Simple banking application

**Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**M.Nishitha 24KB1A05AT**

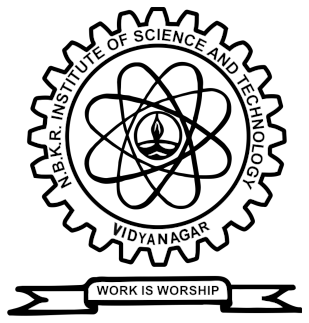
**M.Aloukitha Surin 24KB1A05Y2**

**M.Sohitha 24KB1A05Y5**

**M.Rishithanjali 24KB1A05Z9**

**Under the Guidance of**

**Smt.Sruthi**

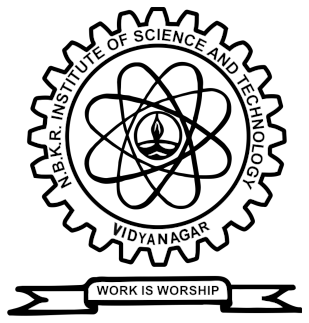
****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

1. **B.K.R. INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(AUTONOMOUS)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



###### CERTIFICATE

This is to certify that the project report entitled YOUR PROJECT TITLE being submitted by

M.Nishitha 24KB1A05AT

M.Aloukitha Surin 24KB1A05Y2

M.Sohitha 24KB1A05Y5

M.Rishithanjali 24KB1A05Z9

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the N.B.K.R.IST is a record of bonafied work carried out under my guidance and supervision.

| Smt.SruthiDesignation | Dr.A.RajaSekhar Reddy  M.Tech, Ph.D  Head of the Department |
| --- | --- |

**DECLARATION**

I hereby declare that the dissertation entitled Simple Banking Application submitted for the B.Tech Degree is my original work and the dissertation has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Place: vidyanagar M.Nishitha

Date: 5 may2025 24KB1A05AT

Acknowledgment:

I would like to express my sincere gratitude to all those who contributed to the successful completion of this simple banking application project.

First and foremost, I am deeply thankful to my [Supervisor’s/Instructor’s Name], whose valuable guidance, support, and encouragement were instrumental throughout the development process. Their insights and feedback helped me overcome challenges and improve the overall quality of this project.

I would also like to extend my appreciation to my peers and colleagues who provided useful suggestions and assistance during various stages of the project.

Lastly, I am grateful to my family and friends for their constant support, motivation, and understanding throughout the duration of this work.

Abstract:

The simple banking application is a lightweight, user-centric mobile solution aimed at delivering essential banking functionalities such as account creation, balance inquiry, fund transfers, and transaction tracking. Developed with a focus on security, responsiveness, and intuitive design, the application ensures that users can manage their finances conveniently from their mobile devices. It demonstrates the integration of secure authentication mechanisms and real-time data processing, making it suitable for both educational and practical implementations. This project highlights the potential of mobile technology in enhancing everyday financial operations and provides a foundation for more advanced digital banking systems.

Introduction:

This simple banking application is designed to provide users with basic financial services through a user-friendly mobile interface. It allows users to create accounts, check balances, transfer funds, and view transaction history securely. The app prioritizes ease of use, data privacy, and reliability, making banking accessible and efficient for individuals with minimal technical expertise. Ideal for both learning and practical purposes, this application demonstrates core concepts of mobile banking systems, including authentication, real-time updates, and secure data handling.

Software Requirement Analysis:

Software Requirement Analysis involves identifying the essential functional and non-functional requirements necessary to design and implement the banking application. It ensures that the system meets user needs and behaves reliably in a production environment.

1. Functional Requirements

These define the core operations that the application must perform:

User Authentication: Users must be able to register, log in, and log out securely.

Account Management: Users can view their profile and manage account settings.

Balance Inquiry: Users can check their current account balance.

Fund Transfer: Users can send money to other registered accounts.

Transaction History: Users can view a history of their transactions.

Notifications: Users receive alerts for transactions and updates.

2. Non-Functional Requirements

These define the quality attributes of the system:

Security: All data must be encrypted; authentication and session handling should be secure.

Performance: The app should respond quickly, with transactions processed in real-time.

Usability: The interface must be user-friendly and accessible to all users.

Scalability: The system should be designed to handle increased user load in the future.

Maintainability: Code should be modular and well-documented for easy updates.

Software Design:

Software design is a critical phase in system development that defines the architecture, components, interfaces, and data flow of the application. The objective is to create a robust, secure, and user-friendly system that meets functional and non-functional requirements.

1. System Architecture

The application follows a Client-Server Architecture, where:

The client is the mobile application developed using Flutter.

The server/backend handles authentication, data storage, and business logic, built using Firebase or a custom backend (e.g., Node.js + MySQL).

2. Architecture Diagram

User (Mobile App)

|

v

[Flutter Frontend]

|

v

[Backend Server or Firebase]

|

v

[Database: Firestore / MySQL]

3. Use Case Diagram

Actors:

User

System (App)

Use Cases:

Register/Login

View Balance

Transfer Funds

View Transaction History

Receive Notifications

4. Class Diagram (Simplified)

+------------------+

| User |

+------------------+

| userID |

| name |

| email |

| password |

+------------------+

|

v

+------------------+

| Account |

+------------------+

| accountID |

| balance |

| userID |

+------------------+

+------------------+

| Transaction |

+------------------+

| transactionID |

| amount |

| senderID |

| receiverID |

| date |

+------------------+

5. Data Flow Diagram (Level 1)

1. User logs in

2. System verifies credentials via backend

3. User performs action (e.g., check balance, transfer funds)

4. Backend communicates with database

5. Results returned to user interface

6. UI Design Overview

Login/Signup Page

Home Dashboard

Balance Overview

Transfer Money Interface

Transaction History Page

Proposed System:

The proposed system is a mobile-based simple banking application designed to provide users with essential banking functionalities such as account creation, balance inquiry, fund transfers, and transaction history tracking. This system aims to improve user accessibility, security, and operational efficiency over traditional or semi-automated banking processes.

1. Objectives of the Proposed System

Provide a secure and user-friendly mobile banking platform.

Enable real-time transactions and balance updates.

Simplify core banking functions for daily users.

Reduce dependency on physical bank visits for basic tasks.

Ensure data protection through secure authentication and encryption.

2. Key Features

User Authentication: Secure login and registration with password hashing or OTP.

Account Dashboard: View account details and real-time balance.

Money Transfer: Send and receive funds between users/accounts.

Transaction History: Access to a detailed and searchable record of past transactions.

Notification System: Alerts for transactions and account activity.

3. Advantages Over Existing Systems

Available 24/7, accessible from any location.

Intuitive UI tailored for non-technical users.

Faster processing time compared to manual banking methods.

Scalable for future enhancements such as bill payments or QR code payments.

4. Technology Stack

Frontend: Flutter (cross-platform mobile UI)

Backend: Firebase / Node.js / PHP (based on preference)

Database: Firebase Firestore or MySQL

Authentication: Firebase Auth or custom JWT-based system

Cloud Storage (if needed): Firebase Storage

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_ACCOUNTS 100

#define MAX\_NAME\_LENGTH 50

// Structure for a bank account

typedef struct {

int accountNumber;

char accountHolder[MAX\_NAME\_LENGTH];

float balance;

} Account;

// Structure for a mini statement entry

typedef struct MiniStatementEntry {

char transactionType[10]; // "Deposit" or "Withdraw"

float amount;

struct MiniStatementEntry\* next;

} MiniStatementEntry;

// Structure for a bank account with a mini statement

typedef struct {

Account account;

MiniStatementEntry\* miniStatementHead;

} BankAccount;

// Function prototypes

void createAccount(BankAccount\* accounts, int\* accountCount);

void deposit(BankAccount\* accounts, int accountCount);

void withdraw(BankAccount\* accounts, int accountCount);

void printMiniStatement(BankAccount\* accounts, int accountCount);

void freeMiniStatements(BankAccount\* accounts, int accountCount);

int main() {

BankAccount accounts[MAX\_ACCOUNTS];

int accountCount = 0;

int choice;

while (1) {

printf("\nSimple Banking Application\n");

printf("1. Create Account\n");

printf("2. Deposit\n");

printf("3. Withdraw\n");

printf("4. Print Mini Statement\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

createAccount(accounts, &accountCount);

break;

case 2:

deposit(accounts, accountCount);

break;

case 3:

withdraw(accounts, accountCount);

break;

case 4:

printMiniStatement(accounts, accountCount);

break;

case 5:

freeMiniStatements(accounts, accountCount);

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

}

void createAccount(BankAccount\* accounts, int\* accountCount) {

if (\*accountCount >= MAX\_ACCOUNTS) {

printf("Maximum account limit reached.\n");

return;

}

BankAccount newAccount;

newAccount.account.accountNumber = \*accountCount + 1;

printf("Enter account holder name: ");

scanf("%s", newAccount.account.accountHolder);

newAccount.account.balance = 0.0;

newAccount.miniStatementHead = NULL;

accounts[\*accountCount] = newAccount;

(\*accountCount)++;

printf("Account created successfully! Account Number: %d\n", newAccount.account.accountNumber);

}

void deposit(BankAccount\* accounts, int accountCount) {

int accountNumber;

float amount;

printf("Enter account number: ");

scanf("%d", &accountNumber);

if (accountNumber < 1 || accountNumber > accountCount) {

printf("Invalid account number.\n");

return;

}

printf("Enter amount to deposit: ");

scanf("%f", &amount);

if (amount <= 0) {

printf("Invalid deposit amount.\n");

return;

}

accounts[accountNumber - 1].account.balance += amount;

// Add to mini statement

MiniStatementEntry\* newEntry = (MiniStatementEntry\*)malloc(sizeof(MiniStatementEntry));

strcpy(newEntry->transactionType, "Deposit");

newEntry->amount = amount;

newEntry->next = accounts[accountNumber - 1].miniStatementHead;

accounts[accountNumber - 1].miniStatementHead = newEntry;

printf("Deposited %.2f to account number %d. New balance: %.2f\n",

amount, accountNumber, accounts[accountNumber - 1].account.balance);

}

void withdraw(BankAccount\* accounts, int accountCount) {

int accountNumber;

float amount;

printf("Enter account number: ");

scanf("%d", &accountNumber);

if (accountNumber < 1 || accountNumber > accountCount) {

printf("Invalid account number.\n");

return;

}

printf("Enter amount to withdraw: ");

scanf("%f", &amount);

if (amount <= 0 || amount > accounts[accountNumber - 1].account.balance) {

printf("Invalid withdraw amount.\n");

return;

}

accounts[accountNumber - 1].account.balance -= amount;

// Add to mini statement

MiniStatementEntry\* newEntry = (MiniStatementEntry\*)malloc(sizeof(MiniStatementEntry));

strcpy(newEntry->transactionType, "Withdraw");

newEntry->amount = amount;

newEntry->next = accounts[accountNumber - 1].miniStatementHead;

accounts[accountNumber - 1].miniStatementHead = newEntry;

printf("Withdrew %.2f from account number %d. New balance: %.2f\n",

amount, accountNumber, accounts[accountNumber - 1].account.balance);

}

void printMiniStatement(BankAccount\* accounts, int accountCount) {

int accountNumber;

printf("Enter account number: ");

scanf("%d", &accountNumber);

if (accountNumber < 1 || accountNumber > accountCount) {

printf("Invalid account number.\n");

return;

}

BankAccount\* account = &accounts[accountNumber - 1];

printf("\nMini Statement for Account Number: %d (%s)\n",

account->account.accountNumber, account->account.accountHolder);

printf("Current Balance: %.2f\n", account->account.balance);

printf("Transactions:\n");

MiniStatementEntry\* current = account->miniStatementHead;

if (!current) {

printf("No transactions found.\n");

}

while (current != NULL) {

printf("%s: %.2f\n", current->transactionType, current->amount);

current = current->next;

}

}

void freeMiniStatements(BankAccount\* accounts, int accountCount) {

for (int i = 0; i < accountCount; i++) {

MiniStatementEntry\* current = accounts[i].miniStatementHead;

while (current != NULL) {

MiniStatementEntry\* temp = current;

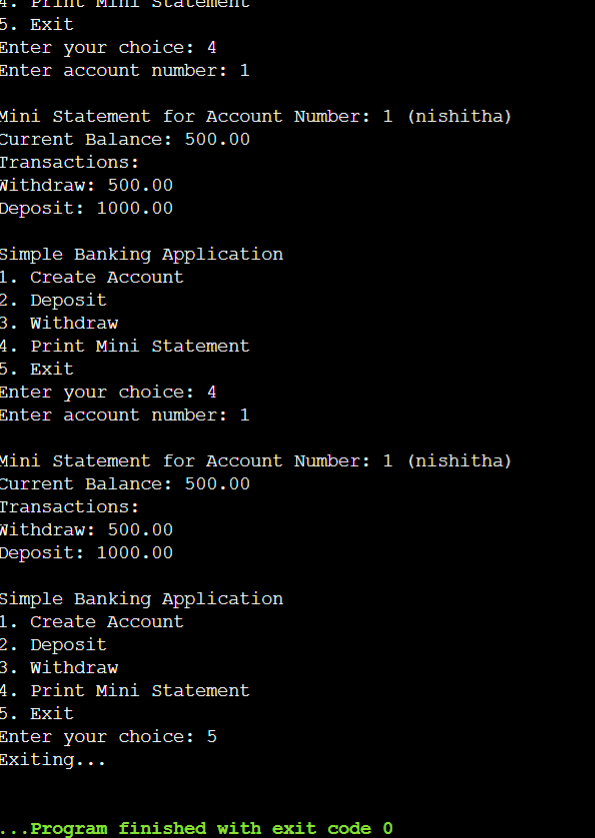
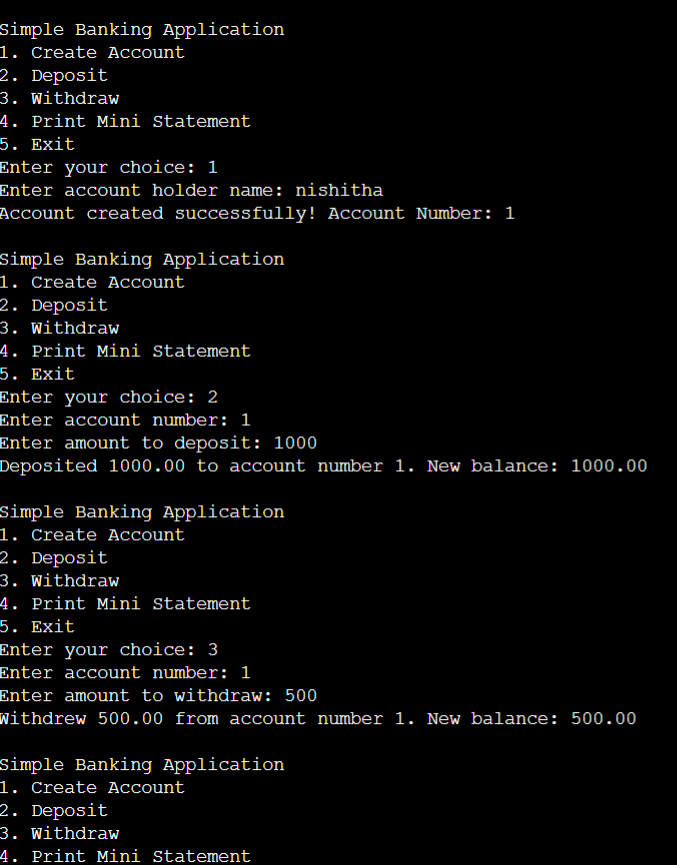
current = current->next;

free(temp);

}

}

}



Conclusion:

The development of this simple banking application demonstrates the effective use of modern mobile technologies to provide essential banking services in a secure, accessible, and user-friendly manner. Through features like user authentication, balance inquiry, fund transfers, and transaction history, the application meets the basic needs of users while ensuring data security and ease of use.

The project showcases the successful integration of front-end and back-end technologies, emphasizing responsive design and real-time data processing. It also highlights the potential of mobile platforms to enhance banking accessibility, especially in areas with limited access to traditional banking infrastructure.

This application serves as a foundational model that can be extended with additional functionalities such as bill payments, QR code-based transactions, and AI-powered financial advice. With further development and user feedback, the system can evolve into a more comprehensive mobile banking solution.

References:

1. R. Sharma, Mobile Banking Technology and Services, Journal of Banking and Financial Technology, 2021.

2. A. Agarwal, “A Study on User Experience in Mobile Banking Applications,” International Journal of Computer Applications, vol. 182, no. 12, pp. 15–20, 2020.

3. Firebase Documentation – https://firebase.google.com/docs

4. Flutter Documentation – https://docs.flutter.dev

5. MySQL Reference Manual – https://dev.mysql.com/do

6. Android Developers Guide – https://developer.android.com/guide

7. Jacob Nielsen, Usability Engineering, Morgan Kaufmann, 1993.

8. OWASP Mobile Security Project – https://owasp.org/www-project-mobile-security/

9. GitHub – Sample Flutter Banking App Projects – https://github.com

10. PayPal Developer Docs – <https://developer.paypal.com/>

THANK YOU