##### **HooBank : Banking Application**

Submitted to the

Department of Master of Computer Applications in partial fulfilment of the requirements for the Mini Project in

**Full Stack Development – MCA26**

**by**

**Mohammed Afreed Pasha**

**1MS24MC059**

**Under the Guidance of**

**Ms Sahana**

**Department of Master of Computer Applications**

**RAMAIAH INSTITUTE OF TECHNOLOGY**

(Autonomous Institute, Affiliated to VTU)

Accredited by National Board of Accreditation & NAAC with ‘A+’ Grade

MSR Nagar, MSRIT Post, Bangalore-560054

[www.msrit.edu](http://www.msrit.edu/)

**2025**

**DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS**

**CERTIFICATE**

This is to certify that the project entitled “HooBank : Banking Application” is carried out by Mohammed Afreed Pasha – 1MS24MC059 student of 2nd semester, in partial fulfillment for the Mini Project in Full Stack development – MCA26, during the academic year 2024-2025.

Ms Sahana Dr. S Ajitha

Guide Head of the Department

Name of Examiners Signature with Date

**ACKNOWLEDGEMENT**

With deep sense of gratitude, I am thankful to our Respected HoD, **Dr. S Ajitha** for her pillared support and encouragement. I would like to convey my heartfelt thanks to my Project Guide **Ms Sahana**, Department of Master of Computer Applications, for giving me an opportunity to embark upon this topic and for her/his constant encouragement.

It gives me great pleasure to acknowledge the contribution of all people who helped me directly or indirectly realize it. First I would like to acknowledge the love and tremendous sacrifices that my Parents made to ensure that I always get an excellent education. For this and much more, I am forever in their debt.

I am thankful to all my friends for their moral and material support throughout the course of my project. Finally, I would like to thank all the Teaching and Non-Teaching Staff of Department of Master of Computer Applications for their assistance during various stages of my project work.

**Mohammed Afreed Pasha**

**ABSTRACT**

HooBank is a modern, full-stack digital banking platform built on the MERN stack (MongoDB, Express.js, React.js, Node.js), designed to provide users with a seamless and intuitive banking experience. The application offers a responsive and visually appealing interface, ensuring ease of use and accessibility across various devices. It features a secure authentication process where users sign up with a predefined account number and PIN before setting up their email and password for logins. The back-end, built with Node.js and Express.js, provides a robust RESTful API for handling user authentication, data retrieval, and transaction management. User passwords and PINs are securely hashed using bcrypt, and a lockout mechanism is implemented to prevent brute-force attacks. The dashboard presents real-time banking data, allowing users to track their financial activities effectively. A key feature of HooBank is the expense chart, which provides a visual representation of spending patterns, enhancing financial tracking and insights. This project showcases expertise in full-stack development, UI/UX design, data visualization, and secure authentication workflows, laying the groundwork for a scalable and interactive banking application.

**Table of Contents**

[**1.** **Introduction** 1](#_Toc204302197)

[**1.1.** **Problem Definition** 1](#_Toc204302198)

[**2.** **Implementation** 2](#_Toc204302199)

[**2.1.** **System Specification and Architecture** 2](#_Toc204302200)

[**2.1.1.** **System Specification** 2](#_Toc204302201)

[**2.1.2.** **System Architecture** 2](#_Toc204302202)

[**2.2.** **Code snippet** 5](#_Toc204302203)

[**3.** **User Interface Screenshots** 8](#_Toc204302204)

[**4.** **Conclusion and Future Enhancement** 10](#_Toc204302205)

# **Introduction**

## **Problem Definition**

In the modern banking landscape, users often face challenges in accessing a simple, user-friendly, and efficient digital banking platform. Many existing banking applications require complex authentication processes, lack intuitive interfaces, or do not provide seamless access to essential banking features such as transaction tracking and financial insights. Additionally, users may struggle with managing and visualizing their expenses, making financial planning more difficult.

Another significant challenge in digital banking is ensuring a smooth and responsive user experience. Many platforms suffer from slow performance, inefficient UI design, and limited accessibility across different devices. Furthermore, integrating transaction data dynamically while maintaining an optimized and interactive frontend requires careful development and state management.

To address these challenges, HooBank is developed as a modern and responsive banking application that offers a streamlined and interactive user experience. The platform provides an easy-to-navigate interface while ensuring fast and responsive performance across devices. The application features a structured authentication system, allowing users to sign up using a predefined account number and PIN before setting up their login credentials.

Additionally, HooBank enhances financial management by incorporating an expense chart, implemented using Recharts, to provide users with a clear visual representation of their spending patterns. This feature allows users to track their transactions effectively and gain valuable financial insights.

By offering a secure, interactive, and visually appealing banking experience, HooBank addresses key pain points in digital banking. The project showcases expertise in full-stack development, UI/UX design, and data visualization, laying the foundation for a scalable and efficient banking solution.

# **Implementation**

## **System Specification and Architecture**

### **System Specification**

**Hardware Specifications**

* **Processor:** Intel Core i3 or equivalent (Recommended: Core i5 or higher)
* **RAM:** 4 GB (Recommended: 8 GB or higher)
* **Hard Disk:** 20 GB of free space

**Software Specifications**

* **Operating System:** Windows, macOS, or Linux
* **Node.js:** Version 14.x or higher
* **npm:** Version 6.x or higher
* **MongoDB:** Version 4.x or higher (local instance or a cloud-based service like MongoDB Atlas)
* **Web Browser:** Google Chrome, Mozilla Firefox, or any modern web browser
* **Code Editor:** Visual Studio Code (recommended) or any other code editor

### **System Architecture**

The project follows a client-server architecture. The front-end, built with React, serves as the client, while the back-end, built with Node.js and Express.js, acts as the server. Communication between the two is handled via a RESTful API.

**Back-End Development**

**Technologies Used:**

* **Node.js:** A JavaScript runtime environment that executes server-side code.
* **Express.js:** A web application framework for Node.js, used to build the RESTful API.
* **MongoDB:** A NoSQL database used to store user, account, and transaction data.
* **Mongoose:** An object data modeling (ODM) library for MongoDB and Node.js.
* **JSON Web Tokens (JWT):** Used for secure user authentication and session management.
* **Bcrypt.js:** A library for hashing user passwords before storing them in the database.

**Key Features Implemented:**

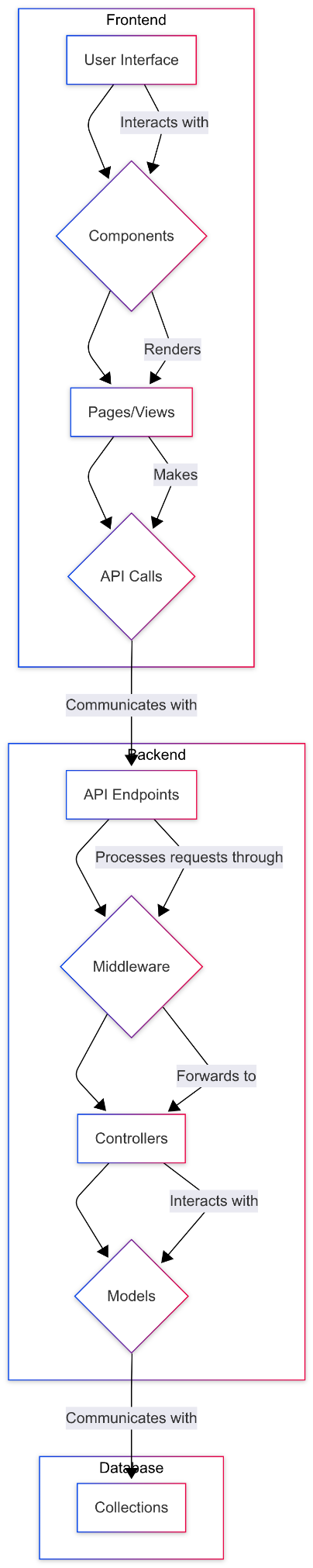
* **RESTful API:** A comprehensive API was developed to handle all client-server communication. The API includes endpoints for:
  + **User Authentication:**
    - **POST /api/auth/signup**: For user registration.
    - **POST /api/auth/login**: For user login.
    - **GET /api/auth/verify**: To verify the user's token and retrieve their data.
* **Database Models:** Mongoose schemas were created for the following data models:
  + **User:** Stores user information such as email and hashed password.
  + **Account:** Stores account details like account number, balance, and account holder name.
  + **Transaction:** Stores transaction data, including type, amount, description, and the balance after the transaction.
* **Authentication and Security:**
  + **JWT Authentication:** The back-end uses JWT to create and verify user tokens, ensuring secure access to protected routes.
  + **Password Hashing:** User passwords and debit card PINs are hashed using bcrypt before being stored in the database to prevent unauthorized access.
  + **Account Lockout:** To enhance security, an account lockout mechanism has been implemented. After five failed login attempts, a user's account is temporarily locked for 15 minutes.

**Front-End and Back-End Integration**

The React front-end was built to seamlessly communicate with the back-end API. This integration involved:

* **API Calls:** The front-end is designed to make API calls to the back-end for user registration, login, and data retrieval, creating a dynamic and interactive user experience.
* **Dynamic Data:** The dashboard fetches and displays real-time account and transaction data from the database, providing users with up-to-date information.
* **State Management:** The front-end's state management is structured to handle the asynchronous nature of API calls and efficiently store user and transaction data

**Architecture Diagram**

****

## **Code snippet**

the complete source code for this project is available on GitHub: [*https://github.com/afreed007/hoobank\_mern*](https://www.google.com/search?q=https://github.com/afreed007/hoobank_mern)

##### **Backend: server.js**

import app from './src/app.js';

import connectDB from './src/config/database.js';

import dotenv from 'dotenv';

dotenv.config();

connectDB();

const PORT = process.env.PORT || 5000;

const server = app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

process.on('unhandledRejection', (err) => {

console.error('Unhandled Promise Rejection:', err.message);

server.close(() => {

process.exit(1);

});

});

process.on('uncaughtException', (err) => {

console.error('Uncaught Exception:', err.message);

process.exit(1);

});

##### **Backend: config/database.js (Database Connection)**

import mongoose from 'mongoose';

import dotenv from 'dotenv';

dotenv.config();

const connectDB = async () => {

try {

mongoose.set('strictQuery', false);

const conn = await mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log(`MongoDB Connected: ${conn.connection.host}`);

mongoose.connection.on('error', (err) => {

console.error('MongoDB connection error:', err);

});

mongoose.connection.on('disconnected', () => {

console.log('MongoDB disconnected');

});

process.on('SIGINT', async () => {

await mongoose.connection.close();

console.log('MongoDB connection closed due to app termination.');

process.exit(0);

});

} catch (error) {

console.error('Database connection failed:', error.message);

process.exit(1);

}

};

export default connectDB;

##### **Frontend: App.jsx**

import React, { useState } from "react";

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";

import styles from "./style";

import { Navbar, Hero, Business, Stats, Billing, CardDeal, Testimonials, Clients, CTA, Footer, LoginModal, Dashboard } from "./components";

const App = () => {

const [isLoginOpen, setIsLoginOpen] = useState(false);

return (

<Router>

<Routes>

{/\* Home Route \*/}

<Route

path="/"

element={

<div className="bg-primary w-full overflow-hidden">

<div className={`${styles.paddingX} ${styles.flexCenter}`}>

<div className={`${styles.boxWidth}`}>

<Navbar />

</div>

</div>

<div className={`bg-primary ${styles.flexStart}`}>

<div className={`${styles.boxWidth}`}>

<Hero setIsLoginOpen={setIsLoginOpen} />

</div>

</div>

<div className={`bg-primary ${styles.paddingX} ${styles.flexCenter} ml-10`}>

<div className={`${styles.boxWidth}`}>

<Stats />

<Business />

<Billing />

<CardDeal />

<Testimonials />

<Clients />

<CTA isLoginOpen={isLoginOpen} setIsLoginOpen={setIsLoginOpen} />

<Footer />

</div>

</div>

{/\* Login Modal \*/}

<LoginModal isOpen={isLoginOpen} onClose={() => setIsLoginOpen(false)} />

</div>

}

/>

{/\* Dashboard Route \*/}

<Route path="/dashboard" element={<Dashboard />} />

</Routes>

</Router>

);

};

export default App;

##### **Frontend: Dashboard.jsx**

import React, { useState, useEffect } from "react";

import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from "recharts";

import { useNavigate, useLocation } from "react-router-dom";

const Dashboard = () => {

const [user, setUser] = useState(null);

const [transactions, setTransactions] = useState([]);

const [balance, setBalance] = useState(0);

const [loading, setLoading] = useState(true);

const [error, setError] = useState("");

const navigate = useNavigate();

const location = useLocation();

const API\_BASE = "http://localhost:5000/api/auth";

useEffect(() => {

const initializeDashboard = async () => {

// ... (Dashboard initialization logic)

};

initializeDashboard();

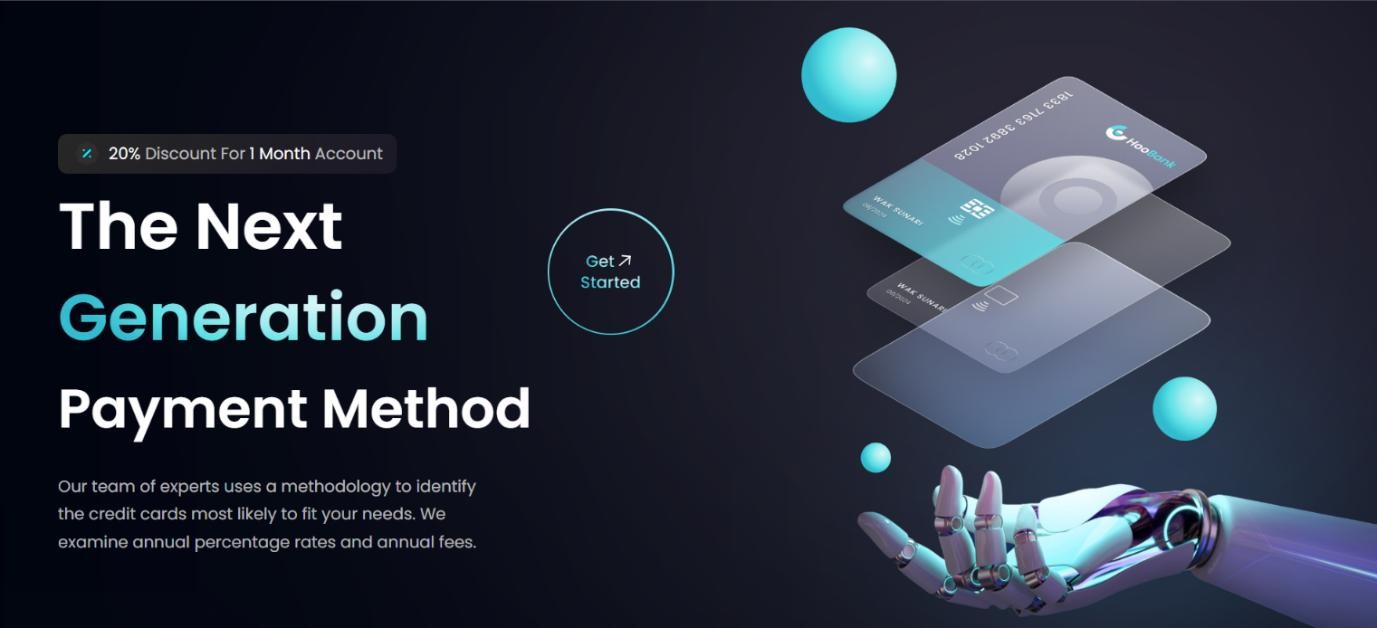
}, [navigate, location.state]);

// ... (Other functions and JSX for the dashboard)

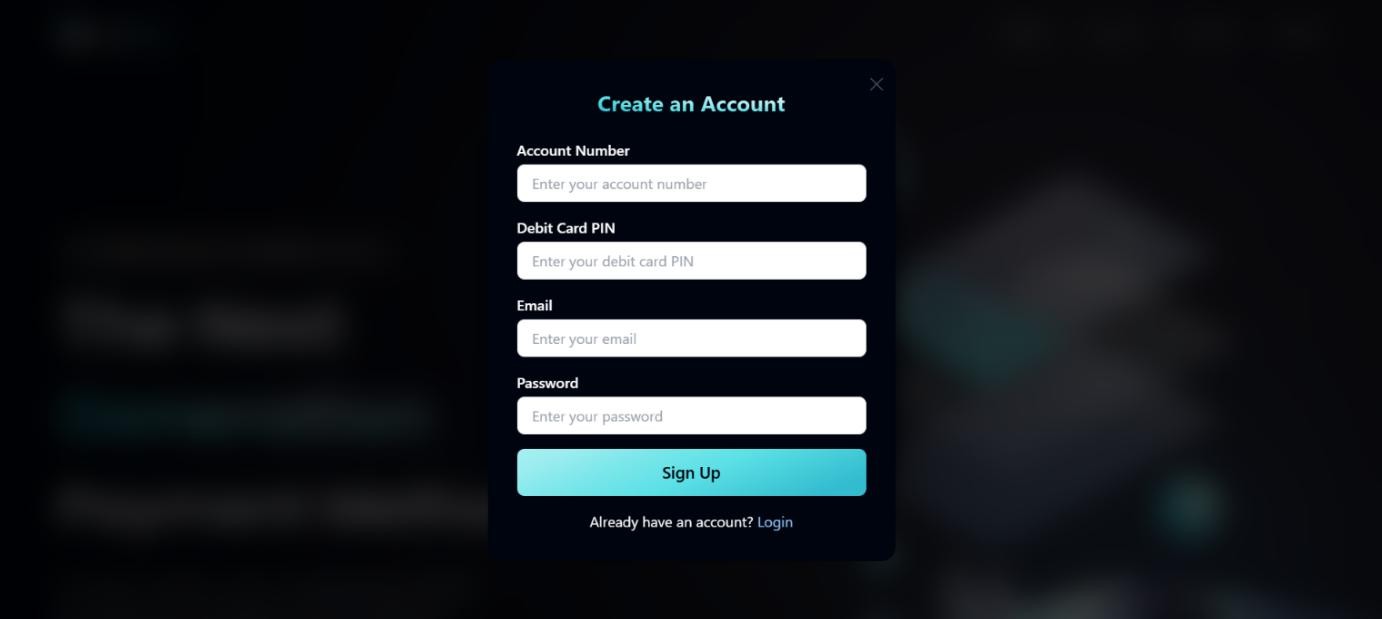
};

export default Dashboard;

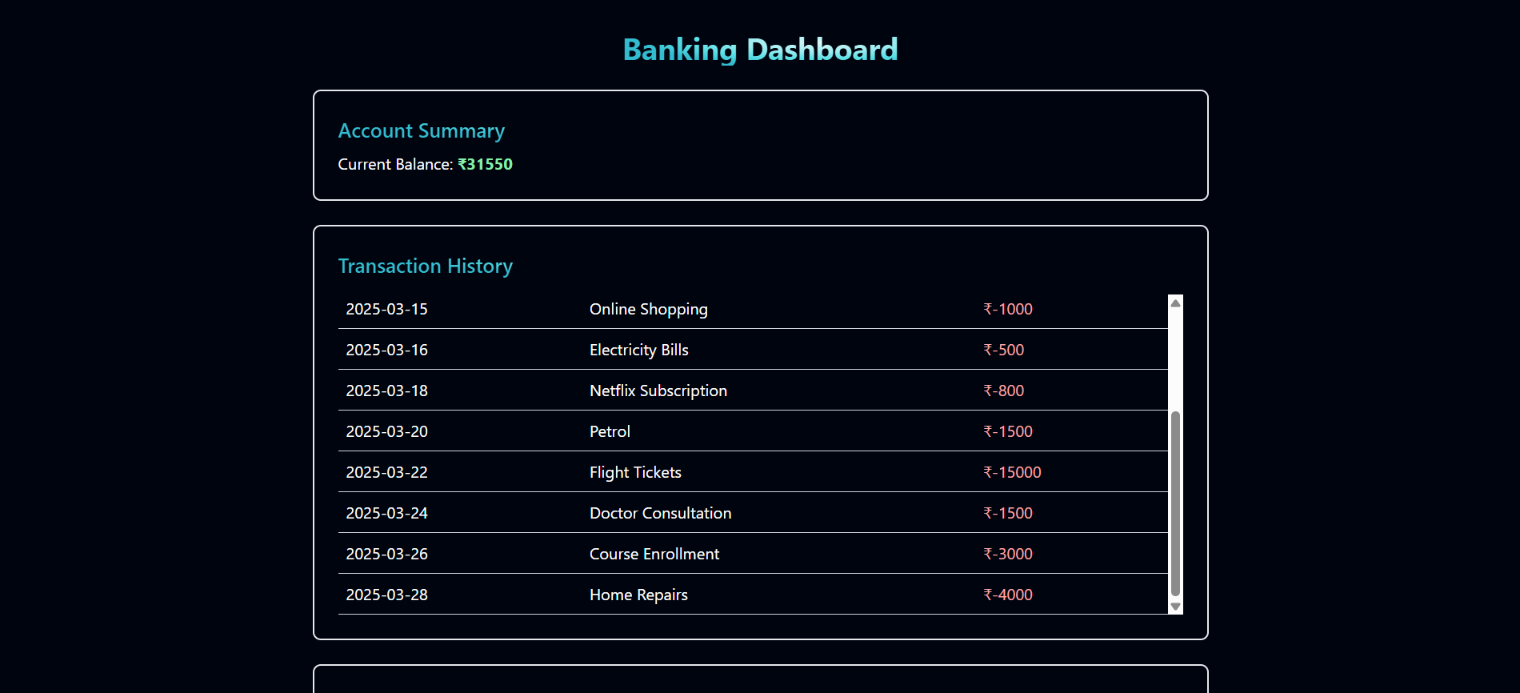
# **User Interface Screenshots**



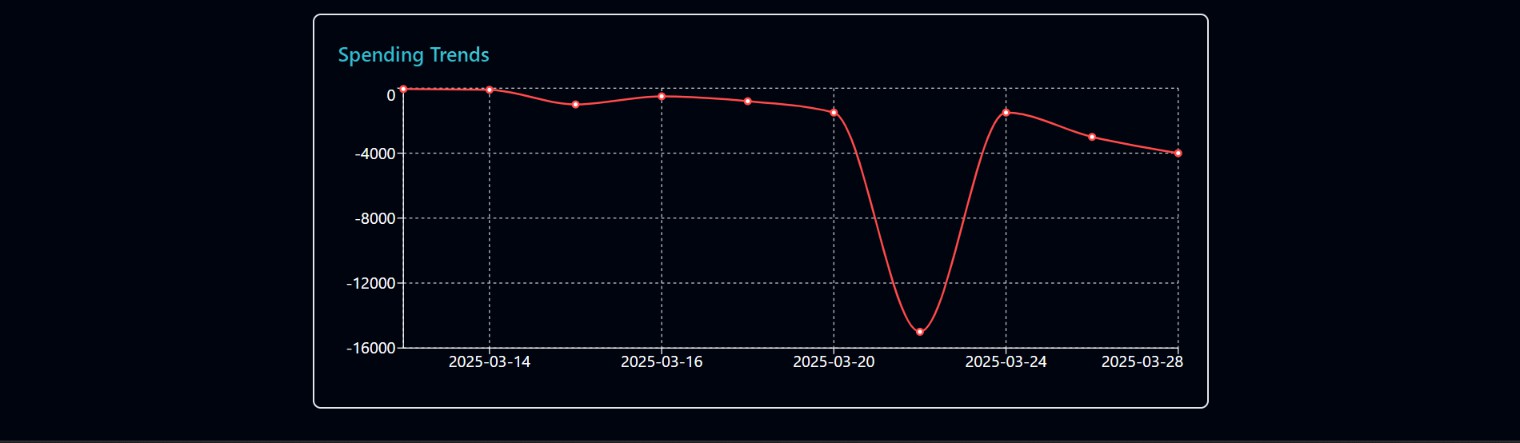
*Figure 3.1: HomePage*



*Figure 3.2: Sign-up and login Section*



*Figure 3.3: Dashboard Section*



*Figure 3.4: spendings chart Section*

# **Conclusion and Future Enhancement**

HooBank successfully delivers a modern, user-friendly, and responsive digital banking experience. The platform ensures smooth navigation and optimal performance across various devices. The application provides essential banking features, including a structured authentication system, transaction history tracking, and an expense chart for visualizing financial data using Recharts. These features enhance user engagement and make financial management more intuitive.

While the current implementation provides a solid foundation, future enhancements could include:

* Secure API Integration: Further secure transaction processing and real-time account updates.
* User Authentication Enhancements: Strengthen security with multi-factor authentication and encrypted credentials.
* Personalized Financial Insights: Introduce AI-driven analytics to provide users with spending insights and budget recommendations.
* Expanded Banking Functionalities: Add features like fund transfers, bill payments, and investment tracking.
* Dark Mode Support: Offering a customizable interface for improved user experience.
* Performance Optimization: Enhancing frontend efficiency through better state management and caching mechanisms for faster load times.

With these advancements, HooBank aims to evolve into a comprehensive and scalable digital banking solution, offering a seamless and secure experience for users while demonstrating expertise in full-stack development, UI/UX design, and financial data visualization.