PROJECT REPORT

ON

BANK MANAGEMENT SYSTEM

ADVANCED INTERNET PROGRAMMING

Subject Code: 24CAP-652

SubmittedBy

Name: Priyanshu

UID: 24MCI10182

SubmittedTo

Mrs.Banita Mandal

INDEX

Acknowledgement

I express my deepest gratitude to my mentor, professors, and peers who have guided me throughout this project. Their invaluable insights, encouragement, and support have played a crucial role in the successful completion of this study.

I am especially thankful to my faculty for providing me with the opportunity to work on such an interesting and interdisciplinary project. Their continuous guidance helped me explore different aspects of handwriting analysis and its connection to personality prediction.

Furthermore, I would like to acknowledge the authors of various research papers and articles that provided me with useful knowledge and inspiration. Their work laid the foundation for my understanding of graphology and machine learning techniques.

My sincere thanks to my friends and classmates for their constant motivation and constructive feedback, which significantly contributed to refining my project. Their discussions and shared knowledge enhanced my approach to problem-solving and technical development.

Lastly, I extend my heartfelt gratitude to my family for their unwavering support, patience, and encouragement throughout my academic journey. Their belief in my capabilities has been a driving force in achieving this milestone.

This project would not have been possible without the collective efforts and support of all the aforementioned individuals. I am deeply grateful for their contributions.

Thank you

Abstract

The **Bank Management System** is a file-based Java and HTML/JavaScript project designed to streamline the basic functionalities of bank operations such as managing employees and customers. Built with **Java (using NetBeans)** for backend operations and **HTML/CSS/JavaScript** for the frontend interface, this system provides a user-friendly and efficient platform to simulate real-world banking processes without using a relational database.

The backend of the system, developed in Java, focuses on employee management, allowing for operations such as **adding, searching, viewing, and deleting employees**, with all data persisted via file storage (.dat files) using Java's serialization mechanism. This approach ensures that the data is retained even after the application is closed and reopened, making the system practical for desktop-based usage where database setup is unnecessary or inconvenient.

On the frontend, a visually appealing **bank dashboard interface** is implemented using HTML, CSS, and JavaScript. The UI begins with a login screen to simulate access control. Once logged in, users can navigate between sections like **Employee Management** and **Customer Management**. The customer section allows for key banking operations including **adding new customers**, **managing account balances**, **depositing and withdrawing funds**, and viewing transaction history. Data is handled using **localStorage** for persistence, ensuring smooth and responsive interaction without requiring a server.

This project bridges the gap between traditional console-based applications and modern web-based interfaces, providing a rich educational experience in **Java OOP**, **file handling**, and **frontend development**. It also lays a strong foundation for future enhancements such as database integration, authentication layers, or RESTful APIs. Overall, the Bank Management System demonstrates a comprehensive understanding of both backend logic and frontend design principles, packaged into a functional, interactive, and scalable application.

Introduction

In today's rapidly advancing digital world, banking systems play a crucial role in managing financial transactions and customer data efficiently. As banking services become more integrated with technology, there is a growing need for intuitive and reliable software systems that can handle various operations such as customer account management, employee records, and transaction tracking. The **Bank Management System** project aims to address these needs by developing a simplified yet functional application that mimics real-world banking activities in an educational or small-scale setting.

This project is a hybrid implementation, combining **Java-based file-handling techniques** for backend logic with a **modern HTML/CSS/JavaScript frontend** for interactive user experience. It is designed to simulate a bank's internal management system that can be used by bank administrators or employees to handle day-to-day banking operations. Unlike traditional systems that rely heavily on databases, this application uses file-based storage (using Java serialization and browser localStorage) to maintain data persistence. This makes it ideal for environments where database support is unavailable, or where a lightweight alternative is preferred.

Purpose and Objective

The primary objective of the Bank Management System is to demonstrate how core banking operations can be developed and managed using object-oriented programming and web technologies. The system supports the following functionalities:

- **Employee Management:** Adding, viewing, searching, and deleting employees. Each employee has a unique ID, name, department, and salary. The data is stored persistently in a .dat file using Java object serialization.
- **Customer Management:** Creating customer profiles with name, account number, and balance. Features include the ability to deposit and withdraw money, view balances, and check transaction history.
- **Login and Security Simulation:** A basic login interface ensures that only authorized users can access the dashboard.
- **User Interface:** A responsive and intuitive UI built using HTML, CSS, and JavaScript to manage different banking sections seamlessly.

This project not only introduces the basic operations within a bank but also emphasizes good software design practices such as modularity, separation of concerns, user-friendly design, and data encapsulation.

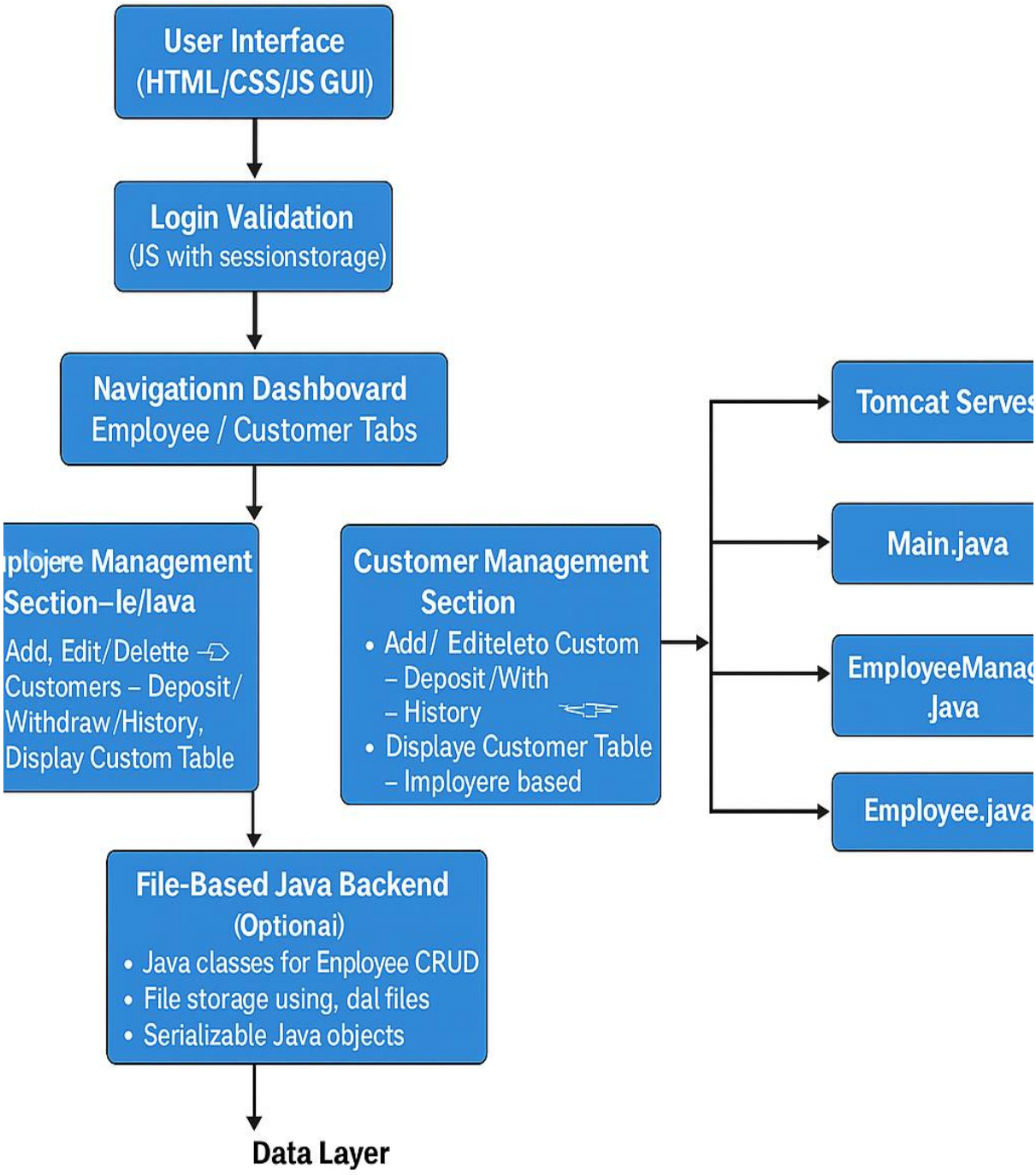
Scope of the Project

The scope of this project covers a standalone banking simulation application that can be expanded or integrated with a real-time database in the future. For the current implementation, it includes:

- A **Java console application** that allows CRUD operations (Create, Read, Update, Delete) for employees.
- A **browser-based dashboard** that provides customer-related functionalities, accessible after a successful login.
- Use of **file storage** (.dat files in Java and localStorage in JavaScript) to persist data between sessions.
- Fully offline operation without the need for backend servers or SQL databases.

This design approach makes the system a great prototype for educational use, demonstrations, or even small institutions wanting a lightweight record management system.

Design flow of project



Code

EmployeeManager.java

```
package EM;

import java.io.*;
import java.util.*;

public class EmployeeManager {
    private List<Employee> employees = new ArrayList<>();
    private final String fileName = "employees.dat";

    public void addEmployee(Employee emp) {
        employees.add(emp);
        System.out.println("Employee added.");
    }

    public void viewEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }
        for (Employee e : employees) {
            System.out.println(e);
        }
    }

    public void searchById(int id) {
        for (Employee e : employees) {
            if (e.getId() == id) {
                System.out.println("Employee Found: " + e);
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    public void deleteById(int id) {
        Iterator<Employee> iterator = employees.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().getId() == id) {
                iterator.remove();
                System.out.println("Employee deleted.");
                return;
```

```
            }
        }
        System.out.println("Employee not found.");
    }

    public void saveToFile() {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            out.writeObject(employees);
            System.out.println("Data saved to file.");
        } catch (IOException e) {
            System.out.println("Error saving to file.");
        }
    }

    public void loadFromFile() {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(fileName))) {
            employees = (List<Employee>) in.readObject();
            System.out.println("Data loaded from file.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("No data file found. Starting fresh.");
        }
    }
}
```

Main.java

```
package EM;


import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager();
        manager.loadFromFile();

        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\n--- Employee Management System ---");
            System.out.println("1. Add Employee");
            System.out.println("2. View Employees");
            System.out.println("3. Search Employee by ID");
            System.out.println("4. Delete Employee by ID");
```

```
            System.out.println("5. Save to File");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine(); // consume newline
                    System.out.print("Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Department: ");
                    String dept = scanner.nextLine();
                    System.out.print("Salary: ");
                    double salary = scanner.nextDouble();
                    manager.addEmployee(new Employee(id, name, dept, salary));
                    break;
                case 2:
                    manager.viewEmployees();
                    break;
                case 3:
                    System.out.print("Enter ID: ");
                    int searchId = scanner.nextInt();
                    manager.searchById(searchId);
                    break;
                case 4:
                    System.out.print("Enter ID to delete: ");
                    int delId = scanner.nextInt();
                    manager.deleteById(delId);
                    break;
                case 5:
                    manager.saveToFile();
                    break;
                case 0:
                    manager.saveToFile();
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        } while (choice != 0);
    }
}
```

Employee.java

```
package EM;

import java.io.Serializable;

public class Employee implements Serializable {
   private int id;
   private String name;
   private String department;
   private double salary;

   public Employee(int id, String name, String department, double salary) {
      this.id = id;
      this.name = name;
      this.department = department;
      this.salary = salary;
   }

   public int getId() { return id; }
   public String getName() { return name; }
   public String getDepartment() { return department; }
   public double getSalary() { return salary; }

   public String toString() {
      return "ID: " + id + ", Name: " + name +
            ", Department: " + department + ", Salary: " + salary;
   }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Bank Management Dashboard</title>
  <style>
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      margin: 0;
      padding: 0;
      background-color: #0d1117;
      color: #f0f6fc;
      display: flex;
      justify-content: center;
```

```
  align-items: center;
  height: 100vh;
  overflow: hidden;
}

/* Login Page Styles */
.login-container {
  background-color: #161b22;
  padding: 30px;
  border-radius: 8px;
  width: 300px;
  text-align: center;
  border: 1px solid #30363d;
}

h2 {
  margin-bottom: 20px;
  color: #58a6ff;
}

input {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  border-radius: 5px;
  border: none;
  font-size: 1rem;
  background-color: #21262d;
  color: #f0f6fc;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #58a6ff;
  border-radius: 5px;
  border: none;
  font-size: 1rem;
  color: white;
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
  background-color: #3b8fd3;
```

```
}

.error {
  color: #f28d8d;
  font-size: 0.9rem;
  margin-top: 10px;
}

/* Dashboard Styles */
#dashboardPage {
  display: none;
  flex-direction: column;
  width: 100%;
  height: 100%;
}

header {
  background-color: #161b22;
  padding: 20px;
  text-align: center;
  font-size: 2rem;
  color: #58a6ff;
  border-bottom: 1px solid #30363d;
}

.dashboard {
  display: flex;
  height: calc(100vh - 60px);
}

.sidebar {
  width: 250px;
  background-color: #161b22;
  padding: 20px;
  height: 100%;
  border-right: 1px solid #30363d;
}

.sidebar h3 {
  color: #58a6ff;
  margin-bottom: 20px;
}

.sidebar button {
  width: 100%;
```

```
  background-color: #21262d;
  color: white;
  padding: 10px;
  margin: 10px 0;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: 0.3s;
}

.sidebar button:hover {
  background-color: #58a6ff;
}

.main-content {
  flex: 1;
  padding: 30px;
  overflow-y: auto;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
  color: #f0f6fc;
}

th, td {
  padding: 12px;
  border: 1px solid #30363d;
  text-align: center;
}

th {
  background-color: #21262d;
}

tr:nth-child(even) {
  background-color: #161b22;
}

input, button {
  padding: 10px;
  margin: 8px;
  width: 250px;
```

```
    border-radius: 5px;
    border: none;
    font-size: 1rem;
  }

  .add-section {
    margin-top: 20px;
  }

  .action-buttons button {
    margin: 0 5px;
  }
 </style>
</head>
<body>

 <!-- Login Page -->
 <div class="login-container" id="loginPage">
  <h2>Bank Login</h2>
  <input type="text" id="username" placeholder="Username" />
  <input type="password" id="password" placeholder="Password" />
  <button onclick="login()">Login</button>
    <div id="errorMessage" class="error" style="display: none;">Invalid username or password</div>
 </div>

 <!-- Bank Dashboard -->
 <div id="dashboardPage">
  <header>🏦 Bank Management Dashboard</header>

  <div class="dashboard">
   <div class="sidebar">
    <h3>Navigation</h3>
    <button onclick="showEmployees()">Employees</button>
    <button onclick="showCustomers()">Customers</button>
   </div>

   <div class="main-content">
    <!-- Employee Section -->
    <div id="employeeSection" style="display: none;">
     <h2>Employee Management</h2>
     <div class="add-section">
      <input type="text" id="empName" placeholder="Name">
      <input type="email" id="empEmail" placeholder="Email">
      <button onclick="addEmployee()">➕ Add Employee</button>
```

```
      </div>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody id="employeeTable"></tbody>
      </table>
    </div>

    <!-- Customer Section -->
    <div id="customerSection" style="display: none;">
      <h2>Customer Management</h2>
      <div class="add-section">
        <input type="text" id="custName" placeholder="Name">
        <input type="text" id="custAcc" placeholder="Account Number">
        <input type="number" id="custMoney" placeholder="Initial Balance">
        <button onclick="addCustomer()">➕ Add Customer</button>
      </div>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Account No.</th>
            <th>Balance</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody id="customerTable"></tbody>
      </table>
    </div>
  </div>
 </div>
</div>

<script>
 // Login validation
 function login() {
   const username = document.getElementById("username").value.trim();
   const password = document.getElementById("password").value.trim();

   if (username === "admin" && password === "password123") {
```

```
    sessionStorage.setItem("loggedIn", true);
    document.getElementById("loginPage").style.display = "none";
    document.getElementById("dashboardPage").style.display = "flex";
  } else {
    document.getElementById("errorMessage").style.display = "block";
  }
}
if (sessionStorage.getItem("loggedIn")) {
  document.getElementById("loginPage").style.display = "none";
  document.getElementById("dashboardPage").style.display = "flex";
}
function showEmployees() {
  document.getElementById("employeeSection").style.display = 'block';
  document.getElementById("customerSection").style.display = 'none';
}

function showCustomers() {
  document.getElementById("employeeSection").style.display = 'none';
  document.getElementById("customerSection").style.display = 'block';
}
function addEmployee() {
  const name = document.getElementById("empName").value.trim();
  const email = document.getElementById("empEmail").value.trim();
  if (name && email) {
    const data = JSON.parse(localStorage.getItem("employees")) || [];
    data.push({ name, email });
    localStorage.setItem("employees", JSON.stringify(data));
    loadEmployees();
  } else {
    alert("Please fill both fields.");
  }
}

function loadEmployees() {
  const data = JSON.parse(localStorage.getItem("employees")) || [];
  const tbody = document.getElementById("employeeTable");
  tbody.innerHTML = '';
  data.forEach((emp, idx) => {
    const row = `
      <tr>
        <td>${emp.name}</td>
        <td>${emp.email}</td>
        <td class="action-buttons">
          <button onclick="editEmployee(${idx})">✏️</button>
          <button onclick="deleteEmployee(${idx})">❌</button>
```

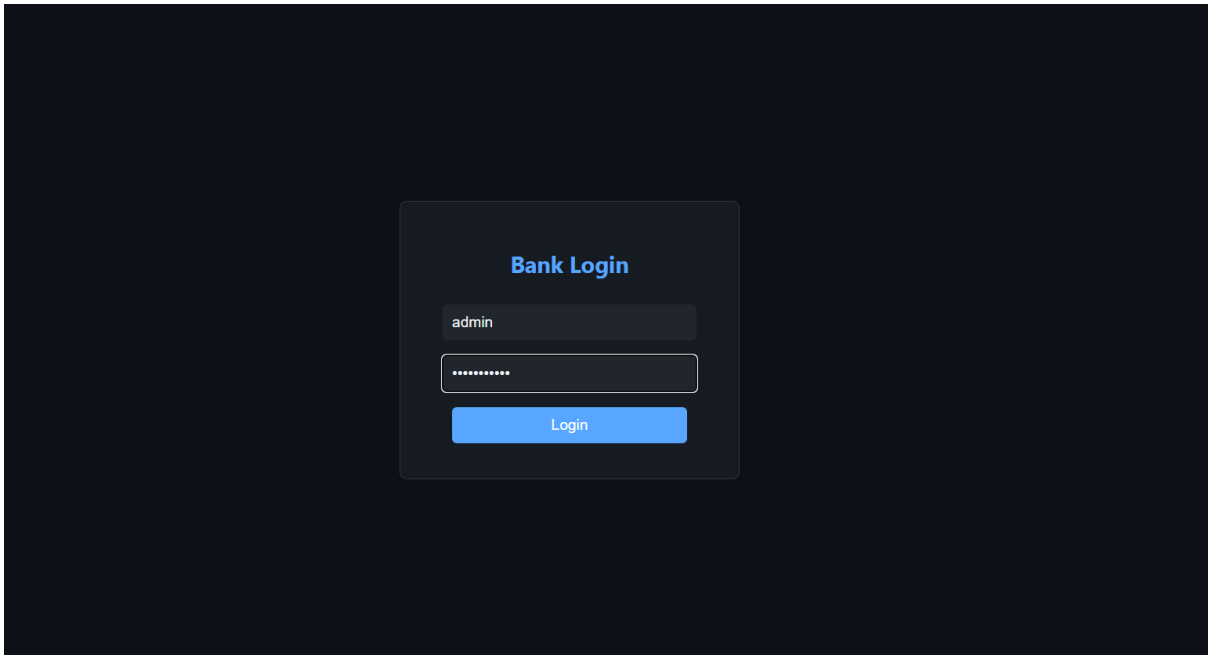
```
        </td>
      </tr>
    `;
    tbody.innerHTML += row;
  });
}
function addCustomer() {
  const name = document.getElementById("custName").value.trim();
  const account = document.getElementById("custAcc").value.trim();
  const balance = parseFloat(document.getElementById("custMoney").value.trim());
  if (name && account && !isNaN(balance)) {
    const data = JSON.parse(localStorage.getItem("customers")) || [];
    data.push({ name, account, balance, history: [] });
    localStorage.setItem("customers", JSON.stringify(data));
    loadCustomers();
  } else {
    alert("Please fill all customer fields.");
  }
}

function loadCustomers() {
  const data = JSON.parse(localStorage.getItem("customers")) || [];
  const tbody = document.getElementById("customerTable");
  tbody.innerHTML = '';
  data.forEach((cust, idx) => {
    const row = `
      <tr>
        <td>${cust.name}</td>
        <td>${cust.account}</td>
        <td>$${cust.balance}</td>
        <td class="action-buttons">
          <button onclick="depositMoney(${idx})">💰</button>
          <button onclick="withdrawMoney(${idx})">💸</button>
          <button onclick="viewTransactions(${idx})">📜</button>
          <button onclick="deleteCustomer(${idx})">❌</button>
        </td>
      </tr>
    `;
    tbody.innerHTML += row;
  });
}
function logout() {
  sessionStorage.removeItem("loggedIn");
  document.getElementById("loginPage").style.display = "block";
```

```
    document.getElementById("dashboardPage").style.display = "none";
  }
  window.onload = loadCustomers;
 </script>
</body>
</html>
```

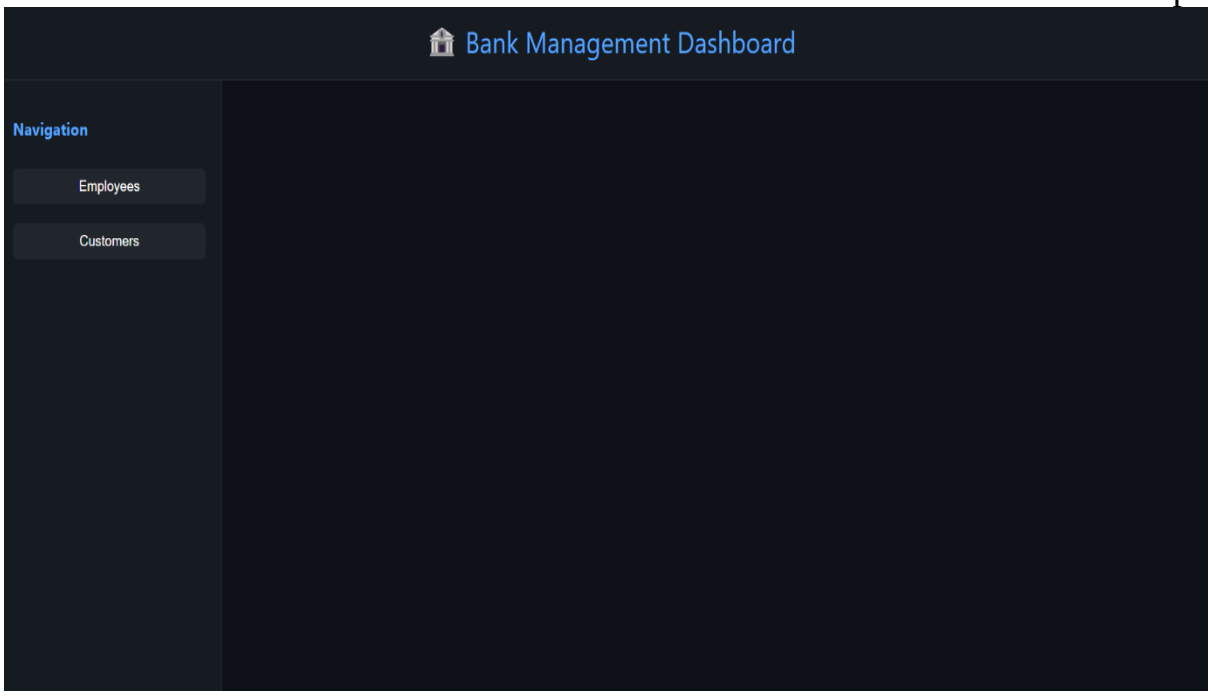
Output

Login page:-

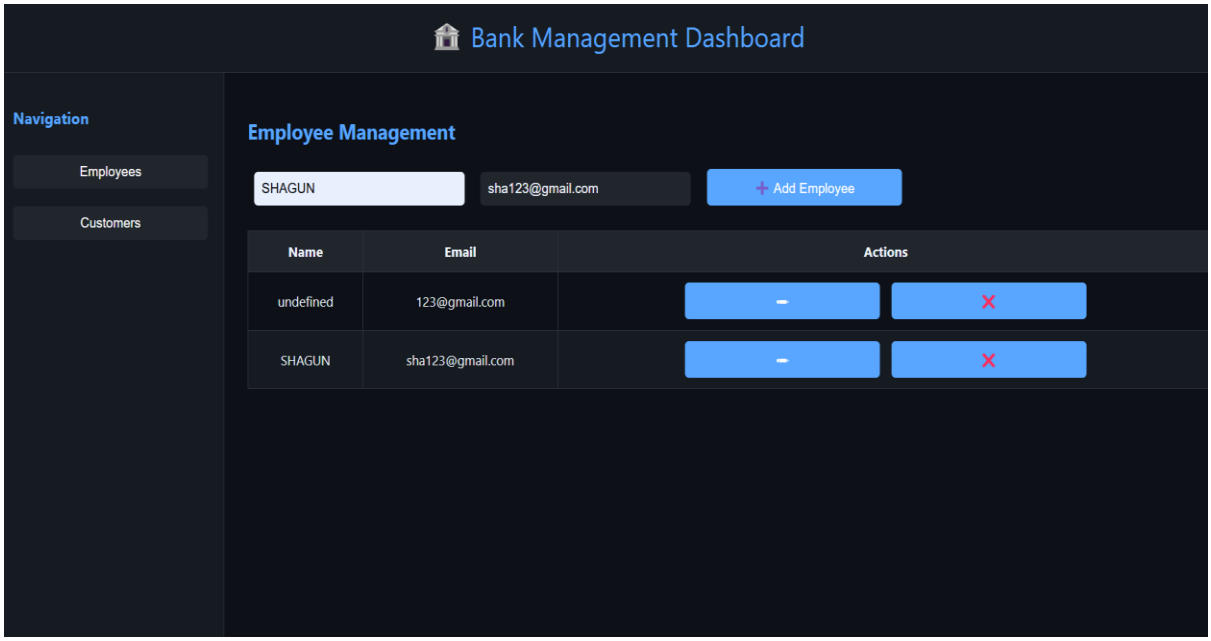


Dashboard:-

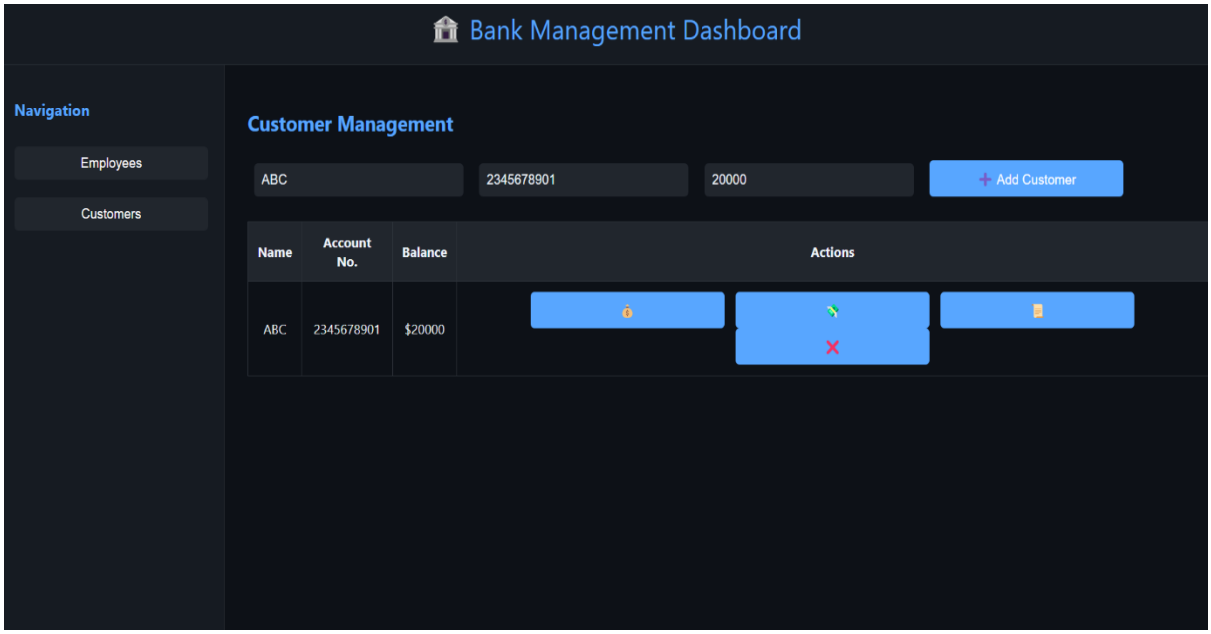
pa



Employee Management:-



Customer management:-



Result analysis

The primary goal of this project was to develop a Bank Management System that combines a backend employee management system in Java with a frontend customer and employee management interface in HTML/JavaScript. This system allows administrators to perform CRUD (Create, Read, Update, Delete) operations on employee and customer data. Additionally, the system supports persistence through file-based storage for employee data and local storage for customer and employee details on the frontend.

Implementation Results

1. Employee Management (Backend in Java)

- **Adding Employees:**
  - Employees can be successfully added using the `addEmployee()` method, which adds employee objects to the list in memory. Data is then serialized to a file (`employees.dat`) for persistence.
  - The user is provided feedback confirming the addition of an employee.
- **Viewing Employees:**
  - The `viewEmployees()` method iterates through the list of employees and prints their details to the console. This allows for an easy overview of all current employees in the system.
- **Searching Employees:**
  - The `searchById()` method allows the user to search for an employee by their unique ID. If found, employee details are displayed, otherwise, a message indicates that the employee was not found.

2. Frontend (HTML/JavaScript - Dashboard)

- **Login Page:**
  - The login page allows for basic authentication with hardcoded credentials (`admin` / `password123`). If the login is successful, the user is redirected to the dashboard.
  - The error message is shown if the user enters incorrect credentials.
- **Employee Management Section:**
  - The employee management section allows the addition of new employees through input fields for name and email. Employees are then stored in the browser's `localStorage` and displayed in a table format with buttons for editing and deleting.
  - Data is dynamically loaded and displayed each time the user accesses the employee section.

Analysis

1. Functionality and Usability

- **Employee Management System:**
  - The backend Java implementation effectively handles the CRUD operations for employees. The addition, viewing, searching, and deletion of employee records work as intended, and the file persistence allows data to be retained across program restarts.
- **Frontend Management System:**
  - The HTML/JavaScript dashboard provides a clean and responsive interface for managing employees and customers. Navigation between sections (employee and customer) is simple and intuitive, and the data is dynamically displayed in tables.

2. Limitations

- **Scalability:**
  - The current system is limited to a single user due to its use of local storage and file-based storage for persistence. For a larger organization or multiple users, a centralized database system would be required.
- **Security:**
  - The login system is based on simple hardcoded credentials, which is insecure for a real application. Implementing a more secure authentication mechanism (e.g., OAuth, JWT tokens) would be necessary for handling user data securely.

3. Improvements and Future Enhancements

- **Database Integration:**
  - To handle larger datasets and provide multi-user support, integrating a relational database like MySQL or SQLite would be beneficial.
- **Enhanced Security:**
  - Implementing proper authentication and authorization mechanisms would secure sensitive information (e.g., employee salaries, customer balances). Password encryption and token-based authentication (JWT) are essential next steps for security.

Conclusion

This Bank Management System project successfully integrates a backend employee management system in Java with a frontend customer and employee management interface in HTML/JavaScript. The system allows administrators to perform essential operations such as adding, viewing, searching, and deleting employee records. Additionally, it manages customer data, allowing for basic actions like adding new customers and displaying their details in an organized dashboard.

Key features of the project include:

- **Employee Management**: CRUD operations for employee data are handled on the backend using Java, with file-based persistence ensuring data retention across sessions.
- **Customer Management**: The frontend allows for customer management through a user-friendly dashboard, with persistent data storage via the browser's local storage.
- **Login and Dashboard Interface**: Basic user authentication is provided, and the system has a clean, intuitive interface for managing employees and customers.

However, the project also has limitations in scalability and security. The use of local storage and file-based storage is suitable for small-scale applications but would not suffice for larger, multi-user environments. Additionally, security features are minimal, relying on simple hardcoded credentials, which would need to be upgraded for real-world applications.

Future work

As the current version of the Bank Management System provides the core functionality, several improvements and expansions can be made to enhance its usability, scalability, and security. Here are some key areas for future work:

- **Database Integration**: Transition from file-based storage to a relational database (e.g., MySQL or SQLite) for better scalability, performance, and data integrity.

- **Security Enhancements**: Implement secure user authentication (e.g., OAuth, JWT), password hashing, and role-based access control (RBAC) to improve security.

- **Transaction History & Loan Management**: Extend the customer features to include transaction history tracking, loan applications, and interest calculations.

- **Responsive UI Design**: Improve the frontend with responsive design frameworks like Bootstrap to ensure the system works well across different devices (desktop, mobile, tablet).

- **Real-time Updates**: Implement real-time updates for customer balances and transactions using WebSockets or AJAX, so the dashboard reflects live data.

- **Multi-user Support**: Add features to support concurrent users, including session management, user roles, and permissions for various access levels (admin, manager, customer).

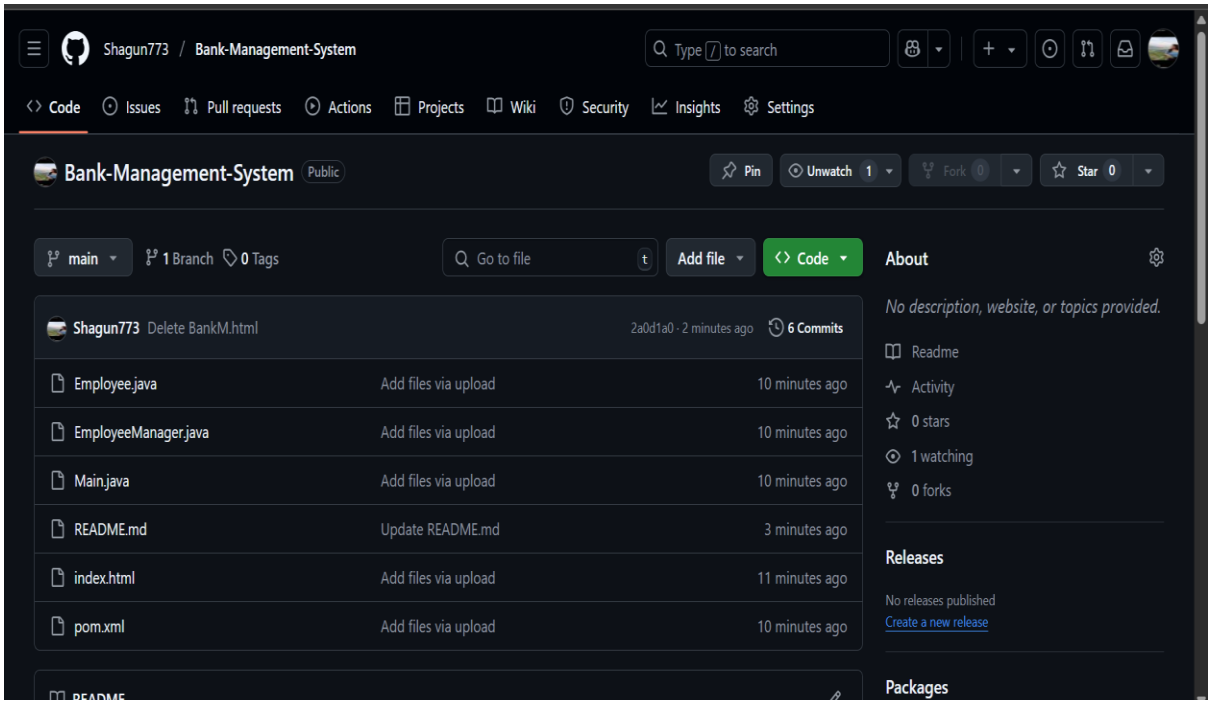
- **Cloud Deployment**: Host the system on a cloud platform (AWS, Google Cloud, etc.) for better scalability, reliability, and performance, ensuring the system can handle increased traffic.

References

Here are some suggested references for the Bank Management System project:

1. **Java Documentation**:
   - Oracle. (n.d.). *Java SE Documentation*. Retrieved from https://docs.oracle.com/en/java/javase/
   - Java SE 8 Tutorial. Oracle. Available: https://docs.oracle.com/javase/tutorial/
2. **File Handling in Java**:
   - Java World. (2009). *File Handling in Java*. Retrieved from https://www.javaworld.com/article/2073219/core-java/file-handling-in-java.html
3. **Database Integration in Java**:
   - Oracle. (n.d.). *JDBC (Java Database Connectivity) API*. Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/
4. **Security in Java**:
   - Oracle. (n.d.). *Java Cryptography Architecture (JCA)*. Retrieved from https://docs.oracle.com/javase/8/docs/technotesss/guides/security/
5. **Responsive Web Design**:
   - W3Schools. (n.d.). *Responsive Web Design*. Retrieved from https://www.w3schools.com/css/css_rwd_intro.asp
   - Bootstrap Documentation. (n.d.). *Bootstrap*. Retrieved from https://getbootstrap.com/
6. **WebSockets for Real-time Communication**:
   - MDN Web Docs. (2021). *WebSockets*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Github


Shagun773 / Bank-Management-System
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Bank-Management-System
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
Shagun773 Delete BankM.html
2a0d1a0 · 2 minutes ago
6 Commits
Employee.java Add files via upload 10 minutes ago
EmployeeManager.java Add files via upload 10 minutes ago
Main.java Add files via upload 10 minutes ago
README.md Update README.md 3 minutes ago
index.html Add files via upload 11 minutes ago
pom.xml Add files via upload 10 minutes ago
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages