High Level Design

Banking Application (Console based)

Revision Number: 1.3

last date of revision: 09/07/23

iNeuron Intelligence private limited.

- Sukumar Tusam

Document Version Control

Revision	Date	Author	Changes
1.1	09.6.23	Sukumar Tusam	Introduction & General Description
1.2	12.6.23		Added process flow application diagrams
1.3	13.6.23		Added KPI’s

Contents

1. Introduction

1.1. 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.

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

.

- Java – A possible programming language to implement banking application.
- Tomcat – a free, open-source implementation of Java Servlet and Java Server Pages technologies developed under the Jakarta project at the Apache Software Foundation.
- Apache - An open source Web server.
- ER – Entity Relation Diagram
- Kernel – Core of an operating system, a kernel manages the machine’s hardware resources (including the processor and the memory), and provides and controls theway any other software component can access these resources.
- DHCP – (Dynamic Host Configuration Protocol) – This is a protocol that letsnetwork administrators centrally manage and automate the assignment of IP Addresses on the corporate network.

2. General Description

2.1. Product Perspective

The Simple Banking Application is designed to be a standalone software product that operates independently of other banking systems or platforms. It provides a self-contained solution for users to perform basic banking operations without the need for complex integrations or dependencies on external systems.

2.2. Tools used

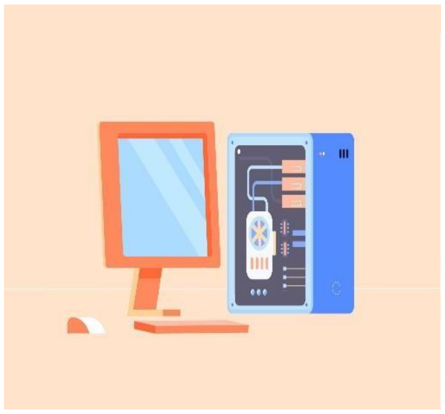
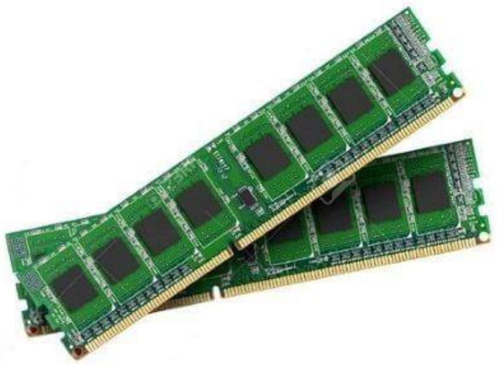
2.2.1. Software Requirements

- Java is a widely-used programming language known for its versatility and platform independence. It is commonly used to develop banking applications due to its robustness, scalability, and extensive libraries and frameworks.

- Integrated Development Environment (IDE): IDEs like Eclipse, IntelliJ IDEA, or NetBeans provide a development environment for writing, compiling, and debugging Java code. These IDEs offer features such as code completion, syntax highlighting, and debugging tools, making the development process more efficient.
- Git : Utilizing a version control system like Git is crucial for managing and tracking changes to the source code. It allows developers to collaborate effectively, maintain different code branches, and roll back changes if needed.

2.2.2. Hardware Requirements

- Computer or server: A machine capable of running Java programs.
- Sufficient RAM: The amount of RAM required will depend on the expected number of concurrent users and the size of the database (if used).
- Storage: Adequate disk space to store the application code, dependencies, and any associated data.

2.3. Problem Statement

Existing banking systems lack a user-friendly interface and efficient functionality for customers to manage their accounts, make deposits, and withdraw funds. Additionally, there is a need for enhanced security measures to protect user data and ensure secure.

2.4. Further Improvement

This Banking application can be added with more use cases, so that the console-based banking application can become more secure, efficient, and maintainable like;

- User Interface: Implementing a more user-friendly console interface with clear instructions and input validation.
- User Authentication and Security: Designing a user authentication system and ensuring secure storage of user account information.
- Account Management: Creating a class to represent bank accounts and implementing methods for managing account details.
- Transaction Handling: Developing transaction handling logic with proper validation, error handling, and balance updates.
- Data Persistence: Implementing a data storage mechanism to persist account information and transaction history.
- Error Handling and Logging: Handling exceptions gracefully, displaying meaningful error messages, and adding logging functionality.
- Modular and Scalable Architecture: Designing a modular architecture using classes and packages, following object-oriented principles and design patterns.

.

2.5. Technical Requirement

This documents addresses the requirements for this project like;

- Balance Management:

The application should provide functionality to check the account balance.
Deposits should be allowed, and the account balance should be updated accordingly.

.

Error Handling and Logging:

Proper error handling should be implemented to handle exceptions and display user-friendly error messages.

- Input Validation:

User inputs should be validated to ensure correct and secure data processing.

Input validation should include checks for data types, length, format, and range.
Invalid inputs should be rejected, and appropriate error messages should be displayed to the user.

- Modularity and Maintainability:

The application should follow a modular architecture, separating concerns into different classes or packages.
Object-oriented principles and design patterns should be employed to promote code reusability and maintainability.

- Testing and Quality Assurance:

The application should undergo rigorous testing to ensure functionality, reliability, and security.
Unit tests should be created to test individual components and methods.

.

2.6. Data Requirement :

Data requirement completely depends on our problem statements. We need the following information for this project ;

Bank Account:

Balance: The current balance of the bank account. It represents the available funds for a user.
Account Number: An identifier for each bank account. It uniquely identifies a specific account.
Transactions:

N/A: Since transaction history is not required, there is no need to track individual transactions or maintain a transaction history.

Data Validation:

Validate inputs for data integrity and consistency.
Sanitize inputs to prevent injection attacks.
Implement checks for data type, length, format, and range.

2.7. Constraints

Constraints for the project include platform compatibility, ensuring the application works across different operating systems, performance optimization to handle concurrent users efficiently, adherence to security best practices to protect user data, scalability to accommodate future growth, validation of user inputs to prevent security vulnerabilities, robust error handling and logging mechanisms, and compliance with relevant banking regulations.

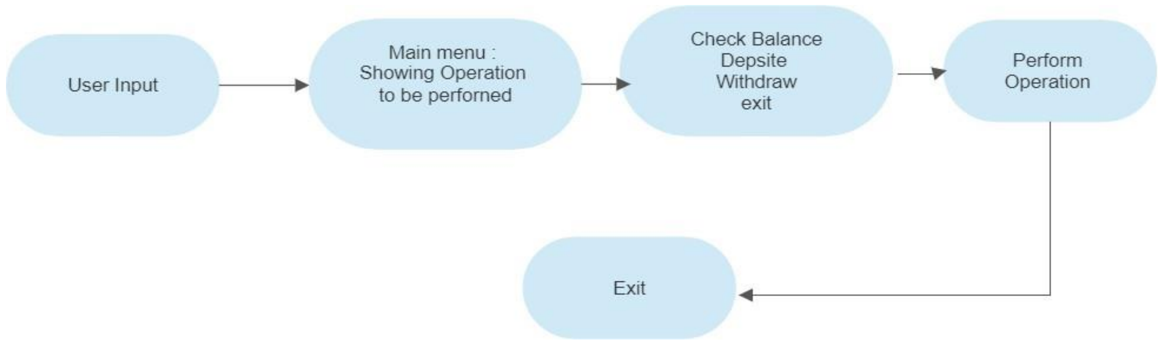
2.8. Assumption

The console-based banking application is designed to provide basic banking functionalities such as balance checking, deposits, withdrawals, and account management. The application assumes a single-user environment, where user input through the console is trusted and properly formatted. It operates in an offline mode without real-time integration with external systems or online banking services.

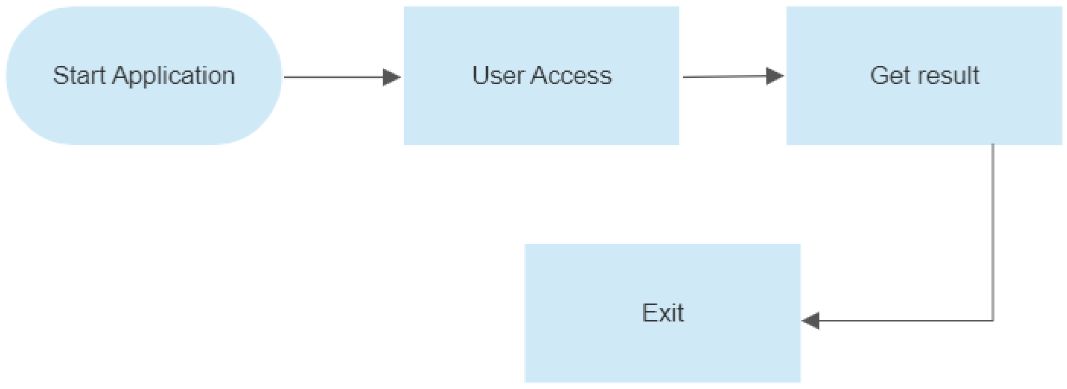
3. Design Details

3.1. Process Flow

Describing the process flow for a console-based banking application after identifying the different types of anomalies;


User Input
Main menu : Showing Operation to be perforned
Check Balance Depsite Withdraw exit
Perform Operation
Exit

3.1.1. Deployment Process



3.2. Event Log

The system should log every event so that user will know what process is running internally

Event Log for Console-Based Banking Application:

- Main Menu Selection

User selects an option from the main menu.
Application captures the selected option.

- Check Balance ::

User selects "Check Balance" option.
Application retrieves user's account information.
Application displays the current account balance.

- Deposit Money:

User selects "Deposit Funds" option.
Application prompts user to enter the amount to deposit.
User enters the deposit amount.
Application validates the amount.
If valid, application updates the account balance and transaction history.
Application displays the updated balance.

- Withdraw Money:

User selects "Withdraw Funds" option.
Application prompts user to enter the amount to withdraw.
User enters the withdrawal amount.

Application validates the amount.
If valid and sufficient funds are available, application updates the account balance and transaction history.
If invalid or insufficient funds, an error message is displayed.

- Exit:

User selects "Exit" option.
Application terminates, and the user session ends.

3.3. Error Handling :

- Application detects errors, exceptions, or invalid inputs.
- Application logs the error details.
- Application displays appropriate error messages to the user.

3.4. Reusability :

The code written and components used should have the ability to be reused with no problems.

3.5. Application Compatibility

- The console-based banking application should be compatible with popular operating systems (e.g., Windows, mac OS, Linux) and their different versions.
- Ensure compatibility with the targeted Java version or JDK used for development and deployment.
- Verify compatibility with the hardware platforms on which the application is intended to run.
- Test the application's console interface compatibility with various terminal emulators and console environments.
- Ensure compatibility of any third-party Java libraries or dependencies used in the application.
- Verify compatibility with the chosen database management system if the application interacts with a database.

3.6. Performance

Performance is going to be very important for this project. The application should provide a responsive and efficient user experience while handling increasing user load and transaction volumes. The application should also aim for quick response times, near-instantaneous operations for balance checking, deposits, and withdrawals.

3.7. Resource Utilization:

When any task is performed, it will likely use all the processing power available until that function is finished.

3.8. Deployment

git GitHub

4. Dashboard

4.1. KPI’s (Key Performance Indicator)

- User Adoption Rate: Measure the percentage of users who actively engage with and use the console banking application.
- Transaction Volume: Track the total number of transactions processed by the application over a specific period.
- Error Rate: Monitor the percentage of transactions or operations that result in errors or failures.
- Response Time: Measure the average time taken by the application to respond to user requests.
- Security: Evaluate the application's security measures to ensure the safety of user data and transactions.
- Application Stability: Monitor the application's stability and uptime, measuring the frequency and duration of any downtime.
- Code Quality: Assess the quality of the code base, including readability, maintainability, and adherence to best practices.
- Cost Efficiency: Evaluate the project's cost-effectiveness by comparing development and maintenance costs to the benefits delivered.

5. Conclusion

In conclusion, the provided code implements a simple banking application that allows users to perform basic banking operations such as checking their account balance, depositing funds, withdrawing funds, and exiting the application. The program utilizes the **Scanner** class to take user input and a **while** loop to continuously display a menu and handle user choices until the user chooses to exit.

6. References

https://www.javatpoint.com/banking-application-in-java for banking app details
Google.com for images