Software Design Document (SDD)

1. Introduction

This document ‘s purpose:

provide a detailed overview of the design and architecture of the IM Service- BE MERN Stack project.

The project aims to create a web application that perform CRUD actions about users and banking data and allows users to view and manage data related to bank accounts, business ‘s addresses, and transaction history etc. The application will have three user levels: normal, manage, and admin, each with different permissions and functionalities.

2. System Overview

The IM Service - BE MERN Stack project will be developed using the following technologies:

Front-end: React.js

Back-end: Node.js, Express.js

Database: MongoDB

3. User Roles and Permissions

The application will have three user roles with different permissions:

3.1. Normal

Can only view their business ‘s data such as bank account number, address, and transaction history etc.

Cannot perform any CRUD operations.

3.2. Manager

Can view information about employees, customers and transaction history.

Only can do update account number, business ‘s address, transaction information and user ‘s information etc.

3.3. Admin

Can perform all CRUD operations on users and datas according to bank(account number, address and transaction information etc).

Can view, create, update, and delete bank account number, addresses, and transaction information.

Can view, create, update, and delete users.

Also add and delete customer of the company.

4. System Architecture

The system architecture will follow the MERN (MongoDB, Express.js, React.js, Node.js) stack. The front-end will be developed using React.js, which will communicate with the back-end API built using Node.js and Express.js. The data will be stored in a MongoDB database.

5. Database Design

The MongoDB database will consist of the following collections:

5.1. Users Collection

Fields:

id: Unique identifier for the user.

name: Name of the user.

email: Email address of the user.

gender: gender of the user

address: Address of the user

company: Name of company

company\_id: Foreign key referencing the company user belonging to

password: Encrypted password of the user.

role: Role of the user (normal user, manager, or admin).

5.2. Transactions Collection

Fields:

id: Unique identifier for the transaction.

amount: Transaction amount.

date: Date of the transaction.

company\_Id: Foreign key referencing the company associated with the transaction.

type: Transaction ‘s type

description: Detailed description about transaction

5.3. Company Collection

Fields:

id: Unique identifier for the customer.

name: Name of the company.

account\_number: Bank account number of the company.

address: Address of the company

customers: list of user(company ‘s customer) id

6. API Endpoints

The following API endpoints will be implemented:

6.1. User Endpoints

GET /api/users: Get all users (all).

GET /api/users/:id: Get user by ID (admin only).

GET /api/users/company/id: Get company ‘s employees by company ‘s ID (all

employees belonged to company).

POST /api/users: Create a new user

POST /api/users/login: Login

POST /api/users/avatar: uploading avatar file

PUT /api/users/:id: Update user (admin and manager only).

DELETE /api/users/:id: Delete user by ID (admin only).

6.2. Transaction Endpoints

GET /api/transactions: Get all transactions (Project manager).

GET /api/transactions/:company\_id: Get transaction by company ID (all users).

POST /api/transactions: Create a new transaction (admin only).

PUT /api/transactions/:id Update the transaction (admin and manager only).

DELETE /api/transactions/:id Delete a transaction (admin only).

6.3. Company Endpoints

GET /api/company: Get all customers (Project manager).

GET /api/ company /customer/:company\_id: Get all customers connected to company (all users).

GET /api/ company /:id: Get company by ID (all users).

POST /api/ company: Create a new company (admin only).

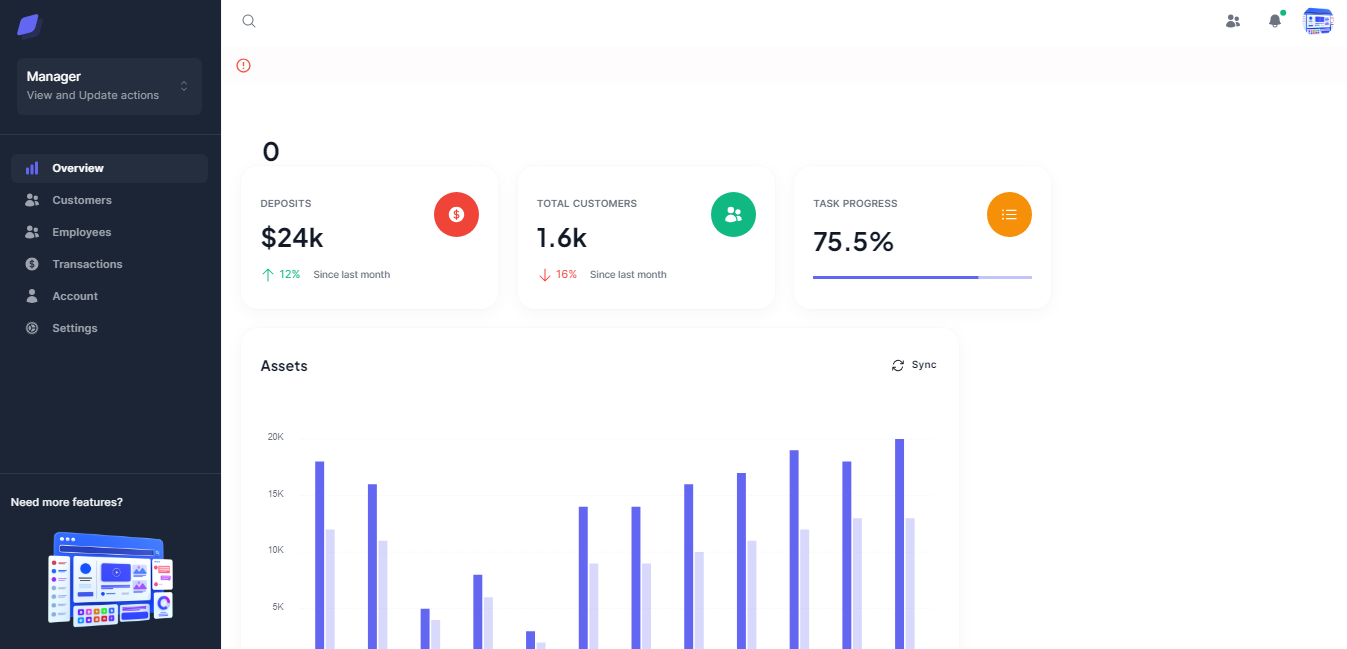
PUT /api/ company /:id: Update company (admin and manager only).

DELETE /api/ company /:id: Delete customer by ID (admin only).

6. Front-end Design

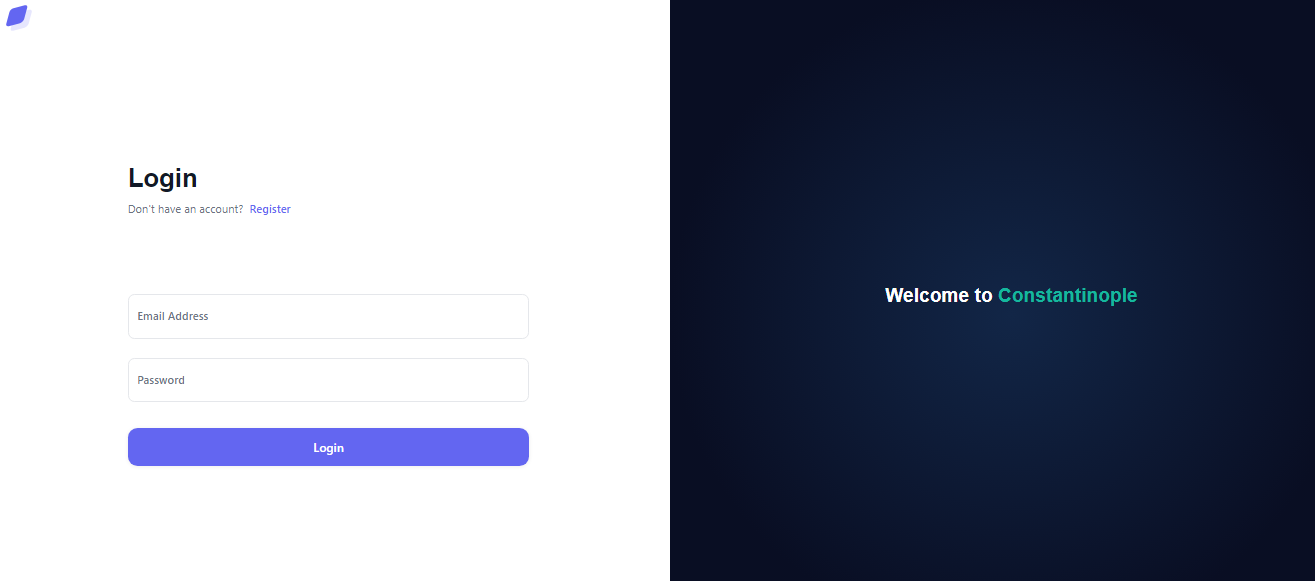
6.1. First page

url: /

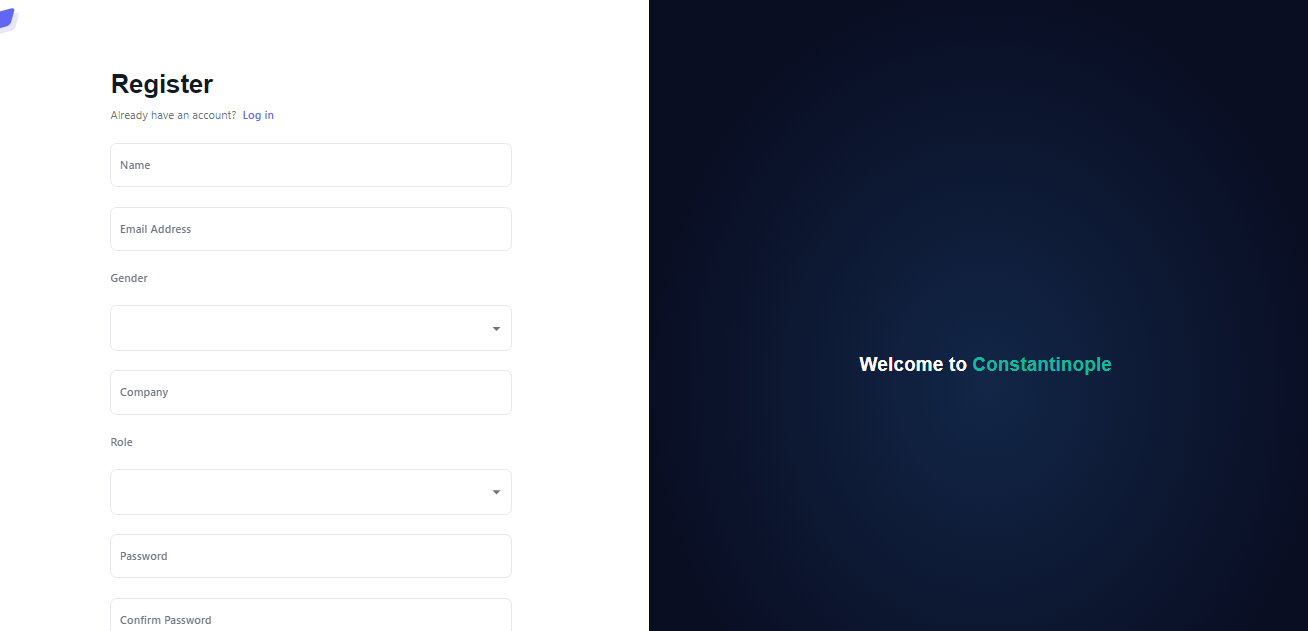


6.2. Login and register page

url: /auth/login

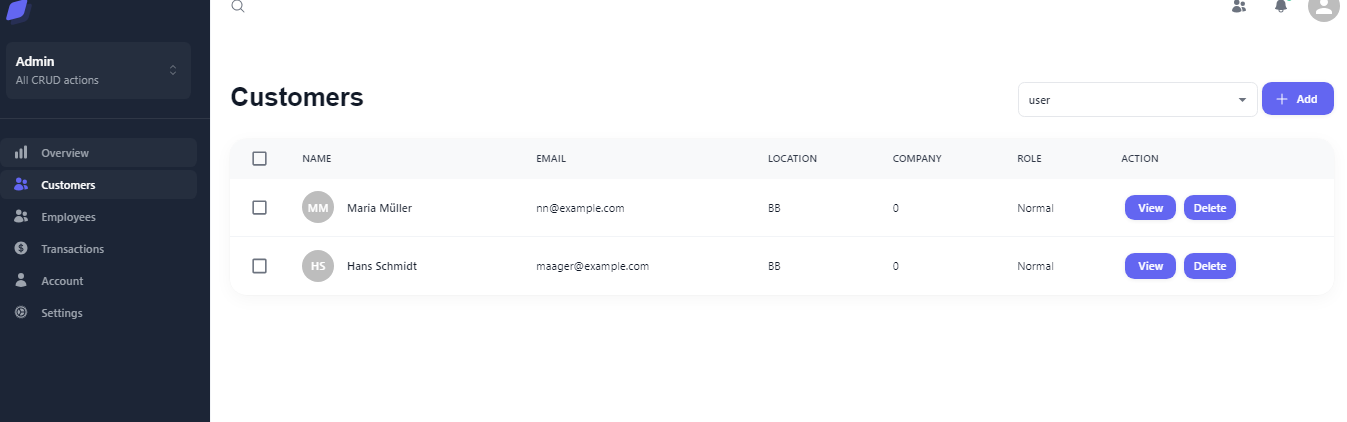


url: /auth/register



6.3. View customer’s information

url: /customers

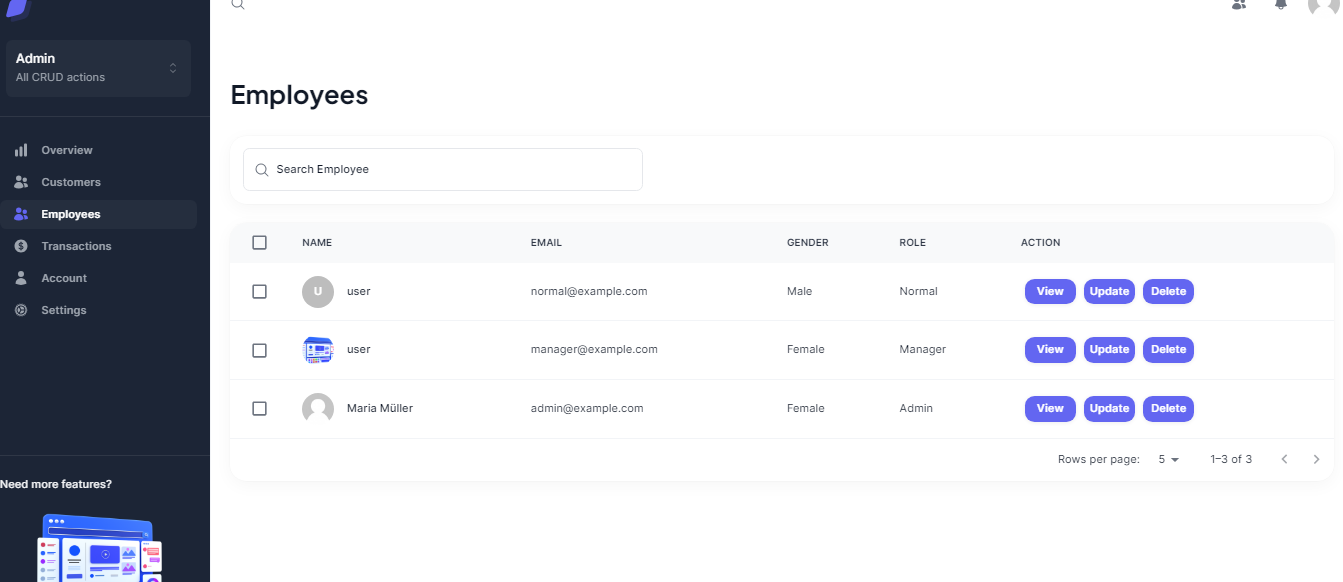


If you are admin, you can add and delete customer

Role of user

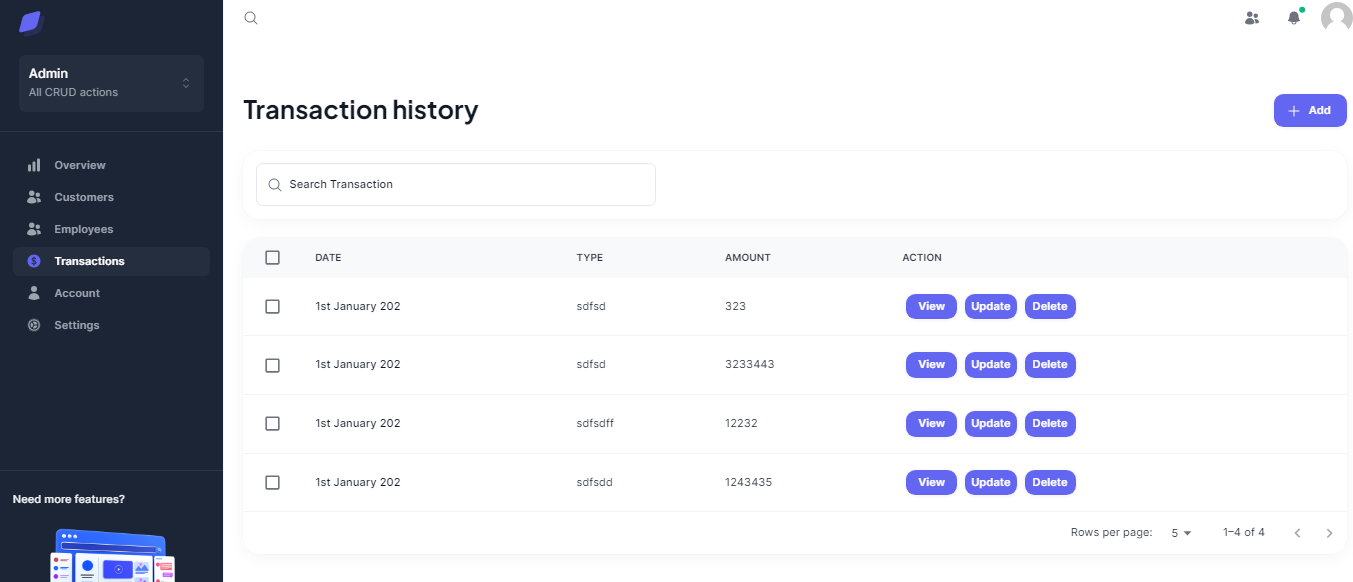
6.4. View employee’s information

url: /employees



6.5. View transaction history

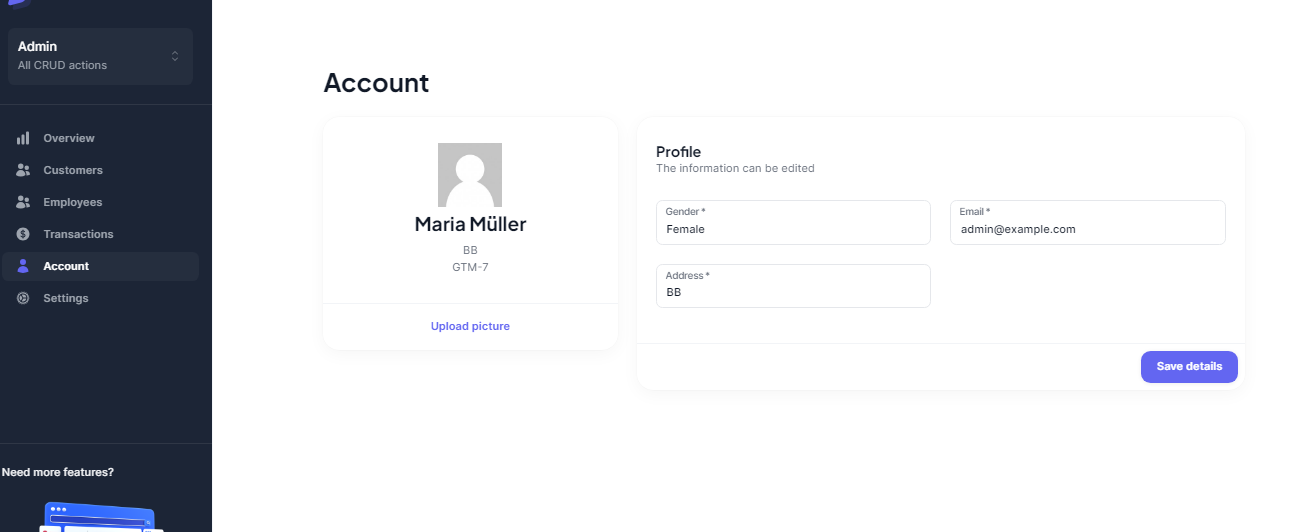
url: /transactions



If you are admin, you can do all crud action

6.6. Account management page

url: /account



Select image to upload avatar

7. Conclusion

In my IM Service project, I strived to create a comprehensive solution that catered to the specific needs of different user roles. The project offered different functionality according to user’s role, viewing, updating, adding, and deleting information such as employee, customer and transaction data . And project also used AWS S3 bucket for seamless updates of user data(avatar).

While time constraints limited the scope of the project, I focused on delivering essential core features while showcasing the system's capabilities. However, it is important to acknowledge that certain aspects of the IM Service logic were not fully realized in this project. Additionally, employing MERN Stack might not have been the optimal choice for development of excellent IM Service project.

The current implementation of the user interface was intentionally restricted to demonstrate specific actions. As such, there are opportunities for improvement in terms of UI design and functionality.

While the project encompassed login capabilities, define role, and restricted action based on user 's role, the complete enforcement of role-based access control (RBAC) remains pending. The envisioned system would empower normal users to only view datas, while manager would have additional privileges to view and edit datas. Admins, on the other hand, would have exclusive access to add, view, edit and delete informations about cutsomers, employees and transaction history.

Implementing robust security features was beyond the project's scope; however, they are vital for enterprise-level applications, particularly multifactor authentication (MFA) and single sign-on (SSO). Incorporating these features would enhance the overall security posture of the system.

Numerous areas within this implementation hold potential for improvement. Notably, bolstering security measures and adopting features more suited to a banking-related system would be paramount. This includes implementing additional authentication measures for customers opening personal or business accounts, such as conducting KYC checks and identity verification to ensure compliance. Strengthening security entails logging login IP addresses, timestamps, and employing MFA to mitigate fraudulent access attempts.

To enhance the system further, I propose the introduction of:

1. Performance Optimization: By fine-tuning algorithms and optimizing code, we can enhance the system's speed and efficiency. This may involve leveraging parallel processing, employing more advanced data structures and algorithms.
2. Scalability: To handle increasing demands and accommodate growing user bases, the system can be made more scalable. This could involve utilizing cloud-based resources to ensure smooth operations even during peak usage periods.
3. UI Enhancements: Improving the UI can greatly enhance overall system usability and user satisfaction. This may include redesigning the interface for better intuitiveness, incorporating responsive design principles for mobile devices, and conducting user testing to gather feedback for continuous improvement.
4. Security and Privacy: Strengthening security measures is crucial in today's digital landscape. The system can be fortified by implementing robust authentication mechanisms, encryption protocols, and secure coding practices. Additionally, privacy features can be enhanced, ensuring compliance with relevant regulations such as data protection laws.

Although my project had its limitations, it served as a solid foundation for further development and highlighted key areas where improvements can be made, especially in terms of security measures and features tailored to the banking domain. By addressing these aspects, the IM Service can evolve into a more robust and efficient solution.