



adil-01 yesterday



439 lines (294 loc) · 23.6 KB

Preview

Code

Blame



Project Documentation: Online Banking System

Table of Contents

1. Introduction

- Purpose of the Document
- Project Overview

2. System Architecture

- Frontend (React)
- Backend (Spring Boot)
- Database (MariaDB)

3. Installation and Setup

- Prerequisites
- Installation Steps

4. Account Management

- User Dashboard
- Account Creation
- Internet Banking Registration
- Account Login & Authentication
- JWT State Management
- Account Management
- Account Recovery

5. Fund Transfers

- Adding Payees
- Fund Transfer Methods
- Transaction Validation

6. Admin Dashboard

- Admin Home Page
- Admin Login
- JWT State Management
- Editing & Deleting Accounts

7. Transaction History

- Viewing Transaction History
- Transaction Details

8. Page Descriptions

- Account Authentication Pages
- Account Management Pages
- Fund Transfer Pages
- Admin Dashboard Pages
- Transaction History Pages

9. Troubleshooting

- Common Issues and Solutions

11. Testing

12. PWA

13. Future Enhancements

- List of Possible Improvements

14. Conclusion

- Summary of the Project

1. Introduction

Purpose of the Document

This document serves as a comprehensive guide to the Online Banking System application, offering detailed information on its features, functionality, and usage instructions for both end-users and administrators.

Project Overview

Online Banking System is a full-stack web application developed using React for the frontend, Spring Boot for the backend, and MariaDB for the database. It consists of several modules, including User Management, Account Management, Fund Transfers, Admin Dashboard, and Transaction History, each catering to specific user and administrative needs.

2. System Architecture

Frontend (React)

The frontend architecture of Online Banking System is built using React, a popular JavaScript library for building user interfaces. It utilizes various components, state management, and routing to create a seamless user experience.

Technologies and Libraries Used

- **React** is a JavaScript library for building user interfaces. It is used to build single-page applications and allows us to create reusable UI components.
- **React Router** is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.
- **LocalStorage** is a read-only property of the window interface that allows you to access a Storage object for the Document's origin; the stored data is saved across browser sessions. LocalStorage is similar to sessionStorage, except that while localStorage data has no expiration time, sessionStorage data gets cleared when the page session ends.
- **Axios** is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and node.js with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests. It can make XMLHttpRequests from the browser, make http requests from node.js, supports the Promise API, intercept request and response, transform request and response data, cancel requests, and more.
- **nanoid** is a tiny, secure, URL-friendly, unique string ID generator for JavaScript. It uses hardware random generator and can be used in clusters. It uses a larger alphabet than UUID (A-Za-z0-9_-), so ID size was reduced from 36 to 21 symbols.
- **jwt-decode** is a small browser library that helps decoding JWTs token which are Base64Url encoded. However, this library doesn't validate the token; any well-formed JWT can be decoded. You should validate the token in your server-side logic by using something like express-jwt, koa-jwt, Owin Bearer JWT, etc²⁵.
- **Jest** is a delightful JavaScript Testing Framework with a focus on simplicity. It works with projects using Babel, TypeScript, Node, React, Angular, Vue and more. Jest aims to work out of the box, config-free, on most JavaScript projects. It can make tests which keep track of large objects with ease using snapshots.

Backend (Spring Boot)

The backend of Online Banking System is powered by Spring Boot, a framework for building Java-based applications. It handles data processing, business logic, and communication with the frontend.

Technologies and Libraries Used

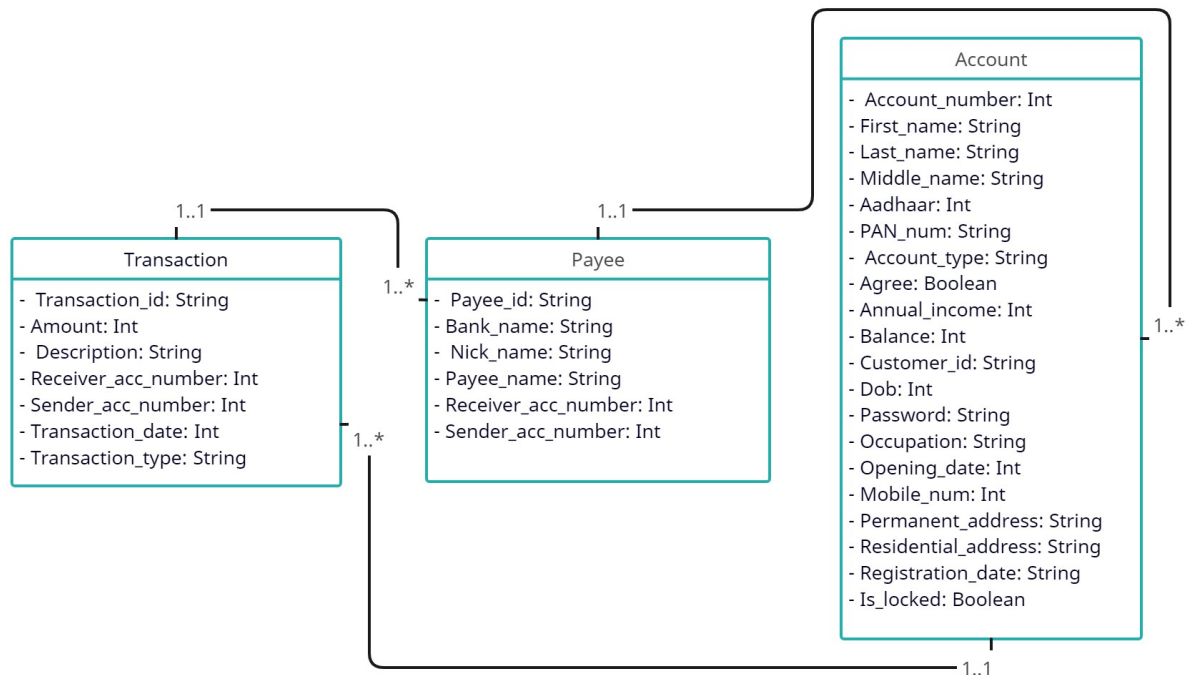
- **Spring Boot:**
 - Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications. It is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

- **Spring Security:**
 - Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications. It is a sub-project of Spring framework which was started in 2003 by Ben Alex. Later on, in 2004, It was released under the Apache License as Spring Security 2.0.0. It overcomes all the problems that come during creating non spring security applications and manage new server environment for the application.
- **Spring Data JPA:**
 - JPA is a Java specification(Jakarta Persistence API) and it manages relational data in Java applications. To access and persist data between Java object(Plain Old Java object)/ class and relational database, we can use JPA. Upon Object-Relation Mapping (ORM), it follows the mechanisms. It has the runtime EntityManager API and it is responsible for processing queries and transactions on the Java objects against the database. The main highlight is it uses JPQL (Java Persistent Query Language) which is platform-independent.
- **Maria DB Driver:** -MariaDB Connector/ODBC is a database driver that uses the industry standard Open Database Connectivity (ODBC) API.
- **Spring Web:** Spring Web is a module of the Spring Framework that provides tools for developing web applications. It includes support for building traditional servlet-based web applications using Spring MVC, as well as reactive web applications using Spring WebFlux.
- **Validator:** A validator is a tool or function that checks if a given input meets certain criteria. Validators can be used to ensure that user input is valid, or to check if data conforms to a specific format. There are many different types of validators, including those for checking HTML markup, string formats, and data types.
- **JWT:** JWT (Java JSON Web Token) is a Java library for creating and verifying JSON Web Tokens (JWTs). JWTs are a compact and secure way of transmitting information between parties as a JSON object. JWT provides an easy-to-use and understand API for working with JWTs on the Java Virtual Machine (JVM) and Android.
- **JAXB-API:** JAXB (Java Architecture for XML Binding) is an API that provides tools for mapping between XML documents and Java objects. It allows developers to work with XML data in a more natural and convenient way by automatically generating Java classes from XML schemas or by using annotations to define the mapping between XML elements and Java classes.
- **MAVEN:** Maven is a software project management and comprehension tool. It uses a Project Object Model (POM) to manage a project's build, dependencies, reporting, and documentation. Maven can automate many aspects of the build process, making it easier to manage complex projects.
- **RESTful APIs:** RESTful APIs are web services that follow the architectural style of Representational State Transfer (REST). REST is an approach to designing APIs that emphasizes simplicity, scalability, and performance. RESTful APIs use standard HTTP methods (such as GET, POST, PUT, DELETE) to perform operations on resources, making them easy to use and understand.

Database (MariaDB)

The application's data is stored and managed using MariaDB, a relational database management system.

Database Schema



- **Transactions table:**

- Transaction_id : Unique identifier for each transaction.
- Amount : The monetary value of the transaction.
- Description : A brief description of the transaction.
- Receiver_acc_number : The account number of the receiver of the transaction.
- Sender_acc_number : The account number of the sender of the transaction.
- Transaction_date : The date on which the transaction occurred.
- Transaction_type : The type of transaction (e.g. deposit, withdrawal, transfer).

- **Payee table:**

- Payee_id : Unique identifier for each payee.
- Bank_name : The name of the bank associated with the payee.
- Nick_name : A nickname for the payee, as specified by the user.
- Payee_name : The name of the payee.
- Receiver_acc_number : The account number of the receiver associated with the payee.
- Sender_acc_number : The account number of the sender associated with the payee.

- **Account table:**

- `Account_number` : Unique identifier for each account.
- `First_name` : The first name of the account holder.
- `Last_name` : The last name of the account holder.
- `Middle_name` : The middle name of the account holder.
- `Aadhar_number` : The Aadhar number of the account holder (a unique identification number issued by the Indian government).
- `PAN_number` : The PAN (Permanent Account Number) of the account holder (a unique identification number used for tax purposes in India).
- `Account_type` : The type of account (e.g. savings, checking).
- `Agree` : Whether or not the account holder has agreed to the terms and conditions.
- `Annual_income` : The annual income of the account holder.
- `Balance` : The current balance of the account.
- `Customer_id` : Unique identifier for each customer associated with an account.
- `DoB` : The date of birth of the account holder.
- `Password` : The password associated with the account.
- `Occupation` : The occupation of the account holder.
- `Opening_date` : The date on which the account was opened.
- `Mobile_num` : The mobile number associated with the account.
- `Permanent_address` : The permanent address of the account holder.
- `Residential_address` : The residential address of the account holder.
- `Registration_date` : The date on which the account was registered.
- `Is_locked` : Whether or not the account is locked.

3. Installation and Setup

Prerequisites

Before setting up Online Banking System, ensure you have the following software/tools installed:

- Java Development Kit (JDK)
- Node.js and npm
- MariaDB
- IDE (Integrated Development Environment) of your choice

Installation Steps

Follow these steps to set up HooBank on your local environment:

1. Clone the Repository:

```
git clone https://github.com/adil-01/hoobank_backend
git clone https://github.com/adil-01/hoobank_frontend
```



2. Backend Setup:

- Open the backend project in your IDE.
- Configure the database connection in `application.properties`.
- Run the Spring Boot application.

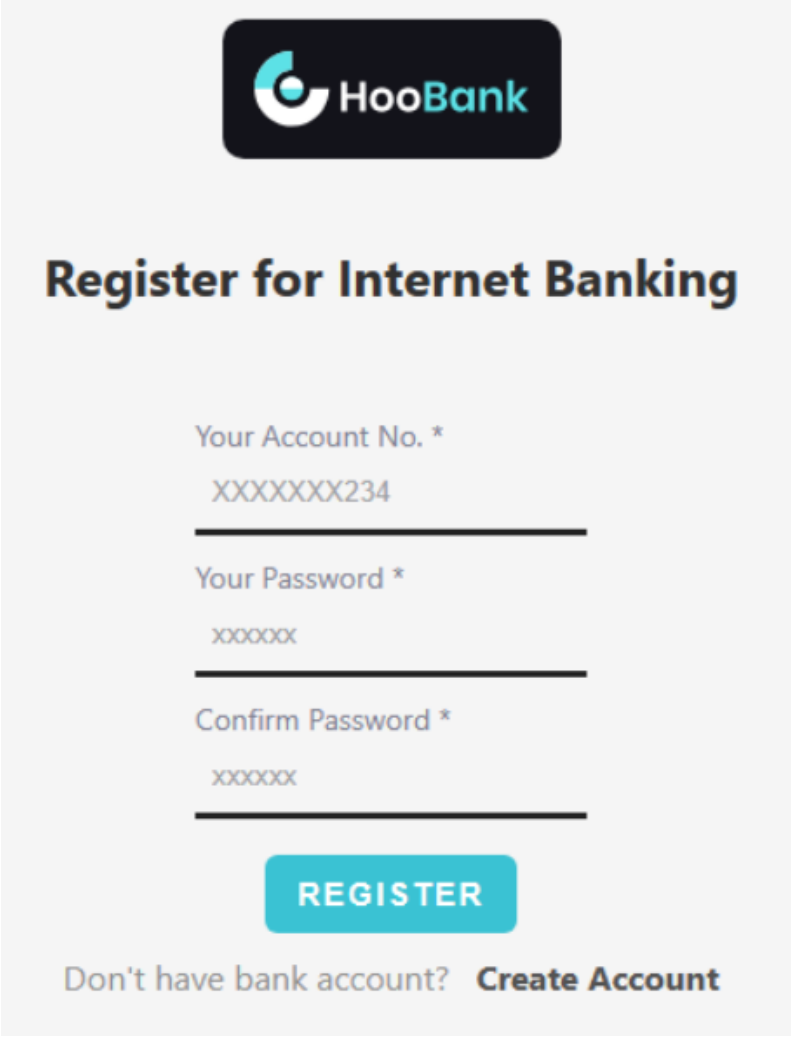
3. Frontend Setup:

- Navigate to the frontend directory: `cd hoobank_frontend`
- Install frontend dependencies: `npm install`
- Start the development server: `npm start`

4. **Access the Application:** Open your web browser and visit `http://localhost:3000` to access HooBank.

5. Account Management

User Registration

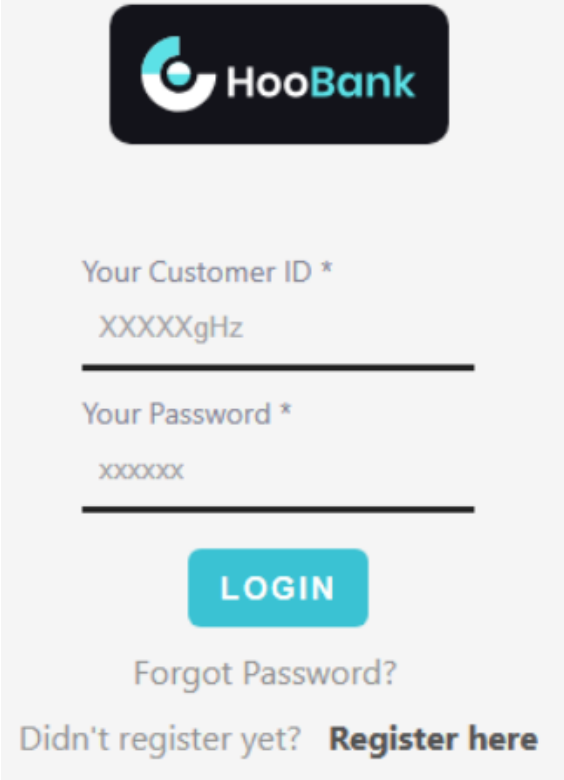


The image shows a registration form for HooBank. At the top is the HooBank logo, which consists of a stylized 'G' icon followed by the text 'HooBank'. Below the logo is the title 'Register for Internet Banking'. The form contains three input fields: 'Your Account No. *' with the placeholder text 'XXXXXXXX234', 'Your Password *' with the placeholder text 'xxxxxx', and 'Confirm Password *' with the placeholder text 'xxxxxx'. Each input field has a horizontal line below it. At the bottom of the form is a blue button with the text 'REGISTER'. Below the button is a link that says 'Don't have bank account? Create Account'.

New users can register for internet banking using the following steps:

1. Navigate to the registration page.
2. Fill in the required details (Account Number, password, etc.).
3. Submit the registration form.

User Login and Authentication



The image shows a login form for HooBank. At the top is the HooBank logo, which consists of a stylized 'G' icon in teal and black, followed by the text 'HooBank' in a sans-serif font. Below the logo are two input fields. The first is labeled 'Your Customer ID *' and contains the text 'XXXXXgHz'. The second is labeled 'Your Password *' and contains the text 'xxxxxx'. Both fields have a horizontal line below them. Below the password field is a teal button with the word 'LOGIN' in white capital letters. Below the button are two links: 'Forgot Password?' and 'Didn't register yet? **Register here**'.


Registered users can log in using their credentials:

1. Access the login page.
2. Enter the customer ID and password.
3. Click the "Login" button.
4. Upon successful authentication, the user will be redirected to their dashboard.

Account Recovery

Users who forget their password, can follow these recovery steps:

2. **Forgot Password:**



New Password *

.....

Confirm Password *

.....

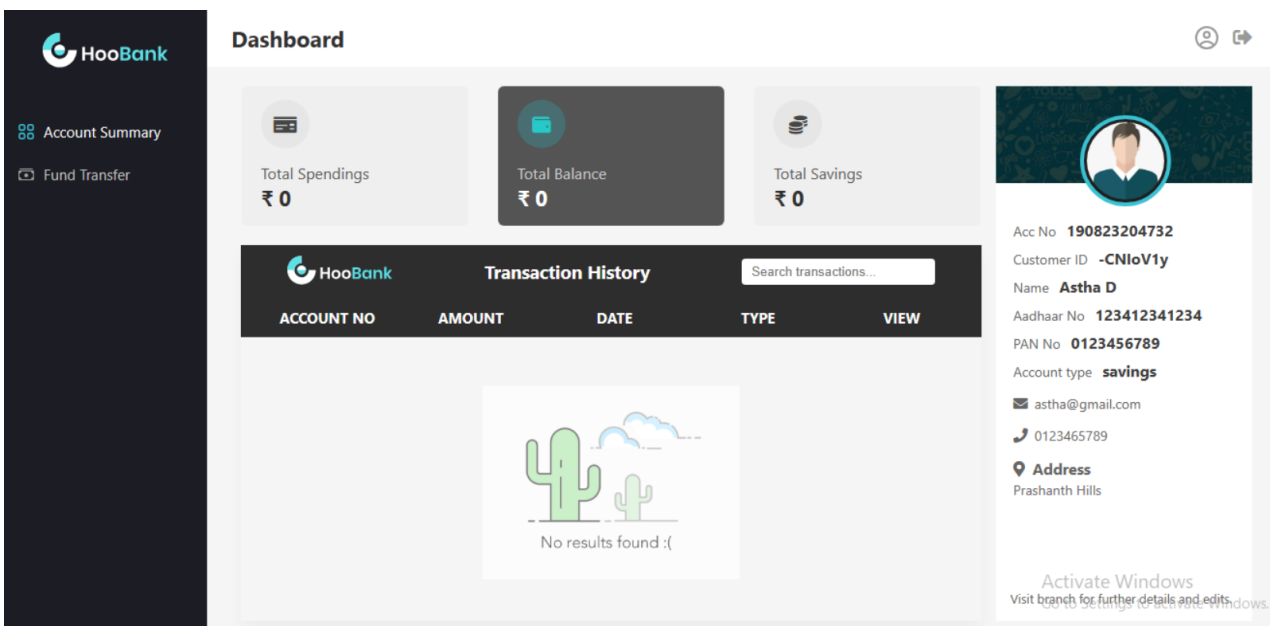
CHANGE PASSWORD

- Click the "Forgot Password" link on the login page.
- Provide the Customer ID.
- Verify with OTP and redirect to create new password page.
- Set a new password and confirm the same.

3. Account Locked:

- If the account is locked due to multiple failed login attempts, the users account can get locked by admin.
- Visit the branch and get your account unlocked by admin.

User Dashboard



Dashboard

Total Spendings ₹ 0

Total Balance ₹ 0

Total Savings ₹ 0

Transaction History

ACCOUNT NO	AMOUNT	DATE	TYPE	VIEW
No results found :(

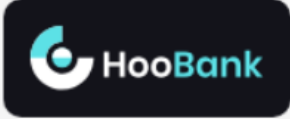
Acc No **190823204732**
 Customer ID **-CNloV1y**
 Name **Astha D**
 Aadhaar No **123412341234**
 PAN No **0123456789**
 Account type **savings**
 ✉ astha@gmail.com
 ☎ 0123465789
 📍 Address
 Prashanth Hills

Activate Windows
 Visit branch for further details and edits.


Once logged in, users are directed to their dashboard, where they can access various account-related features:

1. Account Summary: An overview of their account balances and recent transactions.
2. Account Statement: Detailed view of account transactions within a specified timeframe.
3. Change Password: Option to update the account password.
4. Session Management: View and manage active sessions for enhanced security.

Account Creation



Open an Account

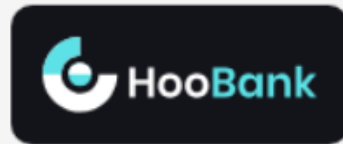
Title *	Aadhaar No *	Date of Birth *
Mr.	XXXX XXXX XXXX	dd - mm - yyyy 
First Name *	Your PAN No *	Your Mobile No *
John	XXXXXXXXXX	9876543210
Middle Name	Residential Address *	Your Email *
Wilson	Enter your address here	hello@gmail.com
Last Name *	Permanent Address *	Your Occupation *
Doe	Enter your address here	Govt. Officer
Father's Name *		Annual Income *
James Jacob		₹ 120000

☐ I agree to terms & conditions *

ADD ACCOUNT

Users can open a new bank account by following these steps:

1. Click the "Open an Account" button.
2. Provide personal and account-related details.
3. Agree to terms and conditions.
4. Submit the application for review.



Register for Internet Banking

Your Account No. *

190823204732

Your Password *

xxxxxx

Confirm Password *

xxxxxx

REGISTER

Don't have bank account? **Create Account**



SUCCESS

Your Acc No: 190823204732



6. Fund Transfers

Adding Payees

Fund Transfer

HooBank

--- select beneficiary ---

Not found? **Add beneficiary**

Amount *
₹ 500

Choose Payment Method
☐ IMPS ☐ Neft ☐ RTGS

Remarks *
Bank deposit

PAY

Paypal

Checkout

Total
\$210

Change

Make Payment

Last Transaction

Dribbble Pro

16 Days ago

▼ -\$250.93

Netflix

4 Days ago

▼ -\$250.93

Manulife Cash

4 Days ago

▲ -\$250.93

Great! Your Payment is successfully.

Before initiating a fund transfer, users must add payees (other users' bank account details) using the "Add Payee" feature:

1. Access the "Add Payee" page.
2. Enter the payee's account details (account number, bank details, etc.).
3. Verify the payee information.
4. Confirm the addition.

Fund Transfer Methods

Choose Payment Method

☐ IMPS ☐ Neft ☐ RTGS

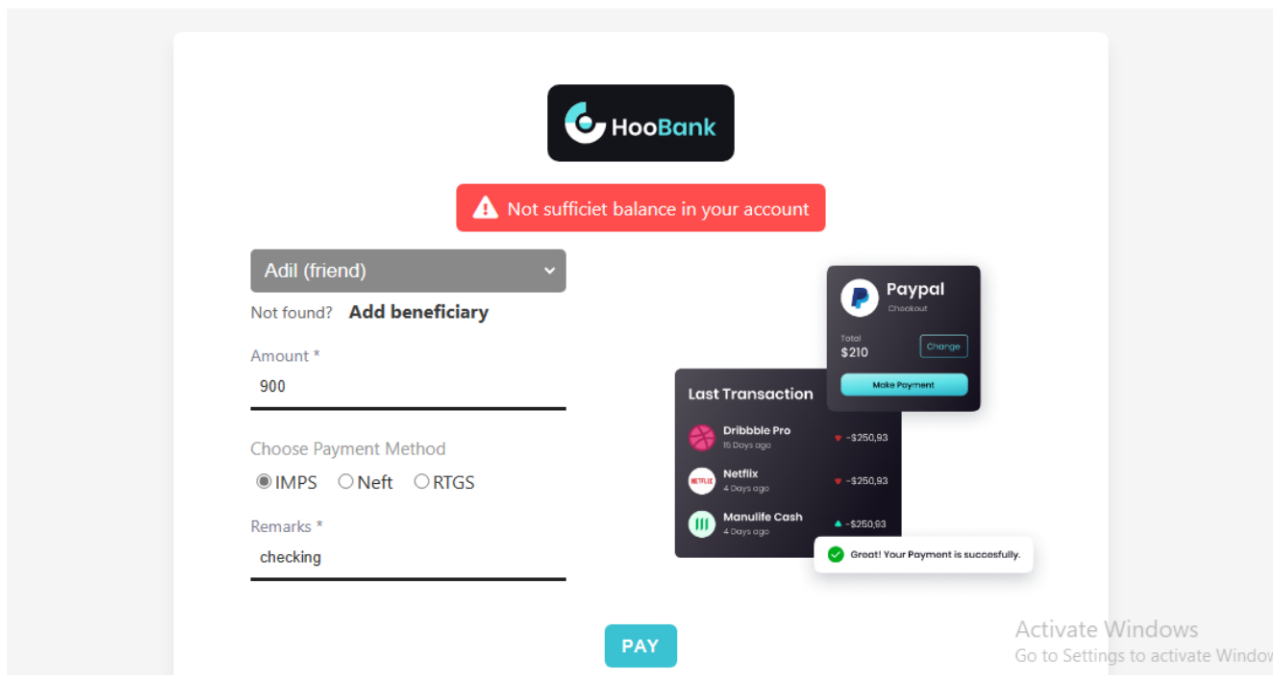
Remarks *

Bank deposit

Users can transfer funds to other accounts using various methods:

1. NEFT (National Electronic Funds Transfer)
2. IMPS (Immediate Payment Service)
3. RTGS (Real Time Gross Settlement)

Transaction Validation








The image shows a mobile app interface for HooBank's fund transfer feature. At the top, the HooBank logo is displayed. Below it, a red error banner states "Not sufficient balance in your account". The main form includes a beneficiary dropdown menu currently showing "Adil (friend)", a link to "Add beneficiary" if not found, an "Amount" field with "900" entered, and a "Choose Payment Method" section with radio buttons for "IMPS", "Neft", and "RTGS". A "Remarks" field contains the text "checking". A large blue "PAY" button is at the bottom. Overlaid on the right is a "Paypal Checkout" modal showing a total of "\$210" and a "Make Payment" button. Below this is a "Last Transaction" list showing three items: "Dribbble Pro" (-\$250.83, 16 days ago), "Netflix" (-\$250.83, 4 days ago), and "Manulife Cash" (+\$250.83, 4 days ago). A green success message at the bottom right says "Great! Your Payment is successfully."

Before confirming a fund transfer, users should review the transaction details and confirm the recipient's information.

- The upper limit of the transaction has been set as Rs 2,00,000.
- Validating the amount whether its negative or greater than the account balance.

7. Admin Dashboard

 Account Details <input data-bbox="1034 1211 1348 1261" type="text" value="Search accounts..."/>				
ACCOUNT NO	BALANCE	NAME	MOBILE	ACTION
180823224742	₹ 150	Adil Adil	9876543211	 
190823204732	₹ 850	Astha D	0123465789	 

Admin Login

Administrators can access a dedicated dashboard to manage user accounts, and system settings.

- Admin Login & Authentication
- JWT State Management

Account Management

Admins can perform various tasks related to Account management:

1. View Account: List of registered users with key details.
2. Edit Account: Modify user details (name, email, etc.).
3. Delete Account: Remove a user account from the system.

8. Transaction History

Viewing Transaction History

The screenshot displays the HooBank Dashboard. On the left is a dark sidebar with the HooBank logo and navigation links: Account Summary, Fund Transfer, and Change Password. The main content area is titled 'Dashboard' and features three summary cards: 'Total Spendings ₹ 200', 'Total Balance ₹ 850', and 'Total Savings ₹ 50'. Below these is a 'Transactions' section with a search bar and date range filters (01-07-2023 to 23-08-2023). A table lists three transactions with columns for Account No, Amount, Date, Type, and View. To the right of the table is a user profile card for 'Astha D' with account details and contact information. A Windows watermark is visible at the bottom right.

ACCOUNT NO	AMOUNT	DATE	TYPE	VIEW
180823224742	₹ 200 ↓	08/19/2023	IMPS	
180823224742	₹ 20 ↑	22-08-2023	RTGS	
180823224742	₹ 30 ↑	08/21/2023	Neft	

User Profile:
Acc No: 190823204732
Customer ID: -CNloV1y
Name: Astha D
Aadhaar No: 123412341234
PAN No: 0123456789
Account type: savings
Email: astha@gmail.com
Phone: 0123465789
Address: Prashanth Hills

Users can view their transaction history by accessing the "Transaction History" section:

1. Auto fetching account using JWT token.
2. Search for the desired transaction.
3. View a list of transactions with details & status.

Transaction Details



Payment Successful

Transaction ID **26673387-ef85-43ba-b0c1-6a3dfa11dd2e**

Payer Acc No

190823204732



Payee Acc No

180823224742

Transaction Information

Transaction type

IMPS

Amount

₹ 200

Transaction date

08/19/2023

Remarks

for testing

PRINT



Dashboard

Clicking on a specific transaction provides users with detailed information about that transaction:

1. Transaction ID
2. Date and Time
3. Remarks
4. Sender and Receiver Details
5. Transaction Amount
6. Payment Method
7. Status (Pending, Completed, Failed, etc.)

9. Page Descriptions

User Management Pages

- Home Page: Landing page after login.
- Login: User authentication page.
- Open an Account: Account creation form.
- Register for Internet Banking: Instructions for online registration.
- Forgot Password: Account recovery page for forgotten passwords.
- Account Locked Page: Information about locked accounts.
- Set New Password: Page for setting a new password after password reset.

Account Management Pages

- Dashboard: User's main account page.
- Account Statement: Detailed transaction history.

- Summary: Account balance and recent transactions.
- Change Password: Form to update the account password.
- Session Expired: Notification when the session expires.

Fund Transfer Pages

- Add Payee: Form to add new payees.
- Fund Transfer: Form for initiating fund transfers.

Admin Dashboard Pages

- Home Page: Admin dashboard landing page.
- User Management: List of registered users with actions.

Transaction History Pages

- Transaction History: Display of transaction history with filters.

10. Troubleshooting

Common Issues and Solutions

- Troubleshooting steps for login issues.
- Account recovery process for forgotten credentials.
- Steps to address transaction errors.
- Instructions for contacting support.

11. Testing

- **API Testing:**
 - API testing is a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer.
 - API testing is used to determine whether APIs return the correct response (in the expected format) for a broad range of feasible requests, react properly to edge cases such as failures and unexpected/extreme inputs, deliver responses in an acceptable amount of time, and respond securely to potential security attacks.
 - API testing involves testing APIs directly (in isolation) and as part of the end-to-end transactions exercised during integration testing. Beyond RESTful APIs, these transactions include multiple types of endpoints such as web services, ESBs, databases, mainframes, web UIs, and ERPs.

- API testing plays a central role in the API-first approach, as it enables teams to continuously verify the quality, health, and performance of their endpoints as they work to deliver a seamless digital experience. There are several types of API tests, and each one plays a distinct role in ensuring the API remains reliable. Traditionally, API testing has occurred at the end of the development phase, but an increasing number of teams are running tests earlier in the API lifecycle. This approach to API testing, which is known as "shifting left," supports rapid iteration by enabling teams to catch and fix issues as soon as they are introduced.

- **Integration Testing:**

- Integration testing is a type of software testing where individual software modules are combined and tested as a group. It is conducted to evaluate the compliance of a system or component with specified functional requirements. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated¹. Integration Testing focuses on checking data communication amongst these modules.
- There are several types of Integration Testing, including Big Bang Testing, Incremental Testing, Bottom-up Integration Testing, Top-down Integration Testing, and Sandwich Testing. The goal of integration testing is to check the correctness of communication among all the modules. Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as integration testing.
- Some best practices for Integration Testing include determining the Integration Test Strategy that could be adopted and later preparing the test cases and test data, studying the Architecture design of the Application and identifying the Critical Modules, designing test cases to verify each interface in detail, choosing input data for test case execution, performing positive and negative integration testing, and communicating bug reports to developers and fixing defects and retesting.

- **Unit Testing:**

- Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.
- Unit Testing is important because it helps to fix bugs early in the development cycle and save costs. It helps the developers to understand the testing code base and enables them to make changes quickly. Good unit tests serve as project documentation. Unit tests help with code re-use.
- There are several techniques for Unit Testing, including Black Box Testing, White Box Testing, and Gray Box Testing. Black Box Testing is used in covering the unit tests for input, user interface, and output parts. White Box Testing is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules. Gray Box Testing is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

React Testing:

The screenshot shows a VS Code editor with a file named `global.test.js` open. The code is a Jest test for a 'card render' component. It uses `render` from `react-dom/test-utils` to render the component with specific props. Then, it uses `screen` from `@testing-library/react` to find the rendered element and asserts that it is in the document and has the correct text content.

```
obs > src > components > __test__ > JS global.test.js > test('card render') callback
5 test('card render', () => {
6   render(<Card cardIcon='fa fa-money-check' cardLabel={"Total Spendings"}
7     cardInfo={`₹ 1000`} theme="dark" />)
8
9   const cardElement = screen.getByTestId('card-1');
10  expect(cardElement).toBeInTheDocument();
11  expect(cardElement).toHaveTextContent('Total Spendings');
12 })
```

The terminal at the bottom shows the test results for `src/components/__test__/global.test.js`. All tests passed, including 'card render', 'logout button render', 'form validation', 'popup render', 'admin dashboard render', 'admin all_accounts api', 'navbar render', 'transaction render', and 'account creation render'. The summary shows 1 test suite passed, 9 tests passed, 0 snapshots, and a total time of 2.085 s.

Test Suites: 1 passed, 1 total
Tests: 9 passed, 9 total
Snapshots: 0 total
Time: 2.085 s
Ran all test suites.

Spring Boot Testing:

The screenshot shows an IDE with a Java file `BackendApplicationTests.java` open. The code is a JUnit test class that uses `@Autowired` to inject `PayeeService` and `@MockBean` to mock `AccountRepository`, `TransactionRepository`, and `PayeeRepository`. The test method `getAllTransactionsByAccNumberServiceTest` sets up a mock transaction and verifies that the service returns the correct list of transactions.



```
41 @Autowired
42 private PayeeService payeeService;
43
44 @MockBean
45 private AccountRepository accountRepository;
46
47 @MockBean
48 private TransactionRepository transactionRepository;
49
50 @MockBean
51 private PayeeRepository payeeRepository;
52
53 @Test
54 public void getAllTransactionsByAccNumberServiceTest() {
55
56   String acc_no = "190823204732";
57   // Mock data records setting
58   when(transactionRepository.findBySenderAccNumber(acc_no))
59     .thenReturn(Stream.of(new Transaction(), new Transaction()).collect(Collectors.toList()));
60 }
```

The bottom of the screenshot shows the JUnit test runner output. It indicates that the tests finished after 8.518 seconds with 5/5 runs, 0 errors, and 0 failures. The list of tests includes `checkBalanceTest`, `getAllPayeesByAccNoServiceTest`, `getAccountByCustomerIdServiceTest`, `getAllTransactionsByAccNumberServiceTest`, and `getAllAccountsServiceTest`.

Finished after 8.518 seconds
Runs: 5/5 Errors: 0 Failures: 0
com.obs.backend.BackendApplicationTests [Runner: JUnit 4] (0.250 s)
checkBalanceTest (0.186 s)
getAllPayeesByAccNoServiceTest (0.025 s)
getAccountByCustomerIdServiceTest (0.006 s)
getAllTransactionsByAccNumberServiceTest (0.005 s)
getAllAccountsServiceTest (0.003 s)


12. PWA (Progressive Web App)

A **Progressive Web App (PWA)** is a type of application built using web technologies, but provides a user experience similar to that of a platform-specific app. PWAs can run on multiple platforms and devices from a single codebase, just like a website. They can also be installed on the device, operate offline and in the background, and integrate with the device and other installed apps.

 http://localhost:3000/ 


There were issues affecting this run of Lighthouse:


- There may be stored data affecting loading performance in this location: IndexedDB. Audit this page in an incognito window to prevent those resources from affecting your scores.





PWA


These checks validate the aspects of a Progressive Web App. [Learn more.](#)


 INSTALLABLE


 Web app manifest and service worker meet the installability requirements





 PWA OPTIMIZED


 Registers a service worker that controls page and `start_url`





 Configured for a custom splash screen





 Sets a theme color for the address bar.





 Content is sized correctly for the viewport





 Has a `<meta name="viewport">` tag with `width` or `initial-scale`



 Provides a valid `apple-touch-icon`




 Manifest has a maskable icon





ADDITIONAL ITEMS TO MANUALLY CHECK (3)


Show


These checks are required by the baseline [PWA Checklist](#) but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.


 Captured at Aug 21, 2023, 10:36 AM GMT+5:30

 Emulated Desktop with Lighthouse 9.6.2

 Single page load

 Initial page load

 Custom throttling

 Using Chromium 105.0.0.0 with devtools

We have tested the PWA functionalities using Google Chrome Developer Tools using Lighthouse tool and the report is shown above.

Overall, PWAs provide a native-like experience to users on supporting devices while being adaptable to different browsers and devices. We have implemented using `serviceWorker.js` by registering, fetching and unregister properties we will store the cache version of website and can use in offline mode in mobile devices also.

13. Future Enhancements

Consider the following potential improvements for HooBank:

- **Multi-factor Authentication:** Enhance security with additional authentication methods.
- **Real-time Notifications:** Notify users about account activities via email or SMS.
- **International Fund Transfers:** Enable cross-border transfers.
- **Enhanced Admin Tools:** Provide more robust user management features for administrators.

14. Conclusion

In conclusion, Online Banking System is a comprehensive banking application that offers users a range of functionalities, including user management, account management, fund transfers, and more. With its intuitive interface and robust features, HooBank aims to provide a seamless banking experience for both users and administrators.