A Report Submitted in

Partial Fulfilment for

award of Bachelor of Technology/Master of Integrated Technology

In

DBMS (BCSE0452)

COMPUTER SCIENCE & ENGINEERING

By

Drishay Chauhan

(2301330100084)

Under the Supervision of

Ms. Neeti Taneja

Assistant Professor

Department of Computer Science & Engineering

School of Computer Science and Information Technology

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA

(An Autonomous Institute)

Affiliated to

DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW

DECLARATION

I hereby declare that the work presented in this report entitled "Secured Online Banking System", was carried out by me. I have not submitted the matter embodied in this report for the award of any degree or diploma of any other University or Institute.

Name: Drishay Chauhan

____

(Candidate's Signature)

CERTIFICATE

Certified that has carried out the project work presented in this Project Report in partial fulfillment of the requirements for the award of **Bachelor of Technology, CSE 2nd Year, 4th Sem** from **Noida Institute of Engineering & Technology** under my supervision.

Signature:_______________	Signature:____________
Ms. Neeti Taneja	Dr. Kumud Saxena
Assistant Professor	Dean SCSIT, HOD
NIET, Gr. Noida	Department of CSE
	NIET, Gr. Noida

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to **Ms. NEETI TANEJA** for their invaluable guidance, support, and constant supervision throughout the development of this project. Their expertise and encouragement have been pivotal in shaping the project.

I would also like to extend my thanks and appreciation to our esteemed **HOD (Dr. Kumud Saxena)** for their continuous motivation and support. Their encouragement has been instrumental in completing this project successfully.

Finally, I am deeply thankful to my peers and everyone who directly or indirectly contributed to this project with their knowledge, advice, and assistance.

ABSTRACT

This project presents a Secured Online Bank Management System that modernizes traditional banking software by replacing outdated terminal-based systems with a secure, scalable, and user-friendly web application. The system is built using a Java backend connected to a MySQL database, and a React frontend to deliver a responsive and accessible user experience.

Core features include user registration, secure login, account management, fund transfers, and transaction tracking. The backend ensures efficient data handling, while the frontend enhances usability through a modern interface.

Security is a key focus, with measures such as password hashing, role-based access control, input validation, and protection against SQL injection to safeguard user data and prevent unauthorized access. These implementations align with best practices for online financial applications.

The system addresses the drawbacks of legacy software—such as poor usability, limited access, and lack of data protection—by introducing a robust, flexible, and extensible digital banking solution. It lays a strong foundation for future enhancements like mobile support, analytics integration, and real-time fraud detection.

In summary, this project delivers a secure and modern banking system prototype suitable for academic demonstration and potential real-world use in small-scale financial environments.

TABLE OF CONTENTS

INTRODUCTION

1.1 Overview

The Secured Online Bank Management System is a web-based application designed to modernize traditional banking practices by replacing legacy, terminal-based systems with a more intuitive and secure platform. Built with a powerful Java backend, a MySQL relational database, and a responsive React frontend, the system ensures seamless interaction between the user interface and backend services.

This project is focused on improving both operational efficiency and user experience in banking environments. By providing a centralized digital platform, the system allows banks to manage accounts, transactions, and customer data with higher accuracy and less manual intervention. A major emphasis has been placed on data security, with robust mechanisms such as password hashing, encrypted communication, and role-based access control to protect sensitive user information from unauthorized access or cyber threats.

Additionally, the system's modular structure supports easy scalability, making it adaptable for future enhancements such as mobile banking, analytics, or AI-powered fraud detection. Overall, the project offers a reliable and efficient foundation for modern digital banking services.

1.2 Objective and Scope

The goal of this project is to design and implement a secure, scalable, and user-friendly online bank management system that meets the core operational needs of a typical banking institution while ensuring high standards of cybersecurity and usability.

Objectives:

- To implement a secure login system with features such as password hashing, encryption, and password recovery options.
- To provide essential banking functionalities, including:
  - Account information display
  - Balance inquiry
  - Deposits and withdrawals
  - Internal fund transfers
- To enforce role-based access control, ensuring that different user roles—such as customers, employees, and managers—have access only to relevant features and data.
- To ensure smooth integration between frontend and backend components for consistent performance and real-time data access.

Scope:

- The project is web-based, developed using Java for backend logic, MySQL for structured data storage, and React for the user interface.
- It focuses on implementing the core features of online banking, ensuring high usability and secure transaction management.
- The system is limited to internal banking functions such as user account management and basic transaction services.
- External modules like real-time stock trading, loan applications, credit scoring, or integration with third-party APIs are outside the scope of this version but can be considered in future iterations.

LITERATURE REVIEW

2.1 Overview of Existing Systems

Traditional bank management systems have long relied on legacy software architectures, many of which are still terminal-based and operate on mainframe infrastructure. These systems were originally built to handle basic banking functions such as account management, deposit/withdrawal processing, and transaction logging. However, they were not designed to support today's digital banking demands, such as real-time data access, multi-user concurrency, and remote user interactions.

The user interfaces in these older systems are often text-based, making them unintuitive and difficult for customers or less tech-savvy staff to navigate. Moreover, many of these applications lack modularity, meaning updates or changes require extensive redevelopment and system downtime.

Recent research and case studies reveal a growing number of vulnerabilities in these systems—especially concerning data security, access control, and compliance with modern financial regulations. As the banking sector becomes increasingly digital, these outdated systems fail to meet evolving customer expectations, regulatory standards, and operational efficiency.

2.2 Limitations of Existing Systems

Despite their foundational role in early digital banking, legacy bank management systems are plagued with several critical limitations:

• Security Issues

- Most older systems lack advanced encryption protocols and multi-factor authentication.
- User credentials may be stored in plaintext or use weak hashing algorithms, making them susceptible to cyberattacks such as phishing and brute-force attacks.

• User Interface

- Terminal-based UIs offer minimal visual feedback and are often non-intuitive, resulting in a poor user experience.
- Lack of support for graphical dashboards, responsive design, and cross-device compatibility makes them unsuitable for modern usage.

• Maintenance & Scalability

- The technology stack used in legacy systems is often obsolete, making it difficult to find skilled developers for maintenance.
- These systems have limited support for scaling operations, especially in environments with increasing customer and transaction volumes.

• Integration Difficulties

- Legacy systems often use closed architectures that hinder integration with newer technologies like cloud platforms, mobile apps, APIs, or data analytics tools.
- As a result, banks find it challenging to introduce innovative services or respond **quickly to changing market demands.**

REQUIREMENTS

3.1 Functional Requirements

- User Authentication: Secure login, password recovery, and role-based access.
- Account Management: Viewing account details, balance inquiry, deposit, withdrawal, and fund transfer functionalities.
- Data Integrity: Ensure that all financial transactions are accurate and recorded securely.

3.2 Non-Functional Requirements

- Security: Use of encryption, password hashing, and secure connection protocols.
- Performance: The system must handle multiple simultaneous transactions with minimal response time.
- Usability: Intuitive user interface for smooth navigation and ease of operation.
- Scalability: Ability to expand features and accommodate a growing user base without significant changes in infrastructure.

3.3 Software and Hardware Requirements

- Software:
  - Backend: Java (JDK 22), Apache Tomcat, MySQL.
  - Frontend: Node.js, React, HTML/CSS/JavaScript.

  - Development Tools: VS Code or Eclipse/IntelliJ, Git for version control.
- Hardware:
  - A standard desktop or server machine with 4GB RAM minimum.
  - Adequate storage for database and application files.

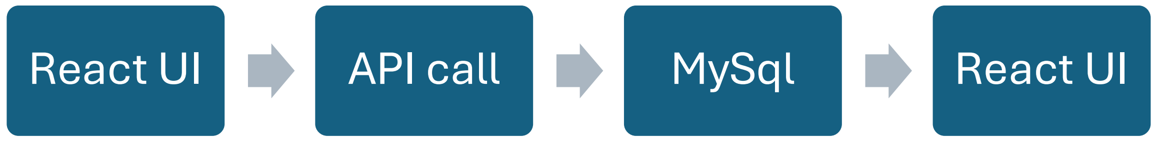
SYSTEM DESIGN AND IMPLEMENTATION

4. System Design and Implementation

4.1 System Architecture (Flow Diagram)

The proposed system adopts a three-tier architecture to ensure modularity, scalability, and maintainability. These three layers are logically separated into:

- Presentation Layer (Frontend):
  - Built using React.js, this layer provides an interactive and responsive user interface.
  - It enables users to perform actions like logging in, viewing account details, and executing transactions.
- Application Layer (Backend):
  - Developed in Java using Servlets, this layer manages all business logic.
  - It handles tasks such as authenticating users, processing transactions, verifying roles, and controlling access.
- Data Layer (Database):
  - Uses MySQL to persist user data, account information, and transaction logs.
  - Ensures data consistency, referential integrity, and secure storage.



4.2 GUI Design

The Graphical User Interface (GUI) emphasizes clarity, role-based accessibility, and ease of navigation. The interface is designed using React with reusable components and a structured layout.

Key Features:

- Landing Page:
  Displays three role-based access options:
    - Customer
    - Employee
    - Manager
- Login Screen:
  Dynamically adjusts based on selected role.
  Includes:
    - Username and password fields
    - Role selection
    - Password reset option (optional enhancement)
- Dashboard Pages:
  Custom dashboards for each user type:
    - Customer: View balance, transfer funds, view transaction history
    - Employee: Manage customer requests, assist transactions

- Manager: Access reports, monitor staff and system logs

Each dashboard features:

- A side navigation menu
- Main content area for transactions and queries
- Real-time status updates and messages

4.3 Database Design

The database is structured to reflect a realistic banking environment with proper normalization and entity relationships.

Key Tables:

- customer_details
  Contains customer profile, login credentials (hashed passwords), and account info.
- staff_details
  Stores employee data, roles, and access levels.
- transaction_history
  Tracks each deposit, withdrawal, and transfer, with fields for timestamps, amounts, and sender/receiver details.

4.4 Implementation Details

The application integrates several modern technologies and secure coding practices:

Backend (Java Servlets):

- Handles API endpoints such as:
  - POST /api/login
  - GET /api/account-details

  - POST /api/transfer
- Performs user authentication, request validation, and transaction logic
- Communicates with the MySQL database through JDBC

Database (MySQL):

- Structured schema with primary and foreign key constraints
- Ensures atomicity and consistency of all transactions
- Stores hashed passwords and sensitive information securely

Frontend (React):

- Uses React components and state management to update UI dynamically
- Calls backend APIs using fetch() or Axios
- Responsive layout compatible with desktop and tablets

Security Features:

- Passwords are hashed using a custom or standard secure hashing algorithm (e.g., SHA-256 or bcrypt)
- Role-based access control prevents unauthorized access to protected routes and APIs
- Cross-Origin Resource Sharing (CORS) configured on backend to safely allow frontend requests

TESTING

5. Testing and Evaluation

5.1 Testing Methodology

To ensure the robustness, reliability, and usability of the Secured Online Bank Management System, a comprehensive multi-phase testing strategy was adopted. This ensured the application performed correctly under normal and edge-case scenarios, and that both functional and non-functional requirements were validated.

1. Unit Testing

- Conducted on individual modules such as login, registration, fund transfer, and account queries.
- Tested using mock data to verify the accuracy of logic and error handling.
- Focused on verifying each function and method behaves as expected in isolation.
- Tools used: JUnit (for backend Java modules), Postman (for API endpoints).

2. Integration Testing

- Validated that data flows seamlessly between the frontend (React), backend (Java Servlets), and MySQL database.
- Ensured that the system components interact correctly, especially for features involving:
    - Authentication

- Transaction processing
- Role-based navigation

- Observed system behavior during multi-step transactions.

3. User Acceptance Testing (UAT)

- Conducted with a test group of 5–10 users acting as customers, employees, and managers.
- Feedback was collected to assess:
    - Ease of use
    - System responsiveness
    - Visual clarity of the interface
- Minor usability enhancements were made based on user feedback, including clearer error messages and navigation prompts.

5.2 Test Cases and Results

The following key functionalities were thoroughly tested with both valid and invalid inputs, and the results were recorded:

Test Case	Description	Expected Result	Actual Result	Status
Login Validation	Test login with correct/incorrect credentials	Access granted or denied with message	Behavior as expected	✅ Passed
Fund Transfer	Transfer funds between valid accounts	Accurate balance update and transaction log	Works correctly	✅ Passed
Balance Inquiry	Fetch account balance post-transfer	Updated balance displayed	Accurate display	✅ Passed

Test Case	Description	Expected Result	Actual Result	Status
Data Consistency	Multiple simultaneous transfers from same account	No data loss; consistent balances	Maintains integrity	✅ Passed
Role-based Access	Attempt unauthorized access (e.g., customer to manager area)	Access denied	NA	NA
API Integration	Test API endpoints with frontend	Smooth data exchange	NA	NA
Session Timeout	Simulate user inactivity	Automatic logout or warning	NA	NA

CONCLUSION AND FUTURE WORK

6. Conclusion and Future Work

6.1 Conclusion

The Secured Online Bank Management System effectively addresses the core challenges presented by outdated terminal-based banking solutions. Through the implementation of a Java-based backend, MySQL database, and a React-driven frontend, the system demonstrates how modern web technologies can deliver secure, scalable, and user-friendly financial services.

Key accomplishments of the project include:

- Secure Authentication: Password hashing and role-based access ensure data confidentiality and protect against unauthorized access.
- Improved User Experience: The intuitive GUI and role-specific dashboards enhance usability for customers, employees, and managers.
- Efficient Banking Operations: The system handles core banking functionalities like account management, balance inquiry, fund transfers, and transaction history with consistency and accuracy.
- Scalable Architecture: The modular design supports future upgrades and easy maintenance.

Overall, the project proves that a web-based solution can significantly enhance the reliability, accessibility, and security of online banking operations, while maintaining data integrity and compliance with modern development standards.

6.2 Future Work

While the current implementation meets the fundamental requirements for an online banking platform, several enhancements can be made to increase system functionality, security, and performance:

1. Enhanced Security Features

- Implement multi-factor authentication (MFA) to provide an additional layer of security.
- Use SSL/TLS encryption and OAuth2 or JWT-based session handling to protect user data during transmission and improve session management.
- Integrate fraud detection algorithms to monitor for suspicious activity.

2. Mobile Application Development

- Build a cross-platform mobile app (using React Native or Flutter) to provide customers with on-the-go access to their accounts.
- Include features like biometric login, push notifications, and QR-based payments for enhanced convenience.

3. Performance and Scalability Improvements

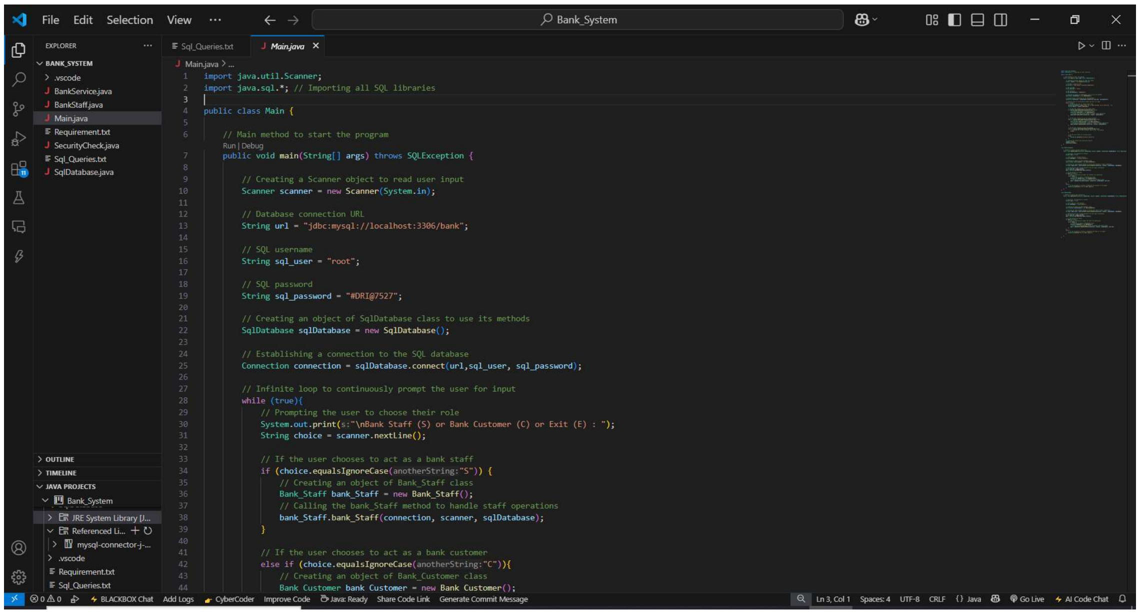
- Optimize database queries and backend services to support high-frequency transactions and real-time updates.
- Introduce caching mechanisms and load balancing for improved performance under concurrent user loads.
- Explore cloud deployment (e.g., AWS, GCP, Azure) for better resource scalability and fault tolerance.

4. Extended Functionalities

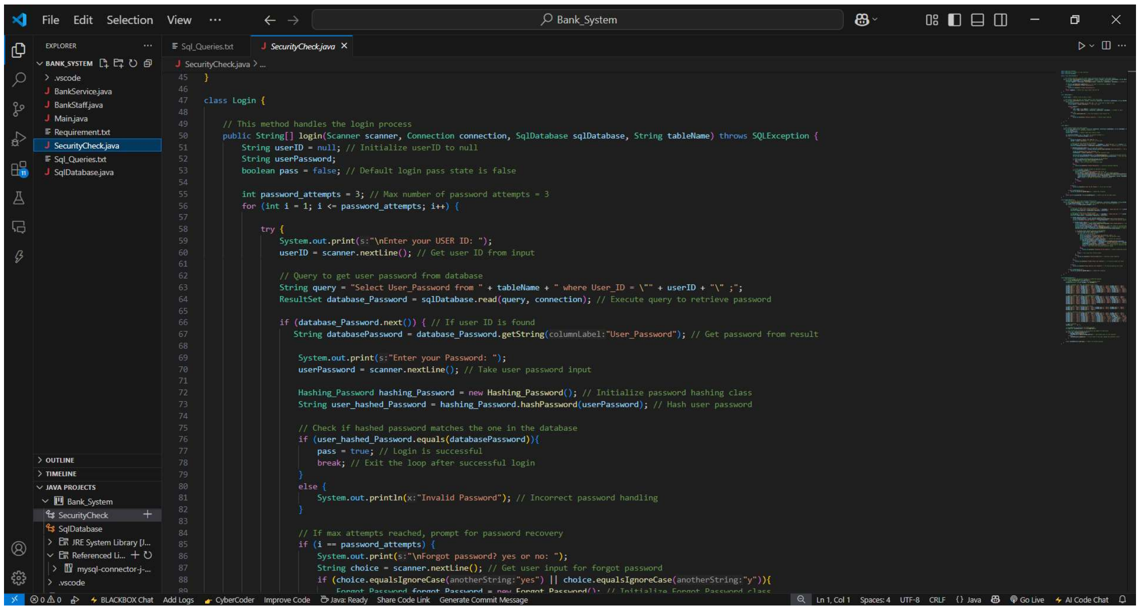
- Add support for loan management, bill payments, and real-time currency exchange.
- Provide audit logging, report generation, and user analytics dashboards for managers.

APPENDIX

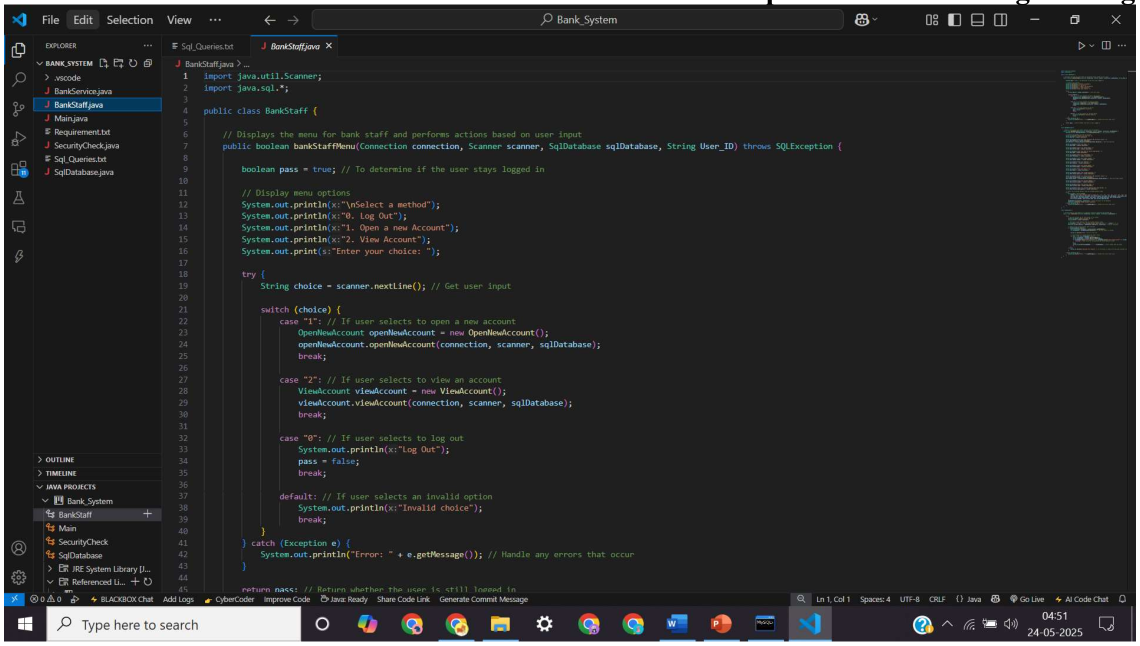
Source Code
Phase-1 of devlopment



```java
import java.util.Scanner;
import java.sql.*; // Importing all SQL libraries

public class Main {

    // Main method to start the program
    public void main(String[] args) throws SQLException {

        // Creating a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Database connection URL
        String url = "jdbc:mysql://localhost:3306/bank";

        // SQL username
        String sql_user = "root";

        // SQL password
        String sql_password = "#DRI@7527";

        // Creating an object of SqlDatabase class to use its methods
        SqlDatabase sqlDatabase = new SqlDatabase();

        // Establishing a connection to the SQL database
        Connection connection = sqlDatabase.connect(url,sql_user, sql_password);

        // Infinite loop to continuously prompt the user for input
        while (true){
            // Prompting the user to choose their role
            System.out.print(s:"\nBank Staff (S) or Bank Customer (C) or Exit (E) : ");
            String choice = scanner.nextLine();

            // If the user chooses to act as a bank staff
            if (choice.equalsIgnoreCase(anotherString:"S")) {
                // Creating an object of Bank_Staff class
                Bank_Staff bank_Staff = new Bank_Staff();
                // Calling the bank_Staff method to handle staff operations
                bank_Staff.bank_Staff(connection, scanner, sqlDatabase);
            }

            // If the user chooses to act as a bank customer
            else if (choice.equalsIgnoreCase(anotherString:"C")){
                // Creating an object of Bank_Customer class
                Bank_Customer bank_Customer = new Bank_Customer();
```

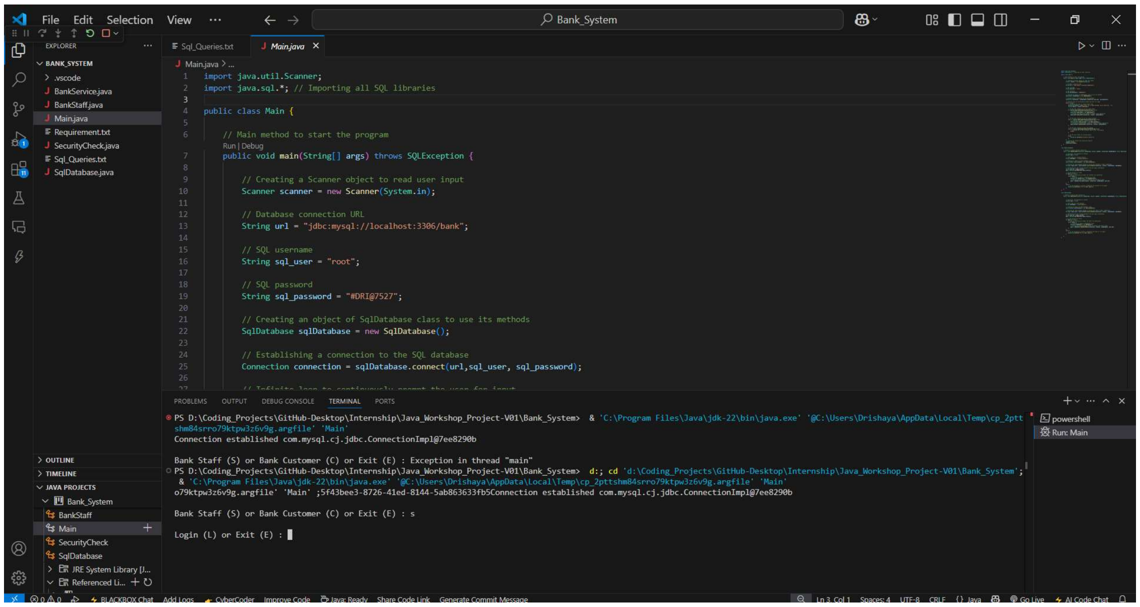


```java
}

class Login {

    // This method handles the login process
    public String[] login(Scanner scanner, Connection connection, SqlDatabase sqlDatabase, String tableName) throws SQLException {
        String userID = null; // Initialize userID to null
        String userPassword;
        boolean pass = false; // Default login pass state is false

        int password_attempts = 3; // Max number of password attempts = 3
        for (int i = 1; i <= password_attempts; i++) {

            try {
                System.out.print(s:"\nEnter your USER ID: ");
                userID = scanner.nextLine(); // Get user ID from input

                // Query to get user password from database
                String query = "Select User_Password from " + tableName + " where User_ID = \"" + userID + "\" ;";
                ResultSet database_Password = sqlDatabase.read(query, connection); // Execute query to retrieve password

                if (database_Password.next()) { // If user ID is found
                  String databasePassword = database_Password.getString(columnLabel:"User_Password"); // Get password from result

                    System.out.print(s:"Enter your Password: ");
                    userPassword = scanner.nextLine(); // Take user password input

                    Hashing_Password hashing_Password = new Hashing_Password(); // Initialize password hashing class
                    String user_hashed_Password = hashing_Password.hashPassword(userPassword); // Hash user password

                    // Check if hashed password matches the one in the database
                    if (user_hashed_Password.equals(databasePassword)){
                        pass = true; // Login is successful
                        break; // Exit the loop after successful login
                    }
                    else {
                        System.out.println(x:"Invalid Password"); // Incorrect password handling
                    }

                    // If max attempts reached, prompt for password recovery
                    if (i == password_attempts) {
                        System.out.print(s:"\nForgot password? yes or no: ");
                        String choice = scanner.nextLine(); // Get user input for forgot password
                        if (choice.equalsIgnoreCase(anotherString:"yes") || choice.equalsIgnoreCase(anotherString:"y")){
                            Forgot_Password forgot_Password = new Forgot_Password(); // Initialize Forgot_Password class
```



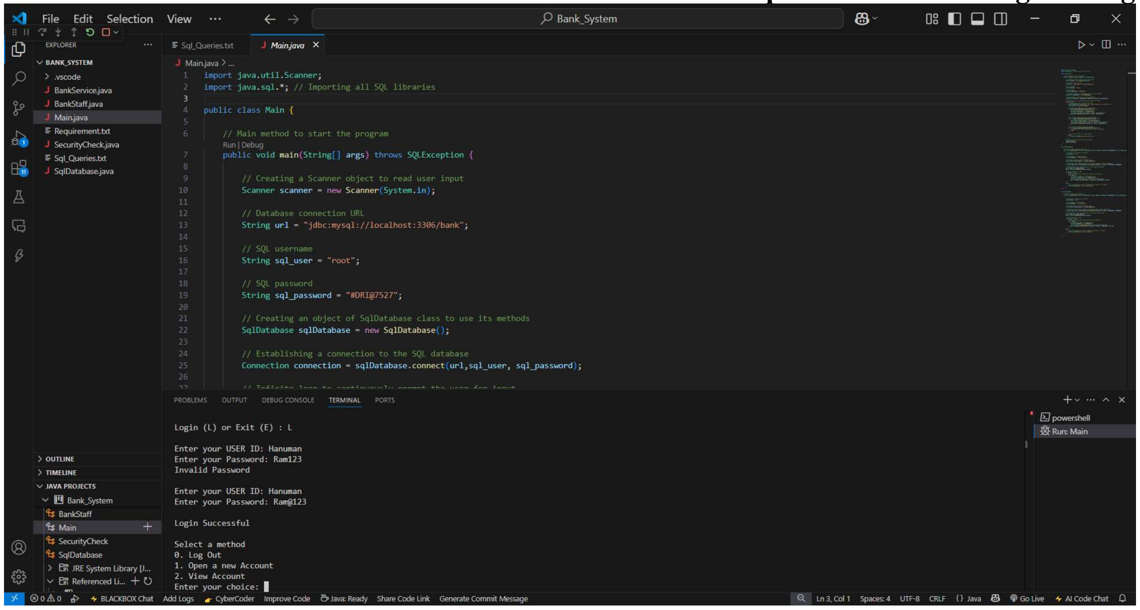
```java
import java.util.Scanner;
import java.sql.*;

public class BankStaff {

    // Displays the menu for bank staff and performs actions based on user input
    public boolean bankStaffMenu(Connection connection, Scanner scanner, SqlDatabase sqlDatabase, String User_ID) throws SQLException {

        boolean pass = true; // To determine if the user stays logged in

        // Display menu options
        System.out.println(x:"\nSelect a method");
        System.out.println(x:"0. Log Out");
        System.out.println(x:"1. Open a new Account");
        System.out.println(x:"2. View Account");
        System.out.print(s:"Enter your choice: ");

        try {
            String choice = scanner.nextLine(); // Get user input

            switch (choice) {
                case "1": // If user selects to open a new account
                    OpenNewAccount openNewAccount = new OpenNewAccount();
                    openNewAccount.openNewAccount(connection, scanner, sqlDatabase);
                    break;

                case "2": // If user selects to view an account
                    ViewAccount viewAccount = new ViewAccount();
                    viewAccount.viewAccount(connection, scanner, sqlDatabase);
                    break;

                case "0": // If user selects to log out
                    System.out.println(x:"Log Out");
                    pass = false;
                    break;

                default: // If user selects an invalid option
                    System.out.println(x:"Invalid choice");
                    break;
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage()); // Handle any errors that occur
        }

        return pass; // Return whether the user is still logged in
```

Implementation-



```java
import java.util.Scanner;
import java.sql.*; // Importing all SQL libraries

public class Main {

    // Main method to start the program
    Run | Debug
    public void main(String[] args) throws SQLException {

        // Creating a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Database connection URL
        String url = "jdbc:mysql://localhost:3306/bank";

        // SQL username
        String sql_user = "root";

        // SQL password
        String sql_password = "#DRI@7527";

        // Creating an object of SqlDatabase class to use its methods
        SqlDatabase sqlDatabase = new SqlDatabase();

        // Establishing a connection to the SQL database
        Connection connection = sqlDatabase.connect(url,sql_user, sql_password);

        // Infinite loop to continuously prompt the user for input
```

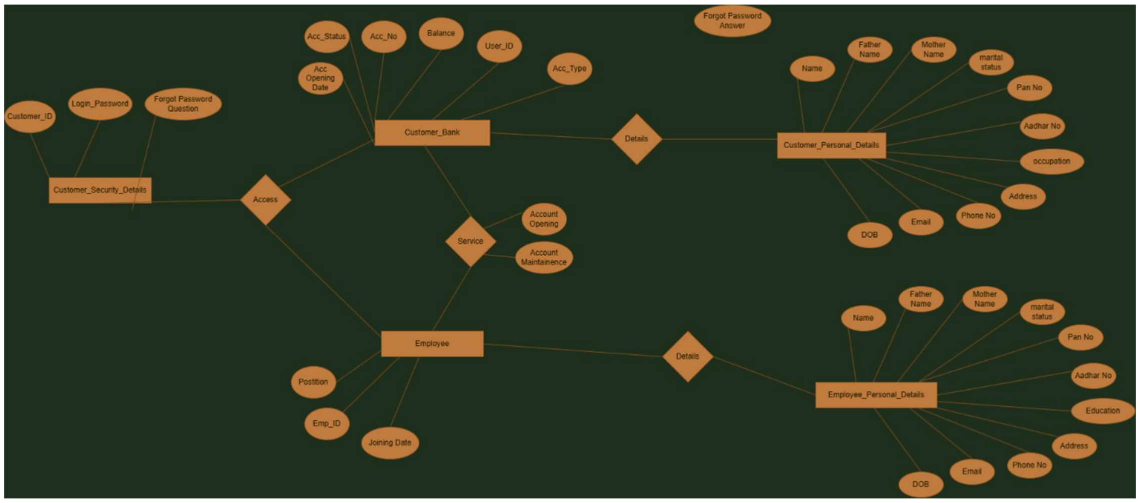
```
PS D:\Coding_Projects\GitHub-Desktop\Internship\Java_Workshop_Project-V01\Bank_System>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '@C:\Users\Drishaya\AppData\Local\Temp\cp_2ptt
shm84srro79ktpw3z6v9g.argfile' 'Main'
Connection established com.mysql.cj.jdbc.ConnectionImpl@7ee8290b

Bank Staff (S) or Bank Customer (C) or Exit (E) : Exception in thread "main"
PS D:\Coding_Projects\GitHub-Desktop\Internship\Java_Workshop_Project-V01\Bank_System>  d:; cd 'd:\Coding_Projects\GitHub-Desktop\Internship\Java_Workshop_Project-V01\Bank_System';
 & 'C:\Program Files\Java\jdk-22\bin\java.exe' '@C:\Users\Drishaya\AppData\Local\Temp\cp_2pttshm84srro79ktpw3z6v9g.argfile' 'Main'
o79ktpw3z6v9g.argfile' 'Main' ;5f43bee3-8726-41ed-8144-5ab863633fb5Connection established com.mysql.cj.jdbc.ConnectionImpl@7ee8290b

Bank Staff (S) or Bank Customer (C) or Exit (E) : s

Login (L) or Exit (E) :
```



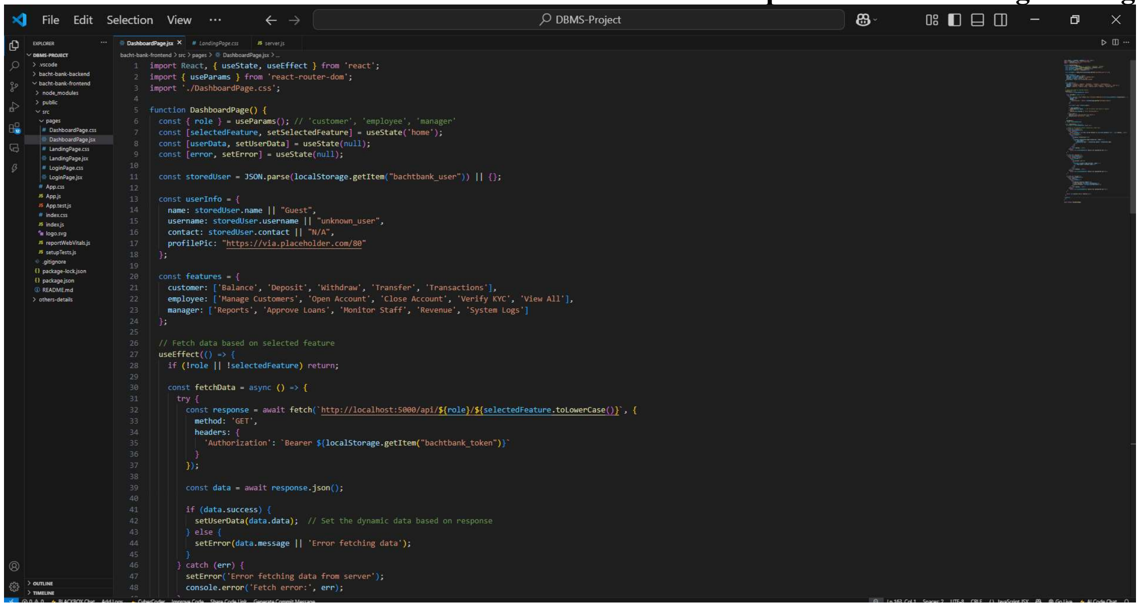
```java
import java.util.Scanner;
import java.sql.*; // Importing all SQL libraries

public class Main {

    // Main method to start the program
    public void main(String[] args) throws SQLException {

        // Creating a Scanner object to read user input
        Scanner scanner = new Scanner(System.in);

        // Database connection URL
        String url = "jdbc:mysql://localhost:3306/bank";

        // SQL username
        String sql_user = "root";

        // SQL password
        String sql_password = "#DRI@7527";

        // Creating an object of SqlDatabase class to use its methods
        SqlDatabase sqlDatabase = new SqlDatabase();

        // Establishing a connection to the SQL database
        Connection connection = sqlDatabase.connect(url,sql_user, sql_password);
```

```
Login (L) or Exit (E) : L

Enter your USER ID: Hanuman
Enter your Password: Ram123
Invalid Password

Enter your USER ID: Hanuman
Enter your Password: Ram@123

Login Successful

Select a method
0. Log Out
1. Open a new Account
2. View Account
Enter your choice:
```

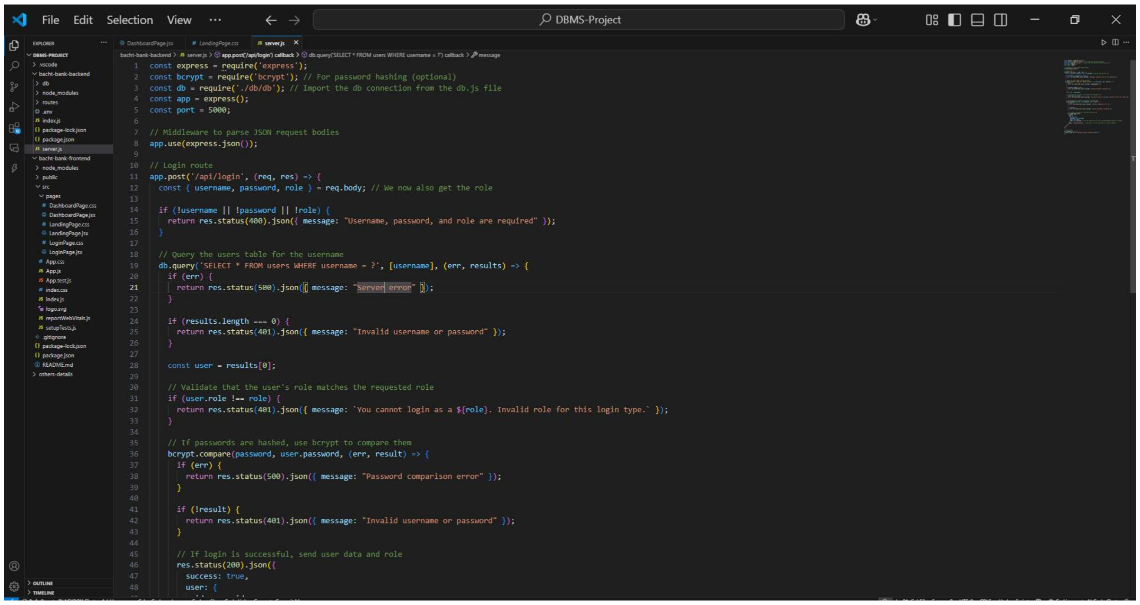
ER diagram / flow of code.



Phase -2 of development

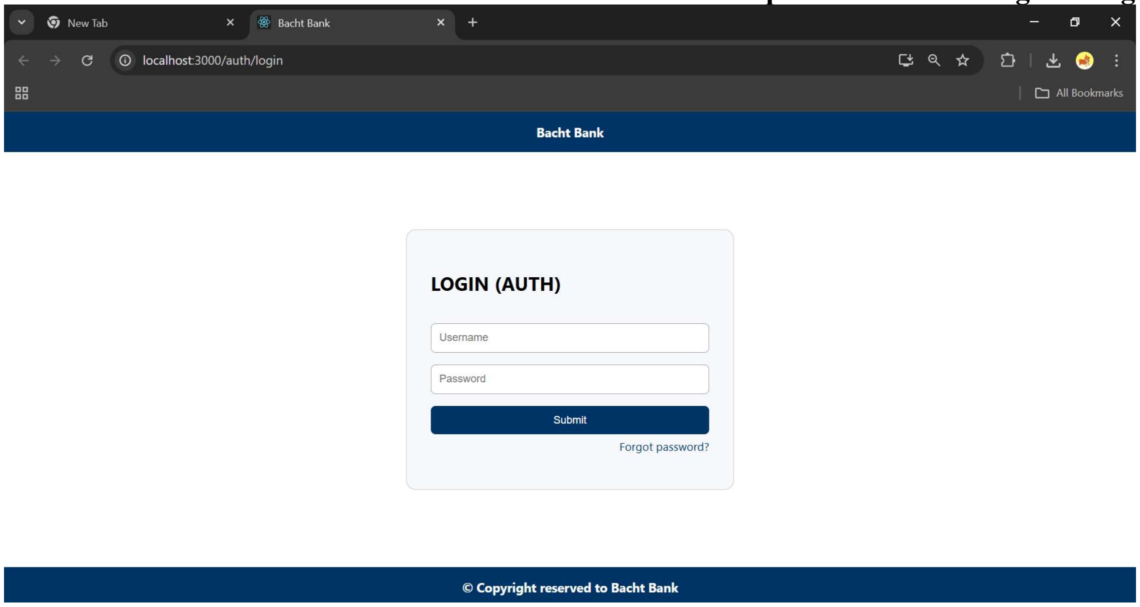


```
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import './DashboardPage.css';

function DashboardPage() {
  const { role } = useParams(); // 'customer', 'employee', 'manager'
  const [selectedFeature, setSelectedFeature] = useState('home');
  const [userData, setUserData] = useState(null);
  const [error, setError] = useState(null);

  const storedUser = JSON.parse(localStorage.getItem("bachtbank_user")) || {};

  const userInfo = {
    name: storedUser.name || "Guest",
    username: storedUser.username || "unknown_user",
    contact: storedUser.contact || "N/A",
    profilePic: "https://via.placeholder.com/80"
  };

  const features = {
    customer: ['Balance', 'Deposit', 'Withdraw', 'Transfer', 'Transactions'],
    employee: ['Manage Customers', 'Open Account', 'Close Account', 'Verify KYC', 'View All'],
    manager: ['Reports', 'Approve Loans', 'Monitor Staff', 'Revenue', 'System Logs']
  };

  // Fetch data based on selected feature
  useEffect(() => {
    if (!role || !selectedFeature) return;

    const fetchData = async () => {
      try {
        const response = await fetch(`http://localhost:5000/api/${role}/${selectedFeature.toLowerCase()}`, {
          method: 'GET',
          headers: {
            'Authorization': `Bearer ${localStorage.getItem("bachtbank_token")}`
          }
        });

        const data = await response.json();

        if (data.success) {
          setUserData(data.data);  // Set the dynamic data based on response
        } else {
          setError(data.message || 'Error fetching data');
        }
      } catch (err) {
        setError('Error fetching data from server');
        console.error('Fetch error:', err);
```



```
const express = require('express');
const bcrypt = require('bcrypt'); // For password hashing (optional)
const db = require('./db/db'); // Import the db connection from the db.js file
const app = express();
const port = 5000;

// Middleware to parse JSON request bodies
app.use(express.json());

// Login route
app.post('/api/login', (req, res) => {
  const { username, password, role } = req.body; // We now also get the role

  if (!username || !password || !role) {
    return res.status(400).json({ message: "Username, password, and role are required" });
  }

  // Query the users table for the username
  db.query('SELECT * FROM users WHERE username = ?', [username], (err, results) => {
    if (err) {
      return res.status(500).json({ message: "Server error" });
    }

    if (results.length === 0) {
      return res.status(401).json({ message: "Invalid username or password" });
    }

    const user = results[0];

    // Validate that the user's role matches the requested role
    if (user.role !== role) {
      return res.status(401).json({ message: `You cannot login as a ${role}. Invalid role for this login type.` });
    }

    // If passwords are hashed, use bcrypt to compare them
    bcrypt.compare(password, user.password, (err, result) => {
      if (err) {
        return res.status(500).json({ message: "Password comparison error" });
      }

      if (!result) {
        return res.status(401).json({ message: "Invalid username or password" });
      }

      // If login is successful, send user data and role
      res.status(200).json({
        success: true,
        user: {
```

Implementation-



Bacht Bank

LOGIN (AUTH)

Username

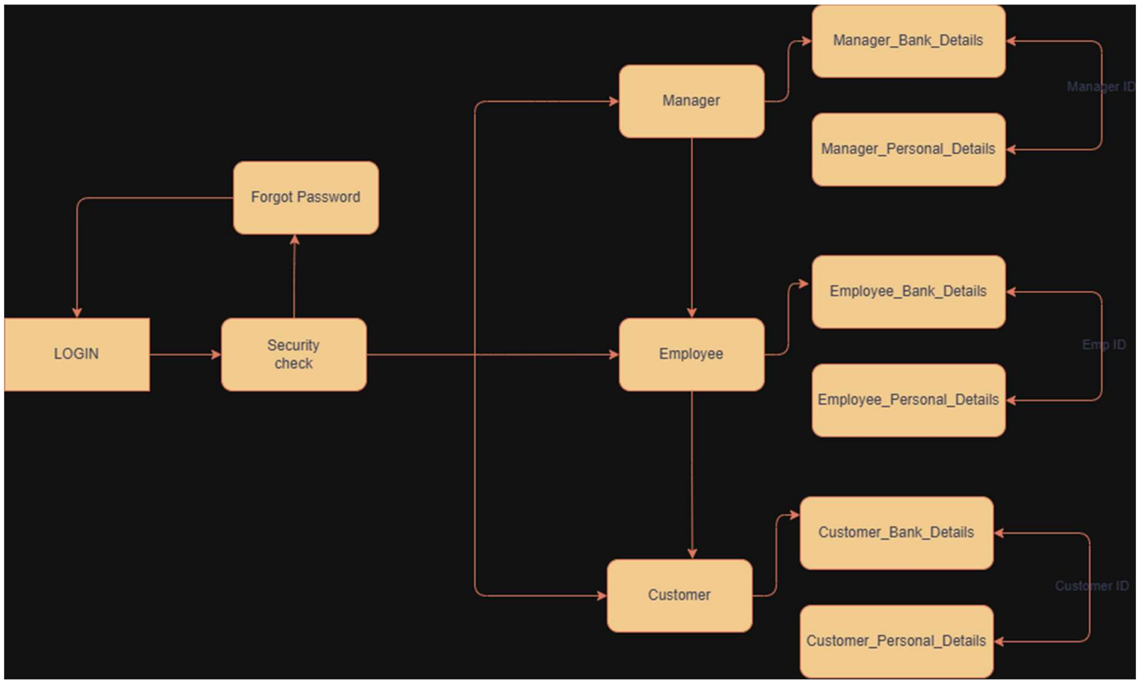
Password

Submit

Forgot password?

© Copyright reserved to Bacht Bank

ER diagram / flow of code



GitHub Repository Link:

https://github.com/Drishay/CollegeProjects/tree/main/DBMS-Project