

**AKSHAYAA S 231801007**

BANK MANAGEMENT SYSTEM

**DATA SCIENCE**

**TABLE OF CONTENTS**

**1. INTRODUCTION**

1.1 INTRODUCTION……………………………………………………………….

1.2 OBJECTIVES……………………………………………………………………

1.3 MODULES………………………………………………………………………

**2. SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION…………………………………………………..

2.2 LANGUAGES…………………………………………………………………..

2.2.1 MySQL………………………………………………………....

2.2.2 Java jdk …………………………………………..…

**3. REQUIREMENTS AND ANALYSIS**

3.1 REQUIREMENT SPECIFICATION………………………………….……

3.2 HARDWARE AND SOFTWARE REQUIREMENTS………………….

3.3 DATA DICTIONARY……………………………………………………

**4.PROGRAM CODE ……………………………………………………….…………………**

**5. RESULTS AND DISCUSSIONS……………………………………………..………....**

**6. CONCLUSION…………………………………………………………………..............**

**7. REFERENCES……………………………………………………………………………..**

**ABSTRACT**

The "Bank Management System" is a comprehensive and efficient project designed to streamline the management of customer accounts in a banking environment. Developed using Java with a Swing-based graphical user interface and a MySQL database back-end, the system offers a user-friendly and robust solution for handling customer data and account operations.

Key functionalities include secure login authentication, adding new accounts, updating existing customer details, deleting accounts, and searching for customer information. The system incorporates intuitive interfaces with text fields, buttons, and confirmation dialogs, ensuring ease of use and preventing errors. The database connectivity is managed through Java’s JDBC framework, enabling real-time operations with high reliability and data integrity.

The application ensures consistency and prevents redundancy by leveraging a well-designed database schema with primary and foreign key constraints. Each operation is supported by robust validation mechanisms to maintain data accuracy and provide meaningful feedback to users. The system also features dynamic reporting capabilities to view customer data, making it easier for administrators to access and manage accounts efficiently.

By automating essential banking tasks, the "Bank Management System" reduces manual effort and error rates while enhancing productivity and data organization. This scalable project can be further extended to include advanced features like transaction history tracking, loan management, and integration with online banking services..

**INTRODUCTION**

**1.1 Introduction**

The "Bank Management System" is a software application designed to automate and simplify the essential operations of a banking institution. This system addresses the limitations of traditional manual processes for managing customer accounts, transactions, and data storage, which are prone to inefficiencies and human errors. By digitizing these processes, the system ensures enhanced accuracy, real-time data retrieval, and streamlined management of banking activities.

Developed using Java for the front-end, the system features a graphical user interface (GUI) that is both intuitive and responsive, allowing bank staff to perform tasks with ease. The back-end is powered by MySQL, hosted via XAMPP, to manage the storage and organization of customer and account data. The robust relational database management system (RDBMS) ensures scalability and reliability, making it capable of handling a large volume of records and transactions efficiently.

The Bank Management System includes core functionalities such as customer account creation, updating account details, deleting accounts, and searching for specific customer information. Additionally, secure login authentication prevents unauthorized access, ensuring data privacy and security. Through its well-structured interface, the system simplifies banking operations, enabling staff to focus on customer service rather than administrative tasks.

The primary goal of this system is to optimize banking workflows by reducing manual efforts, minimizing errors, and improving data consistency. By automating key banking tasks, the application enhances operational efficiency and ensures that data is always up-to-date and accessible. Furthermore, the system's ability to generate real-time reports empowers administrators to monitor account activities and make informed decisions.

This project serves as a modern solution for banking institutions transitioning from manual processes to a digital platform, offering improved accuracy, faster workflows, and greater customer satisfaction. Future enhancements could include features such as transaction management, loan processing, integration with online banking platforms, and comprehensive financial reporting for more advanced banking needs.

**1.2 OBJECTIVES**

Objectives

Primary Objectives

1. Create a User Authentication System: Develop a secure and user-friendly login system for administrators and staff to access the bank management features. The system will validate credentials and restrict access to authorized users only.

2. Enable Account Management: Provide functionality for creating, updating, deleting, and searching customer accounts. This includes capturing essential details such as name, account number, contact information, and account balance.

3. Display Account Details: Ensure all account information is displayed in a structured format in the system, making it easy for staff to review and manage.

4. Enable Account Search: Implement a search mechanism that allows staff to quickly locate customer accounts based on parameters like account number, name, or contact information.

Business Objectives

1. Increase Operational Efficiency: Automate manual tasks such as account creation, updates, and deletions to minimize errors and streamline banking workflows.

2. Enhance User Experience: Provide an intuitive interface that ensures banking staff can perform tasks efficiently and with minimal training.

3. Ensure Real-Time Updates: Maintain up-to-date information on customer accounts, enabling staff to view and manage accounts with real-time accuracy.

4. Maintain Data Integrity and Security: Safeguard sensitive customer and account data with encryption and secure database operations to prevent unauthorized access or data loss.

Modules

**Admin Module**

- Login & Dashboard: Implement a secure login system for administrators. The dashboard will provide an overview of customer accounts, recent activities, and key banking metrics such as total deposits and withdrawals.

- Customer Account Management: Admins can create, update, delete, and search for customer accounts. This includes adding account details, modifying balance information, and deactivating accounts.

- User Access Control: Manage banking staff accounts, assigning roles and permissions based on their responsibilities.

- Reporting & Analytics: Generate reports on account activities, such as account creations, updates, and deletions, for auditing and decision-making.

**User Module**

- Login & Access: Staff users can log in to perform operations on customer accounts. The system will ensure that their access is restricted based on their assigned roles.

- Account Creation: Allow staff to add new customer accounts, capturing key details like name, account number, contact information, and initial balance.

- Account Search & Management: Staff can search for accounts using filters such as name, account number, or contact details. They can view, update, or delete accounts as needed.

- Transaction Overview: Provide staff with an overview of transactions associated with specific accounts, such as deposits and withdrawals.

**Account Module**

- Account Database: Maintain a structured repository of customer accounts with details like account ID, name, account type, balance, and contact information.

- Account Updates: Enable modifications to customer account details, including updates to names, contact details, and account balances.

- Account Deletion: Allow for deactivation or deletion of accounts when necessary, with appropriate confirmation to prevent accidental removal.

**Database Module**

- Customer Data Management: Store and manage customer details in a relational database, ensuring data integrity and ease of access.

- Transaction Tracking: Maintain a log of all account-related activities, including account creations, updates, deletions, and balance changes.

- Audit Trail: Record all actions performed by staff and administrators to ensure accountability and transparency.

**Security Module**

- User Authentication: Implement secure login functionality, ensuring only authorized users can access the system. Use role-based access controls to differentiate between admin and staff operations.

-Data Protection: Encrypt sensitive customer and account information in the database to protect against potential breaches.

- Session Management: Monitor active sessions to prevent unauthorized access and ensure timely session terminations.

- Backup & Recovery: Enable periodic backups of account and transaction data to protect against data loss due to system failures or unforeseen events.

**II. SURVEY OF TECHNOLOGY**

**2.1 Software Description**

**Java JDK (Java Development Kit)**

The Java Development Kit (JDK) is a software development environment used to develop Java applications. It provides the necessary tools, libraries, and runtime environment for developing Java programs. The JDK includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and various other tools for Java development.

**Key Features of Java JDK:**

* **Cross-Platform:** Java programs, once written, can be run on any platform that supports the Java Runtime Environment (JRE), making it platform-independent.
* **Object-Oriented:** Java follows the object-oriented programming paradigm, making it easy to structure and maintain code.
* **Robust Libraries:** The JDK comes with a rich set of libraries for handling file I/O, networking, security, GUI development, and much more.
* **Automatic Memory Management:** Java handles memory management through automatic garbage collection, reducing the risk of memory leaks.
* **Multithreading:** Java supports multithreading, which allows multiple threads to run concurrently, improving the performance of applications.

In this project, Java JDK is used as the primary programming language for developing the back-end functionality of the Library Management System. It is used to create the server-side logic for user management, book management, and database interaction.

**XAMPP (Windows Version 8.0.30-0-VS16-Installer)**

XAMPP is an open-source, cross-platform web server solution stack package. It contains Apache, MySQL, PHP, and Perl, providing everything needed to set up a local server environment on a system for web development and database management. In this project, XAMPP is used to host the MySQL database locally, allowing for efficient and easy access to the back-end database.

**Key Features of XAMPP:**

* **Easy Installation:** XAMPP offers a simple, one-click installation for setting up a local server environment with Apache and MySQL.
* **Local Hosting:** Provides local web server capabilities, allowing developers to test and run websites and applications locally before deployment.
* **Cross-Platform:** Available for Windows, macOS, and Linux, making it suitable for various development environments.
* **Preconfigured Software:** Comes preconfigured with all necessary software components, making it ready to use out of the box.

In this project, XAMPP serves as the local server platform to host the MySQL database, ensuring smooth interaction between the Java-based front-end application and the back-end database.

**MySQL 8.4**

MySQL is an open-source relational database management system (RDBMS) based on Structured Query Language (SQL). It is widely used in many applications to store and manage data. MySQL 8.4 is the version used in this project for storing library data such as user information, book details, and transaction records.

**Key Features of MySQL 8.4:**

* **ACID Compliance:** MySQL ensures data integrity through ACID-compliant transactions (Atomicity, Consistency, Isolation, Durability).
* **High Performance:** MySQL is optimized for high-performance querying and can handle large datasets efficiently.
* **Security Features:** MySQL 8.4 includes advanced security features like data encryption, user authentication, and authorization.
* **Data Consistency:** The system supports complex queries, joins, and transactions to maintain data consistency across the database.
* **Scalability:** MySQL can scale to handle large databases and high volumes of queries, making it ideal for growing systems.

In this project, MySQL 8.4 is used to manage all library data, including books, users, and transactions, enabling efficient data retrieval, insertion, and updates.

**MySQL Connector 9.1**

MySQL Connector is a Java library that allows Java applications to connect to a MySQL database. The MySQL Connector 9.1 version is used in this project to enable the Java-based front-end application to communicate with the MySQL database, ensuring smooth data exchange between the application and the database.

**Key Features of MySQL Connector 9.1:**

* **Seamless Integration:** MySQL Connector 9.1 provides seamless integration between Java applications and MySQL databases, allowing for easy communication between the two.
* **JDBC Support:** The connector is based on Java Database Connectivity (JDBC) API, providing standard methods for querying and manipulating MySQL databases from Java.
* **Efficient Data Handling:** It allows for fast data transfer between Java and MySQL, supporting high-performance applications.
* **Platform Independence:** Being a Java-based connector, it works across all platforms that support Java, ensuring compatibility in cross-platform development environments.
* **Security Features:** Supports SSL connections to MySQL databases for secure data transmission.

In this project, MySQL Connector 9.1 is used to establish a secure and efficient connection between the Java-based Library Management System and the MySQL database, enabling operations such as user registration, book management, and transaction logging.

**Integration of Java, XAMPP, MySQL, and MySQL Connector**

This project integrates **Java** for the front-end user interface and back-end logic, **XAMPP** to host the MySQL database, and **MySQL Connector** for database communication. The integration allows for seamless interaction between the front-end (user interface) and back-end (database), where users can register, search for books, and manage transactions, while administrators can add books, view user data, and generate reports. The combination of Java’s power for back-end processing, MySQL’s robust data management, and the ease of local hosting with XAMPP makes the system efficient, reliable, and scalable.

This breakdown describes the technologies used in your **Library Management System** project and how they integrate to provide a smooth, user-friendly, and efficient system.

**III. REQUIREMENTS AND ANALYSIS**

3.1 Requirements Specification

**User Requirements:**

1. User Registration: The system should allow bank staff to create new customer accounts by entering essential details such as name, contact information, account number, and initial balance.

2. Account Management: Bank staff should have the ability to add, update, delete, and search for customer accounts. Updates may include modifying customer information or adjusting account balances.

3. Account Search: The system should allow staff to search for customer accounts by various criteria, such as account number, name, or contact details. Search results should be displayed in a user-friendly table format.

4. User Authentication: The system should require secure login for all users, with role-based access control to ensure that only authorized personnel (e.g., administrators or staff) can access specific features.

5. Real-Time Updates: Changes to customer accounts should reflect immediately in the account database and UI to maintain consistency.

6. Transaction Overview: The system should allow staff to view a summary of transactions related to a specific customer, such as deposits, withdrawals, and balance adjustments.

7. Report Generation: The system should enable administrators to generate reports summarizing account activities, including new account creations, deletions, and modifications, as well as total funds deposited or withdrawn.

8. Data Security: The system should implement robust encryption to protect sensitive data, ensuring compliance with data protection regulations and preventing unauthorized access.

**System Requirements:**

1. Programming Language: The system will be developed using Java for both the front-end and back-end, offering a secure and efficient environment for application development.

2. Database: MySQL will be used as the relational database to store customer account details, transaction logs, and user credentials.

3. Database Connector: MySQL Connector (Java) will be used to connect the Java application to the MySQL database for seamless data operations.

4. Server: XAMPP (Windows version) will be employed to host the MySQL database locally and support smooth communication between the front-end and back-end components.

5. Operating System Compatibility: The system will be compatible with Windows, macOS, and Linux operating systems.

6. Security: The system will use secure authentication mechanisms, encrypted data storage, and session management to safeguard sensitive customer and transaction data.

7. User Interface: The graphical user interface will be designed using Java Swing to provide an intuitive and responsive experience for bank staff and administrators.

8. Backup & Recovery: Periodic backups of the database will be supported to ensure that data is protected against loss due to unforeseen circumstances, with recovery mechanisms in place for quick restoration.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Software Requirements:**

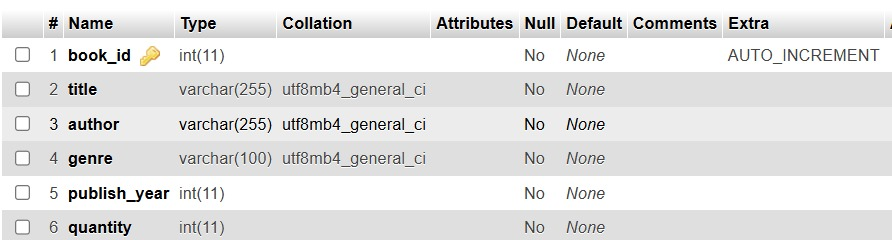
* **Operating System:**
  + **Windows 10/11**, **macOS**, **Linux** (for server and client applications)
* **Front End:**
  + **Java (JDK 8 or higher)**: Used for developing the graphical user interface (GUI) and back-end logic of the system.
* **Back End:**
  + **MySQL 8.4**: Used as the database to store user, book, and transaction data.
  + **MySQL Connector 9.1**: Enables communication between the Java application and the MySQL database.
* **Database Server:**
  + **XAMPP (Windows version 8.0.30-0-VS16)**: Used for setting up the local MySQL server environment.

**Hardware Requirements:**

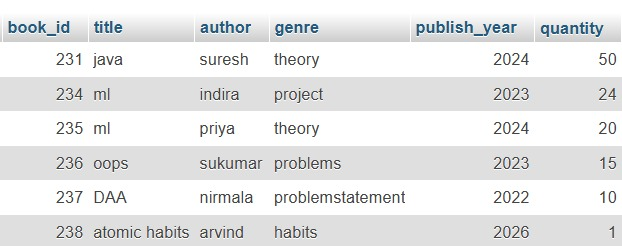
* **System Type:**
  + **Desktop PC or Laptop** with sufficient resources for development and running the system.
* **Processor:**
  + **Intel® Core™ i3-6006U CPU** or equivalent (minimum requirement).
* **Memory:**
  + **4.00 GB RAM** or higher for smooth operation.
* **Storage:**
  + At least **500 MB** of free disk space for the operating system, software, and database storage.
* **Operating System:**
  + **Windows 10/11** or any other supported OS (macOS/Linux).
* **Monitor:**
  + **1024 x 768** resolution or higher to ensure proper display of the graphical interface.
* **Input Devices:**
  + **Keyboard** and **Mouse** for system navigation and interaction.
* **Printer (Optional):**
  + Printer can be optionally used for generating hard copies of reports (though this is not mandatory for basic system functionality).

3.3 DATA DICTIONARY

BOOK TABLE:



Stored added books



**IV. PROGRAM CODE**

**LOGIN PAGE:**

import javax.swing.\*;

import java.awt.event.\*;

import java.sql.\*;

public class LoginPage {

public static void main(String[] args) {

JFrame frame = new JFrame("Bank Login");

JLabel userLabel = new JLabel("User ID:");

JLabel passLabel = new JLabel("Password:");

JTextField userField = new JTextField();

JPasswordField passField = new JPasswordField();

JButton loginButton = new JButton("Login");

userLabel.setBounds(30, 30, 100, 30);

userField.setBounds(150, 30, 150, 30);

passLabel.setBounds(30, 80, 100, 30);

passField.setBounds(150, 80, 150, 30);

loginButton.setBounds(100, 130, 100, 30);

frame.add(userLabel);

frame.add(userField);

frame.add(passLabel);

frame.add(passField);

frame.add(loginButton);

loginButton.addActionListener(e -> {

String username = userField.getText();

String password = new String(passField.getPassword());

try (Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement("SELECT \* FROM users WHERE username=? AND password=?")) {

ps.setString(1, username);

ps.setString(2, password);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

JOptionPane.showMessageDialog(frame, "Login Successful!");

frame.dispose();

MainMenuPage.main(null);

} else {

JOptionPane.showMessageDialog(frame, "Invalid Credentials!");

}

} catch (SQLException ex) {

ex.printStackTrace();

}

});

frame.setSize(400, 250);

frame.setLayout(null);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

**MAIN MENU PAGE :**

import javax.swing.\*;

public class MainMenuPage {

public static void main(String[] args) {

JFrame frame = new JFrame("Main Menu");

JButton addButton = new JButton("Add");

JButton updateButton = new JButton("Update");

JButton deleteButton = new JButton("Delete");

JButton searchButton = new JButton("Search");

addButton.setBounds(50, 30, 100, 30);

updateButton.setBounds(200, 30, 100, 30);

deleteButton.setBounds(50, 80, 100, 30);

searchButton.setBounds(200, 80, 100, 30);

addButton.addActionListener(e -> AddAccount.main(null));

updateButton.addActionListener(e -> UpdateAccount.main(null));

deleteButton.addActionListener(e -> DeleteAccount.main(null));

searchButton.addActionListener(e -> SearchAccount.main(null));

frame.add(addButton);

frame.add(updateButton);

frame.add(deleteButton);

frame.add(searchButton);

frame.setSize(400, 200);

frame.setLayout(null);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

**ADD ACCOUNT:**

import javax.swing.\*;

import java.sql.\*;

public class AddAccount {

public static void main(String[] args) {

JFrame frame = new JFrame("Add Account");

JLabel nameLabel = new JLabel("Name:");

JLabel accNumLabel = new JLabel("Account Number:");

JLabel balanceLabel = new JLabel("Balance:");

JTextField nameField = new JTextField();

JTextField accNumField = new JTextField();

JTextField balanceField = new JTextField();

JButton addButton = new JButton("Add");

nameLabel.setBounds(30, 30, 100, 30);

nameField.setBounds(150, 30, 150, 30);

accNumLabel.setBounds(30, 80, 120, 30);

accNumField.setBounds(150, 80, 150, 30);

balanceLabel.setBounds(30, 130, 100, 30);

balanceField.setBounds(150, 130, 150, 30);

addButton.setBounds(100, 180, 100, 30);

addButton.addActionListener(e -> {

String name = nameField.getText();

String accNum = accNumField.getText();

String balance = balanceField.getText();

try (Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement("INSERT INTO accounts (name, account\_number, balance) VALUES (?, ?, ?)")) {

ps.setString(1, name);

ps.setString(2, accNum);

ps.setDouble(3, Double.parseDouble(balance));

ps.executeUpdate();

JOptionPane.showMessageDialog(frame, "Account Added Successfully!");

} catch (SQLException ex) {

ex.printStackTrace();

}

});

frame.add(nameLabel);

frame.add(nameField);

frame.add(accNumLabel);

frame.add(accNumField);

frame.add(balanceLabel);

frame.add(balanceField);

frame.add(addButton);

frame.setSize(400, 300);

frame.setLayout(null);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

**UPDATE ACCOUNT :**

import javax.swing.\*;

import java.sql.\*;

public class UpdateAccount {

public static void main(String[] args) {

JFrame frame = new JFrame("Update Account");

JLabel idLabel = new JLabel("Account ID:");

JLabel nameLabel = new JLabel("Name:");

JLabel balanceLabel = new JLabel("Balance:");

JTextField idField = new JTextField();

JTextField nameField = new JTextField();

JTextField balanceField = new JTextField();

JButton updateButton = new JButton("Update");

idLabel.setBounds(30, 30, 100, 30);

idField.setBounds(150, 30, 150, 30);

nameLabel.setBounds(30, 80, 100, 30);

nameField.setBounds(150, 80, 150, 30);

balanceLabel.setBounds(30, 130, 100, 30);

balanceField.setBounds(150, 130, 150, 30);

updateButton.setBounds(100, 180, 100, 30);

updateButton.addActionListener(e -> {

String accountId = idField.getText();

String name = nameField.getText();

String balance = balanceField.getText();

try (Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement("UPDATE accounts SET name=?, balance=? WHERE account\_id=?")) {

ps.setString(1, name);

ps.setDouble(2, Double.parseDouble(balance));

ps.setInt(3, Integer.parseInt(accountId));

int rowsUpdated = ps.executeUpdate();

if (rowsUpdated > 0) {

JOptionPane.showMessageDialog(frame, "Account Updated Successfully!");

} else {

JOptionPane.showMessageDialog(frame, "Account ID Not Found!");

}

} catch (SQLException ex) {

ex.printStackTrace();

}

});

frame.add(idLabel);

frame.add(idField);

frame.add(nameLabel);

frame.add(nameField);

frame.add(balanceLabel);

frame.add(balanceField);

frame.add(updateButton);

frame.setSize(400, 300);

frame.setLayout(null);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

**DELETE ACCOUNT :**

import javax.swing.\*;

import java.sql.\*;

public class UpdateAccount {

public static void main(String[] args) {

JFrame frame = new JFrame("Update Account");

JLabel idLabel = new JLabel("Account ID:");

JLabel nameLabel = new JLabel("Name:");

JLabel balanceLabel = new JLabel("Balance:");

JTextField idField = new JTextField();

JTextField nameField = new JTextField();

JTextField balanceField = new JTextField();

JButton updateButton = new JButton("Update");

idLabel.setBounds(30, 30, 100, 30);

idField.setBounds(150, 30, 150, 30);

nameLabel.setBounds(30, 80, 100, 30);

nameField.setBounds(150, 80, 150, 30);

balanceLabel.setBounds(30, 130, 100, 30);

balanceField.setBounds(150, 130, 150, 30);

updateButton.setBounds(100, 180, 100, 30);

updateButton.addActionListener(e -> {

String accountId = idField.getText();

String name = nameField.getText();

String balance = balanceField.getText();

try (Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement("UPDATE accounts SET name=?, balance=? WHERE account\_id=?")) {

ps.setString(1, name);

ps.setDouble(2, Double.parseDouble(balance));

ps.setInt(3, Integer.parseInt(accountId));

int rowsUpdated = ps.executeUpdate();

if (rowsUpdated > 0) {

JOptionPane.showMessageDialog(frame, "Account Updated Successfully!");

} else {

JOptionPane.showMessageDialog(frame, "Account ID Not Found!");

}

} catch (SQLException ex) {

ex.printStackTrace();

}

});

frame.add(idLabel);

frame.add(idField);

frame.add(nameLabel);

frame.add(nameField);

frame.add(balanceLabel);

frame.add(balanceField);

frame.add(updateButton);

frame.setSize(400, 300);

frame.setLayout(null);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

}

**SEARCH ACCOUNT:**

import javax.swing.\*;

import java.sql.\*;

public class SearchAccount {

public static void main(String[] args) {

JFrame frame = new JFrame("Search Account");

JLabel idLabel = new JLabel("Account ID:");

JTextField idField = new JTextField();

JTextArea resultArea = new JTextArea();

JButton searchButton = new JButton("Search");

idLabel.setBounds(30, 30, 100, 30);

idField.setBounds(150, 30, 150, 30);

searchButton.setBounds(100, 80, 100, 30);

resultArea.setBounds(30, 130, 300, 150);

searchButton.addActionListener(e -> {

String accountId = idField.getText();

try (Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement("SELECT \* FROM accounts WHERE account\_id=?")) {

ps.setInt(1, Integer.parseInt(accountId));

ResultSet rs = ps.executeQuery();

if (rs.next()) {

String result = "Name: " + rs.getString("name") + "\n" +

"Account Number: " + rs.getString("account\_number") + "\n" +

"Balance: " + rs.getDouble("balance");

resultArea.setText(result);

} else {

resultArea.setText("Account ID Not Found!");

}

} catch (SQLException ex) {

ex.printStackTrace();

}

});

frame.add(idLabel);

frame.add(idField);

frame.add(searchButton);

frame.add(resultArea);

frame.setSize(400, 350);

frame.setLayout(null);

frame.setVisible(true);

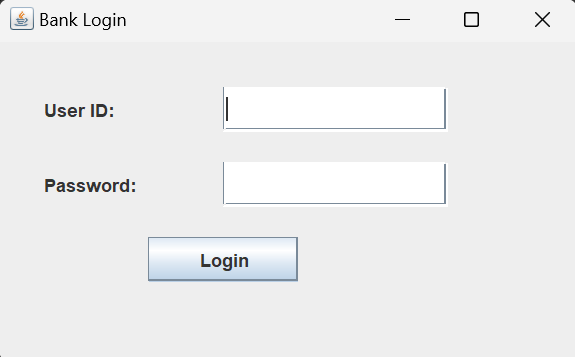
frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

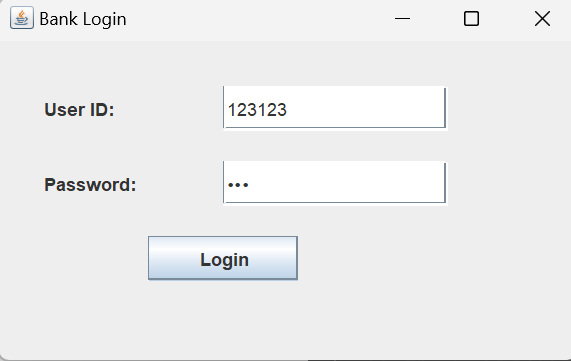
}

}

**V. RESULT AND DISCUSSION**

**LOGIN PAGE :**

****



**RESULTS**

**1.User Features:**

- User Registration:

- Implementation: The user registration functionality was successfully implemented, allowing customers (both bank staff and users) to register by providing essential details such as name, contact information, and account type. Upon successful registration, users can log into the system using their credentials, ensuring secure access to the platform.

- Result: Customers were able to successfully create accounts, and secure login mechanisms ensure only authorized users can access their respective functionalities.

- Add Account (Admin Functionality):

- Implementation: The admin functionality to add new bank accounts for customers was implemented successfully. Bank staff can enter details such as customer name, contact number, account number, and initial balance. Once added, the accounts are stored in the database and become part of the bank's active records.

- Result: Accounts are correctly created, and the system immediately reflects these new entries in the account inventory, accessible for further management or viewing.

- View Account (User & Admin Functionality):

- Implementation: Both administrators (bank staff) and users can view their respective account details. The system displays key details such as account number, account holder name, and balance. Users can check their balances, while admins can view and manage all accounts.

- Result: Both users and administrators are able to easily view accounts, ensuring that accurate and up-to-date information is available for decision-making.

**2. Admin Functionality:**

- Manage Accounts:

- Implementation: The admin module allows bank staff to manage customer accounts effectively. Staff can view, update, or delete accounts, which helps maintain the integrity and accuracy of customer records.

- Result: Administrators can easily manage accounts, ensuring that outdated or inactive accounts are removed, and customer details are regularly updated.

**3. Performance & Security:**

- Performance:

- Observation: The system performed well during testing, with users registering accounts, viewing accounts, and adding new accounts without any significant issues. Data consistency was maintained, and changes to customer accounts were immediately reflected in the database.

- Result: The system handled small-scale operations efficiently, though further performance testing with a larger user base is required to ensure that it can handle high concurrent usage.

- Security:

- Implementation: Basic security measures were successfully implemented, including password encryption and secure login. Only authenticated users can access account-related functionalities, with role-based access control for admins and regular users.

- Result: The system ensures secure access by encrypting user passwords and protecting sensitive data. However, additional security features, such as two-factor authentication and session management improvements, are recommended for further strengthening.

**DISCUSSION**

**1.User Experience:**

- Strengths:

- The system's user interface is simple, intuitive, and easy to navigate. Customers and administrators can quickly register, view accounts, and perform transactions without difficulty.

- The overall system performance during initial testing was stable and responsive, allowing smooth access to core features.

- Areas for Improvement:

- The interface could benefit from additional filtering options, such as account type or transaction history, to enhance usability.

- The system could be optimized for mobile devices to improve accessibility for users who prefer using smartphones and tablets for banking.

**2. Email Integration:**

- Strengths:

- Email notifications worked well to confirm successful account registration and transactions, keeping users informed of important updates.

- Areas for Improvement:

- As the number of customers grows, email integration should be scaled. A more robust email service may be required to handle higher volumes of notifications efficiently.

3. Admin Efficiency:

- Strengths:

- The admin module allows bank staff to efficiently manage customer accounts. Adding, editing, and deleting accounts is a straightforward process, enabling the bank to keep track of account activities.

- Areas for Improvement:

- Including features such as generating transaction reports and auditing capabilities could provide more visibility for administrators, allowing them to track activity and identify irregularities in the system.

- The addition of a search or filter function for quickly locating accounts would help admins work more efficiently, especially with large databases.

4. Security Concerns:

- Discussion:

- While basic security measures like password encryption and secure login were implemented, the addition of two-factor authentication for both users and administrators would significantly improve the security of the system.

- Session management, including automatic timeouts and protection against unauthorized access, would further enhance the system's security.

5. Performance:

- Observation:

- The system performed well during testing with a small user base, but performance testing should be conducted with a larger user group and higher transaction volumes to ensure the system can scale effectively. Stress testing will help evaluate the system’s robustness under heavy loads.

VII. REFERENCES

Java Development and Backend Resources:

- Java Documentation: Official Java documentation for learning and implementing Java-based applications. Available at: https://docs.oracle.com/en/java/

Database Management:

- MySQL Documentation: Official documentation for MySQL, covering database setup, queries, and best practices. Available at: https://dev.mysql.com/doc

- MySQL Connector: Documentation for MySQL Connector/J, which allows Java applications to communicate with MySQL databases. Available at: https://dev.mysql.com/doc/connector-j/

Frontend Development and User Interface:

- Java Swing Documentation: Resources for building GUI applications using Java Swing for creating the user interface. Available at: https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html

Project Management and Development Tools

- GitHub: For version control and collaboration during the development process. Documentation available at: https://docs.github.com

.