**ABSTARCT**

The Bank Account Management System is an application for maintaining a person's account in a bank. In this project I tried to show the working of a banking account system and cover the basic functionality of a Bank Account Management System. To develop a project for solving financial applications of a customer in banking environment in order to nurture the needs of an end banking user by providing various ways to perform banking tasks. Also to enable the user’s work space to have additional functionalities which are not provided under a conventional banking project

The Bank Account Management System is an application for maintaining a person's account in a bank. The main aim of this project is to develop software for Bank Account Management System.This project is developed using shell scripting. The project analyzes the user requirements and then comes up with the requirements specifications. The system is designed as an interactive and content management system. The content management system deals with data entry and updating transaction logs while the interactive system deals with system interaction with the files. Thus, above features of this project will save transaction time and therefore increase the efficiency of the system

**INTRODUCTION**

In today's rapidly evolving world, efficient and secure management of banking operations is of utmost importance. A well-designed Banking Management System can streamline processes, enhance customer experience, and ensure the security of financial transactions. Shell scripting, a powerful scripting language commonly found in Unix-like operating systems, can be utilized to create a robust and automated Banking Management System.

The Banking Management System using Shell Script provides a comprehensive solution for handling various banking operations, such as customer management, account management, transaction processing, and report generation. By leveraging the capabilities of shell scripting, this system offers flexibility, reliability, and ease of use.

The key features of the Banking Management System include:

1. Customer Management

2. Account Management

3. Transaction Processing

4. Report Generation

5. Security and Access Control

By leveraging the power of shell scripting, the Banking Management System offers an efficient and secure platform for managing banking operations. It improves operational efficiency, enhances customer experience, and ensures data integrity and security. The flexibility and scalability of shell scripting enable the system to adapt to evolving banking requirements and regulatory changes.

In summary, the Banking Management System using Shell Script is an invaluable tool for banks and financial institutions to streamline operations and deliver exceptional services to their customers. With its comprehensive features and the versatility of shell scripting, this system empowers banks to effectively manage their operations while maintaining the highest standards of security and efficiency.

**SYSTEM REQUIREMENTS**

• Operating System: Shell scripts can be executed on various operating systems such as Linux, macOS, and Windows with a shell environment installed.

• Shell Environment: You need a shell environment like Bash, Zsh, or PowerShell to run the shell script. These are typically pre-installed on Linux and macOS systems, but you may need to install them on Windows.

• Text Editor: You'll need a text editor to write the shell script. There are many options available, such as Vim, Emacs, Nano, or any other text editor of your choice. Choose one that you're comfortable with.

• Terminal or Command Prompt: You'll use the terminal or command prompt to execute the shell script. It's an essential tool for running shell scripts on your system

**WHAT IS OS?**

An operating system (OS) is a software program that acts as an intermediary between the hardware components of a computer system and the applications running on it. It is responsible for managing and controlling the computer's resources, such as memory, CPU, input/output devices, and network connectivity. In other words, an operating system is the foundation of any computer system, without which it cannot function.

The primary function of an operating system is to provide an interface for users to interact with the computer system. This interface includes the graphical user interface (GUI) that allows users to interact with the system using icons, menus, and windows. The OS also provides a command-line interface (CLI) that enables users to execute commands using text- based input.An operating system provides several essential services, including:

Process Management: An OS manages the execution of multiple applications or processes running concurrently on the system. It is responsible for scheduling and prioritizing the execution of these processes, allocating system resources such as memory, CPU time, and input/output operations.

Memory Management: An OS is responsible for managing the computer's memory, including the allocation of memory to different applications and processes, freeing up memory when it is no longer needed, and protecting the memory from unauthorized access.

File Management: An OS manages the computer's file system, including creating, deleting, moving, and copying files and directories.

Device Management: An OS manages the computer's input/output devices, such as the keyboard, mouse, and monitor, as well as external devices such as printers and scanners.

Security: An OS provides security features to protect the computer system and its data from unauthorized access, including user authentication and access control mechanisms.

Networking: An OS manages network connectivity, enabling the computer system to communicate with other systems over a network.

Some popular operating systems include Windows, macOS, Linux, Unix, and Android. Each of these operating systems has unique features and functionalities, and they are designed to meet the specific needs of different types of computer systems and applications

# SHELL SCRIPTING

A shell script is a type of computer program that is designed to be run by a shell, which is a command-line interpreter used in many operating systems, such as Unix and Linux. Shell scripting is a powerful tool for automating tasks and performing various system administration tasks.

Shell scripts can be used to perform a wide variety of tasks, including file manipulation, system administration, and network management. They are also used for automating repetitive tasks, such as backups, updates, and system maintenance.

One of the primary benefits of shell scripting is its ease of use. Shell scripts can be created using simple text editors and run from the command line. They are also highly portable, allowing them to be run on different operating systems without modification.

In addition, shell scripts are highly customizable and can be modified to meet specific requirements. They can also be used in conjunction with other programming languages to create more complex applications.

Overall, shell scripting is an important tool for system administrators and developers alike, and provides a powerful and flexible way to automate tasks and manage operating systems.

# BENEFITS OF SHELL SCRIPTING

Shell scripting can provide a number of benefits for system administrators, developers, and users of Unix-based systems. By leveraging the capabilities of shell scripting, users can streamline workflows, improve efficiency, and reduce errors caused by manual processes.

* Automation: Shell scripting can be used to automate repetitive tasks, such as backups, system updates, and maintenance tasks. This can save time and reduce the risk of errors caused by manual processes.
* Flexibility: Shell scripting allows for the creation of custom scripts that can be tailored to specific needs. This can help to increase efficiency and streamline workflows.
* Portability: Shell scripts are highly portable and can be run on a wide range of Unix- based systems without modification.
* Interoperability: Shell scripting can be used in conjunction with other programming languages, such as Python or Perl, to create more complex scripts and applications.
* Debugging: Shell scripts can be easily debugged using standard Unix tools, such as grep and awk, which can help to identify and fix errors quickly.
* Accessibility: Shell scripting is a widely used tool and has a large community of users, which means that there is a wealth of resources and support available for those who are new to it.

# APPLICATIONS OF SHELL SCRIPTING

Shell scripting has a wide range of applications in Unix-based systems, By leveraging the power and flexibility of shell scripting, users can automate tasks, improve efficiency, and streamline workflows.

* System administration: Shell scripting is commonly used for system administration tasks such as backups, system updates, user management, and log management.
* Network management: Shell scripting can be used for tasks related to network management, such as network monitoring, security management, and configuration management.
* Application development: Shell scripting can be used as a scripting language for developing Unix-based applications.
* Web development: Shell scripting can be used in conjunction with other web development tools to create dynamic web pages and web applications.
* Data analysis: Shell scripting can be used for data processing and analysis tasks, such as data transformation, data filtering, and data visualization.
* Scientific computing: Shell scripting can be used in scientific computing applications for tasks such as data processing, simulation, and visualization.

**BANKING MANAGEMENT SYSTEM**

Banking management in shell scripting can be implemented to automate various banking operations, such as managing customer accounts, performing transactions, and generating reports. Shell scripting provides a convenient way to automate repetitive tasks and handle data manipulation within a banking system.

Here's an example of how banking management can be implemented using shell scripting:

1. Account Creation:

Shell scripting can be used to create customer accounts. You can prompt the user to enter the necessary details, such as account number, customer name, and initial balance. The script can then store this information in a file or a database for future reference.

2. Account Information:

You can create a script that allows users to retrieve account information. It can prompt for an account number and search the database or file for the corresponding account details, such as the account holder's name and current balance.

3. Deposits and Withdrawals:

Shell scripting can handle deposit and withdrawal operations. The script can prompt the user for the account number and the transaction amount. It can then update the account balance accordingly, ensuring that the balance doesn't go below zero for withdrawals.

4. Security Measures:

Shell scripting can enforce security measures within the banking system. For example, you can implement password authentication to restrict unauthorized access to account information or transaction operations. The script can prompt users for a password and verify it before allowing any sensitive actions.

5. Check Balance:

To check the account balance, you can create a shell script that prompts the user to enter their account number and PIN for authentication. Once authenticated, the script can retrieve the account balance from the database or file associated with the account number and display it to the user.

6. PIN Change:

To enable PIN change functionality, you can create a shell script that prompts the user to enter their current PIN and then enter a new PIN. The script should perform necessary checks, such as validating the current PIN and ensuring the new PIN meets the required criteria. Once validated, the script can update the PIN in the database or file associated with the account number.

**Challenges in Implementing a banking system Using Shell Scripting**

Implementing a banking system solely using shell scripting can present several challenges. While shell scripting is a powerful tool for automating tasks and managing system operations, it may not be the most suitable language for developing a complex banking system. Here are some challenges you might encounter:

1. Limited functionality: Shell scripting primarily focuses on executing system commands and managing files. It lacks many advanced programming features necessary for building a robust banking system, such as data structures, complex data manipulation, and object-oriented programming concepts.

2. Security vulnerabilities: Shell scripting languages like Bash are prone to security vulnerabilities, especially when dealing with sensitive financial information. Shell scripts are typically executed with elevated privileges, making them potential targets for exploitation. Building a secure banking system requires implementing robust security measures, such as input validation, encryption, and access controls, which can be challenging to achieve with shell scripting alone.

3. Scalability and performance limitations: Shell scripting is optimized for executing simple and sequential tasks. As a banking system grows in complexity and user base, shell scripts may struggle to handle large-scale operations efficiently. Performance bottlenecks and slow response times may arise, impacting the overall user experience.

4. Lack of transactional integrity: Ensuring transactional integrity, where multiple operations are executed atomically and consistently, is crucial in a banking system. Shell scripting lacks built-in mechanisms to handle complex transactional workflows, making it challenging to guarantee data consistency and integrity.

5. Limited error handling capabilities: Shell scripting has limited error handling capabilities compared to high-level programming languages. In a banking system, it is essential to handle various error scenarios, such as input validation failures, network failures, and database errors, in a robust and user-friendly manner. Achieving comprehensive error handling with shell scripting alone can be cumbersome.

6. Maintainability and code complexity: As a banking system evolves, maintaining and extending shell scripts can become increasingly complex and error-prone. Shell scripting lacks modularization and code organization features found in higher-level languages, making it challenging to manage large codebases effectively.

7. Integration with external systems: Banking systems often need to integrate with external services like payment gateways, credit card processors, and third-party APIs. Shell scripting may not provide native support for interacting with such services, requiring additional tools or workarounds.

Considering these challenges, it is advisable to explore alternative programming languages, such as Python, Java, or C#, which offer more robust features, libraries, and frameworks tailored for building complex banking systems.

# IMPLEMENTATION

# #!/bin/bash

# # Define the file that stores the user information

# USER\_FILE=login.txt

# if [ ! -f "$login" ]; then

# touch "$login"

# fi

# # for storing the data of user balance

# DATA\_FILE=data.txt

# if [ ! -f "$data" ]; then

# touch "$data"

# fi

# # loacl veriables

# PIN=0

# cdepo=0

# name=""

# flag=1

# depo=0

# #menu function for user

# menu() {

# BALANCE=$(cat data.txt | grep "$username" | cut -d':' -f2)

# echo "BANKING MANAGEMENT SYSTEM"

# while true

# do

# echo "+------------------------------+"

# echo "| Menu |"

# echo "| SELECT AN OPTION |"

# echo "+------------------------------+"

# echo "| 1. account detalis |"

# echo "| 2. change PIN |"

# echo "| 3. Deposit |"

# echo "| 4. Withdraw |"

# echo "| 5. Check Balance |"

# echo "| 6. Exit |"

# echo "+------------------------------+"

# read -p "Enter your choice: " choice

# case $choice in

# 1)

# accdetails

# ;;

# 2)

# changepin

# ;;

# 3)

# Deposit

# ;;

# 4)

# withdrawal

# ;;

# 5)

# Check\_Balance

# ;;

# 6)

# echo "Thank you for using the banking system"

# exit 0

# ;;

# \*) # Invalid option

# echo "Invalid option. Please select again"

# ;;

# esac

# done

# }

# # Define the function to register a new user

# register() {

# read -p "Enter your username: " username

# read -p "Enter your password: " password

# # Check if the username already exists

# if grep -q "^${username}:" $USER\_FILE; then

# echo "Username already exists."

# else

# # Add the new user to the user file

# echo "${username}:${password}" >> $USER\_FILE

# BALANCE=5000

# echo "${username}:${BALANCE}" >> $DATA\_FILE

# echo "Registration successful."

# echo "Account already exists for user $username"

# fi

# }

# # Define the function to login a user

# login() {

# read -p "Enter your username: " username

# read -p "Enter your password: " password

# # Check if the username and password match

# if grep -q "^${username}:${password}$" $USER\_FILE; then

# echo "Login successful."

# echo "You can enjoy our services now."

# menu

# else

# echo "Invalid username or password."

# fi

# }

# # Define the main function

# main() {

# echo "Welcome to the Banking system."

# while true; do

# read -p "Do you want to login or register? (l/r): " choice

# if [ "$choice" = "l" ]; then

# login

# break

# elif [ "$choice" = "r" ]; then

# register

# else

# echo "Invalid choice."

# fi

# done

# }

# accdetails(){

# echo "Name: $username"

# echo "PIN: $password"

# # echo "No of Deposits: $cdepo"

# # echo "No of Withdrawals: $cwith"

# bal1=$(cat data.txt | grep "$username" | cut -d':' -f2)

# echo "Current Balance: $bal1"

# }

# changepin(){

# echo "Welcome $username"

# read -p "Please Enter your current PIN: " pin

# if [ $pin -eq $password ]; then

# echo "Pin Matched"

# read -p "Enter you new pin: " pin1

# read -p "Confirm your PIN: " cpin

# if [ $cpin -eq $pin1 ]; then

# password=$cpin

# sed -i "s/^${username}:.\*/${username}:${cpin}/" $USER\_FILE

# echo "PIN changed Succesfully"

# else

# echo " YOU ENTERD INVALID PINS"

# fi

# else

# echo "Incorrect PIN"

# fi

# }

# Deposit(){

# echo "Enter amount to deposit:"

# read amount

# BALANCE=$((BALANCE + amount))

# sed -i "s/^${username}:.\*/${username}:${BALANCE}/" $DATA\_FILE

# echo "Deposit successful. Your new balance is: $BALANCE"

# }

# withdrawal() {

# echo "Enter amount to withdraw:"

# read cwith

# depo=$BALANCE

# depo=$((depo - cwith))

# if [ $depo -le 0 ]; then

# echo "Withdrawal unsuccessful. Insufficient balance."

# else

# BALANCE=$depo

# sed -i "s/^${username}:.\*/${username}:${BALANCE}/" $DATA\_FILE

# echo "Withdrawal successful. Your new balance is: $BALANCE"

# fi

# }

# Check\_Balance(){

# bal=$(cat data.txt | grep "$username" | cut -d':' -f2)

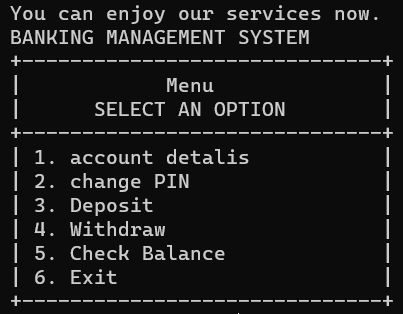
# echo "THE CURRENT BALANCE IS: $bal"

# }

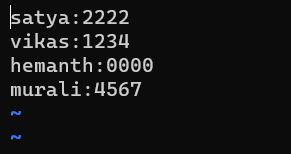
# # Call the main function

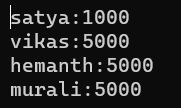
# main

# OUTPUT SCREENS

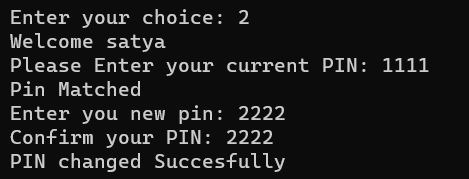


**Fig:1-(Menu bar)**

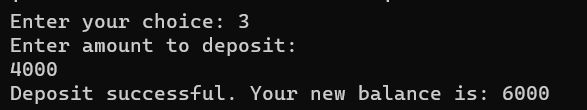


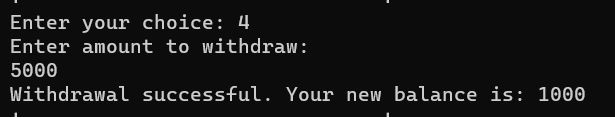


**Fig 2:(Details of Existing Accounts)**



**Fig:3(Change pin)**





**Fig:4(Deposit and Withdraw)**

****

**Fig:5(Check balance)**

****

**Fig:6(Exit)**

**CONCLUSION**

In conclusion, while it is technically possible to implement a banking system using shell scripting, it presents several challenges and limitations. Shell scripting's primary purpose is for automating system tasks and managing files, making it less suited for developing a complex banking system. The challenges include limited functionality, security vulnerabilities, scalability and performance limitations, lack of transactional integrity, limited error handling capabilities, code complexity, and difficulties in integrating with external systems.

To overcome these challenges and build a robust and secure banking system, it is recommended to consider alternative programming languages that offer more advanced features, libraries, and frameworks specifically designed for developing financial applications. Languages like Python, Java, or C# provide a wider range of tools and resources that can better address the requirements of a banking system, including security, transaction handling, scalability, and maintainability. Choosing an appropriate programming language will greatly enhance the effectiveness, reliability, and security of the banking system implementation.

**BIBLIOGRAPHY**

1)<https://github.com/van7b/Customer-Bank-Account-Management-System>

2) <https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/>

3) <https://www.macs.hw.ac.uk/~hwloidl/Courses/LinuxIntro/x864.html>

4) <https://community.spiceworks.com/topic/2362276-write-a-shell-script-called-deposit-sh>

5) <https://www.youtube.com/watch?v=-Cyra7Lxzvw>

6) <https://www.youtube.com/@ProgrammingKnowledge>

7)<https://www.youtube.com/watch?v=txRD_bK062Y&list=PLd3UqWTnYXOnBiXif3SL2W0vceGx6kt_V>