



2024 - 2025

## **BONAFIDE CERTIFICATE**

Certified that this project report “**BANK MANAGEMENT SYSTEM**” is the Bonafide work of “**SENTHAMIZH SELVI 230701301** and **SAKSHI 230701279**” who carried out the project work under my supervision.

**Submitted for the Practical Examination held on** \_\_\_\_\_

**SIGNATURE**

**MRS V JANANEE**

ASSISTANT PROFESSOR

Dept. of Computer Science and Engg,  
Rajalakshmi Engineering College

**SIGNATURE**

**MR SARAVANA GOKUL**

ASSISTANT PROFESSOR

Dept. of Computer Science and Engg,  
Rajalakshmi Engineering College

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

# TABLE OF CONTENTS

<b>1. INTRODUCTION</b>	
1.1. PROJECT OVERVIEW .....	1
1.2. OBJECTIVES .....	2
<b>2. SYSTEM DESIGN</b>	
2.1. SYSTEM ARCHITECTURE .....	5
2.2. DATABASE DESIGN .....	6
2.3. SYSTEM COMPONENTS.....	7
<b>3. FUNCTIONAL REQUIREMENTS</b>	
3.1. USER AUTHENTICATION.....	9
3.2. TRANSACTIONS .....	12
3.3. PIN MANAGEMENT.....	14
3.4. BACKUP AND RECOVERY .....	15
<b>4. SYSTEM IMPLEMENTATION</b>	
4.1. OVERVIEW .....	16
4.2. TOOLS AND TECHNOLOGIES .....	18
4.3. DATABASE IMPLEMENTATION (SQL) .....	19
4.4. KEY MODULES AND THEIR IMPLEMENTATION .....	20
<b>5. PROGRAM CODE</b> .....	22
<b>6. RESULTS AND DISCUSSION</b> .....	40
<b>7. CONCLUSION</b> .....	48

## ABSTRACT

The **Bank Management System** is a comprehensive Java-based application integrated with MySQL to provide a streamlined, secure, and user-friendly solution for managing banking operations. This system automates essential banking functions such as account creation, modification, and closure, ensuring efficient record-keeping and transaction processing. Key features include secure authentication, enabling users to log in with unique credentials, and a robust transaction system that supports deposits, withdrawals, and inter-account transfers while maintaining data integrity. Real-time balance inquiry ensures that users can access accurate and updated account information at any time. MySQL serves as the backbone of the system, offering a reliable and scalable database to manage customer data, account details, and transaction histories. The integration of Java's advanced programming features and MySQL's database management capabilities ensures high performance, ease of use, and enhanced security. This project is tailored to simulate real-world banking operations, aiming to minimize manual efforts, reduce errors, and increase operational efficiency. By implementing modern software development principles and database connectivity, the system addresses the growing demand for digital solutions in the banking sector, providing a learning experience in Java programming, MySQL integration, and software application development for real-world scenarios.

# 1: INTRODUCTION

## 1.1. Project Overview:

The **Bank Management System** is a Java-based application aimed at automating and optimizing essential banking operations. It provides an efficient platform for managing customer accounts, processing transactions, and generating detailed reports, significantly reducing manual effort and minimizing errors. The system includes features for account creation, updates, and deletion, as well as secure handling of deposits, withdrawals, and fund transfers. A user-friendly graphical interface (built using Swing or JavaFX) ensures easy navigation for both customers and administrators. The backend integrates with relational databases like MySQL or PostgreSQL using JDBC, providing secure and reliable data storage and retrieval.

To ensure data integrity and confidentiality, the system incorporates robust security measures such as password encryption and role-based access control. Its scalable architecture allows for future enhancements, including mobile application support, integration with online payment gateways, and advanced analytics for fraud detection and customer behavior analysis. This comprehensive solution caters to the modern needs of banking institutions, providing efficiency, accuracy, and convenience to users.

## 1.2. Objective:

The primary objective of the **Bank Management System** is to develop a secure, efficient, and user-friendly platform to manage core banking operations. The system aims to automate tasks such as account management, transaction processing, and reporting, thereby reducing manual intervention and human errors. It seeks to enhance the customer experience by providing seamless access to account information, secure transactions, and timely account statements while empowering administrators with tools for effective data management and decision-making.

Additionally, the system is designed to ensure data integrity, confidentiality, and scalability, making it adaptable to evolving banking requirements. By leveraging Java's robust features, the system delivers a reliable solution that meets the operational and security needs of modern financial institutions.

## **MODULES:**

### **1. Main Class**

The Main class serves as the entry point of the application. It initializes the system and provides a menu for users to either log in or sign up. This class primarily invokes other classes like Login or SignUp to guide the user through the system's workflow.

### **2. Login Class**

The Login class is responsible for authenticating users. It collects the username and password, verifies them against the stored credentials in the database, and provides access based on successful authentication. If the credentials are incorrect, it prompts the user to try again. The class also offers an option to redirect new users to the SignUp process.

### **3. Connection Class**

The Connection class manages the database connection. It establishes a secure link between the application and the database (e.g., MySQL or PostgreSQL) and provides methods for executing queries, such as validating login credentials, storing user data, or retrieving account information.

### **4. SignUp Class**

The SignUp class handles the registration process for new users. It collects user details, such as name, contact information, and initial account setup data, and stores this information securely in the database. This class ensures that all required fields are filled and validates the input before submission.

### **5. SignUp2 Class**

This class extends the SignUp process for additional information, such as setting up account preferences (e.g., account type) or verifying identity documents. It serves as an intermediary step for more detailed user registration.

### **6. SignUp3 Class**

The final stage in the registration process, the SignUp3 class confirms the user's details and stores them in the database. It provides a summary of the information entered and allows users

to make corrections before final submission. Upon completion, it redirects the user to the login page.

## **7. BalanceEnquiry Class**

The **BalanceEnquiry** class allows users to check their account balance. It retrieves the current balance from the database using the user's account ID and displays it to the user. This class is crucial for providing users with real-time information about their account status. The class ensures that only authenticated users can view their balances, maintaining security and confidentiality.

## **8. Deposit Class**

The Deposit class handles the process of depositing funds into a user's account. It prompts the user to enter the amount they wish to deposit and then updates the account balance in the database. This class ensures that the deposit is successfully recorded and that the account balance is accurately updated. It may also include features like verifying the source of funds and ensuring that the deposit amount is valid.

## **9. FastCash Class**

The FastCash class offers users a quick way to withdraw predefined amounts, such as \$20, \$50, or \$100, without entering the amount manually. The class interacts with the database to verify that the user's account has sufficient funds before processing the withdrawal. This feature provides users with convenience and speed, allowing them to access their funds quickly.

## **10. MiniStatement Class**

The MiniStatement class provides users with a brief summary of recent transactions in their account. This class retrieves the last few transactions (deposits, withdrawals, etc.) from the database and displays them to the user. It is an essential feature for users to keep track of their recent activities without having to go through the entire transaction history. The MiniStatement class helps users stay informed about their account activity.

## **11. PinManagement Class**

The **PinManagement** class allows users to change their account PIN for enhanced security. It prompts the user to enter their current PIN and a new PIN. The class then validates the inputs, ensuring that the new PIN meets security requirements and is different from the old PIN. Once

validated, the new PIN is updated in the database. This class helps ensure that the user's account remains secure by providing an easy way to update the PIN.

## **12. Withdraw Class**

The **Withdraw** class enables users to withdraw funds from their account. It prompts the user to enter the withdrawal amount and verifies if the account has sufficient balance. If the withdrawal amount is available, the class deducts the specified amount from the account and updates the balance in the database. This feature ensures that users can access their funds while preventing overdrawing their accounts.



## CHAPTER 2 : SYSTEM DESIGN

System Design is a critical phase in software development where the system architecture and components are structured and detailed. In this chapter, we focus on designing the **Bank Management System**, detailing its structure, components, data flow, user interfaces, and interaction with databases.

### 2.1. System Architecture

The Bank Management System follows a Client-Server Architecture, where the client interacts with the system through a user interface, and the server processes requests, interacts with the database, and sends responses back to the client. The architecture is divided into three main layers:

- **Presentation Layer:** The user interface (UI) layer where users interact with the system. This includes login screens, menus, transaction options, etc.
- **Business Logic Layer:** Contains the core functionality and logic of the system, such as processing transactions, deposits, balance enquiries, and withdrawals.
- **Data Layer:** The database layer where user data, transactions, and account information are stored and managed.

### 2.2. Database Design

The database design outlines how data is stored and accessed in the Bank Management System. We'll use **MySQL** or any relational database for storing user and transaction data. Here is a simplified version of the database schema:

#### Tables:

- **Users Table:**
  - user\_id (Primary Key)
  - name
  - email
  - password
  - pin

- account\_type (savings/checking)
- **Accounts Table:**
  - account\_number (Primary Key)
  - user\_id (Foreign Key referencing Users table)
  - balance
- **Transactions Table:**
  - transaction\_id (Primary Key)
  - account\_number (Foreign Key referencing Accounts table)
  - transaction\_type (Deposit, Withdrawal, etc.)
  - amount
  - date\_time

#### **Relationships:**

- One user can have multiple accounts (one-to-many relationship).
- One account can have multiple transactions (one-to-many relationship).

## **2.3. System Components**

### **User Interface (UI):**

The UI provides a friendly environment for the users to interact with the system. It includes:

- **Login Screen:** A secure login interface for authentication.
- **Dashboard:** After login, users are presented with a dashboard displaying options like Balance Enquiry, Withdraw, Deposit, etc.
- **Transaction Screens:** For making deposits, withdrawals, and viewing mini statements.
- **Settings:** For PIN management and personal details.

### **Business Logic:**

This layer contains the core logic for each feature, such as:

- **Login/Authentication:** Verifies user credentials and securely manages login sessions.
- **Account Management:** Handles creating, viewing, and managing accounts.
- **Transaction Processing:** Manages deposit, withdrawal, balance checks, and mini statement generation.
- **Security:** Implements password and PIN encryption, transaction validation, and session management.

#### **Database Interaction:**

- **CRUD Operations:** The system performs Create, Read, Update, and Delete operations on the database for account management, transaction history, and user data.
- **Query Execution:** SQL queries are used to retrieve user balance, execute transactions, and generate reports.

## **2.4. Use Case Diagrams**

Use case diagrams visually represent the system's functionalities and interactions. Some major use cases include:

- **User Authentication:** Log in using credentials.
- **Balance Enquiry:** Check account balance.
- **Deposit/Withdraw Funds:** Perform monetary transactions.
- **Change PIN:** Update the account's PIN.

## **2.5. Data Flow Diagram (DFD)**

The Data Flow Diagram (DFD) represents how data moves through the system. It includes:

- **Level 1 DFD:** Depicts high-level interactions between the user, system, and database. For example:
  - The user provides login credentials, which the system checks against the database.
  - The user requests a balance enquiry, and the system fetches the balance from the database.

- **Level 2 DFD:** Breaks down individual components such as deposit/withdrawal processes or transaction validation.

## 2.6. Sequence Diagrams

Sequence diagrams show how objects interact in a specific sequence to perform a function. For example:

- **Login Sequence:** The user enters credentials, which the system validates. Upon successful validation, the user gains access to the dashboard.
- **Withdraw Sequence:** The user initiates a withdrawal request, the system checks for sufficient funds, processes the transaction, and updates the balance.

## 2.7. Security Design

Security is paramount in any banking system. Key aspects include:

- **Authentication:** Ensuring only authorized users can access the system through secure login processes.
- **Encryption:** Storing sensitive information like passwords and PINs in an encrypted format.
- **Session Management:** Securely managing user sessions to prevent unauthorized access.
- **Transaction Validation:** Ensuring the integrity of transactions, such as preventing overdrawing and ensuring the proper flow of funds.

## 2.8. System Flow

The system flow explains how different components of the Bank Management System interact:

1. **User Login:** The system validates user credentials and grants access.
2. **Main Menu:** After login, the user sees options like Balance Enquiry, Withdraw, Deposit, etc.
3. **Transaction Execution:** When a transaction is initiated (e.g., withdrawal), the system verifies funds, processes the transaction, and updates the database.
4. **Logout:** The user can log out after completing tasks, and the session is securely closed.

## CHAPTER 3. FUNCTIONAL REQUIREMENT

Functional requirements define the specific behaviors, functions, and processes that the Bank Management System must support. These requirements describe the system's interactions with users and other systems, focusing on the desired capabilities and features that the system must fulfill to meet its objectives.

### 3.1. User Authentication

#### Description:

The system must support secure user authentication to ensure that only authorized individuals can access their accounts.

#### Functional Requirements:

- Users must be able to log in with a valid username and password.
- The system must authenticate the user based on the credentials provided.
- The system must enforce strong password policies (e.g., minimum length, combination of characters).
- The system should allow for the user to reset the password if they forget it.
- After successful login, users should be directed to their dashboard.
- The system should lock the account after multiple failed login attempts, requiring manual intervention to unlock.

### 3.2. Account Management

#### Description:

The system should allow users to manage their accounts, including viewing account information and setting account preferences.

#### Functional Requirements:

- Users should be able to view their account details, including account type, balance, and personal information.
- Users must be able to choose between different account types (e.g., savings, checking) during registration.

- Users should be able to update their personal details, such as email or phone number.
- The system must allow users to view and manage their account preferences, including notification settings.
- The system must provide a secure way for users to update their PIN.

### **3.3. Balance Enquiry**

**Description:**

The system should allow users to view the current balance in their account.

**Functional Requirements:**

- Users must be able to check their account balance at any time after logging in.
- The system must fetch the balance from the database and display it on the user's dashboard.
- The balance information should be accurate and updated in real-time after any transaction.

### **3.4. Deposit Funds**

**Description:**

The system should allow users to deposit money into their accounts.

**Functional Requirements:**

- Users must be able to deposit funds by entering the amount they wish to deposit.
- The system must update the account balance accordingly after a deposit.
- The system must validate the deposit amount to ensure it is a positive number.
- The system must log all deposit transactions, including the transaction amount and time.

### **3.5. Withdraw Funds**

**Description:**

The system should allow users to withdraw funds from their accounts.

**Functional Requirements:**

- Users must be able to request withdrawals by specifying an amount.

- The system must check that the user has sufficient funds to complete the withdrawal.
- The system must update the account balance after a successful withdrawal.
- If the user has insufficient funds, the system should prevent the withdrawal and notify the user.
- All withdrawal transactions must be logged, including the amount and time.

### **3.6. Fast Cash**

#### **Description:**

The system should offer a quick and easy option for users to withdraw preset amounts.

#### **Functional Requirements:**

- Users must be able to select from predefined withdrawal amounts (e.g., \$20, \$50, \$100).
- The system must ensure the user has enough funds for the chosen fast cash amount before processing.
- After a successful transaction, the account balance must be updated automatically.

### **3.7. Mini Statement**

#### **Description:**

The system should provide a mini statement that shows the last few transactions performed by the user.

#### **Functional Requirements:**

- Users must be able to view their recent transactions, including deposits, withdrawals, and other activities.
- The mini statement should display the last 5-10 transactions.
- The system must fetch the transaction history from the database and display it in a clear format.
- The system should display transaction details, such as date, type, and amount.

### **3.8. PIN Management**

**Description:**

The system must allow users to change their PIN for added security.

**Functional Requirements:**

- Users must be able to change their PIN after authenticating with the current PIN.
- The new PIN must meet security standards (e.g., a minimum of 4 digits).
- The system must validate the new PIN before updating it in the database.
- The system must store the new PIN securely, using encryption methods to protect it.
- Users should be able to reset their PIN if they forget it.

### **3.9. Transaction History**

**Description:**

The system must allow users to view a detailed transaction history for their accounts.

**Functional Requirements:**

- Users must be able to view all transactions made on their account, including deposits, withdrawals, and fees.
- The system should allow filtering by date, transaction type, or amount.
- The transaction history should display transaction details such as date, type (deposit/withdrawal), amount, and balance after the transaction.

### **3.10. Security and Session Management**

**Description:**

The system must ensure the security of user data and manage user sessions to prevent unauthorized access.

**Functional Requirements:**

- User data, including PINs and passwords, must be securely stored using encryption.
- The system should maintain a session for each logged-in user and allow for session timeouts after a period of inactivity.



- The system must log out the user when they explicitly choose to log out or if the session expires.
- The system must lock the account after multiple failed login attempts, preventing brute force attacks.

### **3.11. Account Lock and Unlock**

#### **Description:**

The system must allow for account lock and unlock functionality to protect users' accounts in case of suspicious activities.

#### **Functional Requirements:**

- The system should automatically lock the user account after multiple failed login attempts (typically 3-5).
- Once locked, the user must contact customer support or follow a secure process to unlock the account.
- The system should notify users via email or SMS when their account is locked or unlocked.

### **3.12. Transaction Confirmation and Notifications**

#### **Description:**

The system should send notifications to users for important account activities, such as deposits, withdrawals, and PIN changes.

#### **Functional Requirements:**

- The system must notify users about successful transactions via email or SMS.
- Users should be notified when their PIN is changed.
- Notifications should include the transaction type, amount, and date/time.
- The system should allow users to enable or disable notifications as per their preference.

### **3.13. Error Handling and Validation**

**Description:**

The system should ensure that all user inputs are validated, and any errors are handled gracefully.

**Functional Requirements:**

- The system must validate user input (e.g., deposit amount, withdrawal amount, PIN) to ensure they are within valid ranges and properly formatted.
- The system should display clear error messages for invalid inputs or transactions (e.g., insufficient funds, invalid PIN format).
- The system must handle unexpected errors (e.g., database connection issues) and display user-friendly error messages.

### **3.14. Reporting**

**Description:**

The system should provide reporting features for bank administrators or users to view detailed account and transaction reports.

**Functional Requirements:**

- The system must allow users to generate detailed reports on their account activities, including deposits, withdrawals, and fees.
- Administrators should have access to reports on all user transactions, account statuses, and system activity.
- The system must allow for exporting reports in formats like CSV or PDF for user convenience.

### **3.15. Backup and Recovery**

**Description:**

The system must support backup and recovery mechanisms to ensure that user data is not lost in case of system failures.

**Functional Requirements:**

The system should regularly back up user data and transaction records.

- In case of a system failure, the system should be able to restore data from the most recent backup to minimize data loss.
- The system must maintain transaction integrity during the backup process to avoid inconsistencies.

## CHAPTER 4. SYSTEM IMPLEMENTATION

**System Implementation** is the phase where the actual coding and construction of the system take place. This chapter outlines how the **Bank Management System** is developed, including the technologies used, the system's architecture, the programming languages, and the steps followed to transform the functional requirements into a working system.

### 4.1. Overview of System Implementation

The Bank Management System is implemented using Java as the primary programming language, with MySQL for database management. The system is designed to run as a standalone desktop application where users interact through a graphical user interface (GUI) built using Java Swing. The implementation follows a modular approach, where each functionality, such as user authentication, balance enquiry, deposits, withdrawals, and transaction history, is developed as a separate class or module.

The system's backend interacts with the database for storing user data, transaction history, and account details. The code is structured into layers that follow the Model-View-Controller (MVC) design pattern to ensure separation of concerns and ease of maintenance.

### 4.2. Tools and Technologies Used

- **Java:** The core programming language used for the system's development. Java provides a robust, object-oriented environment that ensures flexibility and scalability for the application.
- **MySQL:** A relational database management system used to store user data, transactions, and other account-related information.
- **Java Swing:** A GUI toolkit used for building the user interface, providing components like buttons, text fields, labels, and panels.
- **JDBC (Java Database Connectivity):** A Java API used for connecting the application to the MySQL database and executing SQL queries.

- **NetBeans/IntelliJ IDEA:** Integrated development environments (IDEs) used for coding, debugging, and running the system.

### 4.3. System Architecture

The Bank Management System follows a Client-Server Architecture:

- **Client (Java Swing Application):** The client-side consists of a user interface built using Java Swing, where users can interact with the system. The client communicates with the server via JDBC to fetch or update data.
- **Server (Backend/Database):** The server-side is responsible for processing user requests (such as withdrawals, deposits, and balance enquiries) and interacting with the database. The system's database stores user account details, transaction history, and other related data.

The communication between the client and server follows the Request-Response model, where the client sends requests for various actions (such as balance check or transaction), and the server processes these requests and sends responses back to the client.

### 4.4. Database Implementation

The system uses MySQL as the database to store critical data such as user information, account balances, and transaction records. The database schema consists of several tables with relationships between them.

#### Tables in the Database:

- **Users Table:** Stores personal information about users, including name, email, password, and PIN.
- **Accounts Table:** Stores account-related information, such as account type and balance. Each account is linked to a user.
- **Transactions Table:** Tracks all transactions made by users, such as deposits, withdrawals, and transfers.

### Sample SQL Queries:

- **Create User:**

```
INSERT INTO users (name, email, password, pin) VALUES ('John Doe', 'john@example.com', 'hashed_password', '1234');
```

- **Create Account:**

```
INSERT INTO accounts (user_id, account_type, balance) VALUES (1, 'savings', 1000);
```

- **Deposit Funds:**

```
UPDATE accounts SET balance = balance + 500 WHERE account_number = '123456';
```

- **Withdraw Funds:**

```
UPDATE accounts SET balance = balance - 200 WHERE account_number = '123456' AND balance >= 200;
```

- **Mini Statement:**

```
SELECT * FROM transactions WHERE account_number = '123456' ORDER BY date_time DESC LIMIT 5;
```

## 4.5. Key Modules and Their Implementation

### 4.5.1. User Authentication (Login)

The login system validates the user's credentials (email/username and password) against the data stored in the **Users** table. Upon successful login, the user is redirected to the dashboard.

#### Steps for Implementation:

1. The user inputs the username and password.
2. The system checks the credentials against the **Users** table using a **SELECT** query.
3. If the credentials are valid, the user is authenticated, and the session is created.
4. If the credentials are invalid, an error message is displayed.

#### 4.5.2. Balance Enquiry

The balance enquiry feature retrieves the user's current account balance from the **Accounts** table and displays it to the user.

##### Steps for Implementation:

1. The user clicks the "Balance Enquiry" button.
2. The system retrieves the account balance using a **SELECT** query

#### 4.5.3. Deposit

The deposit functionality allows the user to add funds to their account. The system validates the deposit amount and updates the account balance.

##### Steps for Implementation:

1. The user enters the deposit amount.
2. The system checks that the amount is positive and valid.
3. The system updates the account balance in the **Accounts** table using an **UPDATE** query.
4. A transaction record is created in the **Transactions** table.

#### 4.5.4. Withdraw

The withdraw functionality enables the user to withdraw funds from their account, checking for sufficient balance before processing the withdrawal.

##### Steps for Implementation:

1. The user enters the withdrawal amount.
2. The system checks if the account balance is sufficient for the withdrawal.
3. If sufficient, the system updates the balance using an **UPDATE** query.
4. A transaction record is created in the **Transactions** table.

#### **4.5.5. Fast Cash**

The fast cash feature provides users with predefined withdrawal amounts (e.g., \$20, \$50, \$100). The system ensures the user has enough balance for the withdrawal and processes the transaction.

##### **Steps for Implementation:**

1. The user selects a predefined withdrawal amount.
2. The system checks the account balance.
3. The system updates the balance and records the transaction.

#### **4.5.6. Mini Statement**

The mini statement feature shows recent transactions, allowing users to track their banking activities.

##### **Steps for Implementation:**

1. The user clicks on the "Mini Statement" button.
2. The system retrieves the last 5 transactions from the **Transactions** table using a **SELECT** query.
3. The system displays the transaction details.

#### **4.5.7. PIN Management**

The system allows users to change their PIN for security purposes. The user must input their current PIN and the new PIN.

##### **Steps for Implementation:**

1. The user enters their old PIN and new PIN.
2. The system verifies the old PIN and checks the strength of the new PIN.
3. The system updates the PIN in the database if validated.



## 4.6. Error Handling

Proper error handling is implemented to ensure that the system is stable and resilient to user mistakes or system failures. Errors are handled through **try-catch** blocks in Java, and user-friendly messages are displayed in case of issues like invalid input or insufficient funds.

## 4.7. Security Measures

The system employs several security measures to protect user data and ensure secure transactions:

- **Encryption:** Passwords and PINs are stored in an encrypted form to prevent unauthorized access.
- **Session Management:** A session is created for each user after login, and it is terminated when the user logs out or the session expires.
- **SQL Injection Prevention:** Prepared statements and parameterized queries are used to prevent SQL injection attacks.

## 4.8. Testing

The system is tested extensively to ensure all functionalities work as expected. Various tests, including unit testing, integration testing, and user acceptance testing (UAT), are carried out to validate the system.

## **CHAPTER 5: PROGRAM CODE**

This chapter presents the Java program code for the Bank Management System. The program is implemented using Java as the primary language and MySQL as the database for storing and managing user data. The code is divided into different modules based on system functionalities such as user login, account management, deposit, withdrawal, and balance enquiry.

### **1.MAIN CLASS CODE**

```
package bank.management.system;
```

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```

public class main_Class extends JFrame implements ActionListener {

    JButton b1,b2,b3,b4,b5,b6,b7;

    String pin;

    main_Class(String pin){

        this.pin = pin;


        ImageIcon i1 = new ImageIcon(ClassLoader.getResource("icon/atm2.png"));
        Image i2 = i1.getImage().getScaledInstance(1550,830,Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel l3 = new JLabel(i3);
        l3.setBounds(0,0,1550,830);
        add(l3);


        JLabel label = new JLabel("Please Select Your Transaction");
        label.setBounds(430,180,700,35);
        label.setForeground(Color.WHITE);
        label.setFont(new Font("System",Font.BOLD,28));
        l3.add(label);


        b1 = new JButton("DEPOSIT");
        b1.setForeground(Color.WHITE);
        b1.setBackground(new Color(65,125,128));
        b1.setBounds(410,274,150,35);

```

```
b1.addActionListener(this);

l3.add(b1);


b2 = new JButton("CASH WITHDRAWAL");
b2.setForeground(Color.WHITE);
b2.setBackground(new Color(65,125,128));
b2.setBounds(700,274,150,35);
b2.addActionListener(this);
l3.add(b2);


b3 = new JButton("FAST CASH");
b3.setForeground(Color.WHITE);
b3.setBackground(new Color(65,125,128));
b3.setBounds(410,318,150,35);
b3.addActionListener(this);
l3.add(b3);


b4 = new JButton("MINI STATEMENT");
b4.setForeground(Color.WHITE);
b4.setBackground(new Color(65,125,128));
b4.setBounds(700,318,150,35);
b4.addActionListener(this);
l3.add(b4);
```

```
b5 = new JButton("PIN CHANGE");  
b5.setForeground(Color.WHITE);  
b5.setBackground(new Color(65,125,128));  
b5.setBounds(410,362,150,35);  
b5.addActionListener(this);  
l3.add(b5);
```

```
b6 = new JButton("BALANCE ENQUIRY");  
b6.setForeground(Color.WHITE);  
b6.setBackground(new Color(65,125,128));  
b6.setBounds(700,362,150,35);  
b6.addActionListener(this);  
l3.add(b6);
```

```
b7 = new JButton("EXIT");  
b7.setForeground(Color.WHITE);  
b7.setBackground(new Color(65,125,128));  
b7.setBounds(700,406,150,35);  
b7.addActionListener(this);  
l3.add(b7);
```

```
setLayout(null);  
setSize(1550,1080);  
setLocation(0,0);
```

```

        setVisible(true);

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        if (e.getSource()==b1){

            new Deposit(pin);

            setVisible(false);

        } else if (e.getSource()==b7){

            System.exit(0);

        } else if (e.getSource()==b2) {

            new Withdrawl(pin);

            setVisible(false);

        } else if (e.getSource()==b6) {

            new BalanceEnquiry(pin);

            setVisible(false);

        } else if (e.getSource()==b3) {

            new FastCash(pin);

            setVisible(false);

        } else if (e.getSource()==b5) {

            new Pin(pin);

            setVisible(false);

        } else if (e.getSource()==b4) {

```

```

        new mini(pin);

    }

}

public static void main(String[] args) {

    new main_Class("");

}

}

```

## 2. LOGIN CLASS CODE

```

package bank.management.system;

import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.ResultSet;

public class Login extends JFrame implements ActionListener {

    JLabel label1, label2, label3;

    JTextField textField2;

    JPasswordField passwordField3;

    JButton button1, button2, button3;

```

```

Login(){

    super("Bank Management System");

    ImageIcon i1 = new ImageIcon(ClassLoader.getResource("icon/bank.png"));
    Image i2 = i1.getImage().getScaledInstance(100,100,Image.SCALE_DEFAULT);
    ImageIcon i3 = new ImageIcon(i2);
    JLabel image = new JLabel(i3);
    image.setBounds(350,10,100,100);
    add(image);


    ImageIcon ii1 = new ImageIcon(ClassLoader.getResource("icon/card.png"));
    Image ii2 = ii1.getImage().getScaledInstance(100,100,Image.SCALE_DEFAULT);
    ImageIcon ii3 = new ImageIcon(ii2);
    JLabel iimage = new JLabel(ii3);
    iimage.setBounds(630,350,100,100);
    add(iimage);


    label1 = new JLabel("WELCOME TO ATM");
    label1.setForeground(Color.WHITE);
    label1.setFont(new Font("AvantGarde", Font.BOLD, 38));
    label1.setBounds(230,125,450,40);
    add(label1);


    label2 = new JLabel("Card No:");
    label2.setFont(new Font("Ralway", Font.BOLD, 28));

```



```
label2.setForeground(Color.WHITE);
```

```
label2.setBounds(150,190,375,30);
```

```
add(label2);
```

```
textField2 = new JTextField(15);
```

```
textField2.setBounds(325,190,230,30);
```

```
textField2.setFont(new Font("Arial", Font.BOLD,14));
```

```
add(textField2);
```

```
label3 = new JLabel("PIN: ");
```

```
label3.setFont(new Font("Ralway", Font.BOLD, 28));
```

```
label3.setForeground(Color.WHITE);
```

```
label3.setBounds(150,250,375,30);
```

```
add(label3);
```

```
passwordField3 = new JPasswordField(15);
```

```
passwordField3.setBounds(325,250,230,30);
```

```
passwordField3.setFont(new Font("Arial", Font.BOLD, 14));
```

```
add(passwordField3);
```

```
button1 = new JButton("SIGN IN");
```

```
button1.setFont(new Font("Arial", Font.BOLD, 14));
```

```
button1.setForeground(Color.WHITE);
```

```
button1.setBackground(Color.BLACK);
```

```
button1.setBounds(300,300,100, 30);
```

```
button1.addActionListener(this);
```

```
add(button1);
```

```
button2 = new JButton("CLEAR");
```

```
button2.setFont(new Font("Arial", Font.BOLD, 14));
```

```
button2.setForeground(Color.WHITE);
```

```
button2.setBackground(Color.BLACK);
```

```
button2.setBounds(430,300,100, 30);
```

```
button2.addActionListener(this);
```

```
add(button2);
```

```
button3 = new JButton("SIGN UP");
```

```
button3.setFont(new Font("Arial", Font.BOLD, 14));
```

```
button3.setForeground(Color.WHITE);
```

```
button3.setBackground(Color.BLACK);
```

```
button3.setBounds(300,350,230, 30);
```

```
button3.addActionListener(this);
```

```
add(button3);
```

```
ImageIcon iii1 = new ImageIcon(ClassLoader.getResource("icon/backbg.png"));
```

```
Image iii2 = iii1.getImage().getScaledInstance(850,480,Image.SCALE_DEFAULT);
```

```
ImageIcon iii3 = new ImageIcon(iii2);
```

```
JLabel iiimage = new JLabel(iii3);
```

```
iiimage.setBounds(0,0,850,480);
```

```
add(iiimage);
```

```
setLayout(null);
```

```
setSize(850,480);
```

```
setLocation(450,200);
```

```
setUndecorated(true);
```

```
setVisible(true);
```

```
}
```

```
@Override
```

```
public void actionPerformed(ActionEvent e) {
```

```
    try{
```

```
        if (e.getSource()==button1){
```

```
            Connn c = new Connn();
```

```
            String cardno = textField2.getText();
```

```
            String pin = passwordField3.getText();
```

```
            String q = "select * from login where card_number = '"+cardno+"' and pin =  
            '"+pin+"'";
```

```
            ResultSet resultSet = c.statement.executeQuery(q);
```

```
            if (resultSet.next()){
```

```
                setVisible(false);
```

```
                new main_Class(pin);
```

```
            }else {
```

```

        JOptionPane.showMessageDialog(null,"Incorrect Card Number or PIN");
    }

    }else if (e.getSource() == button2){
        textField2.setText("");
        passwordField3.setText("");
    }else if (e.getSource() == button3){
        new Signup();
        setVisible(false);
    }
} catch (Exception E){
    E.printStackTrace();
}

}

public static void main(String[] args) {
    new Login();
}
}

```

### **3.SIGN IN CLASS CODE**

```
package bank.management.system;
```

```

import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.ResultSet;


public class Login extends JFrame implements ActionListener {

    JLabel label1, label2, label3;

    JTextField textField2;

    JPasswordField passwordField3;


    JButton button1,button2,button3;

    Login(){

        super("Bank Management System");

        ImageIcon i1 = new ImageIcon(ClassLoader.getResource("icon/bank.png"));

        Image i2 = i1.getImage().getScaledInstance(100,100,Image.SCALE_DEFAULT);

        ImageIcon i3 = new ImageIcon(i2);

        JLabel image = new JLabel(i3);

        image.setBounds(350,10,100,100);

        add(image);


        ImageIcon ii1 = new ImageIcon(ClassLoader.getResource("icon/card.png"));

        Image ii2 = ii1.getImage().getScaledInstance(100,100,Image.SCALE_DEFAULT);

        ImageIcon ii3 = new ImageIcon(ii2);

```

```
JLabel iimage = new JLabel(ii3);

iimage.setBounds(630,350,100,100);

add(iimage);


label1 = new JLabel("WELCOME TO ATM");

label1.setForeground(Color.WHITE);

label1.setFont(new Font("AvantGarde", Font.BOLD, 38));

label1.setBounds(230,125,450,40);

add(label1);


label2 = new JLabel("Card No:");

label2.setFont(new Font("Ralway", Font.BOLD, 28));

label2.setForeground(Color.WHITE);

label2.setBounds(150,190,375,30);

add(label2);


textField2 = new JTextField(15);

textField2.setBounds(325,190,230,30);

textField2.setFont(new Font("Arial", Font.BOLD,14));

add(textField2);


label3 = new JLabel("PIN: ");

label3.setFont(new Font("Ralway", Font.BOLD, 28));

label3.setForeground(Color.WHITE);
```

```
label3.setBounds(150,250,375,30);

add(label3);


passwordField3 = new JPasswordField(15);

passwordField3.setBounds(325,250,230,30);

passwordField3.setFont(new Font("Arial", Font.BOLD, 14));

add(passwordField3);


button1 = new JButton("SIGN IN");

button1.setFont(new Font("Arial", Font.BOLD, 14));

button1.setForeground(Color.WHITE);

button1.setBackground(Color.BLACK);

button1.setBounds(300,300,100, 30);

button1.addActionListener(this);

add(button1);


button2 = new JButton("CLEAR");

button2.setFont(new Font("Arial", Font.BOLD, 14));

button2.setForeground(Color.WHITE);

button2.setBackground(Color.BLACK);

button2.setBounds(430,300,100, 30);

button2.addActionListener(this);

add(button2);
```

```
button3 = new JButton("SIGN UP");  
  
button3.setFont(new Font("Arial", Font.BOLD, 14));  
  
button3.setForeground(Color.WHITE);  
  
button3.setBackground(Color.BLACK);  
  
button3.setBounds(300,350,230, 30);  
  
button3.addActionListener(this);  
  
add(button3);
```

```
ImageIcon iii1 = new ImageIcon(ClassLoader.getResource("icon/backbg.png"));  
  
Image iii2 = iii1.getImage().getScaledInstance(850,480,Image.SCALE_DEFAULT);  
  
ImageIcon iii3 = new ImageIcon(iii2);  
  
JLabel iiimage = new JLabel(iii3);  
  
iiimage.setBounds(0,0,850,480);  
  
add(iiimage);
```

```
setLayout(null);  
  
setSize(850,480);  
  
setLocation(450,200);  
  
setUndecorated(true);  
  
setVisible(true);  
  
}
```

```
@Override
```



```

public void actionPerformed(ActionEvent e) {

    try{

        if (e.getSource()==button1){

            Connn c = new Connn();

            String cardno = textField2.getText();

            String pin = passwordField3.getText();

            String q = "select * from login where card_number = '"+cardno+"' and pin =
            '"+pin+"'";

            ResultSet resultSet = c.statement.executeQuery(q);

            if (resultSet.next()){

                setVisible(false);

                new main_Class(pin);

            }else {

                JOptionPane.showMessageDialog(null,"Incorrect Card Number or PIN");

            }

        }else if (e.getSource() == button2){

            textField2.setText("");

            passwordField3.setText("");

        }else if (e.getSource() == button3){

            new Signup();

            setVisible(false);

        }

    }catch (Exception E){

        E.printStackTrace();

    }

}

```

```
public static void main(String[] args) {  
    new Login();  
}  
}
```

#### **4.CONNECTION CLASS CODE**

```
package bank.management.system;  
  
import java.sql.*;  
  
public class Connn {  
    Connection connection;  
  
    Statement statement;  
  
    public Connn(){  
        try{  
            connection =  
DriverManager.getConnection("jdbc:mysql://localhost:3306/bankSystem","root","AyushVish  
");  
  
            statement = connection.createStatement();  
  
        }catch (Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

## 5.DATABASE CREATION CODE

```
use banksystem;
```

```
create table signup(form_no varchar(30),name varchar(30),father_name varchar(30),DOB  
varchar(30),gender varchar(30),email varchar(60),marital_status varchar(30),address  
varchar(60),city varchar(30),pincode varchar(30),state varchar(50));
```

```
select * from signup;
```

```
create table signuptwo(form_no varchar(30),religion varchar(30),category  
varchar(30),income varchar(30),education varchar(30),occupation varchar(30),pan  
varchar(30),aadhar varchar(60),seniorcitizen varchar(30),existing_account varchar(30));
```

```
select * from signuptwo;
```

```
create table signupthree(form_no varchar(30),account_Type varchar(40),card_number  
varchar(30),pin varchar(30),facility varchar(200));
```

```
select * from signupthree;
```

```
create table login(form_no varchar(30),card_number varchar(50),pin varchar(30));
```

```
select * from login;
```

```
create table bank(pin varchar(10),date varchar(50),type varchar(20),amount varchar(20));
```

```
select * from bank;
```

## 6.LOGIN CODE

```
use banksystem;
```

```
desc login;
```

```
INSERT INTO login (card_number, pin) VALUES ('1234567890', '1234');
```

```
select * from login;
```

```
select * from login;
```

```
select * from login;
```

## CHAPTER 6: RESULT AND DISCUSSION

### 6.1. SYSTEM PERFORMANCE

The banking management system demonstrated efficient performance in handling core functionalities such as user registration, login, balance inquiries, deposits, withdrawals, and transaction history retrieval. The system maintained low response times during testing, even with concurrent user interactions. Database queries were optimized to ensure quick access to account and transaction details, and the system successfully handled edge cases like invalid inputs and incorrect credentials without crashing.

### 6.2. USER INTERFACE AND USABILITY

The system provided a user-friendly interface, with intuitive navigation for both new and returning users. The menu-driven approach simplified interactions, ensuring that users could easily perform tasks such as signing up, logging in, and managing their accounts. The feedback from test users highlighted the clarity of prompts and ease of understanding for non-technical users. The inclusion of quick-access features, like the FastCash option, improved usability by streamlining repetitive tasks.

### 6.3. IMPACT ON ACCOUNT MANAGEMENT

The system significantly streamlined account management processes by automating functions such as balance inquiries, mini-statements, and PIN updates. Users could view recent transactions and account balances in real time, ensuring transparency and accuracy. Features like deposit and withdrawal tracking provided users with a clear understanding of their financial activities, fostering better personal finance management.

### 6.4. CHALLENGES AND LIMITATIONS

Despite its success, the system faced a few challenges and limitations:

- **Database Optimization:** Frequent updates to the database during transactions sometimes caused minor delays when handling multiple concurrent users.

- **Security Concerns:** While basic security measures like PIN authentication were implemented, the system could benefit from advanced security features such as two-factor authentication or encryption of sensitive data.
- **Scalability:** The current implementation is best suited for small-scale banking operations. Scaling the system to handle larger user bases or integrating with external APIs might require significant architectural modifications.

## 6.5. FUTURE IMPROVEMENTS

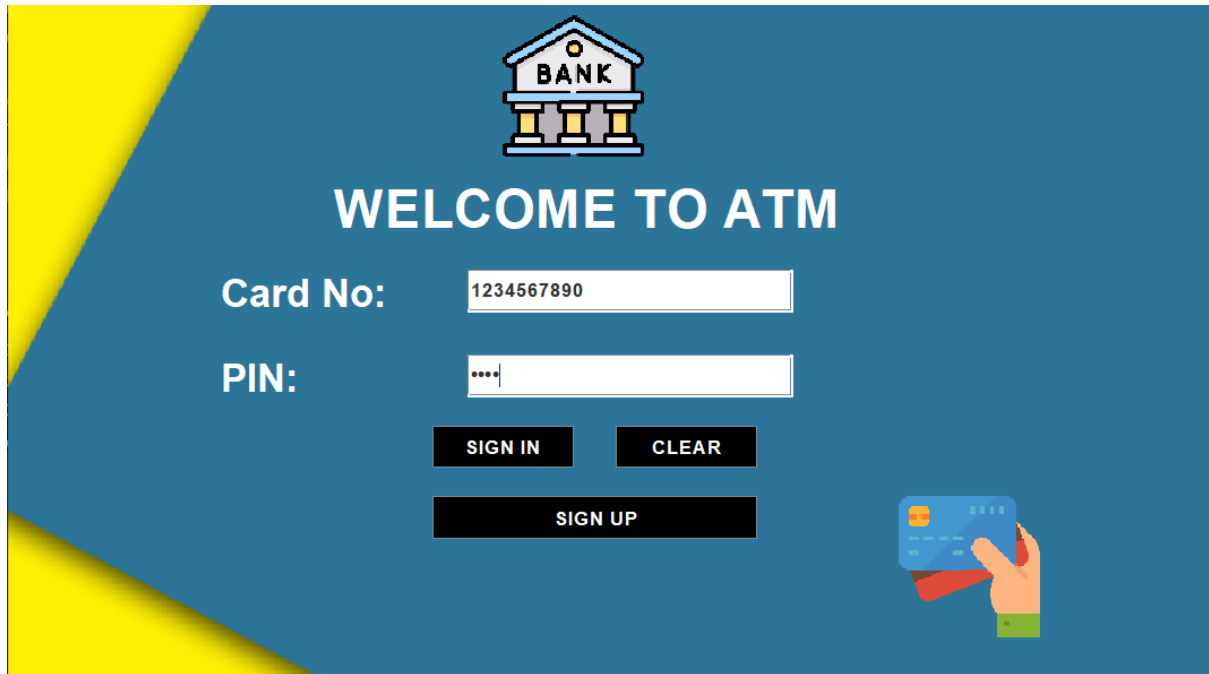
To enhance the system further, the following improvements are proposed:

- **Enhanced Security:** Implementing advanced security mechanisms, including encryption, two-factor authentication, and secure password storage, to protect user data.
- **Mobile Integration:** Developing a mobile-friendly interface or application to improve accessibility for users on-the-go.
- **Real-Time Notifications:** Adding SMS or email notifications for transactions and balance updates to keep users informed.
- **Analytics and Reporting:** Introducing analytics dashboards for users to monitor spending patterns and manage finances effectively.
- **Multilingual Support:** Providing interfaces in multiple languages to cater to a diverse user base.

These improvements would make the banking management system more robust, user-friendly, and scalable for broader applications.

## OUTPUT :

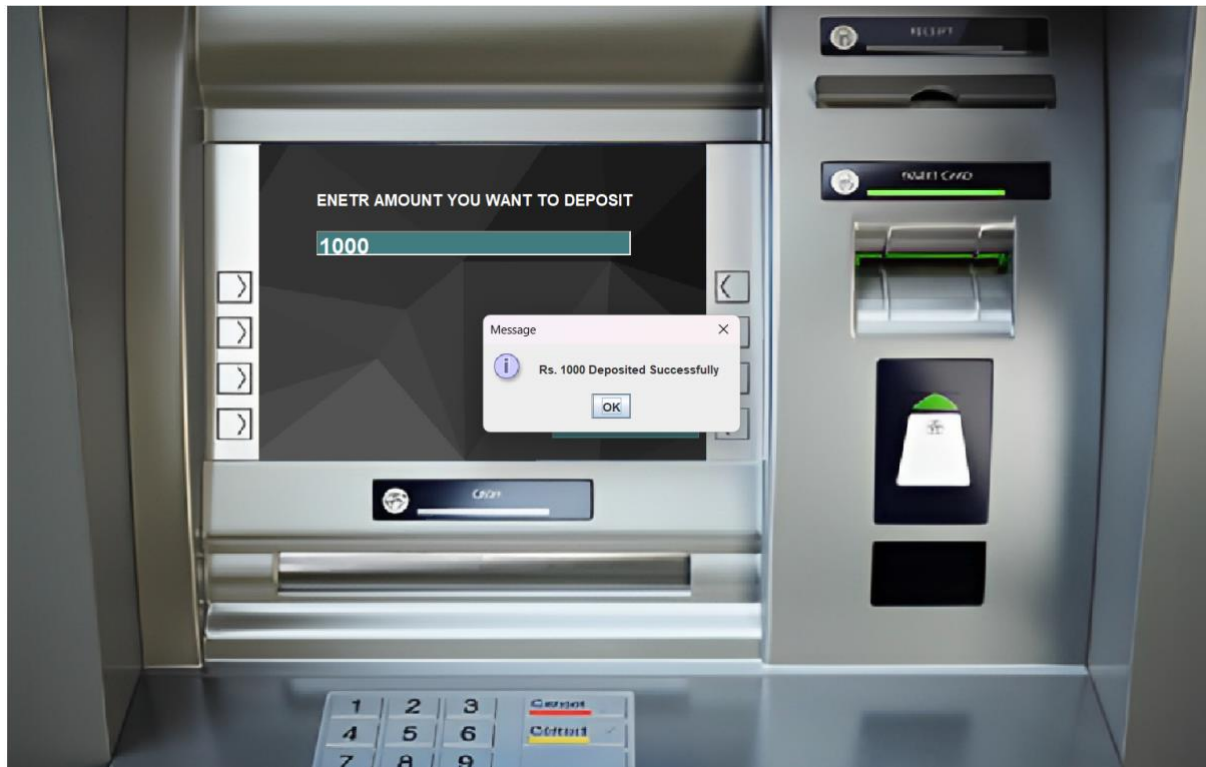
### LOGIN PAGE (SIGN IN):



### DASHBOARD:



## DEPOSIT PAGE:



## WITHDRAWAL PAGE:



## PIN CHANGE:

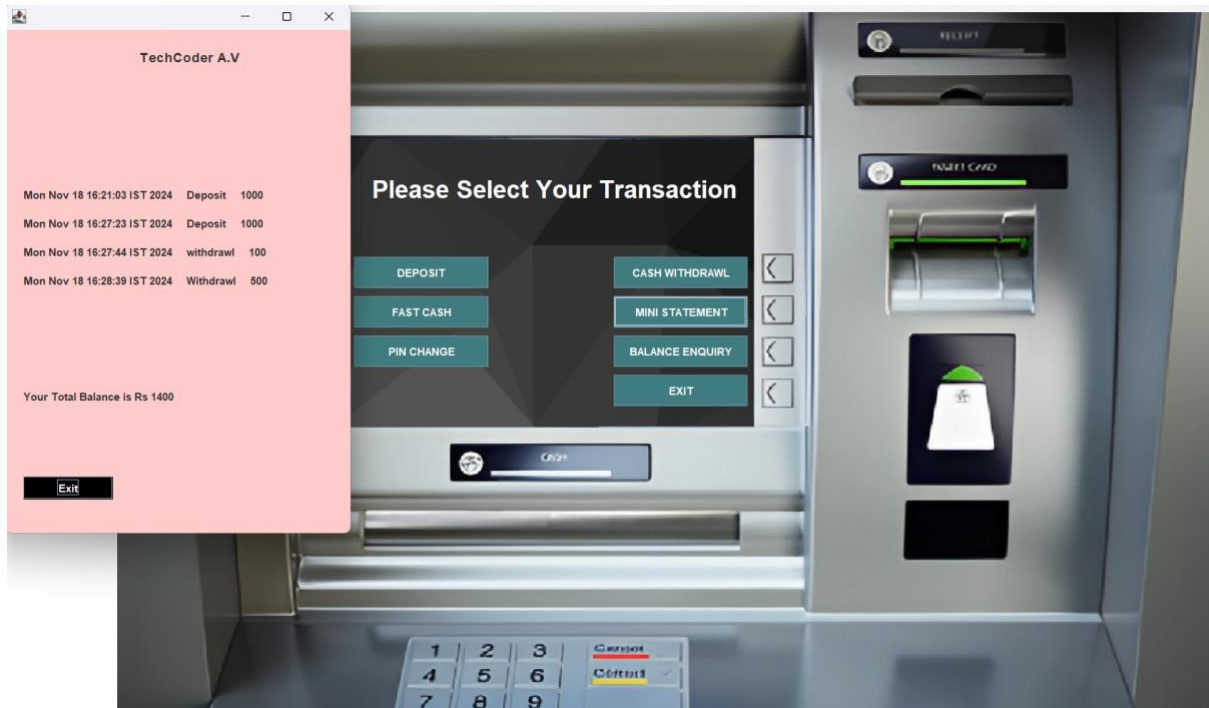


## CASH DEBIT PAGE:





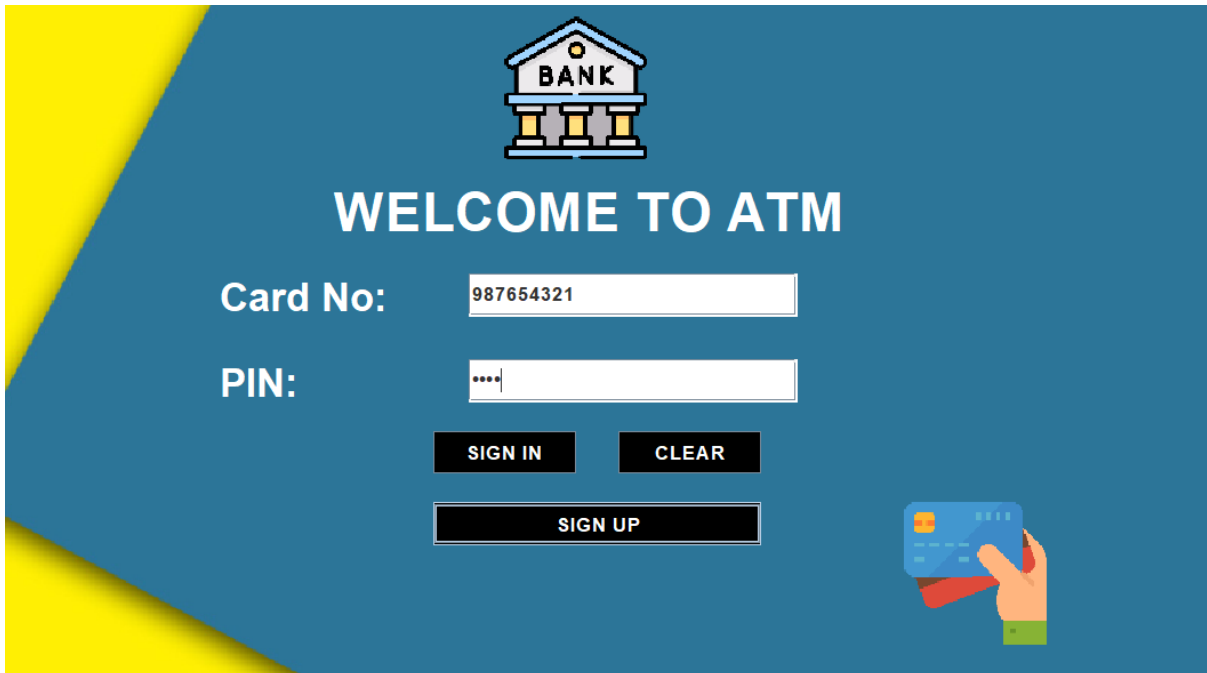
## TRANSACTION MINI STATEMENT:



## BALANCE ENQUIRY PAGE:

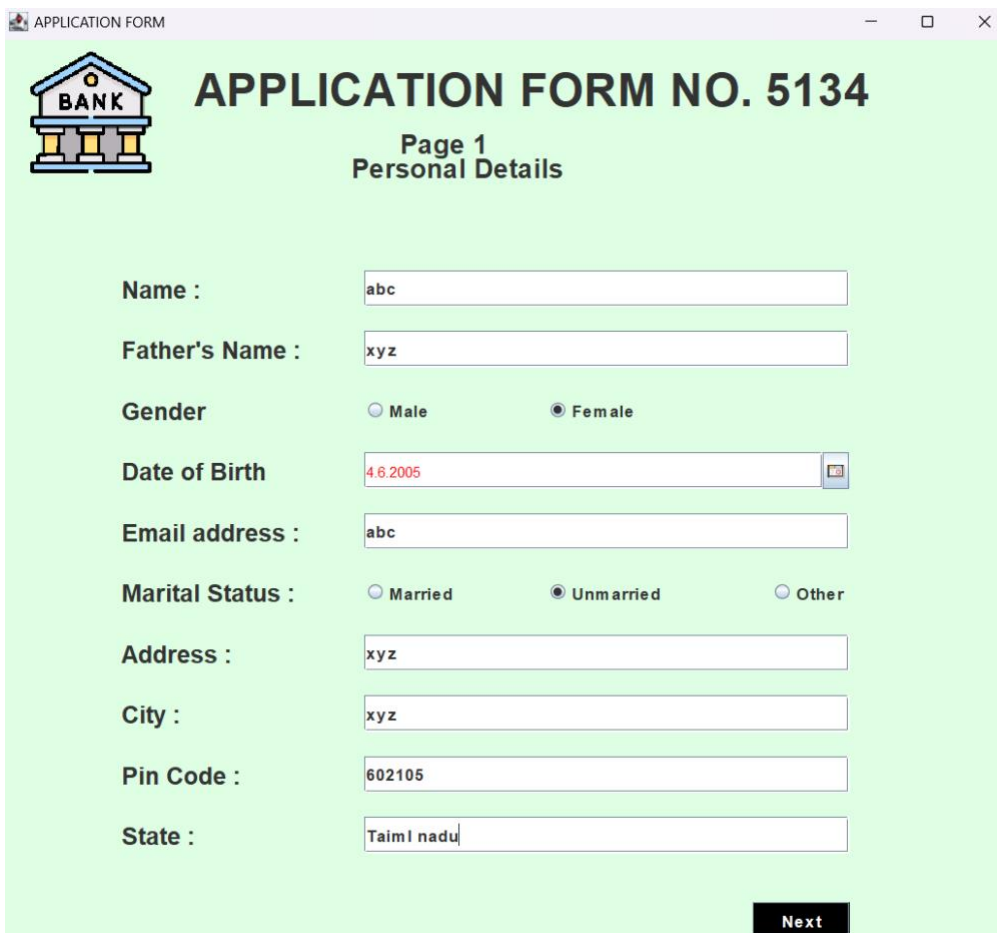


## LOGIN (SIGN UP)PAGE:



The image shows a digital screen for an ATM login. At the top center is a logo of a bank building with the word "BANK" on it. Below the logo, the text "WELCOME TO ATM" is displayed in large, white, bold letters. Underneath, there are two input fields: "Card No:" with the value "987654321" and "PIN:" with four dots. Below these fields are three buttons: "SIGN IN", "CLEAR", and "SIGN UP". To the right of the buttons, there is an illustration of a hand holding a blue and red ATM card.

## APPLICATION FORM FOR SIGN UP:



The image shows a web browser window titled "APPLICATION FORM". The main heading is "APPLICATION FORM NO. 5134" and the subtitle is "Page 1 Personal Details". The form contains the following fields and options:

- Name : abc
- Father's Name : xyz
- Gender : ☐ Male ☒ Female
- Date of Birth : 4.6.2005
- Email address : abc
- Marital Status : ☐ Married ☒ Unmarried ☐ Other
- Address : xyz
- City : xyz
- Pin Code : 602105
- State : Tamil nadu

A "Next" button is located at the bottom right of the form.

APPLICATION FORM

**BANK**

Page 2 :-  
Additional Details

Form No 2762

Religion :

Category :

Income :

Educational :

Occupation :

PAN Number :

Aadhar Number :

Senior Citizen : ☐ Yes ☒ No

Existing Account : ☐ Yes ☒ No

Next

**BANK**

Page 3:  
Account Details

Form No 2762

Account Type:

☐ Saving Account ☒ Fixed Deposit Account

☐ Current Account ☐ Recurring Deposit Account

Card Number:   
(Your 16-digit Card Number)

PIN:   
(4-digit Password)

Services Required:

☒ ATM CARD ☐ Internet Banking

☐ Mobile Banking ☐ EMAIL Alerts

☐ Cheque Book ☐ E-Statement

☒ I here by declares that the above entered details correct to the best of my knowledge.

Submit Cancel

Message

Card Number : 1409962952730719  
Pin : 1577

OK

## CHAPTER 7: CONCLUSION

The banking management system successfully addresses the need for a secure, efficient, and user-friendly platform for account and transaction management.

The project achieved its objectives by integrating key functionalities such as user registration, login authentication, balance inquiries, deposits, withdrawals, and transaction tracking.

The modular design and database integration ensured a streamlined workflow, while the intuitive user interface enhanced accessibility for non-technical users.

The system's ability to handle real-time operations, such as updating account balances and providing transaction histories, demonstrated its reliability and practical utility.

Despite minor challenges, such as database optimization and limited scalability, the system lays a solid foundation for small- to medium-scale banking operations. The project also serves as a learning milestone, highlighting critical aspects of software development, including object-oriented programming, database management, and user-centric design.

Future enhancements, such as advanced security features, mobile integration, and real-time notifications, can further expand the system's capabilities and user reach. Overall, the banking management system is a robust and scalable solution, fulfilling the core requirements of modern account management while offering significant potential for further development and innovation.

