A Project Report on

BANK BUDDY:SIMPLIFYING YOUR BANKING EXPERIENCE

Industrial Internship Project report submitted in partial fulfillment of the Requirements for the award of the degree in

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

Y.NARENDRA	**218X1A05G9**
D.KASIM BASHA	**218X1A05E1**
Y.KRISHNA REDDY	**218X1A05H9**
Y.SRI RAM BRAHMA REDDY	**218X1A05I9**

Under the Esteemed Guidance of

Dr. G. VENKATESWARA RAO MTech (Ph.D)

professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

ACCREDITED BY NAAC WITH 'A' GRADE

(APPROVED BY AICTE, AFFLIATED TO JNTUK, KAKINADA)

NH-5, CHOWDAVARAM, GUNTUR – 522019.

2021 - 2025

KALLAM HARANADHAREDDY INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

ACCREDITED BY NAAC WITH 'A' GRADE

(APPROVED BY AICTE, AFFLIATED TO JNTUK, KAKINADA)

NH-5, CHOWDAVARAM, GUNTUR - 522019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Industrial Internship Project work entitled **BANK BUDDY: SIMPLIFYING YOUR BANKING EXPERIENCE** being submitted by

Y.NARENDRA	**218X1A05G9**
D.KASIM BASHA	**218X1A05E1**
Y.KRISHNA REDDY	**218X1A05H9**
Y.SRI RAM BRAHMA REDDY	**218X1A05I9**

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering in the Kallam Haranadhareddy Institute of Technology is a record of bonafide work carried out by them.

Internal Guide

Dr. G VENKATESWARA RAO

Professor

Head of the Department

Dr. V RAJIV JETSON

Professor & HOD

External Examiner

CERTIFICATE FROM INTERN ORGANIZATION

This is to certify that YADALA NARENDRA, DUDEKULA KASIM BASHA, YARASI KRISHNA REDDY and YENUMULA SRI RAM BRAHMA REDDY Reg.No. 218X1A05G9, 218X1A05E1, 218X1A05H9 and 218X1A05I9 of Kallam Haranadhareddy Institute of Technology underwent industrial internship in SMART INTERNZ from 15-07-2024 to 02-11-2024. The overall performance of the intern during his/her internship is found to be Satisfactory.

Y.NARENDRA	**218X1A05G9**
D.KASIM BASHA	**218X1A05E1**
Y.KRISHNA REDDY	**218X1A05H9**
Y.SRI RAM BRAHMA REDDY	**218X1A05I9**

Authorized Signatory with Date and Seal

DECLARATION

We YADALA NARENDRA (218X1A0G9), DUDEKULA KASIM BASHA(218X1A05E1), YARASI KRISHNA REDDY(218X1A05H9), YENUMULA SRI RAM BRAHMA REDDY(218X1A05E1), hereby declare that the project report titled "**BANK BUDDY:SIMPLIFYING YOUR BANKING EXPERIENCE**" under the guidance of **Dr.G. VENKATESWARA RAO** is submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering.

This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project have not been submitted to any other university for the award of any degree.

Y.NARENDRA	**218X1A05G9**
D.KASIM BASHA	**218X1A05E1**
Y.KRISHNA REDDY	**218X1A05H9**
Y.SRI RAM BRAHMA REDDY	**218X1A05I9**

ACKNOWLEDGEMENT

We profoundly grateful to express our deep sense of gratitude and respect towards **Sri. KALLAM MOHAN REDDY**, **Chairman, KHIT**, **Guntur** for his precious support in the college**.**

We are thankful to **Dr. M. UMA SANKAR REDDY**, Director, KHIT, Guntur for his encouragement and support for the completion of the project.

We are much thankful to **Dr. B. SIVA BASIVI REDDY**, Principal, KHIT, Guntur for his support during and until the completion of the project.

We are greatly indebted to **Dr. V RAJIV JETSON, M.Tech, Ph.D** Professor &Head of the department, Computer Science and Engineering, KHIT, Guntur for providing the laboratory facilities fully as and when required and for giving us the opportunity to carry the project work in the college during Industry Internship.

We are also thankful to our Project Coordinator **Mr. G MANTHRU NAIK** who helped us in each step of our Project.

We extend our deep sense of gratitude to our Internal Guide **Dr.G. VENKATESWARA RAO**, Assistant Professor and other Faculty Members & Support staff for their valuable suggestions, guidance and constructive ideas in each and every step, whichwas indeed of great help towards the successful completion of our project.

Y.NARENDRA	**218X1A05G9**
D.KASIM BASHA	**218X1A05E1**
Y.KRISHNA REDDY	**218X1A05H9**
Y.SRI RAM BRAHMA REDDY	**218X1A05I9**

ABSTRACT

This project involves the development of a MERN stack-based banking system designed to streamline essential banking operations for both administrators and users. The system features two main roles: Admin and User. Admins can log in to manage various functionalities, including adding and viewing users, reviewing loan applications, updating loan statuses, and logging out. Users can register, log in, and perform essential financial transactions such as crediting, debiting, and transferring funds. Additionally, users have access to their bank statements, can apply for loans, view loan statuses, manage their profiles, and log out. By leveraging the power of the MERN stack (MongoDB, Express.js, React.js, and Node.js), the application ensures a robust, user-friendly platform for managing banking activities efficiently. This project aims to enhance the digital banking experience by providing seamless financial services and efficient user management through a single integrated system

Keywords: Admin, User, MERN

TABLE OF CONTENTS

Contents		Page No
Certificate		iii
Declaration		vi
Acknowledgement		vii
Abstract		viii
Table of Content		
List of Figures		
1. INTRODUCTION		**01-03**
1.1.	Project Idea	02
1.2.	Motivation of the Project	02
1.3.	Problem Statement	02
1.4.	Statement of scope	03
1.5.	Goals and objectives	03
2. LITERATURE SURVEY		**04-07**
2.1.	Literature Survey	04-05
2.2.	Existing System	06
2.3.	Disadvantages of the Existing System	06
2.4.	Feasibility Study	06-07
3. PROPOSED SYSTEM		**08-10**
3.1.	Proposed System	09
3.2.	Methodology	09-10
3.3.	Advantages	10
3.4.	Approaches	10-11
4. SYSTEM REQUIREMENTS SPECIFICATION		12-16
4.1.	Software Requirements	13
4.2.	Hardware Requirements	13
4.3.	Functional Requirements	13-14

4.4 4.4 Non-Functional Requirements

4.5 Technologies used 16

5.SYSTEM DESIGN 17-25

5.1. System Architecture 18

5.2. Module Description 18-19

5.3. E-R Diagram 19-20

5.4 Data Flow Diagram 20-21

5.5 UML Diagrams 22-25

5.5.1 Class Diagram 22

5.5.2 Sequence Diagram 22-23

5.5.3 Activity Diagram 23-24

5.6 Use Case Diagram 25

6 IMPLEMENTATION 26-45

6.1 Implementation 27

6.1.1 Admin Module 27

6.1.2 User Module 27

6.2 Code 28-45

7 SYSTEM TESTING 46-53

7.1 Testing Introduction 47

7.2 Types of Testing 47

7.2.1 Unit testing 47

7.2.2 Integration testing 48

7.2.3 Functional testing 48-49

7.2.4 White Box Testing 49

7.2.5 Black Box Testing 49

7.3 Test Cases 50-53

8 OUTPUT SCREENS **54-59**

9 CONCLUSION **60-61**

9.1 Future Enhancement 61

10 REFERENCES **62-63**

LIST OF FIGURES

Contents	Page No
Fig 5.1: Architecture	18
Fig 5.3: E-R Diagram	20
Fig 5.4.1: DFD Diagram(Level-1)	21
Fig 5.4.2: DFD Diagram(Level-2)	21
Fig 5.5.1: Class Diagram	22
Fig 5.5.2: Sequence Diagram	23
Fig 5.5.3: Activity Diagram	24
Fig 5.6:Use Case Diagram	25
Fig 8.1:Home Page	55
Fig 8.2:About Page	55
Fig 8.3:Service Page	56
Fig 8.4:Admin Login Page	56
Fig 8.5:Add Users Page	57
Fig 8.6:View Users Page	57
Fig 8.7:Admin Dashboard	58
Fig 8.8:User Signup page	58
Fig 8.9:User Login Page	59
Fig 9.0:Loan Request Page	59

LIST OF TABLES

Contents	Page No
Table 1: Test Cases	50-53

CHAPTER-1

INTRODUCTION

1.1 PROJECT IDEA

The rapid evolution of digital banking has transformed how financial services are accessed and managed. This project presents the development of a full-fledged banking system using the MERN stack (MongoDB, Express.js, React.js, and Node.js), designed to streamline essential banking operations for both administrators and users. The system is built with two distinct roles: Admin and User.

Admins are responsible for overseeing and managing users, reviewing and updating loan applications, and maintaining the overall functionality of the platform. Users, on the other hand, can register and log into perform critical financial transactions such as crediting, debiting, and transferring funds. Additionally, users can access their bank statements, apply for loans, manage loan statuses, and update their profiles.

This project aims to enhance the digital banking experience by providing a user-friendly interface and efficient, secure management of banking activities. By leveraging the power of the MERN stack, the system ensures scalability, flexibility, and reliability in handling both administrative and user-level banking operations**.**

1.2 MOTIVATION

The motivation behind developing this system is to address the limitations of traditional banking processes, which are often time-consuming and inconvenient for users. By offering a digital solution, the system aims to simplify banking operations, improve user experience, and provide secure and accessible financial services to all users.

1.3 PROBLEM STATEMENT

The current manual banking processes are inefficient, error-prone, and inconvenient for users. There is a need for a digital banking platform that provides secure, efficient, and user-friendly services, enabling users to perform financial transactions, apply for loans, and manage their accounts seamlessly from anywhere.

1.4 SCOPE

The scope of the proposed system includes developing a web-based platform for digital banking services, supporting both user and admin roles. It covers secure financial transactions, loan management, user profile management, and real-time status updates. The system aims to enhance overall banking efficiencyand user satisfaction

1.5 OBJECTIVE OF THE PROJECT

The objective of the proposed system is to provide a secure, efficient, and user-friendly digital banking platform. It aims to streamline financial transactions, improve loan management, and enhance user convenience by integrating key banking features within a single, accessible application.

CHAPTER-2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

Related Work

1. **Author**: Akhavan, N., & Jafari, H.

   Title:A Comparative Study of Web-Based Banking Systems

   Outcome:

   The study highlights the transformation of traditional banking systems into web-based platforms, showcasing improved customer convenience, real-time access to banking services, and reduced operational costs. It emphasizes the importance of digital platforms in enhancing customer Satisfaction and operational efficiency.

   Disadvantage:

   The paper points out that transitioning to web-based systems can be costly for smaller banks and may face resistance from customers unfamiliar with digital banking technologies.

2. **Author**: Bennett, M., & Chin, S.

   Title:A Comparative Study of Web-Based Banking Systems

   Outcome:

   This research identifies security vulnerabilities in digital banking systems and provides recommendations for enhancing security, including encryption, multi-factor authentication, and secure session management. It demonstrates the critical importance of robust security protocols in the development of online banking application.

   Disadvantage:

   The study notes that implementing high-level security measures can lead to a more complex system architecture, potentially slowing down transaction processes and increasing development costs

2.1 Existing System:

The existing system relies on manual processes for handling banking activities, such as visiting branches for transactions and record-keeping. These processes are time-consuming, prone to human error, and lack remote accessibility, making banking inconvenient for users who seek quick and efficient services.

2.2 Disadvantages:

- Time-consuming and inefficient, requiring users to visit physical branches for transactions.
- Prone to errors due to manual data entry and record-keeping.
- Limited accessibility, making it challenging for users to manage their accounts remotely.

2.3 Feasibility Study

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economic feasibility
- Technical feasibility
- Social feasibility

Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system

CHAPTER-3

PROPOSED SYSTEM

3.1 Proposed System:

The proposed system is a MERN stack-based banking application that provides an integrated platform for users and administrators. Users can register, perform financial transactions (credit, debit, transfer), apply for loans, and view loan statuses. Admins can add and manage users, view and update loan statuses. The system aims to offer a seamless and efficient digital banking experience, enhancing user convenience and supporting effective management of financial activities.

3.2 METHODOLOGY

MODULES AND FUNCTIONALITY

ADMIN MODULE

1) **Login :** Here Admin will do login with default credentials to access the admin dashboard
2) **Add Users:** Here admin can create user and generate the account number for each user
3) **View Users:**All registered users in BankBuddy can be able to see the users
4) **View Loans:**User Applied for the loans that admin can view
5) **Change the status:** Can change the status of loan process
6) **Logout:** After performing all operations admin can do logout

USER MODULE

1) **Register:** User need to register for the portal , can do registration or admin can add the user
2) **Login:** with given valid credentials user will do login and can access the user dashboard
3) **Credit:**Here user is crediting the self account balance
4) **Debit:** User can withdraw the amount and that can be subtracted from the main balance
5) **Transfer amount:**Logged in user transferring the amount to registered user account
6) **bank statement:** User can view the entire bank statement which he made till the date
7) **Apply for Loan:**Here user applying for loan that amount creding to the account balance after approval
8) **View loan status:**User gets the update here whether loan processed or rejected
9) **Profile:**User views all his personal details here which he entered during Registration
10) **Logout:**After all operations user can do logout securely

3.3 Advantages:

- User-friendly interface for easy navigation and interaction.
- Real-time financial transactions and loan application processing.
- Secure platform with role-based access for users and admins.
- Centralized management of user accounts and financial activities.
- Efficient loan application and status tracking for better user experience.

3.4 APPROACHES

1. Streamline User Interfaces

Unified Dashboard: Consolidate all financial products (accounts, loans, investments) into one easy-to-navigate dashboard, where users can view balances, transactions, and reports in a single view.
Minimalist Design: Simplify digital banking apps with clear icons, easy navigation, and intuitive workflows.
Customizable Features: Allow users to personalize their app layout based on frequently used features.

2. Enhance Customer Service with AI and Chatbots

24/7 Support: Implement AI-driven chatbots that can answer common banking questions, assist with simple transactions, and provide guidance on using digital services.

Proactive Assistance: AI can monitor user patterns and notify customers about unusual spending, payment due dates, or saving opportunities.

Multi-Channel Support: Ensure consistent support through all channels (app, website, social media, phone), so customers have a seamless experience.

3. Improve Transaction Processes

One-Click Transfers: Implement faster transfer options with fewer steps for intra-bank and inter-bank transfers.

Automatic Bill Pay and Reminders: Enable features that automatically handle recurring bill payments and alert users of upcoming payments.

Instant Fund Availability: Utilize real-time payment systems to reduce the lag between deposits and availability in the account.

4. Optimize Account Onboarding

Digital KYC: Use video verification or digital ID checks to streamline the onboarding process, enabling account setup within minutes.

Pre-Filled Forms: Allow customers to pre-fill application forms by integrating with digital ID databases, minimizing the manual input of details.

5. Focus on Security and Trust

Biometric Authentication: Use fingerprint, facial recognition, or voice recognition to simplify login while enhancing security.

Multi-Factor Authentication: Educate users about the importance of MFA and make it a standard for large transactions.

Real-Time Fraud Alerts: Implement instant notifications for any suspicious transactions with a quick one-click option to report them.

CHAPTER-4

SYSTEM REQUIREMENTS

AND SPECIFICATION

4.1 SOFTWARE REQUIREMENTS

SOFTWARE SYSTEM CONFIGURATION:

- Operating System : Windows 7/8/10
- Server side Script : Express js
- Programming Language : TypeScript
- IDE/Workbench : VS Code
- Database : Mongodb
- Clint Side : React js

4.2HARDWARE REQUIREMENTS:

- Processor - I3/Intel Processor
- RAM - 4GB (min)
- Hard Disk - 160GB

4.3 Functional Requirements:

Admin module

2.3.1 **Login :** Here Admin will do login with default credentials to access the admin dashboard

2.3.2 **Add Users:** Here admin can create user and generate the account number for each user

2.3.3 **View Users:**All registered users in BankBuddy can be able to see the users

2.3.4 **View Loans:**User Applied for the loans that admin can view

5 **Change the status:** Can change the status of loan process

6 **Logout:** After performing all operations admin can do logout

User module

1. **Register:** User need to register for the portal , can do registration or admin can add the user
2. **Login:** with given valid credentials user will do login and can access the user dashboard
3. **Credit:**Here user is crediting the self account balance
4. **Debit:** User can withdraw the amount and that can be subtracted from the main balance

5. **Transfer amount:**Logged in user transferring the amount to registered user account
6. **bank statement:** User can view the entire bank statement which he made till the date
7. **Apply for Loan:**Here user applying for loan that amount creding to the account balance after approval
8. **View loan status:**User gets the update here whether loan processed or rejected
9. **Profile:**User views all his personal details here which he entered during Registration
10. **Logout:**After all operations user can do logout securely.

4.4 Non-Functional Requirements

1.Performance:

The system should be able to handle at least 100 concurrent users without a noticeable decrease in response time.Transaction processing (crediting, debiting, transferring funds) should be completed within 2 seconds under normal load conditions.

2.Security

All user data, including login credentials and financial information, must be encrypted both in transit and at rest.The system must implement role-based access control to ensure that only authorized users can access specific functionalities (e.g., Admin functionalities should not be accessible to regular users).

3.Scalability:

The system should be designed to scale horizontally, allowing for the addition of more servers to handle increased load as the user base grows.

4.Usability:

The user interface should be intuitive and user-friendly, enabling users to complete essential banking operations with minimal training or guidance.User feedback mechanisms (e.g., tooltips, help icons) should be incorporated to assist users in navigating the system.

5. **Availability:**

The system should ensure 99.9% uptime, allowing users to access banking services at any time, excluding scheduled maintenance windows.

6. **Maintainability:**

The codebase should follow best practices for modular design, making it easy to update or add new features without significant changes to the existing code.

7. **Compliance:**

The system must comply with relevant banking regulations and standards (e.g., PCI DSS for payment processing) to ensure the security and privacy of user financial data.

8. **Backup and Recovery:**

The system should implement regular backup procedures to ensure that user data can be restored in case of data loss, with a recovery time objective (RTO) of less than 4 hours.

4.4 Languages and Technologies used:

- HTML, CSS
- React Bootstrap (An HTML, CSS, and JS library)
- Material UI
- JavaScript, Express.js
- MongoDB

CHAPTER-5

SYSTEM DESIGN

5.1 ARCHITECTURE DIAGRAM

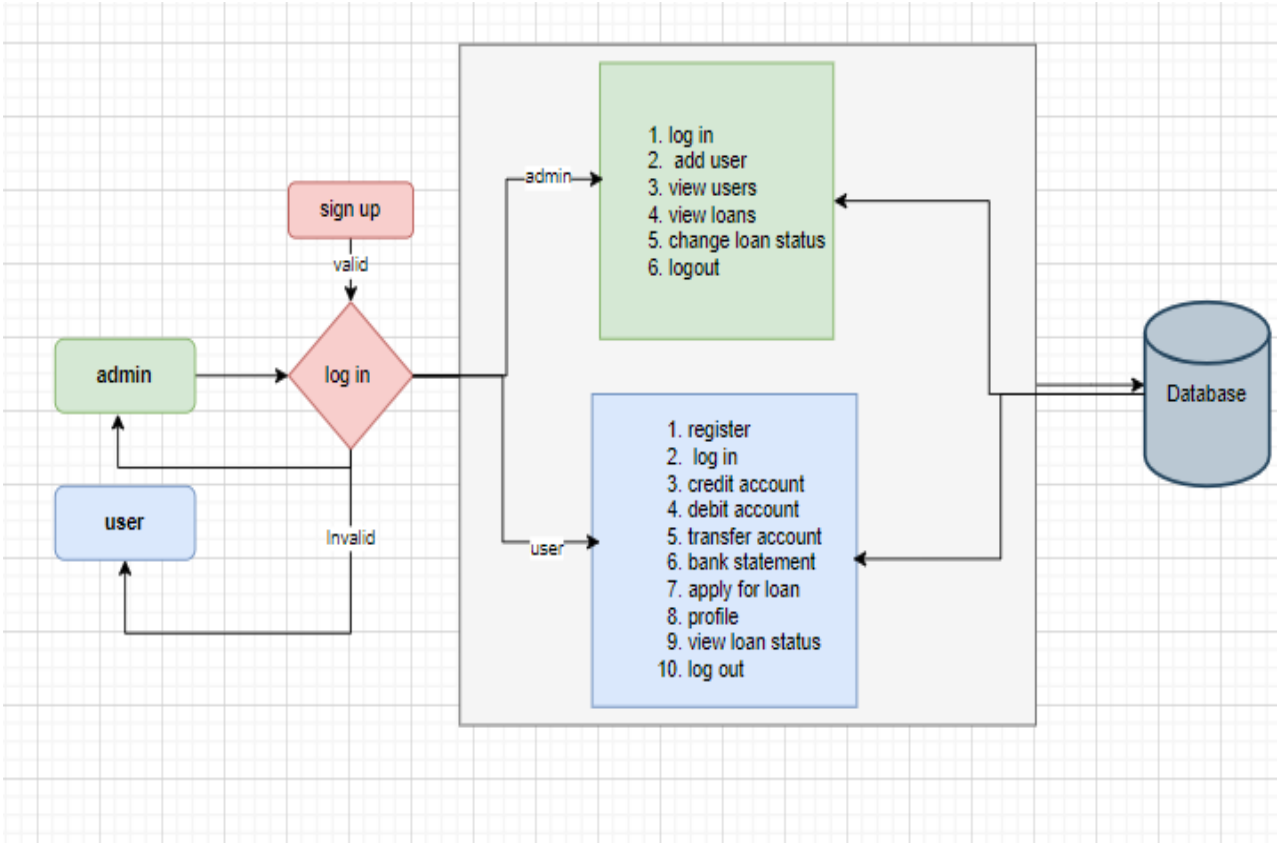


FIG 5.1 Architecture Diagram

MODULE DESCRIPTION

5.2 ADMIN MODULE

7) **Login :** Here Admin will do login with default credentials to access the admin dashboard
8) **Add Users:** Here admin can create user and generate the account number for each user
9) **View Users:**All registered users in BankBuddy can be able to see the users
10) **View Loans:**User Applied for the loans that admin can view
11) **Change the status:** Can change the status of loan process
12) **Logout:** After performing all operations admin can do logout

USER MODULE

1. **Register:** User need to register for the portal , can do registration or admin can add the user
2. **Login:** with given valid credentials user will do login and can access the user dashboard

3. **Credit:**Here user is crediting the self account balance
4. **Debit:** User can withdraw the amount and that can be subtracted from the main balance
5. **Transfer amount:**Logged in user transferring the amount to registered user account
6. **bank statement:** User can view the entire bank statement which he made till the date
7. **Apply for Loan:**Here user applying for loan that amount creding to the account balance after approval
8. **View loan status:**User gets the update here whether loan processed or rejected
9. **Profile:**User views all his personal details here which he entered during Registration
10. **Logout:**After all operations user can do logout securely

5.3 ER Diagram:

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of ER model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Let’s have a look at a simple ER diagram to understand this concept.

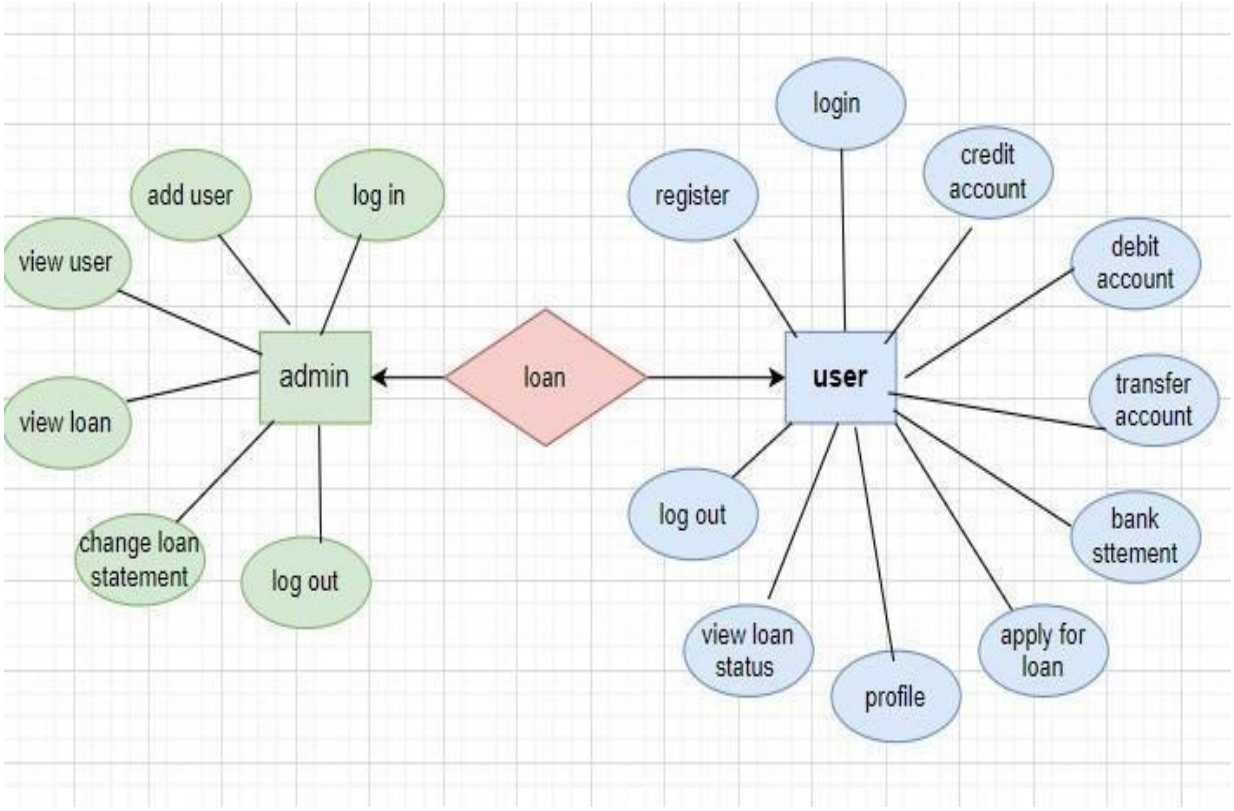


FIG-5.3 E-R DIAGRAM

5.4 DATA FLOW DIAGRAM(DFD)

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

LEVEL-1 DIAGRAM

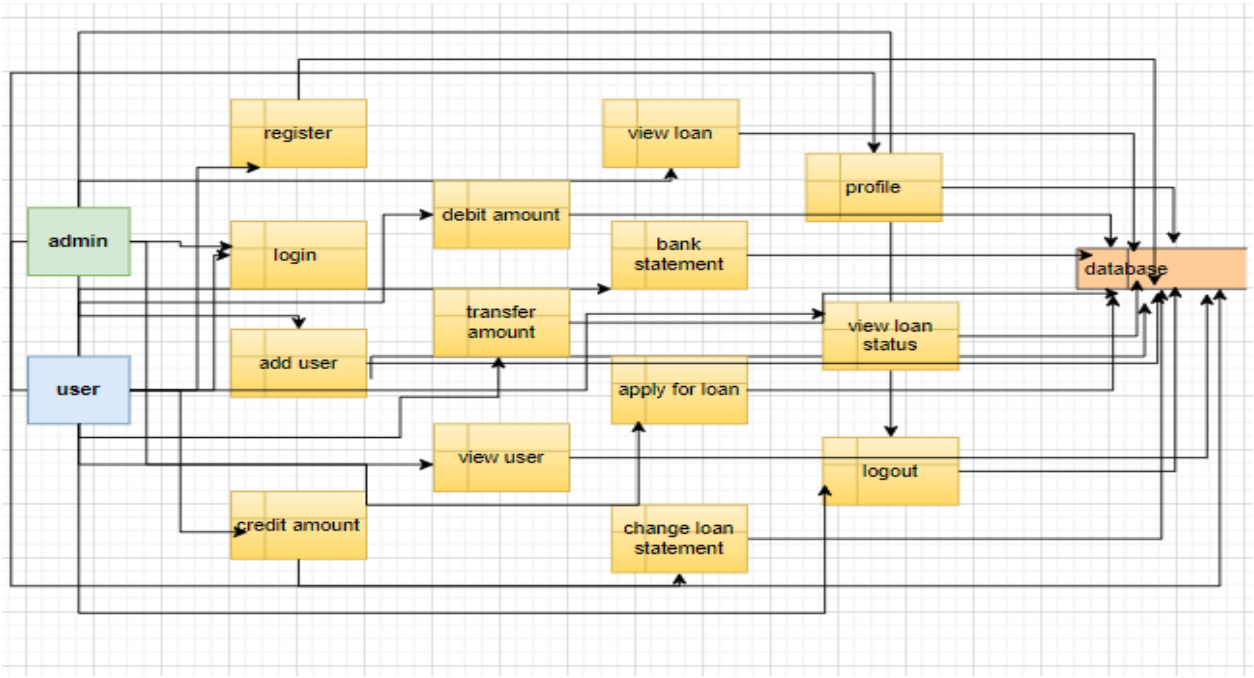


FIG 5.4.1 LEVEL-1 DFD Diagram

LEVEL-2 DIAGRAM

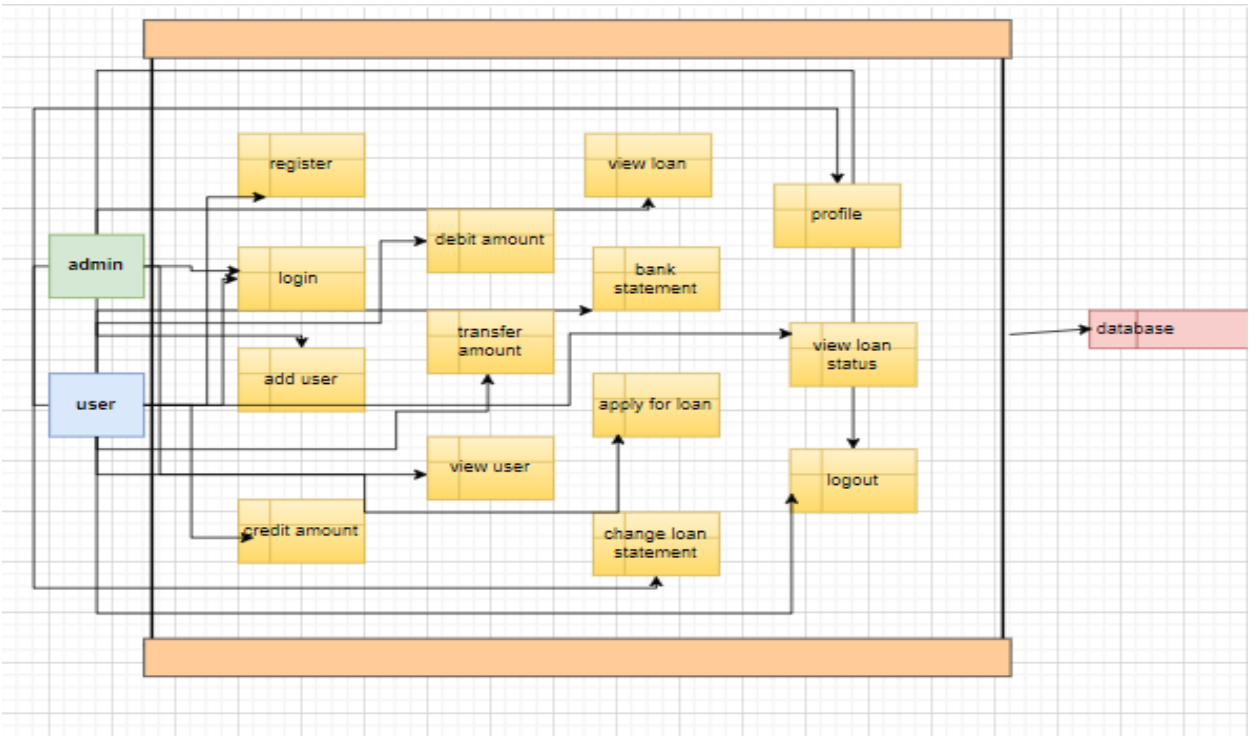


FIG-5.4.2 LEVEL-2 DFD Diagram

5.5 UML DIAGRAM

5.5.1 Class Diagram:

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

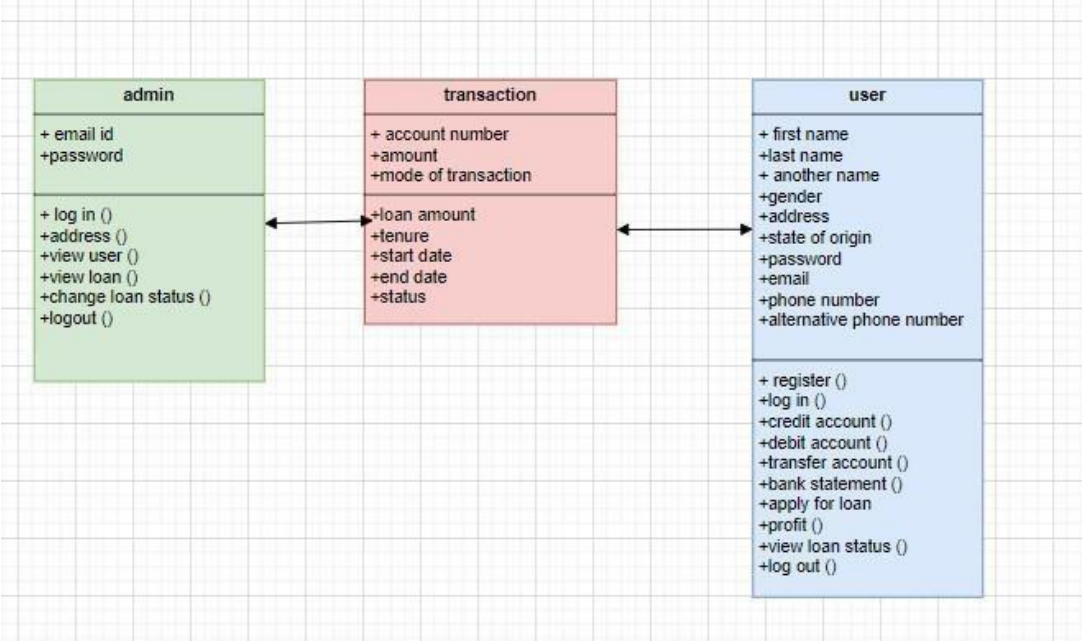


FIG 5.5.1 Class Diagram

5.5.2 Sequence Diagram

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram thatshows how processes operate with one another and in what order. It is a construct of a Message SequenceChart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

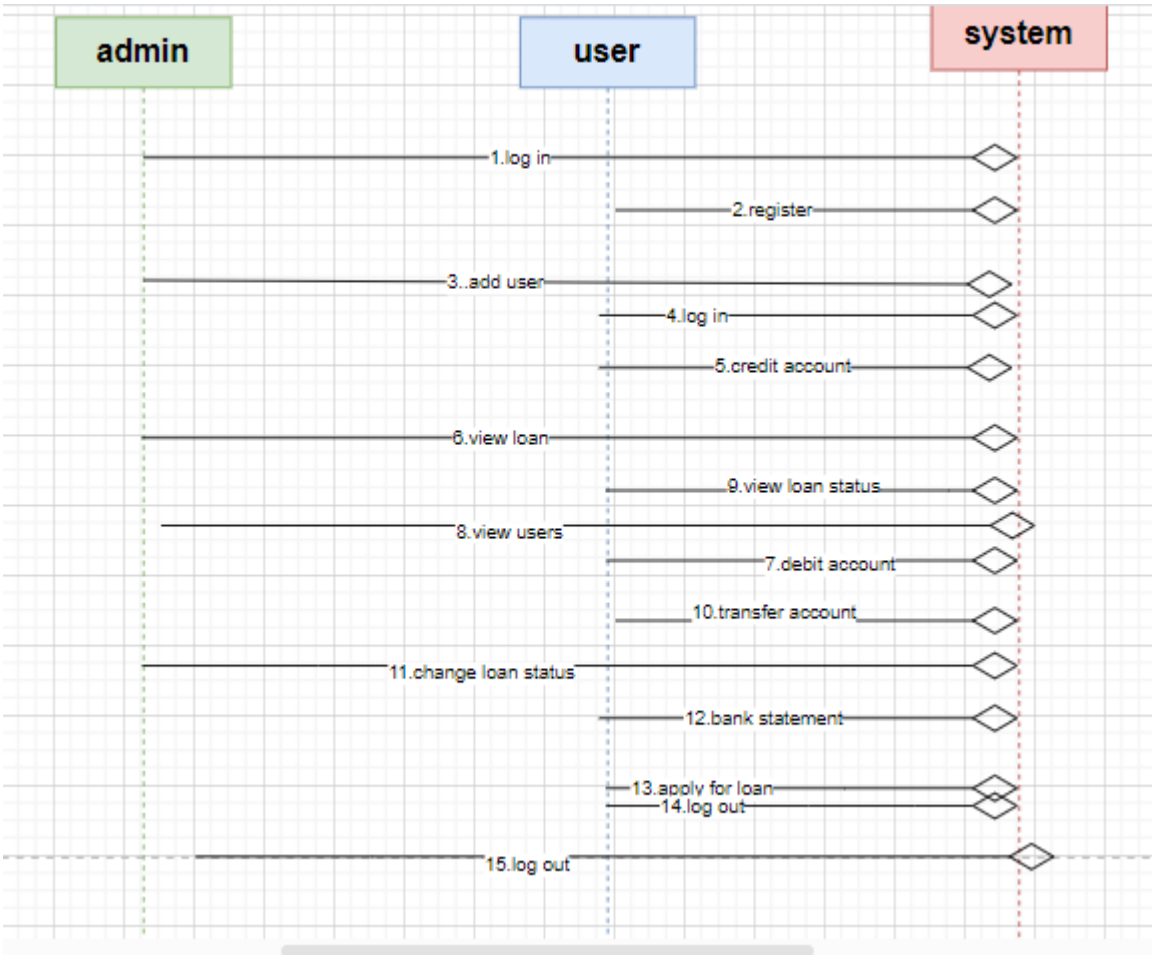


FIG 5.5.2-Sequence Diagram

5.5.3 Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational stepbystep workflows of components in a system. An activity diagram shows the overall flow of control.

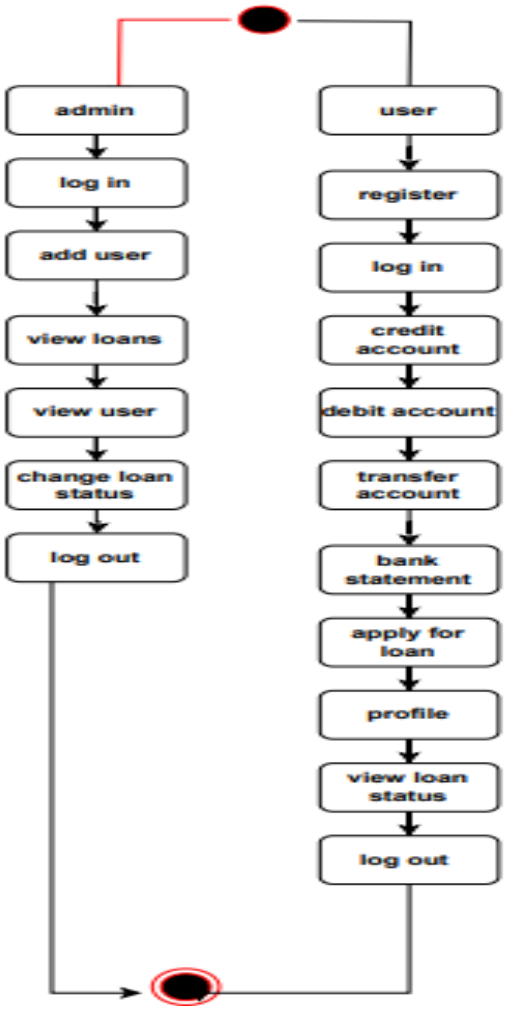


FIG 5.5.3-Activity Diagram

5.6 Use Case Diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Usecase analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

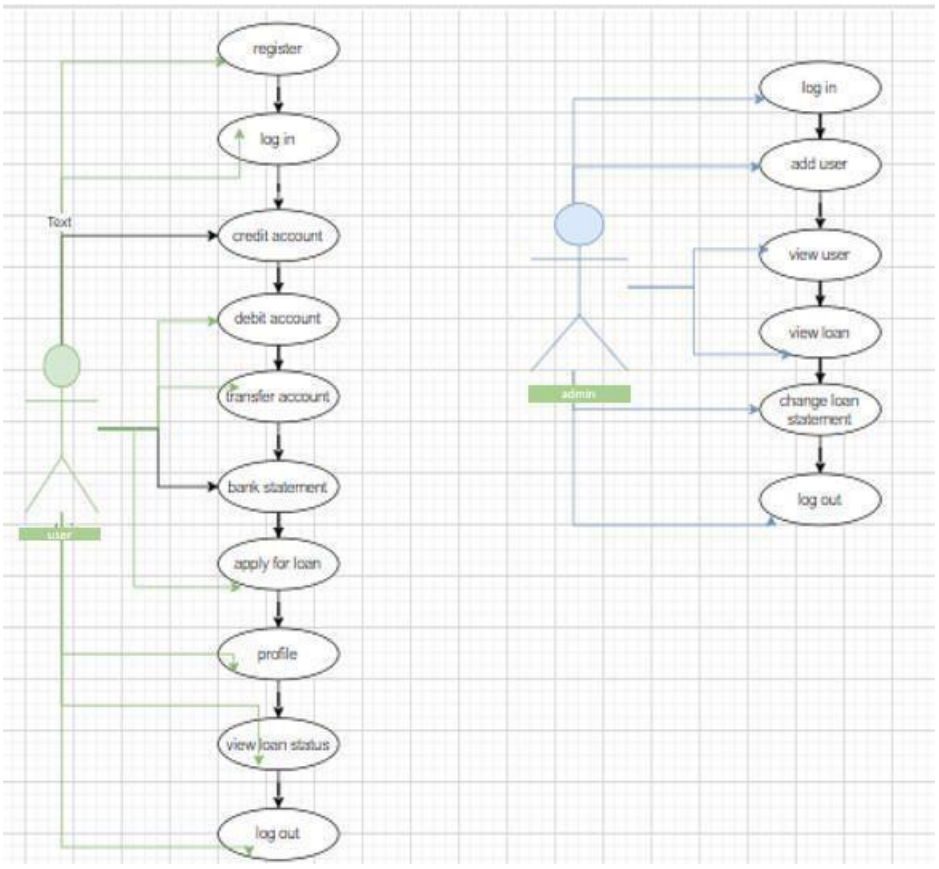


FIG 5.6-Use Case Diagram

CHAPTER-6

IMPLEMENTATION AND CODING

6. 1 IMPLEMENTATION

6.1.1ADMIN MODULE

13) **Login :** Here Admin will do login with default credentials to access the admin dashboard

14) **Add Users:** Here admin can create user and generate the account number for each user

15) **View Users:**All registered users in BankBuddy can be able to see the users

16) **View Loans:**User Applied for the loans that admin can view

17) **Change the status:** Can change the status of loan process

18) **Logout:** After performing all operations admin can do logout

6.1.2 USER MODULE

11) Register: User need to register for the portal , can do registration or admin can add the user

12) Login: with given valid credentials user will do login and can access the user dashboard

13) Credit: Here user is crediting the self account balance

14) Debit: User can withdraw the amount and that can be subtracted from the main balance

15) Transfer amount: Logged in user transferring the amount to registered user account

16) bank statement: User can view the entire bank statement which he made till the date

17) Apply for Loan: Here user applying for loan that amount creding to the account balance after approval

18) View loan status: User gets the update here whether loan processed or rejected

19) Profile: User views all his personal details here which he entered during Registration

20) Logout: After all operations user can do logout securely

6.2 Coding

ADMIN CODE

```
 import { Request, Response } from "express";
import { matchedData } from "express-validator";
import { Account, Admin, DebitCard, User } from "../schema/usersSchema";
import {
  comparePassword,
  hashPassword,
  sendDebitCardRejectEmail,
  sendDebitCardStatusEmail,
} from "../utils/helpers";
import { adminTokens } from "../utils/tokenGeneration";
import { verifyAdminToken } from "../middlewares/verifyTokens";
import { getDateIST, getTimeIST } from "../utils/getIstDateTime";

// Admin registering
export const registerAdmin = async (
  req: Request,
  res: Response
): Promise<any> => {
  const data = matchedData(req);
  const { username, password, email } = data;
  if (!username || !password || !email) {
    return res
      .status(401)
      .send({ msg: "username and password and email are required" });
  }
  try {
    const user = await Admin.findOne({ username });
    if (user) return res.status(401).send({ msg: "User already exists" });

    const hashedPassword = hashPassword(password);

    const newUser = new Admin({
      username,
      password: hashedPassword,
      email,
    });

    const savedUser = await newUser.save();
    return res.status(201).send({
      msg: `Registered Successful  .Pls login ${savedUser.username}`,
```

```
    });
  } catch (err: any) {
    return res.status(400).send({ error: err });
  }
};

// Login the Admin
export const adminLogin = async (req: Request, res: Response): Promise<any> => {
  try {
    const body = req.body;
    const pass = body.password;

    const findUser = await Admin.findOne({
      $or: [{ username: body.username }, { email: body.email }],
    });

    if (!findUser) throw new Error("Admin not found");
    if (findUser.role !== 1) {
      throw new Error("Unauthorized: Admin access required");
    }
    if (!comparePassword(pass, findUser.password))
      throw new Error("Bad Credentials");
    const finalResult = findUser.toObject();
    const { accessToken } = adminTokens(findUser);
    const { password, ...admin } = finalResult;

    return res.send({
      msg: "Login Successful",
      admin,
      accessToken,
      role: findUser.role,
    });
  } catch (err: any) {
    return res.status(400).send({ error: err.message });
  }
};

// Admin can view All users
export const viewAllUsers = async (
  req: Request,
  res: Response
): Promise<any> => {
  try {
    const result = await User.find({}).select("-password");
    res.send(result);
  } catch (err: any) {
    return res.status(400).send({ error: err.message });
  }
```

```
};

// Get All the Debit Card Details
export const getAllPendingDebitCards = async (
  req: Request,
  res: Response
): Promise<any> => {
  try {
    const debitCardData: any = await DebitCard.find({
      status: "pending",
    }).select(["-updatedAt"]);
    if (debitCardData.length === 0) {
      return res.send({ msg: "Till Now No one can Apply for Debit Cards" });
    }

    // Fetch user details for each pending debit card
    const debitCardDetailsWithUsers = await Promise.all(
      debitCardData.map(async (debitCard: any) => {
      console.log(debitCard);
        const user = await User.findById(debitCard.userid).select([
          "firstname",
          "lastname",
          "email",
        ]);
        const accountDetails: any = await Account.findOne({
          accUser: debitCard.userid,
        });
        return {
          ...debitCard.toObject(),
          email: user?.email,
          accNumber: accountDetails.accNumber,
          date: getDateIST(debitCard.createdAt),
          time: getTimeIST(debitCard.createdAt),
          accBalance: accountDetails.accBalance,
        };
      })
    );

    res.send(debitCardDetailsWithUsers);
  } catch (err: any) {
    console.error("Error:", err);
    return res.status(400).send({ error: err.message });
  }
};

const generateUniqueCardNumber = async () => {
  let cardNumber;
  let isUnique = false;
```

```
  while (!isUnique) {
    cardNumber  =
    "4" +
      Math.floor(Math.random() * 1000000000)
        .toString()
        .padStart(11, "0");

    const existingCard = await DebitCard.findOne({ cardNumber });
    if (!existingCard) {
      isUnique = true;
    }
  }

  return cardNumber;
};
// Function to generate a unique CVV
const generateUniqueCVV = async () => {
let cvv;
  let isUnique = false;

  while (!isUnique) {
    cvv = Math.floor(100 + Math.random() * 900).toString();

    const existingCard = await DebitCard.findOne({ cvv });
    if (!existingCard) {
      isUnique = true;
    }
  }

  return cvv;
};

const generateExpiryDate = async () => {
  const month = Math.floor(Math.random() * 12) + 1;
  const year = new Date().getFullYear() + Math.floor(Math.random() * 5);
  let expiryDate = `${month.toString().padStart(2, "0")}/${year}`;
  return expiryDate;
};

// Admin can approve or reject the debit card application
export const reviewDebitCardApplication = async (
  req: Request,
  res: Response
): Promise<any> => {
  const { debitid, status } = req.body;
  try {
    const debitCardApplication: any = await DebitCard.findOne({ _id: debitid });
```

```
    if (!debitCardApplication) {
      return res.status(404).send({ msg: "Application not found" });
    }
    const debitCardAccount: any = await User.findOne({
      _id: debitCardApplication.userid,
    });
    if (debitCardApplication) {
      let name = debitCardApplication.cardHolder;
      let email = debitCardAccount?.email;
      sendDebitCardStatusEmail(name, email, status);
    }
    if (status === "accepted") {
      const cardNumber = await generateUniqueCardNumber();
      const cvv = await generateUniqueCVV();
      const expiryDate = await generateExpiryDate();
      debitCardApplication.status = status;
      debitCardApplication.cardNumber = cardNumber;
      debitCardApplication.cvv = cvv;
      debitCardApplication.expiryDate = expiryDate;
      const savedCard = await debitCardApplication.save();
      console.log(savedCard, " savedcard ");

      return res.send({ msg: "Debit Card Approved", debitCardApplication });
    } else if (status === "rejected") {
      debitCardApplication.status = status;
      const rejectedStatus = await debitCardApplication.save();
      if (rejectedStatus) {
        let name = debitCardApplication.cardHolder;
        let email = debitCardAccount?.email;
        sendDebitCardRejectEmail(name, email, status);
      }
      return res.send({ msg: "Debit Card Rejected", debitCardApplication });
    } else {
      return res.status(400).send({ msg: "Invalid action" });
    }
  } catch (err: any) {
    console.error("Error:", err);
    return res.status(400).send({ error: err.message });
  }
};
```

USER CODE

```
import {
  Account,
  Credit,
  Debit,
  DebitCard,
  Transfer,
  User,
} from "../schema/usersSchema";
import { Request, Response } from "express";
import {
  comparePassword,
  hashPassword,
  sendAppliedDebitCardEmail,
  sendCreditEmail,
  sendDebitEmail,
  sendRegistrationEmail,
  sendTransferEmail,
} from "../utils/helpers";
import { userTokens } from "../utils/tokenGeneration";
import { verifyUserToken } from "../middlewares/verifyTokens";
        // User registering
export const registerUser = async (
  req: Request,
  res: Response
): Promise<any> => {
  const {
    firstname,
    lastname,
    otherName = null,
    gender,
    address, stateofOrigin,

    password.
```

```
import QRCode from "qrcode";

// Function to generate a unique bank account number
const generateAccountNumber = async () => {
  let accountNumber;
  let isUnique = false;

  while (!isUnique) {
    // Generate a random 10-digit account number
    accountNumber =
      "100" +
      Math.floor(Math.random() * 1000000000)
        .toString()
        .padStart(9, "0");

    const existingAccount = await Account.findOne({ accNumber: accountNumber });
    if (!existingAccount) {

      isUnique = true;
    }
  }

  return accountNumber;-
};

// User registering
export const registerUser = async (
  req: Request,
  res: Response
): Promise<any> => {
  const {
    firstname,
    lastname,
    otherName = null,
    gender,
    address,
    stateofOrigin,
```

```
  password,
  email,
  phoneNumber,
  alternativePhoneNumber = null,
 } = req.body;

 try {
  const user = await User.findOne({ email });

  if (user) return res.status(401).send({ msg: "User already exists" });

  const hashedPassword = hashPassword(password);

  const newUser = new User({
   firstname,
   lastname,
   otherName,
   gender,
   address,
   stateofOrigin,

   password: hashedPassword,
   email,
   phoneNumber,
   alternativePhoneNumber,
  });

  const savedUser = (await newUser.save()).toObject();
  if (savedUser) {
   let name = firstname + " " + lastname;
   sendRegistrationEmail(email, name);
  }
  // Generate unique account number
  const accNumber = await generateAccountNumber();

  // Create account for the user
  const newAccount = new Account({
```

```
      accNumber,
      accUser: savedUser._id,
      accHolder: `${firstname} ${lastname}`,
      accBalance: 0,
    });

    const savedData = (await newAccount.save()).toObject();

    return res.status(201).send({
      msg: `Registered Successfully ✅. Please login ${savedUser.firstname}
      ${savedUser.lastname}`, account: {
        accNumber: newAccount.accNumber,
        accHolder: newAccount.accHolder,
      },
    });
  } catch (err: any) {
    return res.status(400).send({ error: err });
  }
};
// Login the User
export const userLogin = async (req: Request, res: Response): Promise<any> => {
  try {
    const body = req.body;

    const pass = body.password;
    const findUser = await User.findOne({ email: body.email });

    if (!findUser) throw new Error("User not found");
    if (findUser.role !== 2) {
      throw new Error("Unauthorized: User access required");
    }

    if (!comparePassword(pass, findUser.password))
      throw new Error("Bad Credentials");

    const { password, ...user } = findUser.toObject();
    const { accessToken } = userTokens(findUser);
```

```
    return res.send({
      msg: "Login Successful",
      user,
      accessToken,
      role: findUser.role,
    });
  } catch (err: any) {
    console.log(err);
    return res.status(400).send({ error: err.message });
  }
};

// Credit the Amount
export const creditAmount = async (
  req: Request,
  res: Response
): Promise<any> => {
  const amount = Number(req.body.amount);
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }
  const token = authHeader.split(" ")[1];

  try {
    const decoded: any = verifyUserToken(token);

    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });

  res: Response
): Promise<any> => {
  const amount = Number(req.body.amount);
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }
```

```
const token = authHeader.split(" ")[1];

try {
  const decoded: any = verifyUserToken(token);
  if (!decoded) {
    return res.status(401).send({ msg: "Access Token is Invalid" });
  }

  const userid = decoded.id;
  const getAccount: any = await Account.findOne({ accUser: userid });
  if (amount > getAccount.accBalance) {
    return res.send({ msg: `Your account balance is insufficient` });
  }
  getAccount.accBalance -= amount;
  const savedAccount = await getAccount.save();
  const debitedTransaction = await Debit.create({
  accountNumber: getAccount.accNumber,
  amount: amount,
    accUser: getAccount.accUser,
  });
  const date = getDateIST(debitedTransaction?.date);
  const time = getTimeIST(debitedTransaction?.date);
  const userDetails = await User.findOne({ _id: userid });
  if (savedAccount) {
    let userFullName = userDetails?.firstname + " " + userDetails?.lastname;
    let useremail: any = userDetails?.email;
    let dateTime = date + " " + time;
    let balance = savedAccount?.accBalance;
    let accountnumberis = savedAccount?.accUser;

    let transactionID = debitedTransaction?._id;
    sendDebitEmail(
      userFullName,

      useremail,
      amount,
      dateTime,
```

```
        balance,
        accountnumberis,
        transactionID
      );
    }
    return res.send({ msg: "Debited Amount Successfully", savedAccount });
  } catch (err: any) {
    console.log(err);
    return res.status(400).send({ error: err.message });
  }
};

// Transfer the Amount
export const transferAmount = async (
  req: Request,
  res: Response
): Promise<any> => {
  const { destinationAccount } = req.body;
  const amount = Number(req.body.amount);
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }
  const token = authHeader.split(" ")[1];

  try {
    const decoded: any = verifyUserToken(token);
    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });
    }

    const userid = decoded.id;

    const getsourceAccount: any = await Account.findOne({
      accUser: userid,
    });
    if (amount > getsourceAccount.accBalance) {
```

```
  return res.send({
    msg: `Your account balance is insufficient to transfer`,
  });
}
getsourceAccount.accBalance -= amount;
const savedAccount = await getsourceAccount.save();
const getdestinationAccount: any = await Account.findOne({
  accNumber: destinationAccount,
});
if (!getdestinationAccount) {
  return res.send({ msg: "Destination Account Not Found" });
}
getdestinationAccount.accBalance += amount;
const saveddestinationAccount = await getdestinationAccount.save();
const transferAmountTransaction = await Transfer.create({
accountNumber: getsourceAccount.accNumber,
  destinationAccount: destinationAccount,
  accUser: getsourceAccount.accUser,
  destUser: getdestinationAccount.accUser,
  amount: amount,
});
const date = getDateIST(transferAmountTransaction?.date);
const time = getTimeIST(transferAmountTransaction?.date);
const userDetails = await User.findOne({ _id: userid });
if (savedAccount) {
  let userFullName = userDetails?.firstname + " " + userDetails?.lastname;
  let useremail: any = userDetails?.email;
  let dateTime = date + " " + time;
  let balance = savedAccount?.accBalance;
  let accountnumberis = savedAccount?.accUser;
  let transactionID = transferAmountTransaction?._id;
  sendTransferEmail(
    userFullName,
    useremail,

    amount,
```

```
        dateTime,
        balance,
        accountnumberis,
        transactionID,

        destinationAccount
      );
    }

    res.send({
      msg: `Amount transfered from ${getsourceAccount.accNumber} to ${destinationAccount} Successful`,
    });
  } catch (err: any) {
    return res.status(400).send({ error: err.message });
  }
};

// Helper function to get only the date in IST
const getDateIST = (date: Date) => {
  const options: Intl.DateTimeFormatOptions = {
    timeZone: "Asia/Kolkata",
    year: "numeric",
    month: "2-digit",
    day: "2-digit",
  };
  return new Date(date).toLocaleDateString("en-IN", options);
};

// Helper function to get only the time in IST
const getTimeIST = (date: Date) => {
  const options: Intl.DateTimeFormatOptions = {
    timeZone: "Asia/Kolkata",
    hour: "2-digit",
    minute: "2-digit",
    second: "2-digit",
  };
  return new Date(date).toLocaleTimeString("en-IN", options);
```

```
};

// Get all credit transactions based on accountId
export const getAllCreditTransactions = async (
req: Request,
  res: Response

const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }
  const token = authHeader.split(" ")[1];

  try {
    const decoded: any = verifyUserToken(token);
    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });
    }

    const userid = decoded.id;
    const result = await Debit.find({ accUser: userid }).sort({ date: -1 });
    if (result.length === 0) {
      return res.send({
        msg: `No transactions Happened`,
      });
    }

    // Use Promise.all to wait for all promises to resolve
    const finalResult = result.map((eachItem: any) => {
    return {
        ...eachItem.toObject(),
        transactionDate: eachItem?.date ? getDateIST(eachItem.date) : null,
        transactionTime: eachItem?.date ? getTimeIST(eachItem.date) : null,
      };
    });

    res.send(finalResult);
```

```
  } catch (err: any) {
    return res.status(400).send({ error: err.message });
  }
};

// Get transactions by date range
export const getTransactionsByDateRange = async (
  req: Request,
  res: Response
): Promise<any> => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }

  const token = authHeader.split(" ")[1];

  try {
    const decoded: any = verifyUserToken(token);
    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });
    }

    const userid = decoded.id;
    const { startDate, endDate } = req.query;

    // Base filter
    let filter: TransactionFilter = { accUser: userid };

    // If dates are provided, add date filter
    if (startDate && endDate) {
      const start = new Date(startDate as string);
      const end = new Date(endDate as string);
      filter = {
        ...filter,
        date: { $gte: start, $lte: end },
      };
```

```
    }

    // Fetch transactions
    const creditTransactions = await Credit.find(filter).sort({ date: -1 });
    const debitTransactions = await Debit.find(filter).sort({ date: -1 });
    const transferTransactions = await Transfer.find(filter).sort({ date: -1 });


 // Combine results and add transaction type
    const combinedTransactions = [
      ...creditTransactions.map((tx) => ({
        ...tx.toObject(),
        type: "credit",
        destinationAccount: "",
        username: "",
        transactionDate: getDateIST(tx.date),
        transactionTime: getTimeIST(tx.date),
      })),
      ...debitTransactions.map((tx) => ({
        ...tx.toObject(),

        type: "debit",
// User can Apply for Debit card
export const applyDebit = async (req: Request, res: Response): Promise<any> => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }

  const token = authHeader.split(" ")[1]
 try {
    const decoded: any = verifyUserToken(token);
    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });
    }

    const userid = decoded.id;
```

```
    const accountUser = await User.findOne({ _id: userid });
    let fullname = accountUser?.firstname + " " + accountUser?.lastname;
    let userEmail = accountUser?.email!;

    const cardDataAdded: any = await DebitCard.findOne({ userid: userid });
    if (cardDataAdded?.status === "pending") {
      return res.status(402).send({
        msg: "You are Already applied for Debit card .pls Wait for Approval ",
      });
    }

// Get the Debit card Details
export const getDebitCardDetails = async (
  req: Request,
  res: Response
): Promise<any> => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).send({ msg: "Authorization header is missing" });
  }

  const decoded: any = verifyUserToken(token);
    if (!decoded) {
      return res.status(401).send({ msg: "Access Token is Invalid" });
    }
    const userid = decoded.id;
    const getEmail = await User.findOne({ _id: userid });
    return res.send({ email: getEmail?.email });


  } catch (err: any) {
    console.error("Error:", err);
    return res.status(400).send({ error: err.message });
  }
};
```

CHAPTER-7

SYSTEM TESTING

7.1 INTRODUCTION TO SYSTEM TESTING:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.2 Types of Tests

7.2.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

7.2.2 Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

7.2.3 Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

7.2.4 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

7.2.5 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and rsesponses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed

7.3 Test Cases

User Module:

Test Case ID	Test Scenario	Precondition	Test Steps	Expected Result
TC-01	User Registration	User is on the registration page.	1. Open registration page. 2. Enter valid details. 3. Click on "Register."	User account should be successfully created, and a confirmation message should be displayed.
TC-02	User Login	User has registered and is on the login page.	1. Open login page. 2. Enter valid email and password. 3. Click on "Login."	User should be logged in successfully and redirected to the dashboard.
TC-03	Update Profile	User is logged in.	1. Navigate to the "Profile" page. 2. Edit personal details. 3. Click "Save Changes."	The profile details should be updated and saved successfully.
TC-04	View Bank Statements	User is logged in.	1. Navigate to the "Bank Statement" page.	The bank statement for the selected period should be displayed.

			2. Select a date range and click "View Statement."	

Test Case ID	Test Scenario	Precondition	Test Steps	Expected Result
TC-05	View Account Balance	User is logged in.	1. Navigate to the "Dashboard" page. 2. Check available balance in the account section.	The correct account balance should be displayed.
TC-06	Credit Account	User is logged in and on the dashboard.	1. Navigate to the "Transaction" page. 2. Select "Credit." 3. Enter amount and details. 4. Click "Confirm."	The account balance should be updated with the credited amount, and a success message should be displayed.
TC-07	Debit Account	User is logged in and on the dashboard.	1. Navigate to the "Transaction" page. 2. Select "Debit." 3. Enter amount and details. 4. Click "Confirm."	The account balance should be updated with the debited amount, and a success message should be displayed.
TC-08	Transfer Funds	User is logged in.	1. Navigate to the "Transfer" page. 2. Enter recipient details, amount, and	Funds should be transferred, and the sender's balance should reflect the deducted amount.

			transfer reason. 3. Click "Confirm Transfer."	

Test Case ID	Test Scenario	Precondition	Test Steps	Expected Result
TC-09	Apply for Loan	User is logged in.	1. Navigate to the "Loan Application" page. 2. Fill in required loan details. 3. Submit the application.	The loan application should be submitted, and a confirmation message should be displayed.
TC-10	View Loan Status	User has applied for a loan.	1. Navigate to the "Loan Status" page. 2. View current status of loan application.	The loan status should be displayed correctly based on the admin's review.

Admin Module

Test Case ID	Test Scenario	Precondition	Test Steps	Expected Result

TC-11	Admin Review Loan Applications	Admin is logged in.	1. Navigate to the "Loan Applications" page. 2. Review a user's loan application. 3. Update loan status and save changes.	Loan status should be updated, and changes should be saved successfully.

TC-12	Admin Add User	Admin is logged in.	1. Navigate to the "Manage Users" page. 2. Enter user details to add a new user. 3. Click "Add User."	The new user should be added successfully, and a confirmation message should be displayed.

CHAPTER-8

OUTPUT SCREENS/RESULT

8.1 Home Page: This is the landing page for the project.

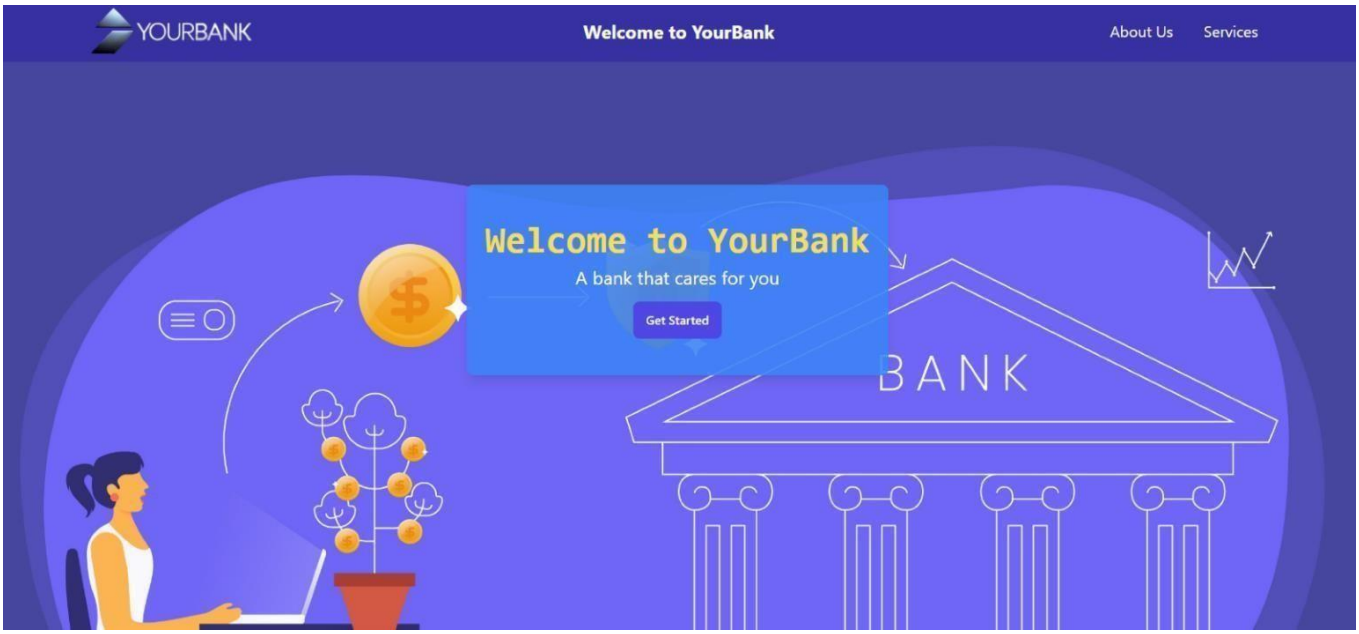


FIG 8.1 HOME PAGE

8.2 About Page: On this page, you can view details about the project.

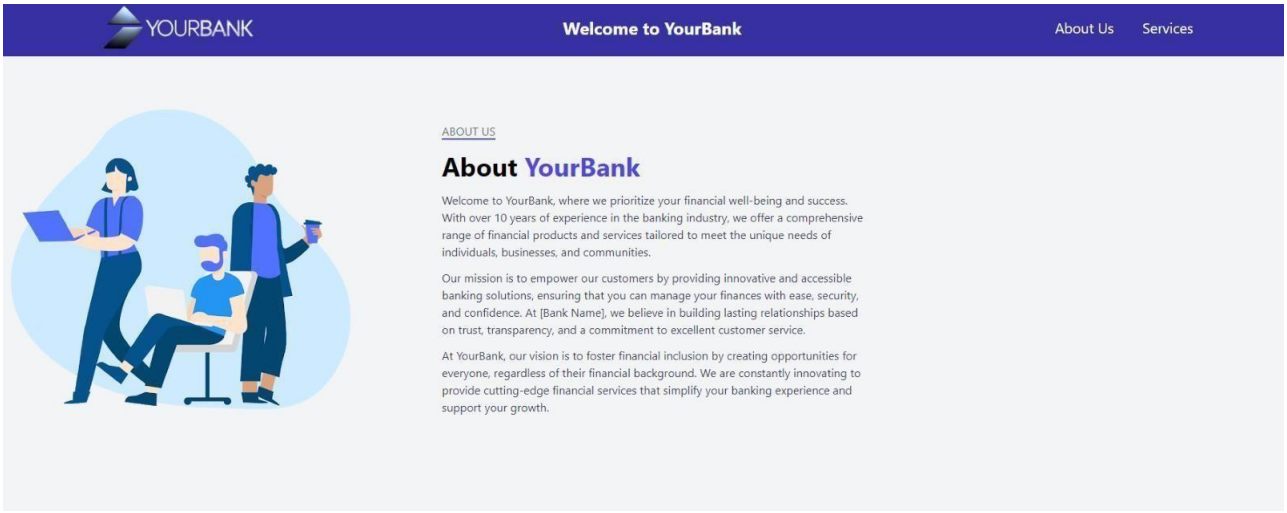


FIG 8.2 ABOUT PAGE

8.3 Services Page: Here, you can view the available services offered in this project.

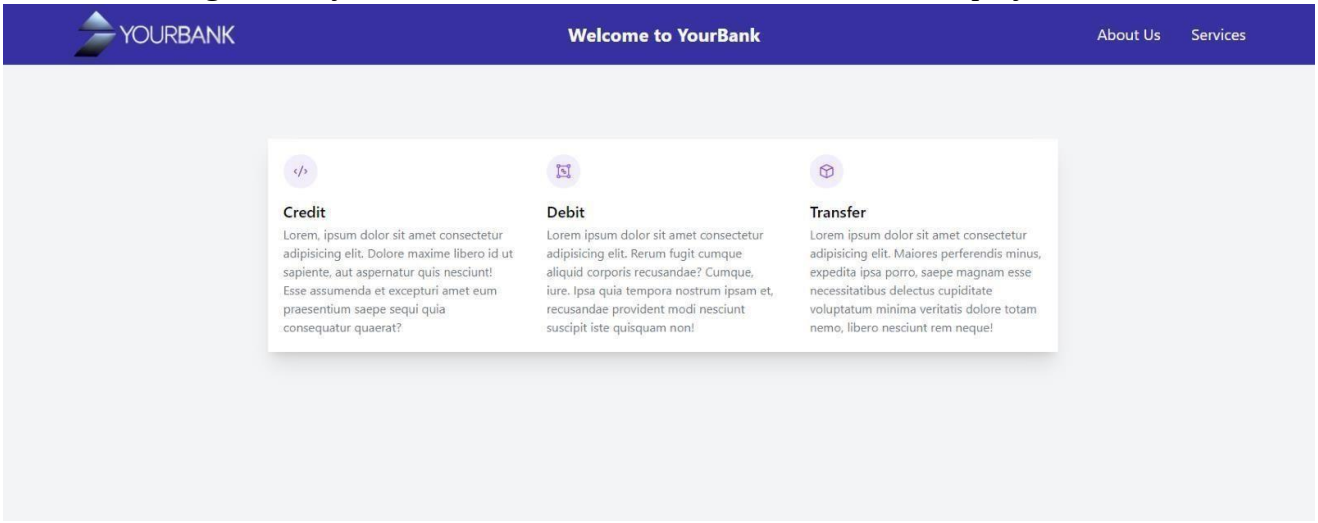


FIG 8.3 SERVICE PAGE

8.4 Admin Login Page: The admin can log in by entering their email and password here**.**

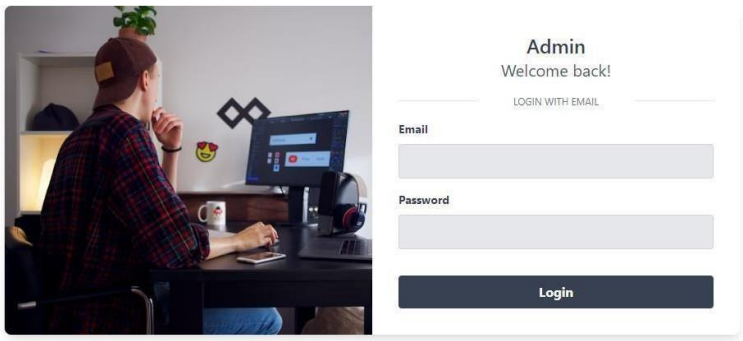


FIG 8.4 ADMIN LOGIN PAGE

8.5 Add Users Page: The admin can add users by filling in the details mentioned below.

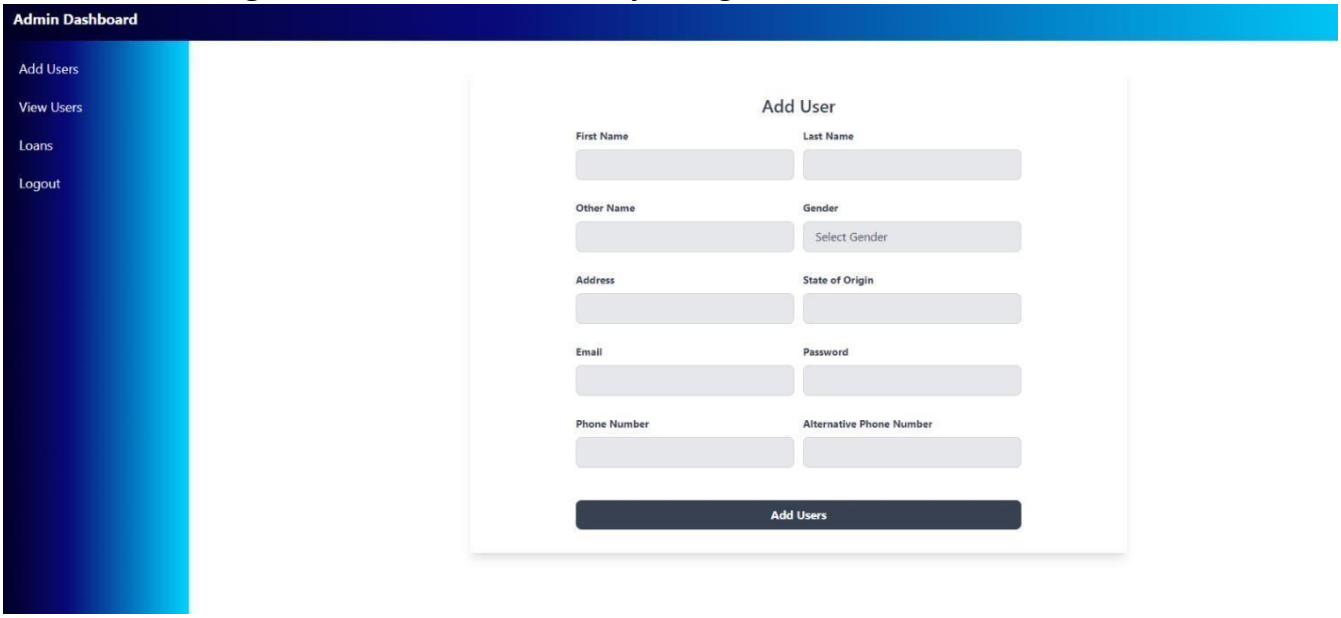
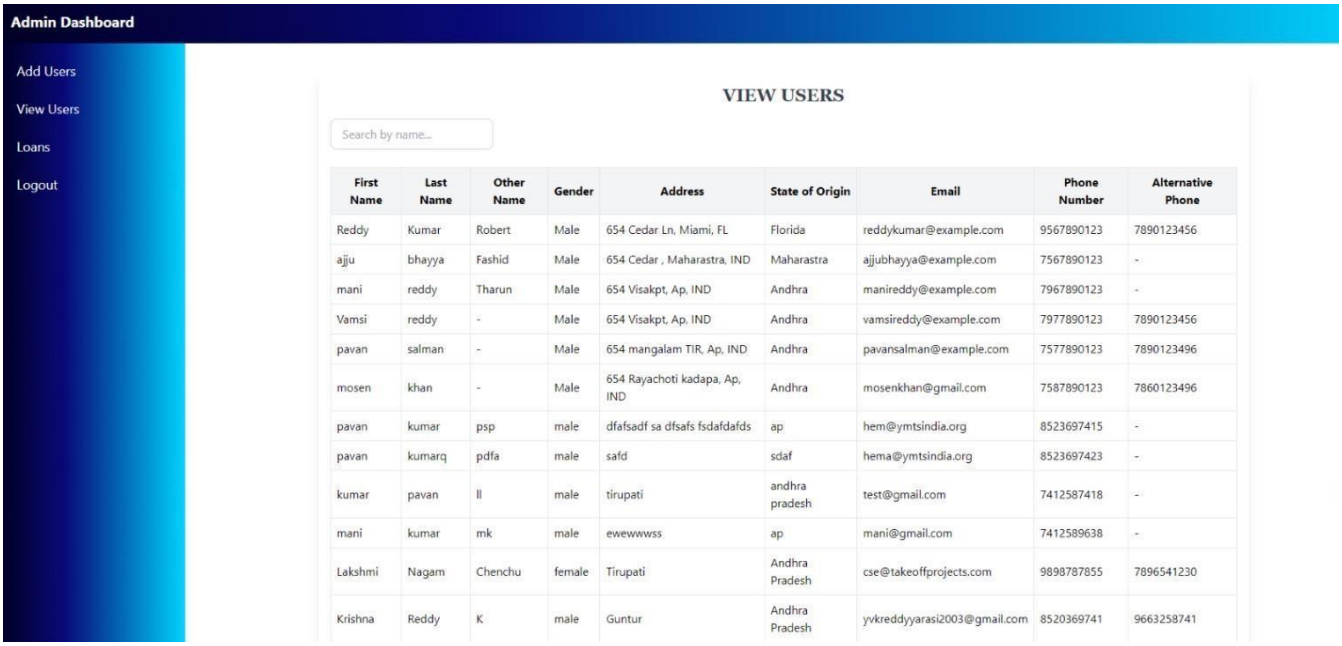


FIG 8.5 ADD USERS PAGE

8.6 View Users Page: The admin can view the details of users they have added, as well as the users registered through the registration form.



First Name	Last Name	Other Name	Gender	Address	State of Origin	Email	Phone Number	Alternative Phone
Reddy	Kumar	Robert	Male	654 Cedar Ln, Miami, FL	Florida	reddykumar@example.com	9567890123	7890123456
ajju	bhayya	Fashid	Male	654 Cedar , Maharastra, IND	Maharastra	ajjubhayya@example.com	7567890123	-
mani	reddy	Tharun	Male	654 Visakpt, Ap, IND	Andhra	manireddy@example.com	7967890123	-
Vamsi	reddy	-	Male	654 Visakpt, Ap, IND	Andhra	vamsireddy@example.com	7977890123	7890123456
pavan	salman	-	Male	654 mangalam TIR, Ap, IND	Andhra	pavansalman@example.com	7577890123	7890123496
mosen	khan	-	Male	654 Rayachoti kadapa, Ap, IND	Andhra	mosenkhan@gmail.com	7587890123	7860123496
pavan	kumar	psp	male	dfafsadf sa dfsafs fsdafdafds	ap	hem@ymtsindia.org	8523697415	-
pavan	kumarq	pdfa	male	safd	sdaf	hema@ymtsindia.org	8523697423	-
kumar	pavan	ll	male	tirupati	andhra pradesh	test@gmail.com	7412587418	-
mani	kumar	mk	male	ewewwwss	ap	mani@gmail.com	7412589638	-
Lakshmi	Nagam	Chenchu	female	Tirupati	Andhra Pradesh	cse@takeoffprojects.com	9898787855	7896541230
Krishna	Reddy	K	male	Guntur	Andhra Pradesh	yvkreddyyarasi2003@gmail.com	8520369741	9663258741

View Loan Requests: The admin can view loan requests raised by users. The admin can either accept or reject the requests. If accepted, the requested amount will be deposited into the respective user's account.

FIG 8.6 VIEW USERS PAGE

8.7 Admin Dashboard

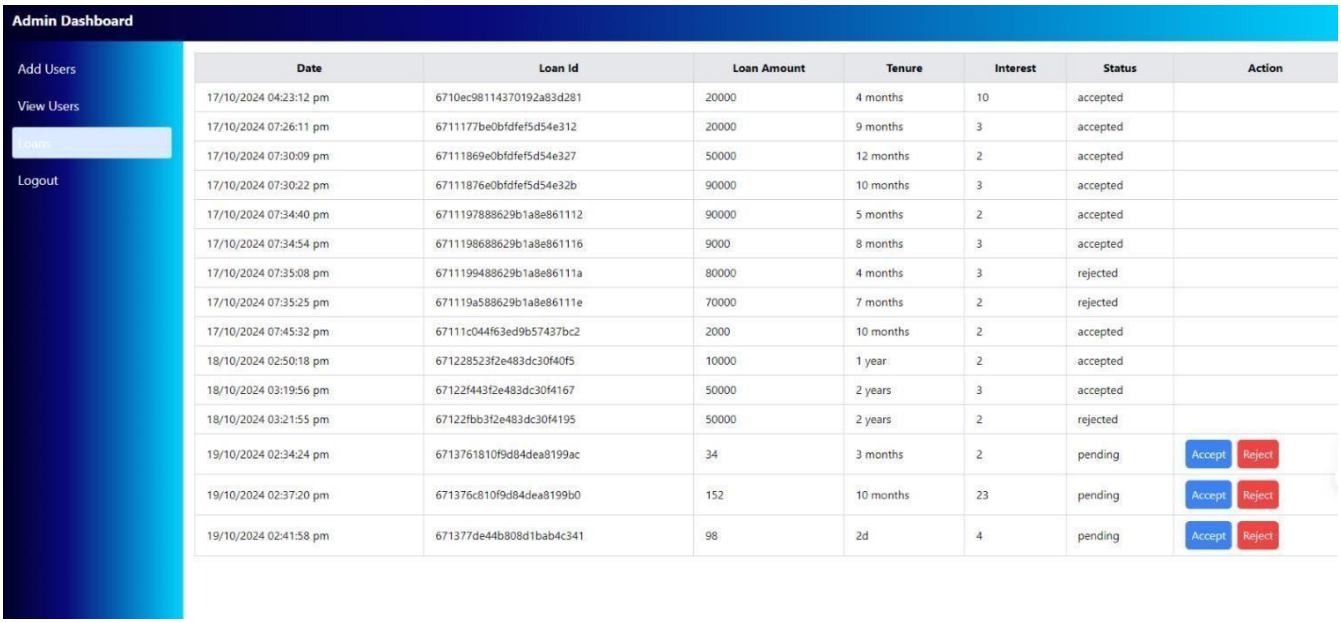


FIG 8.7 ADMIN DASHBOARD

8.8 User/Customer Signup Page: This is the user signup page. By entering the required details, users cancreate an account on this portal.

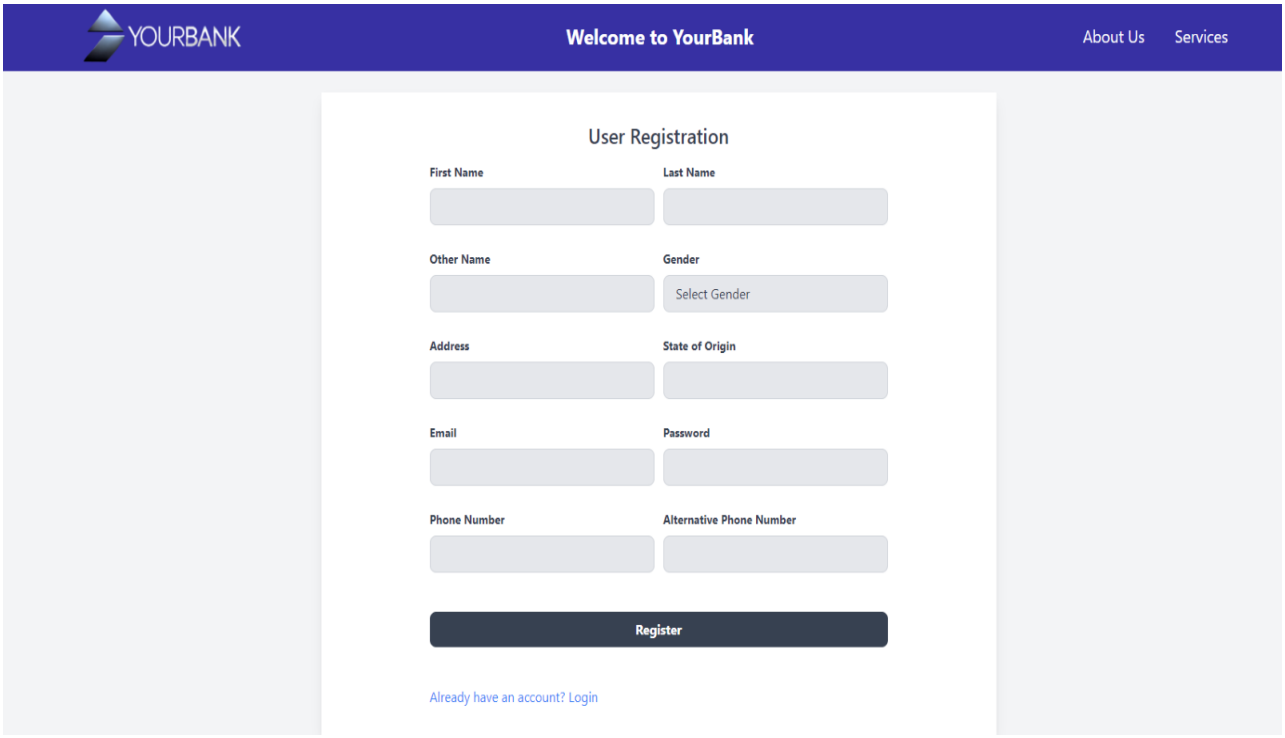


FIG 8.8 USER SIGNUP

8.9 User Login Page: After successful registration, the user can log in by entering their email.

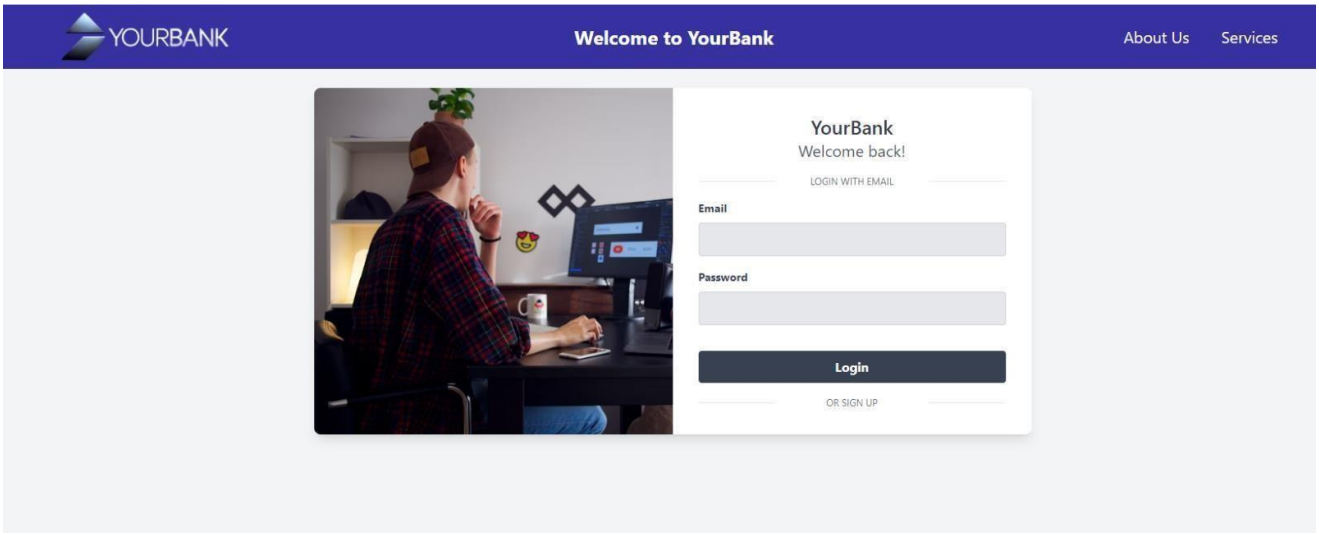


FIG 8.9 USER LOGIN PAGE

9.0 Loan Request Page: If a user wants to apply for a loan, here is the form for submitting a loan request.

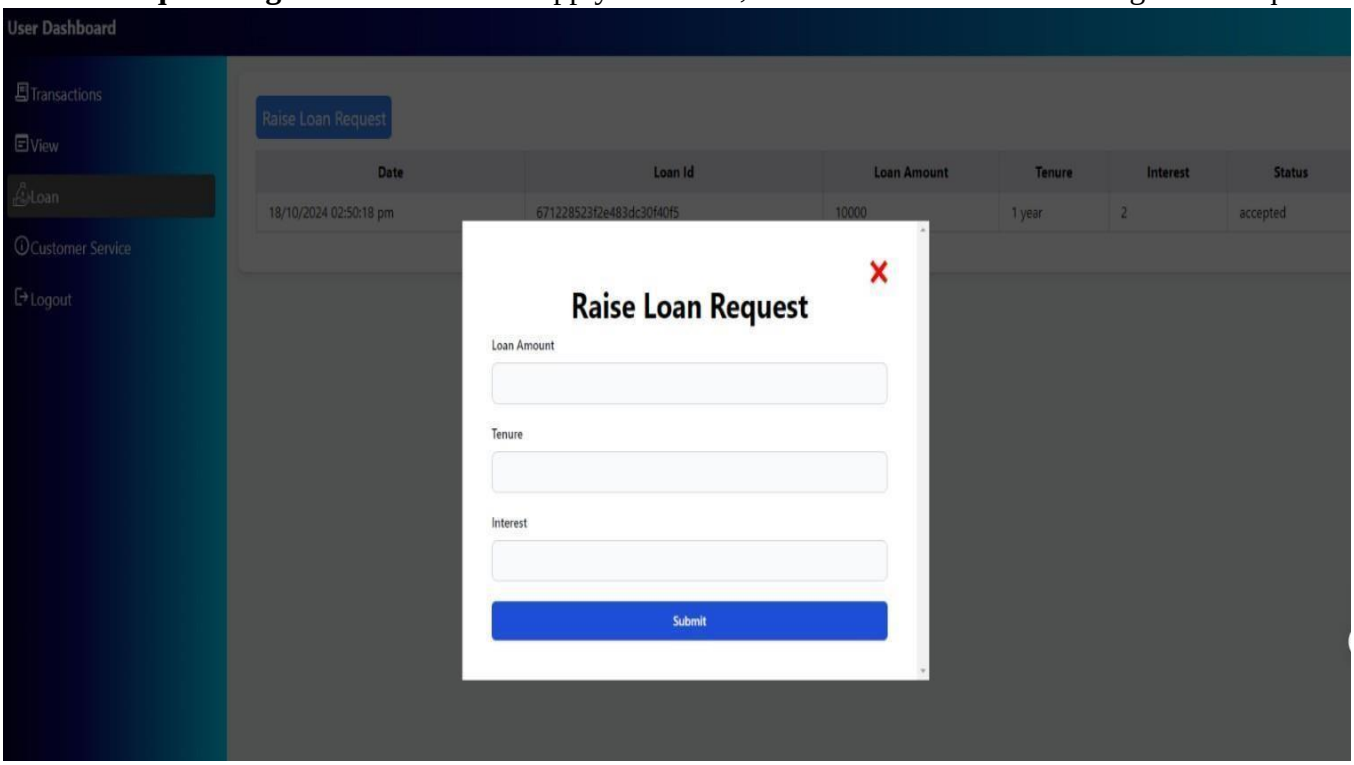


FIG 9.0 LOAN REQUEST PAGE

CHAPTER-9

CONCLUSION

9. CONCLUSION

This MERN stack-based banking system project successfully streamlines core banking operations for both administrators and users. By leveraging MongoDB, Express.js, React.js, and Node.js, it provides a reliable platform for handling tasks such as user management, financial transactions, and loan processing. The system enhances the digital banking experience through its user-friendly design and efficient functionality. The division between Admin and User roles ensures secure, organized management, while the flexibility of the MERN stack allows for scalability and future upgrades. Overall, this project demonstrates the practical application of web technologies in modern banking.

9.1 FUTURE ENHANCEMENT

The system can integrate advanced features like real-time fraud detection using AI, multi-factor authentication for enhanced security, and blockchain for secure and transparent transactions. Adding mobile app support will provide users with more accessibility. Additionally, implementing personalized financial insights and recommendations through machine learning can further improve user experience. Integration with external payment gateways and automated loan risk assessment systems can also broaden the system's capabilities.

CHAPTER-10

REFERENCE

10. REFERANCES

1. **Akhavan, N., & Jafari, H. (2019**) "A Comparative Study of Web-Based Banking Systems." Journal of Internet Banking and Commerce, 24(2), 45-58.
2. **Bennett, M., & Chin, S. (2020**) "A Comparative Study of Web-Based Banking Systems." International Journal of Banking and Financial Services, 34(3), 123-135.
3. **Pandey, V., & Sharma, R. (2021)** "Implementing the MERN Stack for Scalable Web Applications." Journal of Web Development and Design, 12(4), 101-115.
4. **Gomez, J., & Patel, A. (2022)** "User Experience Design in Financial Web Applications." International Journal of Financial Web Systems, 16(1), 67-82.
5. **Kaur, H., & Singh, R. (2021)** "NoSQL Databases in Financial Applications: MongoDB Case Study." Journal of Database Management Systems, 14(3), 200-215.
6. **Khan, M., & Patel, R. (2020)** "The Role of Digital Transformation in Modern Banking: A Case Study of Web-Based Systems." Journal of Financial Innovation, 9(2), 115-130.
7. **Singh, S., & Kaur, P. (2021)** "Security Challenges in Online Banking Systems and Solutions." Journal of Cybersecurity in Financial Systems, 15(1), 25-40.
8. **Sharma, A., & Gupta, V. (2020)** "Leveraging the MERN Stack for Secure and Scalable Banking Solutions." Journal of Web Engineering for Financial Applications, 11(3), 90-105.
9. **Ali, S., & Verma, R. (2021)** "User Experience and Its Impact on Online Banking Adoption: A Study of Customer Preferences." International Journal of Banking and Finance, 14(2), 72-89.
10. **Rai, A., & Soni, P. (2022)** "Analyzing the Efficiency of Web-Based Banking Systems: A Focus on Customer Satisfaction and Operational Performance." Journal of Financial Services Research, 16(4), 150-165.