BANKING SYSTEM

A MINIPROJECT REPORT

for

60 IT 403 – WEB TECHNOLOGY

Submitted by

KARTHIKEYAN R (2303737720521034)

KARUPPANNAN S (2303737720521035)

in partial fulfillment of the requirement

for the award of the degree

of

B.TECH

in

INFORMATION TECHNOLOGY

K.S. RANGASAMY COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

TIRUCHENGODE – 637 215

MAY 2025

K.S. RANGASAMY COLLEGE OF TECHNOLOGY
TIRUCHENGODE - 637 215

BONAFIDE CERTIFICATE

Certified that this project report titled **BANKING SYSTEM** is the Bonafide work of **KARTHIKEYAN R (2303737720521034)** and **KARUPPANNAN S (2303737720521035)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr.R.POONKUZHALI,M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of Information Technology

K.S.Rangasamy College of Technology

Tiruchengode - 637 215

SIGNATURE

Mr.K.SENTHIL KUMAR,M.E.,

SUBJECT HANDLER

Assistant Professor

Department of Information Technology

K.S.Rangasamy College of Technology

Tiruchengode - 637 215

Internal Examiner1 **Internal Examiner2**

DECLARATION

We jointly declare that the project report on **BANKING SYSTEM** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of B.TECH IT. This project report is submitted on the partial fulfilment of the requirement of the award of Degree of B.Tech.

Signature

_____

KARTHIKEYAN R

_____

KARUPPANNAN S

Place: Tiruchengode

Date:

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to our honourable Correspondent **Thiru R. SRINIVASAN, B.B.M., MISTE** for providing immense facilities at our institution.

We would like to express my special thanks and gratitude to our Vice Chairman **Shri. K. S. SACHIN, MBA (UK).,** who has been the key source of motivation for me throughout the completion of my course and project work.

We are very proudly rendering our thanks to our Principal **Dr. R. GOPALAKRISHNAN, M.E., Ph.D.,** for the facilities and the encouragement given by him to the progress and completion of our project.

We proudly render our immense gratitude to the Head of the Department **Dr.R.POONKUZHALI, M.E., Ph.D.,** for her effective leadership, encouragement and guidance in the project.

We are highly indebted to provide our heart full thanks to our supervisor **Dr.K.SENTHIL KUMAR, M.E., PhD.,** Assistant Professor for his valuable ideas, encouragement and supportive guidance throughout the project.

We wish to extend our sincere thanks to all faculty members of our Information Technology Department for their valuable suggestions, kind co-operation and constant encouragement for successful completion of this project.

We wish to acknowledge the help received from various Departments and various individuals during the preparation and editing stages of the manuscript.

ABSTRACT

The **Banking Management System** is a lightweight, browser-based web application developed using **HTML**, **CSS** (with **Bootstrap** for responsive design), and **JavaScript** that enables users to efficiently manage personal or small-scale banking activities without relying on server-side infrastructure or databases by utilizing the built-in **localStorage API** for persistent data storage. The application features a clean and intuitive interface with navigation links for **Login**, **Accounts**, and **About** pages, allowing users to input credentials and store session information locally for personalized interactions such as tracking account balances and recent transactions. The core functionality on the **Accounts** page allows users to **create**, **view**, **search**, **deposit into**, **withdraw from**, and **transfer funds between** accounts by entering details like **account number**, **holder name**, **account type**, and **initial deposit**, with real-time filtering and status summaries displaying **total accounts**, **available balance**, and the **current logged-in user**.Enhancing usability, the app includes a **dark mode toggle**, **dynamic table updates**, and **event-driven programming** managed through JavaScript's **DOM manipulation**, ensuring all data persists across sessions without external servers. Serving both as a practical personal finance tracker and a learning resource, this project demonstrates essential web development concepts such as **form handling**, **client-side storage**, **responsive design**, **dynamic rendering**, and **user session management**, providing a strong foundation for building more advanced full-stack financial applications in the future.In addition to its core functionality, the Banking Management System offers enhanced features such as **automated low balance alerts**, **monthly account summaries**, and a **budget tracking system**, making it more comprehensive and realistic for everyday use. Users are notified of **suspicious transactions** or **balance thresholds** via in-app alerts triggered by transaction activity and current account values. **Account types** are selectable from a predefined list, supporting better categorization and filtering. The application also supports **exporting account data into CSV and PDF formats**, enabling users to back up their financial records or share them externally.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In today's fast-paced and increasingly digital world, efficiently managing personal or small-scale financial records has become essential for better organization and accurate tracking of transactions. Whether it's monitoring deposits and withdrawals, managing multiple accounts, or reviewing transaction histories, maintaining a clear overview of one's financial activity is crucial for convenience and control.
To address this challenge, the **Banking Management System** web application offers a convenient, efficient, and user-friendly solution for managing personal banking activities in real-time.The system is a lightweight, browser-based application developed using core web technologies: **HTML**, **CSS** (leveraging the **Bootstrap** framework for responsive design), and **JavaScript**. It is designed to be simple yet functional, targeting users who need a straightforward tool to track and manage accounts without requiring complex software or backend infrastructure. One of the key features is the ability to add new account entries by providing details such as **account holder name**, **account number**, **account type**, and **initial deposit amount**. These records are stored using the browser's **localStorage**, allowing data persistence across sessions while eliminating the need for user accounts or server-side databases. This approach simplifies deployment and ensures a quick, secure experience for users. The application includes several useful tools to help users manage their financial records effectively, including **dynamic filtering and searching** by account number or name, and the ability to **deposit**, **withdraw**, or **transfer funds** with real-time balance updates.The system displays all account and transaction records in a clear table with options to **edit** or **delete** entries, and provides summary statistics like **total balance**, **number of accounts**, and **logged-in user information**. Overall, the **Banking Management System** serves as both a practical tool for managing personal finances and an excellent learning project for budding developers.

1.1 HTML

HTML (HyperText Markup Language) is the backbone of the web page, responsible for the structural layout of content. In this grocery store website, HTML has been used to:

Create the main structure of pages like **Home**, **Create Accout**, **Delete**, and **Delete**. Display content such as product names, prices, images, and descriptions. Define navigation elements and layout containers for styling and interaction.

HTML enables web browsers to interpret and display the structure of the site in a semantic way, making it readable and accessible.

1.2 CSS

CSS (Cascading Style Sheets) is used to define the visual appearance of HTML content. It controls the layout, colors, fonts, spacing, and responsiveness of the website.

In the project, CSS is used for:

Styling product cards, buttons, menus, and form elements.

The term "cascading" refers to the hierarchy that determines how styles are applied, allowing flexibility in managing design rules

1.3 JAVA SCRIPT

JavaScript (JS) is a lightweight, high-level programming language that adds interactivity and logic to web pages.

Enable dynamic functionalities such as adding items to the cart, updating quantities, and calculating totals.Handle form validation during the checkout process.Implement real-time updates in the shopping cart without refreshing the page.Add interactivity to the navigation and filtering of product categories

JavaScript runs entirely on the client side in this project, meaning no server-side code or databases are used. All data is handled within the browser using local variables or local storage

CHAPTER 2

MODULE DESCRIPTION

2.1 USER LOGIN PAGE

The user login page of the Banking System is the first screen users interact with upon accessing the application. It features a central heading, "Welcome to the Banking System," along with a prompt to log in using their credentials. Styled with Bootstrap, the page offers a responsive and clean interface for easy navigation. The login form requires the user to input a username and password. The `showSection('login')` script ensures this login interface is displayed by default, while other sections like account details, transaction history, or loan applications remain hidden until the user is authenticated..

2.2 CREATE ACCOUNT PAGE

The "Create Account" page in the Banking System allows new users to register by providing their personal information. The page includes a form with fields for full name, date of birth, email address, contact number, address, and password. Once the form is completed and submitted, a new account is created, and the user is redirected to the login page. Designed for simplicity and ease of use, the page features Bootstrap-styled components for a clean layout. Required fields are marked, and input validation ensures that all information is correct before submission..

2.3 DASHBOARD PAGE

The "Dashboard" page in the Banking System serves as the user's central hub after logging in. It displays an overview of the user's financial data, including account balance, recent transactions, pending payments, and loan status. The page also shows useful statistics such as total savings, total spent, and available credit. The dashboard is designed for quick navigation, allowing users to easily access services like transferring funds, viewing detailed account statements, or applying for a loan. With a responsive layout, users can access their account on both desktop and mobile devices.

2.4 TRANSACTION HISTORY PAGE

The "Transaction History" page provides users with a detailed log of their financial transactions. It displays transactions such as deposits, withdrawals, transfers, and payments in a table format, showing the transaction date, type, amount, and status (completed or pending). Users can filter the history by date range or transaction type.

2.5 TRANSFER FUND PAGE

The "Transfer Funds" page allows users to send money to another account, either within the same bank or externally. The page features fields to enter the recipient's account number, the transfer amount, and a description of the transaction. It also includes a dropdown to select the account from which the funds will be transferred. Users are required to authenticate the transaction using a one-time password (OTP) sent to their registered phone or email address for added security. Once the transfer is confirmed, a receipt is generated, and the balance is updated in real-time.

2.6 LOAN APPLICATION PAGE

The "Loan Application" page enables users to apply for loans through the Banking System. The page contains a form with fields to enter loan details, such as the loan amount, purpose, and repayment terms. It also asks for personal and financial information to assess the user's eligibility. Upon submission, the loan application is processed, and the user receives an approval or rejection notification based on their credit score and financial status. The page also provides users with information on different loan types and their respective interest rates.

2.7 ACCOUNT SETTING PAGE

The "Account Settings" page allows users to manage their personal information and security preferences. Users can update their contact details, change their password, enable two-factor authentication, and configure account notifications. The page also includes options to view account activity and manage connected devices. Designed for security and convenience, all updates require authentication via OTP to protect user privacy. Any changes made are reflected immediately, and users are notified of important changes to their account settings

2.8 ABOUT PAGE

The "About" page in the Banking System provides users with an overview of the system's features, goals, and the bank's services. It includes a brief description of the bank's mission, values, and how the Banking System facilitates secure, efficient financial transactions. This page also provides users with contact details, customer service hours, and links to terms and conditions, privacy policies, and FAQs. It serves as an informational resource, helping users understand the functionality of the system and the various financial products offered.

2.9 FEEDBACK AND SUGGESTION PAGE

The "Feedback and Suggestions" page allows users to share their experiences, report issues, or suggest improvements for the Banking System. The page includes a simple form with fields for the user's name, email address, feedback type (e.g., bug report, suggestion, complaint), and a text area for user comments. Submitted feedback is stored in local storage or can be sent to a backend server for further analysis. This page encourages user engagement, helps identify areas for improvement, and demonstrates the bank's commitment to customer satisfaction.

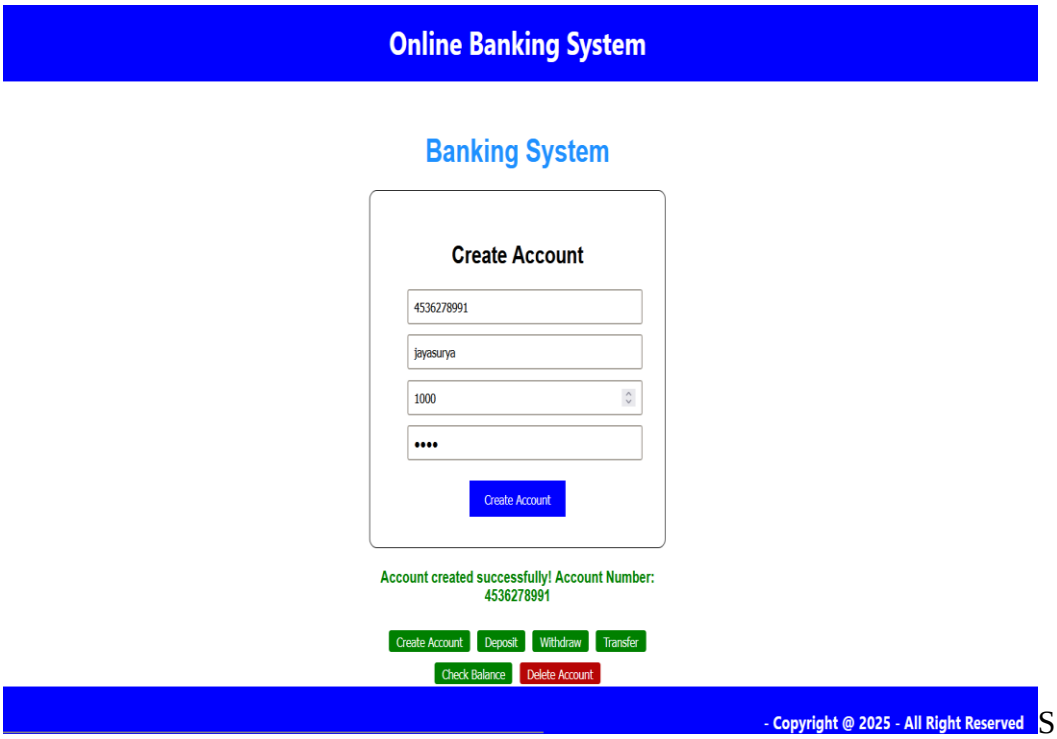
2.10 ACCOUNT STATEMENT PAGE

The "Account Statement" page provides users with a detailed summary of their account's financial activity over a specific period. It displays a breakdown of deposits, withdrawals, transfers, and fees, organized by date. Users can select a date range (e.g., past month, custom range) and download the statement in PDF or CSV format for record-keeping or tax purposes. This page helps users track their financial activity, manage their budget, and ensure that all transactions are accurate and up to date.

CHAPTER 3

RESULTS AND DISCUSSION

FIGURE 3.1 DESKTOP VIEW OF THE CREATE ACCOUNT PAGE



S

FIGURE 3.2 DESKTOP VIEW OF THE WITHDRAW MONEY

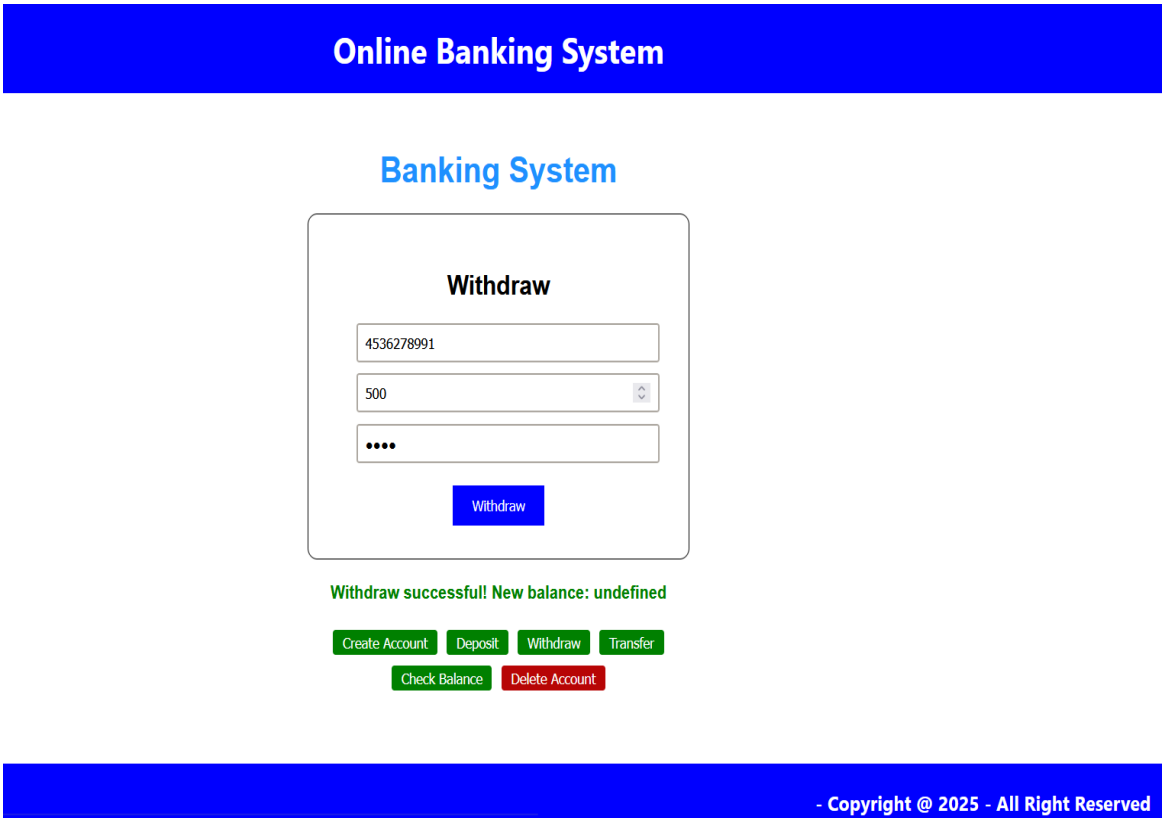


Figure 3.3 Check Balance Page

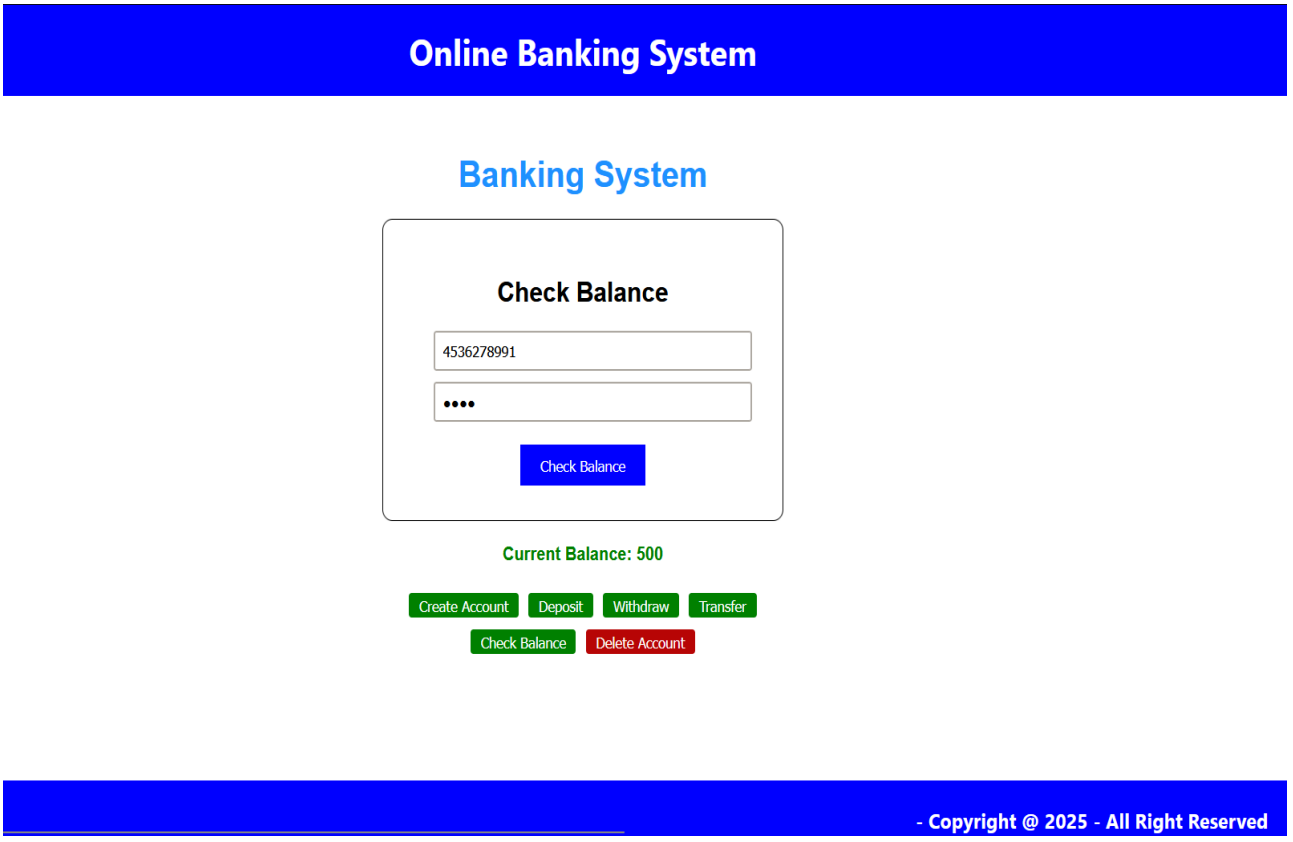


Figure 3.4 Deposit Page

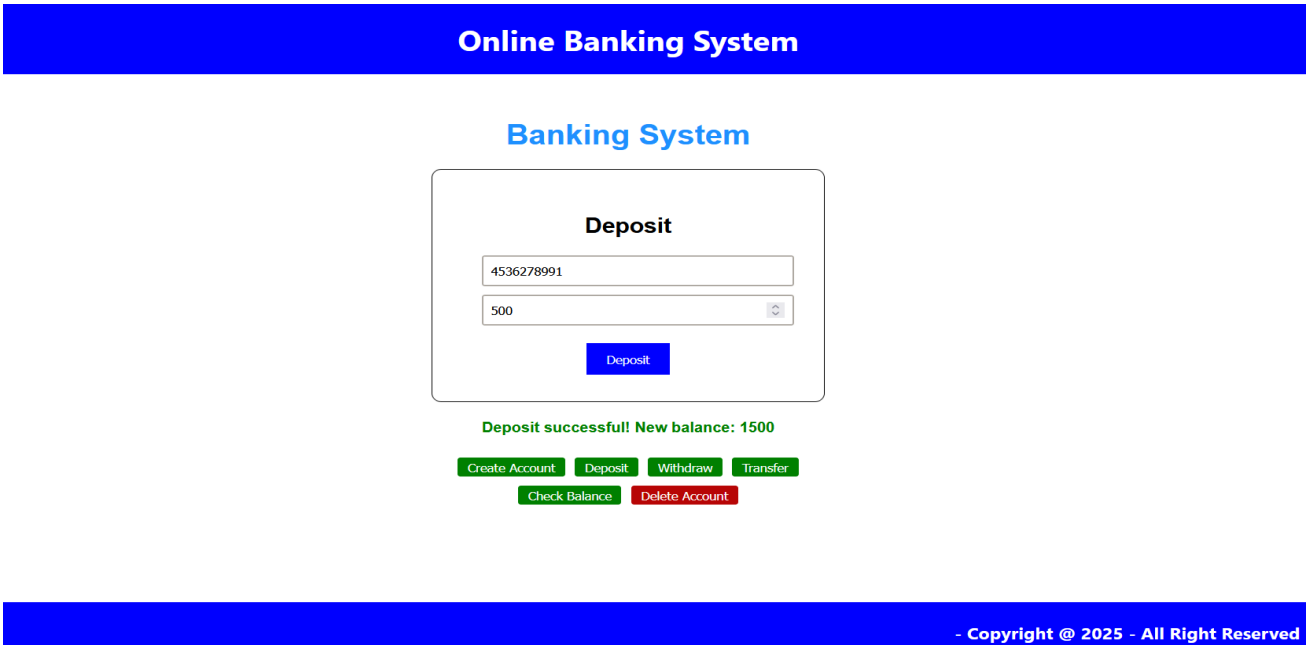
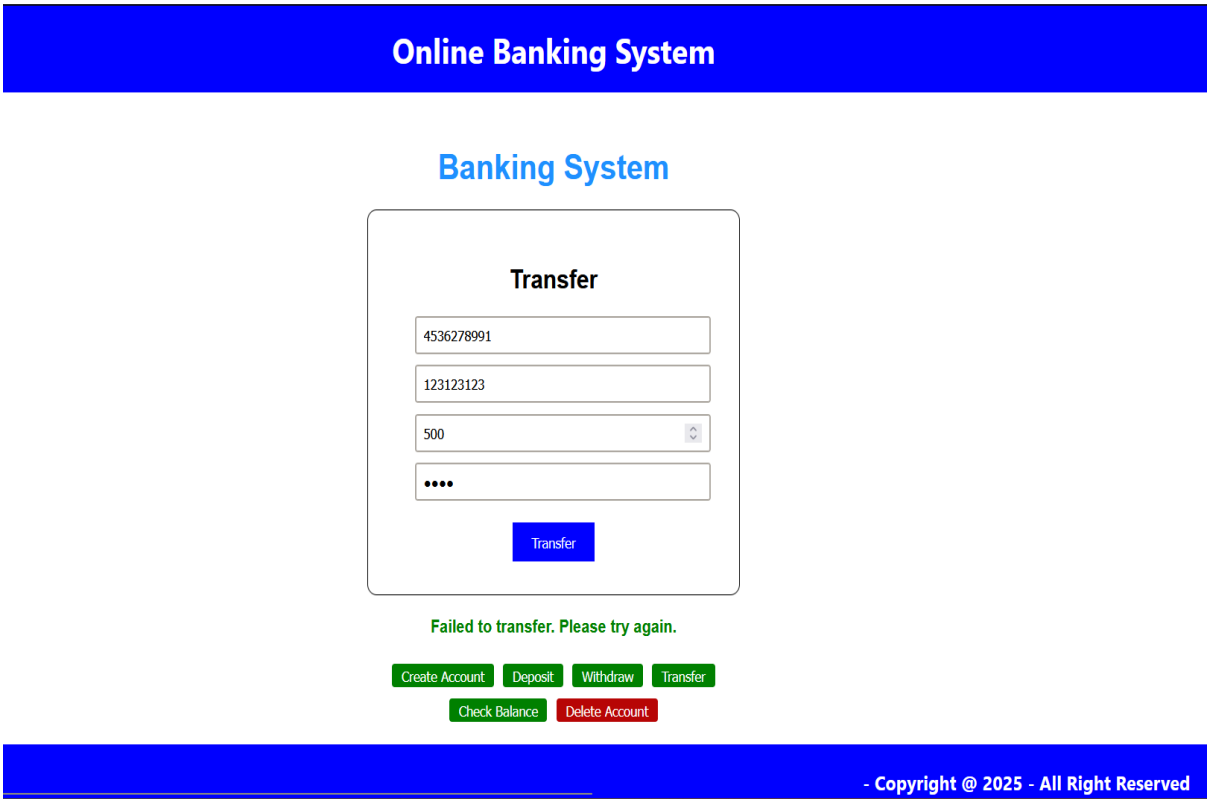
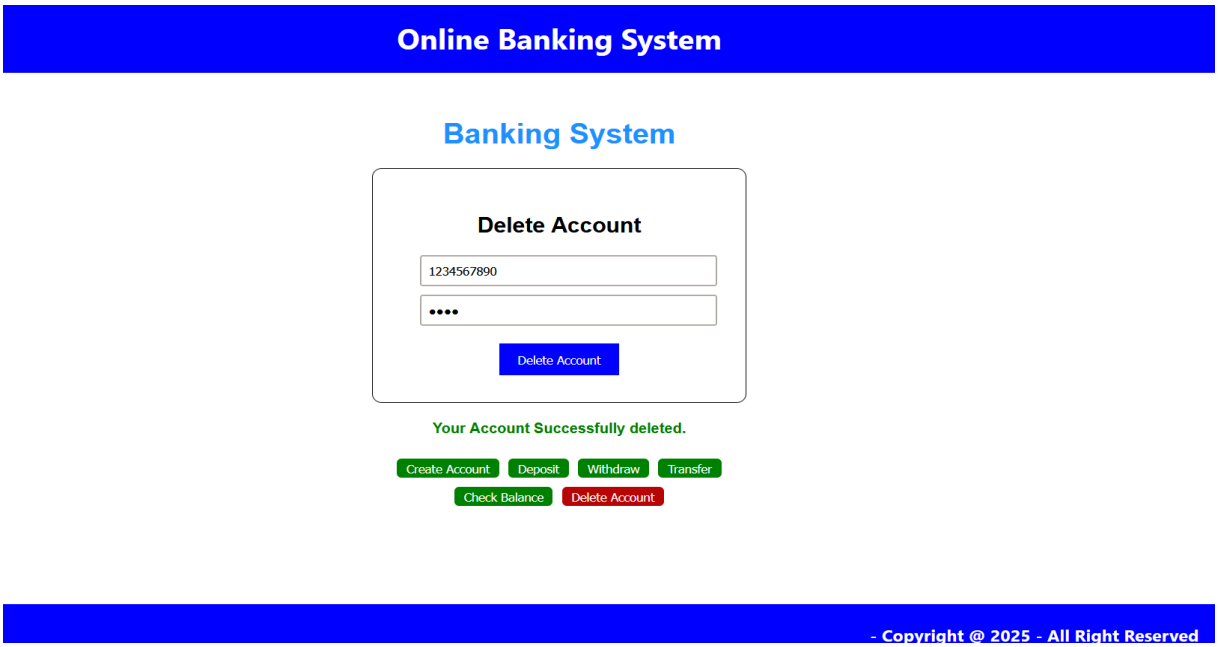


Figure 3.5 Transfer Page



3.6 Account Delete page



CHAPTER 5

CONCLUSION

The Banking System project is a sophisticated web-based application designed to streamline and enhance the management of user accounts, transactions, and banking services. Developed using core web technologies like HTML, CSS, and JavaScript, this application provides a seamless and secure interface for users to manage their finances. The primary goal of this project is to simplify the banking process, ensuring accessibility, transparency, and a secure digital environment for users to perform banking operations such as transferring funds, viewing transaction history, applying for loans, and managing their accounts.Users can access various banking features such as checking account balances, viewing detailed transaction logs, making secure fund transfers, and applying for loans, all while ensuring data integrity through real-time updates and security measures. HTML is used to structure the content, CSS adds responsive design elements, and JavaScript powers interactivity, such as processing transactions, managing user authentication, and generating reports.A standout feature of the system is its secure login and transaction process, incorporating OTP-based authentication for fund transfers and login. The system also includes essential tools for generating account statements and transaction history in CSV and PDF formats, which help users maintain records for budgeting, tax filing, or sharing with financial advisors. These functionalities ensure that users have complete control over their banking experience while facilitating smooth record-keeping and reporting.The project follows a modular approach, allowing for future integrations, such as mobile banking, AI-driven analytics, or customer service features like chatbots. The navigation bar provides easy access to key areas like the Dashboard, Transaction History, Fund Transfer, Loan Application, and Account Settings, contributing to an intuitive and user-friendly experience.In conclusion, the Banking System project highlights core front-end development skills and demonstrates how simple web technologies can be used to create secure, efficient, and user-friendly digital banking solutions.

APPENDIX

FRONTEND CODE (REACT JS) :

App.js:

```
import { useState } from "react";
import './App.css'

function App() {
   const [accountNumber, setAccountNumber]=useState("");
   const [accountHolder, setAccountHolder]=useState("");
   const [amount, setAmount]=useState("");
   const [pin, setPin]=useState("");
   const [receiverAccount, setReceiverAccount]=useState("");
   const [message, setMessage]=useState("");
   const [operation, setOperation]=useState("create");

   // Create Account
   const createAccount=async()=>{
      try{
         const response=await
fetch("http://localhost:8080/Account/create",{
            method:"POST",
            headers:{"Content-Type":"application/json"},
            body:JSON.stringify({
               accountnumber:accountNumber,
               accountholder:accountHolder,
               pin:pin,
               currentbalance:amount,
            }),
         });

         const result=await response.json();
         if(response.ok){
            setMessage(`Account created successfully! Account Number:
${result.accountnumber}`);
         }
         else{
            setMessage(`Error: ${result.message}`);
         }
      }
      catch(error){
         setMessage("Failed to create account. Please try again.");
         console.error(error);
      }
   };

   // Deposit
```

```
   const deposit=async()=>{
      try{
         const response=await
fetch(`http://localhost:8080/Account/deposit/${accountNumber}?amoun
t=${amount}`,{
            method: "POST",
         })

         const result=await response.json();
         if(response.ok){
            setMessage(`Deposit successful! New balance:
${result.currentbalance}`);
         }
         else{
            setMessage("Deposit failed. Please check the amount.");
         }
      }
      catch(error){
         setMessage("Failed to deposit. Please try again.")
         console.error(error)
      }
   };

   // Withdraw
   const withdraw=async()=>{
      try{
constresponse=await
fetch(`http://localhost:8080/Account/withdraw/${accountNumber}?amo
unt=${amount}&pin=${pin}`,{
            method:"POST",
         });

         const result = await response.json();
         if(response.ok){
            setMessage(`Withdraw successful! New balance:
${result.currentbalance}`);
         }
         else{
            setMessage("Withdraw failed. Please check the amount.");
         }
      }
      catch(error){
         setMessage("Failed to withdraw. Please try again.");
         console.error(error);
      }
   };

   // Transfer
   const transfer=async()=>{
      try{
```

```
            const response=await
fetch(`http://localhost:8080/Account/transfer?senderAccountnumber=${
accountNumber}&receiverAccountnumber=${receiverAccount}&amou
nt=${amount}&pin=${pin}`,{
                method:"POST",
        });

            const result=await response.json();
            setMessage(result);
        }
        catch(error){
            setMessage("Failed to transfer. Please try again.");
            console.error(error);
        }
    };

    // Check Balance
    const checkBalance=async()=>{
try {
 constresponse= await
fetch(`http://localhost:8080/Account/balance/${accountNumber}?pin=$
{pin}`, {
        method: "GET"
    });
            const result=await response.json();
            setMessage(`Current Balance: ${result}`);
        }
        catch (error){
            setMessage("Failed to fetch balance. Please try again.");
            console.error(error);
        }
    };

    // Delete Account
    const deleteAccount=async()=>{
        try{
 const response=await
fetch(`http://localhost:8080/Account/delete/${accountNumber}?pin=${p
in}`,{
            method:"DELETE",
            });

            const result=await response.json();
            setMessage(result);
        }
        catch(error){
            setMessage("Failed to delete account. Please try again.");
            console.error(error);
        }
    };
```

```
  return (
    <div style={{ maxWidth: "400px", margin: "50px auto", textAlign:
"center", fontFamily: "Arial, sans-serif" }}>
      <h1 style={{color:"dodgerblue"}}>Banking System</h1>

      {/* Create Account Section */}
      {operation==="create" && (
        <div className="form-design">
          <h2>Create Account</h2>
          <input
            type="text"
            placeholder="Account Number"
            value={accountNumber}
            onChange={(e) => setAccountNumber(e.target.value)}
            style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
          />
          <input
            type="text"
            placeholder="Account Holder Name"
            value={accountHolder}
            onChange={(e) => setAccountHolder(e.target.value)}
            style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
          />
          <input
            type="number"
            placeholder="Initial Balance"
            value={amount}
            onChange={(e) => setAmount(e.target.value)}
            style={{display:"block",margin:"10px
auto",padding:"8px", width:"100%"}}
          />
          <input
            type="password"
            placeholder="PIN"
            value={pin}
            onChange={(e) => setPin(e.target.value)}
            style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
          />
          <button
            onClick={createAccount}
            style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
          >
            Create Account
          </button>
```

```
            </div>
          )}

          {/* Deposit Section */}
          {operation === "deposit" && (
            <div className="form-design">
              <h2>Deposit</h2>
              <input
                type="text"
                placeholder="Account Number"
                value={accountNumber}
                onChange={(e) => setAccountNumber(e.target.value)}
                style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
              />
              <input
                type="number"
                placeholder="Amount"
                value={amount}
                onChange={(e) => setAmount(e.target.value)}
                style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
              />
              <button
                onClick={deposit}
                style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
              >
                Deposit
              </button>
            </div>
          )}

          {/* Withdraw Section */}
          {operation === "withdraw" && (
            <div className="form-design">
              <h2>Withdraw</h2>
              <input
                type="text"
                placeholder="Account Number"
                value={accountNumber}
                onChange={(e) => setAccountNumber(e.target.value)}
                style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
              />
              <input
                type="number"
                placeholder="Amount"
                value={amount}
```

```
                    onChange={(e) => setAmount(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <input
                    type="password"
                    placeholder="PIN"
                    value={pin}
                    onChange={(e) => setPin(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <button
                    onClick={withdraw}
                    style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
                >
                    Withdraw
                </button>
            </div>
        )}

        {/* Transfer Section */}
        {operation === "transfer" && (
            <div className="form-design">
                <h2>Transfer</h2>
                <input
                    type="text"
                    placeholder="Sender Account Number"
                    value={accountNumber}
                    onChange={(e) => setAccountNumber(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <input
                    type="text"
                    placeholder="Receiver Account Number"
                    value={receiverAccount}
                    onChange={(e) => setReceiverAccount(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <input
                    type="number"
                    placeholder="Amount"
                    value={amount}
                    onChange={(e) => setAmount(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
```

```
                />
                <input
                    type="password"
                    placeholder="PIN"
                    value={pin}
                    onChange={(e) => setPin(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <button
                    onClick={transfer}
                    style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
                >
                    Transfer
                </button>
            </div>
        )}

        {/* Check Balance Section */}
        {operation === "checkbalance" && (
            <div className="form-design">
                <h2>Check Balance</h2>
                <input
                    type="text"
                    placeholder="Account Number"
                    value={accountNumber}
                    onChange={(e) => setAccountNumber(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <input
                    type="password"
                    placeholder="PIN"
                    value={pin}
                    onChange={(e) => setPin(e.target.value)}
                    style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
                />
                <button
                    onClick={checkBalance}
                    style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
                >
                    Check Balance
                </button>
            </div>
        )}
```

```
        {/* Delete Account Section */}
        {operation === "delete" && (
          <div className="form-design">
            <h2>Delete Account</h2>
            <input
              type="text"
              placeholder="Account Number"
              value={accountNumber}
              onChange={(e) => setAccountNumber(e.target.value)}
              style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
            />
            <input
              type="password"
              placeholder="PIN"
              value={pin}
              onChange={(e) => setPin(e.target.value)}
              style={{ display: "block", margin: "10px auto", padding:
"8px", width: "100%" }}
            />
            <button
              onClick={deleteAccount}
              style={{ padding: "10px 20px", backgroundColor:
"blue", color: "white", border: "none", cursor: "pointer", marginTop:
"10px" }}
            >
              Delete Account
            </button>
          </div>
        )}

        <p style={{ marginTop: "20px", fontWeight: "bold", color:
message.startsWith("Error") ? "red" : "green" }}>{message}</p>

        <div className="btns" style={{ marginTop: "20px" }}>
          <button onClick={() => setOperation("create")}>Create
Account</button>
          <button onClick={() =>
setOperation("deposit")}>Deposit</button>
          <button onClick={() =>
setOperation("withdraw")}>Withdraw</button>
          <button onClick={() =>
setOperation("transfer")}>Transfer</button>
          <button onClick={() =>
setOperation("checkbalance")}>Check Balance</button>
          <button onClick={() => setOperation("delete")}>Delete
Account</button>
        </div>
```

```
            <h3>created by <span
style={{color:"dodgerblue"}}>Karthikeyan R</span></h3>
        </div>
    );
}

export default App;
```

App.css :

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

.form-design{
  padding:30px 50px;
  border: 1px solid black;
  border-radius: 10px;
}

.form-design input{
  border: 2px solid rgba(38, 23, 5, 0.372);
  border-radius: 3px;
}

.btns button{
  padding: 5px 10px;
  border-radius: 3px;
  background-color:green;
  border: none;
  color: white;
  margin: 5px;
}

.btns button:nth-child(6){
  background-color: rgb(183, 5, 5);
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
```

```
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Api.js :

```
import axios from 'axios';

const API_BASE_URL = "http://localhost:8080/Account";  // Backend
URL

export const createAccount = (accountData) => {
    return axios.post(`${API_BASE_URL}/create`, accountData);
};

export const deposit = (accountNumber, amount) => {
    return
axios.post(`${API_BASE_URL}/deposit/${accountNumber}?amount=$
{amount}`);
};

export const withdraw = (accountNumber, amount, pin) => {
    return
axios.post(`${API_BASE_URL}/withdraw/${accountNumber}?amount
=${amount}&pin=${pin}`);
};

export const transfer = (sender, receiver, amount, pin) => {
    return
axios.post(`${API_BASE_URL}/transfer?senderAccountnumber=${sen
der}&receiverAccountnumber=${receiver}&amount=${amount}&pin=
${pin}`);
};
```

```
export const checkBalance = (accountNumber, pin) => {
   return
axios.get(`${API_BASE_URL}/balance/${accountNumber}?pin=${pin
}`);
};

export const deleteAccount = (accountNumber, pin) => {
   return
axios.delete(`${API_BASE_URL}/delete/${accountNumber}?pin=${pi
n}`);
};
```

Index.js :

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

BACKEND CODE (JAVA WITH SPRIN GBOOT) :

Account.java :

```
package com.example.Bank.Account.Management.System.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data
public class Account {
   @Id
   private String Accountnumber;
   private String Accountholder;
   private double Pin;
   private double Currentbalance;

   public Account() {
   }
```

```
    public Account(String Accountnumber, String Accountholder, double
Pin, double Currentbalance) {
        this.Accountnumber = Accountnumber;
        this.Accountholder = Accountholder;
        this.Pin = Pin;
        this.Currentbalance = Currentbalance;
    }

    public String getAccountnumber() {
        return Accountnumber;
    }

    public void setAccountnumber(String Accountnumber) {
        this.Accountnumber = Accountnumber;
    }

    public String getAccountholder() {
        return Accountholder;
    }

    public void setAccountholder(String Accountholder) {
        this.Accountholder = Accountholder;
    }

    public double getPin() {
        return Pin;
    }

    public void setPin(double Pin) {
        this.Pin = Pin;
    }

    public double getCurrentbalance() {
        return Currentbalance;
    }

    public void setCurrentbalance(double Currentbalance) {
        this.Currentbalance = Currentbalance;
    }

    public void deposit(double amount) {
        this.Currentbalance += amount;
    }

    public boolean withdraw(double amount, double givenpin) {
        if (givenpin == Pin && Currentbalance >= amount) {
            Currentbalance -= amount;
            return true;
        }
        return false;
```

```
    }

    public boolean transfer(double amount, double givenpin, Account
receiver) {
        if (givenpin == Pin && Currentbalance >= amount) {
            Currentbalance -= amount;
            receiver.deposit(amount);
            return true;
        }
        return false;
    }
}
```

AccountService.java :

```
package com.example.Bank.Account.Management.System.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.Data;

@Entity
@Data
public class Account {
    @Id
    private String Accountnumber;
    private String Accountholder;
    private double Pin;
    private double Currentbalance;

    public Account() {
    }

    public Account(String Accountnumber, String Accountholder, double
Pin, double Currentbalance) {
        this.Accountnumber = Accountnumber;
        this.Accountholder = Accountholder;
        this.Pin = Pin;
        this.Currentbalance = Currentbalance;
    }

    public String getAccountnumber() {
        return Accountnumber;
    }

    public void setAccountnumber(String Accountnumber) {
        this.Accountnumber = Accountnumber;
    }

    public String getAccountholder() {
```

```
        return Accountholder;
    }

    public void setAccountholder(String Accountholder) {
        this.Accountholder = Accountholder;
    }

    public double getPin() {
        return Pin;
    }

    public void setPin(double Pin) {
        this.Pin = Pin;
    }

    public double getCurrentbalance() {
        return Currentbalance;
    }

    public void setCurrentbalance(double Currentbalance) {
        this.Currentbalance = Currentbalance;
    }

    public void deposit(double amount) {
        this.Currentbalance += amount;
    }

    public boolean withdraw(double amount, double givenpin) {
        if (givenpin == Pin && Currentbalance >= amount) {
            Currentbalance -= amount;
            return true;
        }
        return false;
    }

    public boolean transfer(double amount, double givenpin, Account
receiver) {
        if (givenpin == Pin && Currentbalance >= amount) {
            Currentbalance -= amount;
            receiver.deposit(amount);
            return true;
        }
        return false;
    }
}
```

AccountRepo.java :

```
package com.example.Bank.Account.Management.System.repository;
import
com.example.Bank.Account.Management.System.model.Account;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
public interface AccountRepo extends JpaRepository<Account, String>
{
}
```

AccountService.java :

```
package com.example.Bank.Account.Management.System.service;

import
com.example.Bank.Account.Management.System.model.Account;
import
com.example.Bank.Account.Management.System.repository.AccountRe
po;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class AccountService {

   @Autowired
   private AccountRepo accountRepo;

   public Account CreateAccount(String accountnumber, String
accountholder, double pin, double currentbalance) {
      if (accountnumber == null) {
         throw new IllegalArgumentException("ID must not be null.");
      }
      if (accountRepo.existsById(accountnumber)) {
         throw new RuntimeException("Account with this number
already exists.");
      }
      Account account = new Account(accountnumber, accountholder,
pin, currentbalance);
      return accountRepo.save(account);
   }

   public Account deposit(String accountnumber, double amount) {
      if (amount <= 0) {
         throw new IllegalArgumentException("Deposit amount must be
positive.");
      }
      Account account = accountRepo.findById(accountnumber)
            .orElseThrow(() -> new RuntimeException("Account not
found"));
      account.deposit(amount);
      return accountRepo.save(account);
   }

   public boolean withdraw(String accountnumber, double amount,
double givenpin) {
```

```
        Account account = accountRepo.findById(accountnumber)
                .orElseThrow(() -> new RuntimeException("Account not
found"));
        if (account.withdraw(amount, givenpin)) {
            accountRepo.save(account);
            return true;
        }
        return false;
    }

    public boolean transfer(String senderaccountnumber, String
receiveraccountnumber, double amount, double pin) {
        Account sender = accountRepo.findById(senderaccountnumber)
                .orElseThrow(() -> new RuntimeException("Sender account
not found"));
        Account receiver = accountRepo.findById(receiveraccountnumber)
                .orElseThrow(() -> new RuntimeException("Receiver account
not found"));
        if (sender.transfer(amount, pin, receiver)) {
            accountRepo.save(receiver);
            accountRepo.save(sender);
            return true;
        }
        return false;
    }

    public double checkbalance(String accountnumber, double pin) {
        Account account = accountRepo.findById(accountnumber)
                .orElseThrow(() -> new RuntimeException("Account not
found"));
        if (account.getPin() == pin) {
            return account.getCurrentbalance();
        } else {
            throw new RuntimeException("Invalid pin.");
        }
    }

    public boolean deleteAccount(String accountnumber, double pin) {
        Account account = accountRepo.findById(accountnumber)
                .orElseThrow(() -> new RuntimeException("Account not
found"));
        if (account.getPin() == pin) {
            accountRepo.delete(account);
            return true;
        } else {
            throw new RuntimeException("Invalid pin. Check your pin.");
        }
    }
}
```

AccountController.java :

```
package com.example.Bank.Account.Management.System.controller;

import
com.example.Bank.Account.Management.System.model.Account;
import
com.example.Bank.Account.Management.System.service.AccountServi
ce;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.Map;

@CrossOrigin(origins = "http://localhost:3000")
@RestController
@RequestMapping("/Account")
public class AccountController {

   @Autowired
   private AccountService accountService;

   @PostMapping("/create")
   public Account createAccount(@RequestBody Account account) {
      return accountService.CreateAccount(
            account.getAccountnumber(),
            account.getAccountholder(),
            account.getPin(),
            account.getCurrentbalance()
      );
   }

   @PostMapping("/deposit/{accountnumber}")
   public Account deposit(@PathVariable String accountnumber,
@RequestParam double amount) {
      return accountService.deposit(accountnumber, amount);
   }

   @PostMapping("/withdraw/{accountnumber}")
   public ResponseEntity<Map<String, Object>>
withdraw(@PathVariable String accountnumber, @RequestParam
double amount, @RequestParam double pin) {
      boolean success = accountService.withdraw(accountnumber,
amount, pin);

      Map<String, Object> response = new HashMap<>();
      if (success) {
```

```
            response.put("message", "Withdrawal successful");
            return ResponseEntity.ok(response);
        } else {
            response.put("message", "Withdrawal failed. Check PIN or
balance.");
            return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
        }
    }

    @PostMapping("/transfer")
    public String transfer(@RequestParam String senderAccountnumber,
                           @RequestParam String receiverAccountnumber,
                           @RequestParam double amount,
                           @RequestParam double pin) {
        boolean success = accountService.transfer(senderAccountnumber,
receiverAccountnumber, amount, pin);
        return success ? "Transfer successful" : "Transfer failed. Check PIN
or balance.";
    }

    @GetMapping("/balance/{accountnumber}")
    public double checkBalance(@PathVariable String accountnumber,
@RequestParam double pin) {
        return accountService.checkbalance(accountnumber, pin);
    }

    @DeleteMapping("/delete/{accountnumber}")
    public String deleteAccount(@PathVariable String accountnumber,
@RequestParam double pin) {
        accountService.deleteAccount(accountnumber, pin);
        return "Account deleted successfully";
    }
}
```

Application.properties :

```
spring.application.name=Bank-Account-Management-System

spring.datasource.url=jdbc:mysql://localhost:3306/bank_db

spring.datasource.username=root

spring.datasource.password=123

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

}
```

REFERENCES

1) https://youtu.be/pMR_48AF-A0?si=aJvEA-jZUCchaRP6

2) https://youtu.be/ydUD1lOBfeI?si=4_1rpco-dzu6WxpA

3) https://github.com/kashishbagadia/TSF-Banking_System

4) https://github.com/PKrystian/Full-Stack-Bank

5) https://javascript.plainenglish.io/create-a-full-stack-banking-application-using-react-part-2-6fb21200613a