**High Level Design (HLD)**

ATM (Automated teller machine) console based Application.

Revision Number : 1.0

Last Date of Revision : 28/07/2023

## **DOCUMENT VERSION CONTROL**

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 28/07/23 | 1 | Initial HLD v1.0 | Sidduganesh Musa |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**DOCUMENT VERSION CONTROL** 2](#_Toc141610022)

[**Abstract** 4](#_Toc141610023)

[**1.** **Introduction** 5](#_Toc141610024)

[1.1 Why this High-Level Design Document ? 5](#_Toc141610025)

[1.2 Scope 5](#_Toc141610026)

[1.3 Definitions 5](#_Toc141610027)

[**2 General Description** 6](#_Toc141610028)

[2.1 Introduction: 6](#_Toc141610029)

[2.2 Scope: 6](#_Toc141610030)

[2.3 Architecture: 6](#_Toc141610031)

[2.4 User Authentication: 7](#_Toc141610032)

[2.5 Data Storage: 7](#_Toc141610033)

[2.6 Core Operations: 7](#_Toc141610034)

[2.7 Error Handling: 8](#_Toc141610035)

[2.8 Security Measures: 8](#_Toc141610036)

[2.9 Tools: 8](#_Toc141610037)

[**3 Design Details** 9](#_Toc141610038)

[3.1 Process Flow 9](#_Toc141610039)

[**4 Performance** 10](#_Toc141610040)

[**CONCLUSION** 12](#_Toc141610041)

# **Abstract**

The Java Console-based ATM Interface is a robust and user-friendly software application designed to provide customers with seamless access to their bank accounts through a text-based console interface. Developed using Java, this application offers essential banking operations, including displaying transactions, making deposits, withdrawals, checking account balances, transferring funds, and exiting the interface.

The primary goal of this project is to create an efficient and secure platform that allows customers to manage their financial activities without the need for a graphical user interface. The purpose of this project is to provide a user-friendly, efficient, and secure way for bank customers to interact with their accounts and perform various banking operations through a text-based console interface.

The application follows an object-oriented design, utilizing core Java concepts and data structures

# **Introduction**

## Why this High-Level Design Document ?

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

* + - Present all of the design aspects and define them in detail
    - Describe the user interface being implemented
    - Describe the hardware and software interfaces
    - Describe the performance requirements
    - Include design features and the architecture of the project
    - List and describe the non-functional attributes like: Security
      * Reliability
      * Maintainability
      * Portability
      * Reusability
      * Application compatibility
      * Resource utilization
      * Serviceability

## 1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

## 1.3 Definitions

|  |  |
| --- | --- |
| Terms | Description |
| ATM | Automatic Teller Machine |
| IDE | Integrated Development Environment |
| JDK | Java Development Kit |

# **2 General Description**

## 2.1 Introduction:

The ATM Interface in Java is a console-based application developed to emulate the functionalities of an Automated Teller Machine (ATM). The primary objective of this application is to offer bank customers a user-friendly, efficient, and secure platform to manage their accounts and perform essential banking operations through a text-based console interface. The High-Level Design Document provides an overview of the overall structure, architecture, and key components of the ATM interface.

## 2.2 Scope:

The scope of the ATM interface covers the development of a command-line application with the following core operations: displaying transactions, making deposits, withdrawals, checking account balances, transferring funds, and exiting the interface. The design emphasizes modularity, maintainability, and adherence to Java best practices.

## 2.3 Architecture:

The application follows a modular design and adopts the Model-View-Controller (MVC) architectural pattern. This design separates the concerns of data management, user interface, and application logic, ensuring scalability and ease of maintenance.

Components:

a. Model:

The Model component is responsible for handling data related to customer accounts, transaction history, and user authentication. It securely manages account information and implements encryption techniques to safeguard sensitive data.

b. View:

The View component represents the user interface for the ATM interface. It displays relevant information to users and receives input for various banking operations through the console interface.

c. Controller:

The Controller component acts as an intermediary between the Model and View components. It processes user input, triggers corresponding actions, and updates the Model and View accordingly.

## 2.4 User Authentication:

The ATM interface incorporates a secure user authentication mechanism. Customers are required to provide their account number and Personal Identification Number (PIN) to access their accounts. Passwords are securely stored using encryption and hashing techniques.

## 2.5 Data Storage:

The application utilizes a data storage system, such as a relational database or file-based storage, to manage customer account details and transaction history. Data access is optimized for efficiency and accuracy.

## 2.6 Core Operations:

a. Show Transactions:

Customers can view their transaction history, displaying a chronological list of past transactions.

b. Deposit:

Customers can securely deposit funds into their accounts. The application immediately updates the account balance upon successful deposits.

c. Withdrawal:

Users can withdraw cash from their accounts, ensuring sufficient funds are available and preventing overdraft scenarios.

d. Show Balance:

Customers can check their account balance in real-time through the console interface.

e. Transfer:

The application facilitates fund transfers between customer accounts, ensuring accuracy and security.

f. Quit:

Exiting the ATM interface is a seamless process, with all data safely saved before the application closes.

## 2.7 Error Handling:

The application includes robust error handling mechanisms to provide informative messages and handle exceptional scenarios gracefully, maintaining a smooth user experience.

## 2.8 Security Measures:

The ATM interface prioritizes data security by implementing password encryption, session management, and other security measures to protect users' sensitive information.

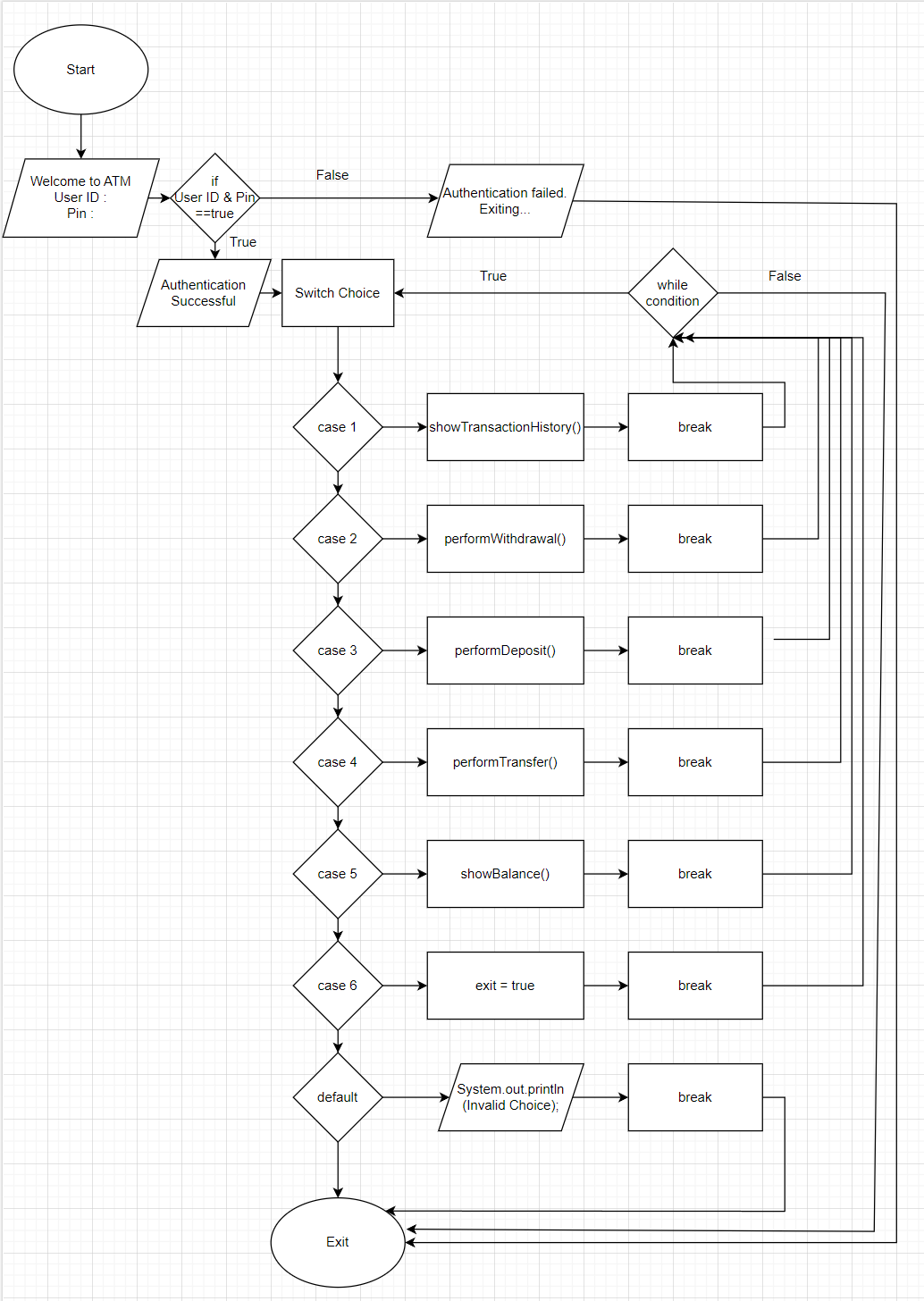
## 2.9 Tools:

The Tools used to build the ATM Interface in Java.

1. VS CODE
2. CMD

# **3 Design Details**

## 3.1 Process Flow



# **4 Performance**

The performance of the ATM interface in a Java console-based application can be evaluated based on several key factors:

Responsiveness:

The ATM interface should respond promptly to user inputs, ensuring that there is minimal delay between user actions and the system's response. Responsiveness is crucial to providing a smooth and efficient user experience during interactions with the application.

Efficiency of Data Retrieval:

The application should efficiently retrieve transaction history, account balances, and other relevant data from the data storage system. Optimized data retrieval ensures that users can quickly view their transactions and balances without experiencing significant delays.

Transaction Processing Speed:

The speed at which the application processes deposit, withdrawal, and transfer transactions impacts user satisfaction. These operations should be executed swiftly and accurately to provide a seamless banking experience.

Error Handling:

The ATM interface should include robust error handling mechanisms to handle exceptional scenarios gracefully. Error messages should be informative and help users understand and resolve issues without causing confusion or frustration.

Security:

While performance is essential, security must not be compromised. The application should implement strong security measures to protect user data, prevent unauthorized access, and ensure that sensitive information, such as passwords and financial details, is stored securely.

Memory and Resource Utilization:

The application should manage memory and system resources efficiently. Excessive memory usage or resource leaks can lead to performance degradation and potentially impact the stability of the application.

Scalability:

As the number of users or transactions increases, the application should maintain its performance and responsiveness. Scalability is crucial to handle concurrent users without significant degradation in performance.

Input Validation:

The ATM interface should perform thorough input validation to prevent incorrect data entry and ensure data integrity. Proper input validation minimizes the risk of errors and enhances the overall performance of the application.

Redundant Operations:

The application should avoid unnecessary or redundant operations, such as repeated database queries, which can impact performance. Caching and other optimization techniques can be employed to mitigate redundant operations.

Load Testing:

Conducting load testing on the ATM interface is essential to evaluate its performance under heavy usage. Load testing helps identify performance bottlenecks and ensures that the application can handle peak loads efficiently.

# **CONCLUSION**

The Java Console-based ATM Interface presents a comprehensive and user-friendly solution for managing bank accounts through a text-based application. With key features such as displaying transactions, making deposits, withdrawals, checking balances, transferring funds, and quitting, the application covers all essential banking operations efficiently.

The user authentication process ensures the security of customer data, with account numbers and PINs being securely validated before granting access to account management features. The implementation of robust error handling mechanisms guarantees a smooth user experience, providing informative error messages to guide users and prevent unexpected behaviour during interactions.

The transaction history functionality allows customers to keep track of their financial activities, empowering them to monitor and review past transactions conveniently. Additionally, support for multiple currencies and denomination options for withdrawals and deposits caters to the diverse needs of users, making the interface more inclusive and versatile.

By incorporating these key operations and functionalities, the ATM interface offers a reliable and secure means for users to conduct their banking activities through a console-based application. The modular design and adherence to best practices in software development ensure maintainability and the potential for further enhancements in the future.

The Java Console-based ATM Interface stands as an efficient and user-centric solution for providing essential banking services. As technology advances, this console application can serve as a stepping stone for further advancements, potentially expanding to incorporate graphical user interfaces or integration with backend databases for a more comprehensive and feature-rich banking experience.