Banking System

Group Members:

1. Sufiyaan Usmani (21K-3195)
2. Qasim Hasan (21K-3210)
3. Ahsan Ashraf (21K-3186)
4. Talha Shaikh (21K-4564)

Appendix:

The main features and functions of this banking system are explained below. To get more details such as Banking System Application, documentation, version history, and project report, please scan the barcodes.

Banking System	*Version History*	*Documentation and User Guide*	*Proposal and Final Report*

Password: fast123

Problem Statement:

The bank management system is an application for maintaining a person's account in a bank. The system provides access to the customer to create an account deposit/withdraw the cash from his account. It also allows the admin to manage the banking system.

Planned:

The bank management system is a user-friendly program that keeps track of a person's bank account. This system allows the customer to open an account and do transactions. On the other hand, this system allows the administrator to manage the Banking System and control it.

Executed:

We have successfully constructed a bank system in which a client may withdraw, deposit, transfer, and examine his balance, while the admin can verify the transaction and customer information, delete accounts that are owing to cash, update currencies, and create Customer's Database Backup.

Existing Problem:

- No admin portal to manage accounts and bank system.
- No money transfers.
- No bank policies.
- No proper checks such as age limits.
- No proper security such as pin codes and login passwords.

Proposed Solution:

- A proper admin portal so that administrator can manage, view, and delete accounts.
- Customers will be able to transfer money to other accounts.
- Proper bank policies will be shown to the customers for which they must agree before creating their account.
- Efficient error checking methods.
- Security features such as login passwords and beeps.

Solution Applied:

- We have created a banking system, and the project is divided into four sections: admin portal, customer portal, create account and applications such as currency rates checker, developer info.
- The admin has been granted authority to verify the client information database, which is one of the key reasons this banking system's application is superior and solves many problems.
- All the passwords will be checked when accessing through filing, and wrong input will stop the user from accessing data.
- The system is designed purely as an actual bank, but due to some restrictions, it still cannot be equivalent to real life, but it is close enough

Timeline Followed:

- On October 21st, brainstorming and the commencement of the project began, and it ended on December 18th.
- After the admin and customer portals were completed, ideas were added.
    1. database backup
    2. Currencies update
    3. Transaction history
    4. Sorting and searching of data
- In addition, we created technical documentation and a user guide for the project.

Important Functions Explanation:

Creating a new account:

Firstly, when we make an account, we need an account number for every user, so a function generateAccountNumber() is used to generate a unique account number and stores it in customer.accountNo. Now first and last name will be input from the user, which will be stored in customer.firstName and customer.lastName. Now the user will enter the age between 18-99, knowing that only integer value can be stored in customer.age. Here, another integerinputonly() ensures that only integer is being input. Similarly, the contact number will also be entered by the user and stored in customer.contactNumber. The user will enter the password, and the program will apply masking in it(i.e., the '*' symbol in place of characters entered). Then we finalize this function by asking yes or no to create the account; if said yes, all the data obtained by the user will be stored in a file using filing. After storing, we will close the file, and we will be back to the main interface

Login:

void loginAsAdmin();

void loginAsCustomer();

The program will first ask the admin/customer to enter their ID/Account Number. The program will then compare this number against the data present in admin.txt/customer.txt. If this ID exists, the program will ask the user to enter their password. The program will then match this entered password with the password associated with this ID (implemented through structures). The program will redirect the user to the admin/ customer portal if the password matches.

Transactions:

void depositMoney();

When the user enters an amount to deposit in his account, the program checks whether the sum of the current balance and entered amount is within the limits of our bank maximum bank balance. If the entered amount is within limits, the entered amount is added to the current balance, and the message is shown of successful withdrawal; else, an error is promoted, and the program asks to re-enter the amount.

void withdrawAmount();

When the user enters an amount to withdraw in his account, the program checks whether the current balance is greater than the requested amount. If the entered amount is less than the current balance, then entered amount is subtracted from the current balance, and the message is shown of successful withdrawal; else, an error is generated, and the program asks to re-enter the amount.

void transferAmount();

The user is asked to enter the account number he wants to transfer money (Program checks whether the account exists in the customer.txt file). If the account exists, he then enters the amount to the transfer program, then follows the algorithm of withdrawAmount() and then followed by depositMoney(). After successful transfer, a message is shown.

A feature involved in all the transaction functions is that a new temporary file temp.txt is created as the user selects one of these functions, and all the updated data is written to that file, and the customer.txt file is deleted, and temp.txt is renamed as customer.txt file.

Searching and Sorting Customer Data:

```
void searchByAccountNumber();
```

The admin enters the account number he wants to search, and the program checks whether the account exists in the customer.txt file. If it exists, the program uses the search algorithm described In (A) below.

```
void searchByName();
```

The admin enters the account number he wants to search, turns the string into uppercase, and checks whether the name exists in the customer.txt file. The program uses the search algorithm described in (A) below.

(A)

The program opens the customer.txt file and searches the entered account number/name. If found, then print the information associated with that account/name.

```
void sortAsc();

void sortDes();
```

The admin is shown the customers' data in ascending/descending order with respect to the balance each account has. This is done through bubble sorting, struct, and array(the size of the array is determined while counting all the characters in the customer.txt file).

Creating Customer Database Backup:

```
void createCustomerDataBaseBackup();
```

When the admin performs this option, the program first fetches the current date and time and stores it in a string. This will be the file name of the new backup file. The program will then create a new file with this string. The program will then read data from customer.txt and copy that data into this new file. This new backup file will be stored in the *backup* folder.

Deleting Account:

```
void deleteAccount();          // for customer to delete his own account

void adminDeleteAccount();     // for admin to delete accounts
```

For a customer, the program confirms from the user if they want to delete their account. If 'y' is pressed, a new temporary file will be made in which all the data from customer.txt will be copied except the account, which is to be deleted. Now the current customer.txt file will be deleted, and the temporary file will be renamed to customer.txt