DANK the BANK

Central Online Banking System

DBMS Project Report
Final-Evaluation

Group 41

Divyam - 2020058

Aamleen - 2020002

Nishaant - 2020091

Kushagra - 2020075

Scope:

With the exponential increase in population, the traffic on Banking portals is tremendous which further results in managing a vast amount of data of customers. This traffic and data is ever increasing and here we present our Banking portal - DANK the Bank (DTB) that effectively uses the concepts of RDBMS to provide a secure, efficient and central banking system to its customers with a user-friendly, interactive interface as a cherry on the top.

Users will be provided with 2 types of accounts - Savings and Current. These accounts will be equipped with multiple user-friendly services online. All the data will be centrally stored and managed using MySQL. The MySQL queries will be optimized and indexes will be used to reduce processing time. The portal is also integrated with features that cater to Employees, which can make it easy for them to manage and for the branches to keep their progress track. Traditionally one of the major hassles faced is while applying and approving loans, as it needs a lot of scrutiny. Through our portal, users can easily apply for loans and the officials on the other end will be provided with all the required data from our Database so that they can reduce the approval time significantly. Once the loan is approved, users can easily pay-back & track their loans annually.

One of the most important features will be to make security our top-most priority. We will achieve this by making our DataBase "smart" by placing proper checks and triggers. Transactions can be done from User-Bank, User-User in various modes and these will be updated on a real-time basis with proper confirmations.

Along with this, we plan to use advanced aggregate functions to get a better idea about branch, employee and user statistics via the variations and stddev in their salaries, balance, transactions, etc.

Thus, using the concepts of DBMS, we aim to deliver a one-stop solution for Online Banking System with Central DataBase.

Stakeholders:

1. Branches Of Banks
    i. Employees Of Banks
        a. Branch Manager
        b. Service Manager
        c. Customer Service
        d. Account Manager
    ii. Loan Department
2. Users/Customers

Project Description/Aims:

An Online Central Banking System portal that integrates users and employees of Bank to effectively enhance the Banking services.

1) Banks can have multiple branches. Storing information of each Branch in the Database.

2) Developing Opcodes for payment, customer, loan and account ID, so that all of them can be effectively linked with each other.

3) Each Branch will have its Employees and Branch Manager. Keeping track of Branch Managers and the Employees under them. Uniquely connect each branch with its employees and manager.

4) Storing Data of Customers/Users and linking them to their accounts.

5) Each Branch will have 2 types of Accounts – Savings & Current. Storing various information related to accounts. Linking Accounts to Branch using Branch Codes. Also linking accounts to users to provide various services to them.

6) Keeping track of transactions made by users, like their type, status, amount, confirmation, etc.

7) Making successful transactions between User & User and also between User & Banks (including loan payment).

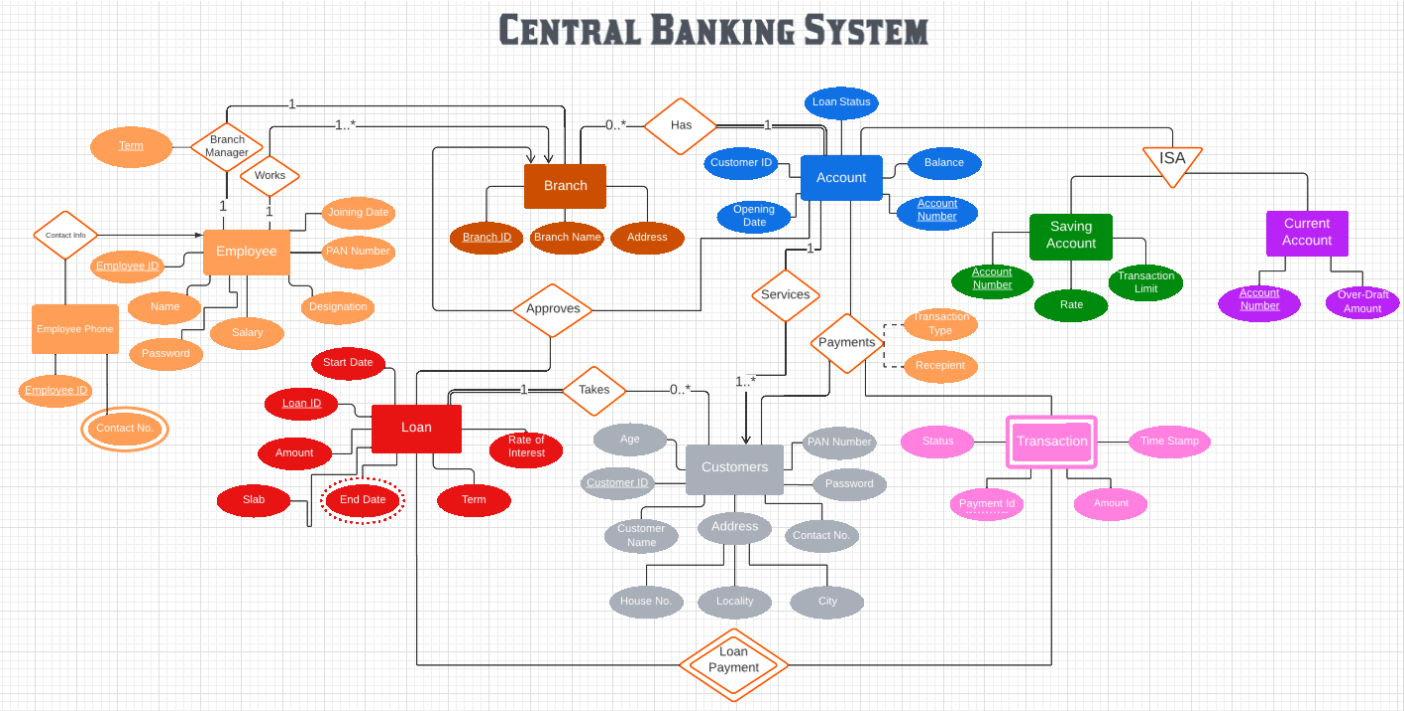
8) Providing Loan opportunities to customers where they can easily apply unless they are defaulters or in the process of paying another loan. We have also provided student loans with 0% interest rates.

9) Tracking the loan status before and after its approval stage. Highlighting significant defaulters & taking actions accordingly.

10) Helping employees to get a better idea about their branch via aggregate MySQL function queries which indicate the areas on which they should improve through statistics.

11) Providing some additional bonus services to the users and admins along with easy to use, interactive UI.

ER DIAGRAM UPDATED



Identification Of Weak Entity

Transaction: Transaction is a weak entity because it depends on the Account and Customer Entity. If there is no account, then there is no point in having any transaction entity itself. Also, the transactions that are made, are done after confirmation from Customer and Account Balance and limits. The Key of Transaction is also dependent on the customer paying.

Identification of Ternary Relationship

Payments: Payments is a ternary relationship because it involves 3 entities. This relationship comes into effect when a customer makes some transactions, which need to be validated by the Account status. Thus 3 entities come into play - Customer, Account & Transaction.

Approves: Approves is a ternary relationship that comes into effect when a customer applies for a loan, which is approved/denied by the branch. Thus 3 entities are involved in this - Customer who applies, Branch which approves/denies, and the Loan which stores all the necessary information and tracks the status before and after loan.

DATABASE SCHEMA

Accounts:

AccountNo	VarChar(100) NOT NULL
OpeningDate	Date NOT NULL
LoanStatus	VarChar(100)
Balance	Double
Customer_ID	VarChar(100) NOT NULL

Primary Key: AccountNo
Foreign Key: Customer_ID

Savings_Accounts:

AccountNo	VarChar(100) NOT NULL
OpeningDate	Date NOT NULL
LoanStatus	VarChar(100)
Balance	Double
Customer_ID	VarChar(100) NOT NULL
Interest Rate	Int DEFAULT ‘6’
TransactionLimit	Double

Primary Key: AccountNo
Foreign Key: AccountNo

Current_Accounts:

AccountNo	VarChar(100) NOT NULL
OpeningDate	Date NOT NULL
LoanStatus	VarChar(100)
Balance	Double
Customer_ID	VarChar(100) NOT NULL
OverdraftLimit	bigint NOT NULL

Primary Key: AccountNo
Foreign Key: AccountNo

Employee:

Employee_ID	VARCHAR(100) NOT NULL
Name	VARCHAR NOT NULL
Salary	Int NOT NULL
Designation	VARCHAR(100)

	NOT NULL
PAN No.	VARCHAR(100) NOT NULL
Joining Date	DATE NOT NULL
Contact No.	VarChar(10) NOT NULL
Password	VARCHAR(30) NOT NULL

Primary Key: Employee_ID

Employee_Phone:

Employee_ID	VARCHAR(100) NOT NULL
PhoneNo	VarChar(10) NOT NULL

Primary Key: Employee_ID, PhoneNo
Foreign Key: Employee_ID

Works:

Employee_ID	VARCHAR(100) NOT NULL
Branch_ID	Bigint NOT NULL

Primary Key: Employee_ID, Branch_ID

Customers:

Customer_ID	VarChar(100) NOT NULL
Name	VarChar(100) NOT NULL
Age	Int NOT NULL
HouseNo	VarChar(20) NOT NULL
Locality	VarChar(100) NOT NULL
City	VarChar(50) NOT NULL
ContactNo	Bigint NOT NULL
PAN	VarChar(30)
Password	VarChar(100) NOT NULL

Primary Key: Customer_ID

Branch:

Branch_ID	Bigint NOT NULL
Branch_Name	VarChar(100) NOT NULL
Address	VarChar(100) NOT NULL

Primary Key: Branch_ID

Transactions:

Payment_ID	VarChar(100) NOT NULL
Amount	Double
Date	DateTime NOT NULL
Status	VarChar(100) NOT NULL

Primary Key: Payment_ID

Loans:

StartDate	Date NOT NULL
Loan_ID	VarChar(100) NOT NULL
Amount	Double NOT NULL
InterestRate	Double
Term	Int NOT NULL
EndDate	Date

Primary Key: Loan_ID

Branch_Managers:

Employee_ID	VARCHAR(100) NOT NULL
Branch_ID	BigInt NOT NULL
Name	VARCHAR(100) NOT NULL
Designation	VARCHAR(100) NOT NULL
Term	Int NOT NULL
Joining_Date	Date NOT NULL

Primary Key: Employee_ID, Branch_ID

Branch_Account:

Branch_ID	BigInt NOT NULL
AccountNo	VarChar(100) NOT NULL

Primary Key: AccountNo, Branch_ID

Branch_Loan_Account:

Branch_ID	BigInt NOT NULL
AccountNo	VarChar(100)

	NOT NULL
Loan_ID	VarChar(100) NOT NULL

Primary Key: AccountNo, Branch_ID, Loan_ID

Customer_Account:

Customer_ID	VarChar(100) NOT NULL
AccountNo	VarChar(100) NOT NULL

Primary Key: AccountNo, Customer_ID

Customer_Account_Transactions:

Customer_ID	VarChar(100) NOT NULL
Payment_ID	VarChar(100) NOT NULL
AccountNo	VarChar(100) NOT NULL
Amount	Double
Transaction_Type	VarChar(100) NOT NULL
Recipient	VarChar(100) NOT NULL

Primary Key: Payment_ID, Customer_ID

Loan_Transactions:

Loan_ID	VarChar(100) NOT NULL
Payment_ID	VarChar(100) NOT NULL
Amount	Double

Primary Key: Payment_ID, Loan_ID

RELATIONAL SCHEMA

Branches (Branch ID, Branch Name, Address)
Employees (Employee ID, Name, PAN Number, Salary, Designation, Joining Date, {Contact No.}, Password)
Employee_Phone(PhoneNo, Employee_ID)

Works (Employee ID, Branch ID)
Branch_Managers (Employee ID, Name, Branch ID, Designation, Term, Joining Date)

Branch_Account (Branch ID, Account No.)
Branch_Loan(Branch ID, Account No, Loan_ID)
Branch_Loan_Account(Branch ID, Loan ID, Customer ID)

Accounts (Account No., Customer ID, Opening Date, Balance, Loan Status)
Savings_Accounts (Account No., Rate, Transaction Limit)
Current_Accounts (Account No., OverDraftLimit)

Customers (Customer ID, Name, PAN Number, Age, Password,Customer Name, Address(House No.,Locality,City))

Customer_Account(Account No., Customer ID)
Customer_Account_Transactions(Payment ID, Customer ID, Account No, Amount, Transaction_Type, Recipient)

Loans(Loan_Id,Term, Rate of Interest, End Date,Start Date,Amount, Slab)
Loan_Transactions(Loan ID, Payment ID, Amount)
Transactions(Payment ID, Status, Amount, Date)

SQL QUERIES

1. List all loan defaulters name, account number, loan_id
2. List account numbers of all employees
3. List Loan_ids that were successfully paid by a customer
4. List customers who are older than 60 and have taken a loan with their rate of interest
5. List all the employees working under a branch manager
6. List status of every type of payment from a customer.
7. List status of transactions of payment > 10000
8. List the customer's number, customer's Name, branch id and loan amount for people who have taken loans.
9. List account number, and net balance across accounts of a customer
10. List the total number of withdrawals and total number of deposits being done by the customer
11. List name, customerID for failed transactions
12. List the managers with salary > 50000 who joined before 2010
13. List the No of customers with savings account and current account
14. List accounts with loan status pending and roi >= 6%
15. List Customers who have both savings and current accounts.

NEW SQL QUERIES

1. For every customer check total transaction and grant him gold, silver, platinum accounts
2. List all the loan defaulters of a branch, and the total amount defaulted, group by branch.
3. Give the loyal customers (>10 years of service with the bank) a one-time gift of Rs. 500 in one of their accounts.
4. List the average salary of all the employees at each branch, and compare it with the salary of the branch manager. Sort the result by average salary at each branch.
5. List the average, maximum and minimum transaction done by a customer
6. Save the current DB state, and then delete all the customers not having PAN as per the RBI rules

7. Accidentally, customers less than 18 years old were also deleted which should not have happened. Revert to the previous state and make the correction. Save the final changes.
8. Calculating the credit score of a customer
9. New loan is to be granted to customers with a credit score > 900
10. Ranking branches on the basis of loans they gave out
11. List all the employees having accounts in the same bank, along with the total balance in all their accounts.
12. Calculate TDS/Tax to be deducted from the Transactions & Deposits made by customers.

ADVANCED AGGREGATE SQL QUERIES

1. Number of transactions in a year, average amount and stddev
2. Ranking customers in the order of transactions they made
3. Ranking customers on the basis of their loan payments
4. Stddev and ranking on branches on the basis of payments made through them
5. Variance calculation of balance and loans issued by a bank branch, compared with its assets

EMBEDDED SQL QUERIES

1. Displaying the user's savings accounts

```
def userSavings():
    if request.method == 'POST':
        columns = ["AccountNo", "Opening_Date", "LoanStatus", "Balance",
"Customer_ID"]
        branch = request.get_json()['id'][:4]
        myCursor.execute("SELECT * FROM accounts WHERE Customer_ID = %s",
(request.get_json()['id'],))
        l = []
        for x in myCursor.fetchall():
            if(x[0][6:8] == "00"):
                l.append(dict(zip(columns, x)))
        print(l)
        columns = ["Payment_ID", "Amount", "Date", "Status"]
        transactionID = []
        for x in l:
```

```
            myCursor.execute("SELECT * FROM customer_account_transaction WHERE
AccountNo = %s", (x['AccountNo'],))
            transactionID.append({x['AccountNo']:[j[1] for j in
myCursor.fetchall()]})
        print(transactionID)
        transactions = []
        finalReturn = []

        for p in transactionID:
            for q in p:
                for x in p[q]:
                    myCursor.execute("SELECT * FROM transactions WHERE Payment_ID =
%s", (x,))
                    i = myCursor.fetchall()
                    print(i)
                    transactions.append(dict(zip(columns, i[0])))
                finalReturn.append({q:transactions})
                transactions = []
        listOfListsOfDicts = [l, finalReturn]
        if(listOfListsOfDicts[0] != []):
            if(l == []):
                return "No Savings Account"
            else:
                # take tuple[0] to get the list and then map it to the savings
account
                return {0:listOfListsOfDicts}
        else:
            return "Failure"
```

2. Creating a new loan

```
def newLoan():
    if request.method == 'POST':
        account = request.get_json()['account']
        myCursor.execute("SELECT LoanStatus FROM accounts WHERE AccountNo = %s",
(account,))
        loanStatus = myCursor.fetchall()[0][0]
        print(loanStatus)
        if loanStatus == "NULL" or loanStatus == "PAID":
            loanStatus = "OKAY"
        else:
            return "Loan Cannot be Created"
```

```
        myCursor.execute("UPDATE accounts SET LoanStatus = %s WHERE AccountNo =
%s", (loanStatus, account))
        myCursor.execute("SELECT * FROM branch_loan_account WHERE AccountNo = %s",
(account,))
        loanNo = 1
        if myCursor.rowcount >= 1:
            loanNo += myCursor.rowcount
        loanID = ""
        if loanNo < 10:
            loanID = account[:4]+"03"+account[6:8]+account[-2:]+"000"+str(loanNo)
        else:
            loanID = account[:4]+"03"+account[6:8]+account[-2:]+"00"+str(loanNo)
        slab =
(int(request.get_json()['amount'])/int(request.get_json()['term']))*(1+(int(request
.get_json()['roi'])/100))
        myCursor.execute("INSERT INTO loans (Loan_ID, StartDate, Amount,
InterestRate, Term, EndDate, Slab) VALUES (%s, CURDATE(), %s, %s, %s,
DATE_ADD(CURDATE(), INTERVAL %s YEAR), %s)", (loanID,
request.get_json()['amount'], request.get_json()['roi'],
request.get_json()['term'], request.get_json()['term'], slab))
        myCursor.execute("INSERT INTO branch_loan_account (Branch_ID, AccountNo,
Loan_ID) VALUES (%s, %s, %s)", (account[:4], account, loanID))
        db.commit()
        if(myCursor.rowcount >= 1):
            return "Success"
        else:
            return "Failure"
```

3. Creating a new transaction

```
def userTransactions():
    if request.method == 'POST':
        columns = ["Payment_ID", "Amount", "Date", "Status"]
        branch = request.get_json()['id'][:4]
        myCursor.execute("SELECT * FROM customer_account_transaction WHERE
Customer_ID = %s", (request.get_json()['id'],))
        transactionID = []
        for x in myCursor.fetchall():
            transactionID.append(x[1])
        transactions = []
        for p in transactionID:
            myCursor.execute("SELECT * FROM transactions WHERE Payment_ID = %s",
(p,))
            transactions.append(dict(zip(columns, myCursor.fetchall()[0])))
```

```
        print(transactions)
        print(myCursor.rowcount)
        if(myCursor.rowcount >= 1):
            if(transactions == []):
                return "No Transactions"
            else:
                print(transactions)
                return {0:transactions}
        else:
            return "Failure"
```

4. Displaying the loans of the user

```
def userLoans():
    if request.method == 'POST':
        columns = ["StartDate","Loan_ID", "Amount", "InterestRate", "Term",
"Status"]
        accounts = []
        branch = request.get_json()['id'][:4]
        myCursor.execute("SELECT * FROM accounts WHERE Customer_ID = %s",
(request.get_json()['id'],))
        loanStatus = []
        for x in myCursor.fetchall():
            accounts.append(x[0])
            loanStatus.append(x[2])
        print(accounts)
        for x in accounts:
            myCursor.execute("SELECT * FROM branch_loan_account WHERE AccountNo =
%s", (x,))
        loanID = []
        for x in myCursor.fetchall():
            print(x)
            loanID.append(x[1])
        loans = []
        for l in loanID:
            myCursor.execute("SELECT L.StartDate, L.Loan_ID, L.Amount,
L.InterestRate, L.Term, A.LoanStatus FROM loans L, Accounts A, branch_loan_account
B WHERE L.Loan_ID = %s AND A.AccountNo = B.AccountNo AND L.Loan_ID = B.Loan_ID",
(l,) )
            loans.append(dict(zip(columns, myCursor.fetchall()[0])))
        print(loans)
        if(myCursor.rowcount >= 1):
            if(loans == []):
                return "No Loans"
```

```
            else:
                return {0:loans}
        else:
            return "Failure"
```

5. Loan Transaction

```
def loanPayments():
    if request.method == 'POST':
        account = request.get_json()['AccountNo']
        branch = account[:4]
        customerID = ""
        myCursor.execute("SELECT Customer_ID FROM accounts WHERE AccountNo = %s",
(account,))
        customerID = myCursor.fetchall()[0][0]
        myCursor.execute("SELECT * FROM transactions WHERE Payment_ID LIKE %s",
(branch+'%', ))
        loanPaymentID = ""
        if(myCursor.rowcount < 10):
            loanPaymentID =
branch+"0202"+account[-2:]+branch[-2:]+account[-2:]+"0"+str(myCursor.rowcount +
1)+account[6:8]+"CX"
        else:
            loanPaymentID =
branch+"0202"+account[-2:]+branch[-2:]+account[-2:]+str(myCursor.rowcount +
1)+account[6:8]+"CX"
        amount = request.get_json()['Amount']
        myCursor.execute("SELECT * FROM loans WHERE Loan_ID = %s",
(request.get_json()['LoanID'],))
        loan = myCursor.fetchall()[0]
        term = loan[4]
        startDate = loan[0]
        loanAmount = loan[2]
        slab = loan[6]
        myCursor.execute("SELECT Balance,LoanStatus FROM accounts WHERE AccountNo =
%s", (account,))
        loanPaymentStatus = "PROCESSED"
        accountData = myCursor.fetchall()[0]
        loanStatus = accountData[1]
        if loanStatus == "PAID" or loanStatus == "DEFAULTER":
            return "Loan Payment Not Allowed"
        if date.today() > startDate.replace(year= startDate.year + int(term)):
            if loanStatus != "DEFAULTER":
                loanStatus = "DEFAULTER"
```

```
                return "Loan Term Expired"
        else:
            if int(amount) > int(loanAmount):
                return "Payment is Greater than Yearly Slab"
            elif int(amount) == int(loanAmount):
                check = (date.today() - (startDate.replace(year= startDate.year +
int(term)))).days/365
                if check <= 0 and check >= -1 and loanAmount > 0:
                    loanStatus = "PAID"
                    loanAmount = "0"
                else:
                    loanAmount = slab
                    loanStatus = "PENDING"
            elif int(amount) < int(loanAmount):
                loanAmount = str(int(loanAmount) - int(amount))
                loanStatus = "PENDING"
        if(int(accountData[0]) - int(amount) < 0):
            loanPaymentStatus = "FAILED"
        elif loanPaymentStatus != "FAILED":
            myCursor.execute("UPDATE accounts SET Balance = Balance - %s WHERE
AccountNo = %s", (amount, account))
            myCursor.execute("UPDATE accounts SET LoanStatus = %s WHERE AccountNo =
%s", (loanStatus, account))
            myCursor.execute("UPDATE loans SET Amount = %s WHERE Loan_ID = %s",
(loanAmount, request.get_json()['LoanID']))
        # loan Status and update
        myCursor.execute("INSERT INTO transactions (Payment_ID, Amount, Date,
Status) VALUES (%s, %s, CURDATE(), %s)", (loanPaymentID, amount,
loanPaymentStatus))
        myCursor.execute("INSERT INTO loan_transaction (Loan_ID, Payment_ID,
Amount) VALUES (%s, %s, %s)", (request.get_json()['LoanID'], loanPaymentID,
amount))
        transactionType = ""
        if loanPaymentID[6:8] == "01":
            transactionType = "Customer to Customer"
        elif loanPaymentID[6:8] == "02":
            transactionType = "Loan Payment"
        elif loanPaymentID[6:8] == "03":
            transactionType = "Deposit/Withdrawal"
        myCursor.execute("INSERT INTO customer_account_transaction (Customer_ID,
AccountNo, Payment_ID, Amount, Transaction_Type, Recipient) VALUES (%s, %s, %s, %s,
%s, %s)", (customerID, account, loanPaymentID, amount, transactionType, account))
        # db.commit()
```

```
        if(myCursor.rowcount >= 1):
            return loanPaymentStatus
        else:
            return loanPaymentStatus
```

6. Displaying customers of bank branch

```
def adminCustomers():
    if request.method == 'POST':
        adminId = request.get_json()['id']
        branch = adminId[:4]
        myCursor.execute("SELECT * FROM customers WHERE Customer_ID LIKE %s",
(branch+"%",))
        columns = [column[0] for column in myCursor.description]
        finalReturn = []
        customerID = []
        for i in myCursor.fetchall():
            finalReturn.append(dict(zip(columns,i)))
            customerID.append(i[0])
        accounts = []
        finalAccounts = []
        for i in customerID:
            myCursor.execute("SELECT * FROM accounts WHERE Customer_ID = %s", (i,))
            columns = [column[0] for column in myCursor.description]
            listOfAccounts = []
            for j in myCursor.fetchall():
                listOfAccounts.append(dict(zip(columns,j)))
            finalAccounts.append({i:listOfAccounts})
        # print(finalReturn)
        print(finalAccounts)
        if(myCursor.rowcount >= 1):
            return {"customer":finalReturn, "accounts":finalAccounts}
        else:
            return "Failure"
```

7. Displaying customers of bank branch

```
def newEmployee():
    if request.method == 'POST':
        name = request.get_json()['Name']
        age = request.get_json()['Age']
        salary = request.get_json()['Salary']
        designation = request.get_json()['Designation']
        pan = request.get_json()['PAN']
```

```
        password = request.get_json()['Password']
        branch = request.get_json()['Branch_ID']
        joiningDate = request.get_json()['Joining_Date']
        myCursor.execute("SELECT * FROM employees WHERE Employee_ID LIKE %s",
(branch+"%",))
        employeeID = branch+"0000"+str(myCursor.rowcount + 1)
        myCursor.execute("INSERT INTO employees (Employee_ID, Name, Salary,
Designation, Joining_Date, PAN, Password) VALUES (%s, %s, %s, %s, %s, %s, %s)",
(employeeID, name, salary, designation, joiningDate, pan, password))
        # db.commit()
        if(myCursor.rowcount == 1):
            return "Success"
        else:
            return "Failure"
```

INDEXES

1. CREATE INDEX Customer_ID_idx ON Customers (Customer_ID);
2. CREATE INDEX LoanStatus_idx ON Accounts (LoanStatus);
3. CREATE UNIQUE INDEX AccountNo_idx ON Accounts (AccountNo);
4. CREATE UNIQUE INDEX Employee_ID_idx ON Employees (Employee_ID);
5. CREATE UNIQUE INDEX Payment_ID_idx ON Transactions (Payment_ID);
6. CREATE INDEX Customer_Account_idx ON Accounts (Customer_ID, AccountNo);
7. CREATE INDEX Status_idx ON Transactions (Status);
8. CREATE INDEX Designation_idx ON Employees (Designation);

many more in index.sql

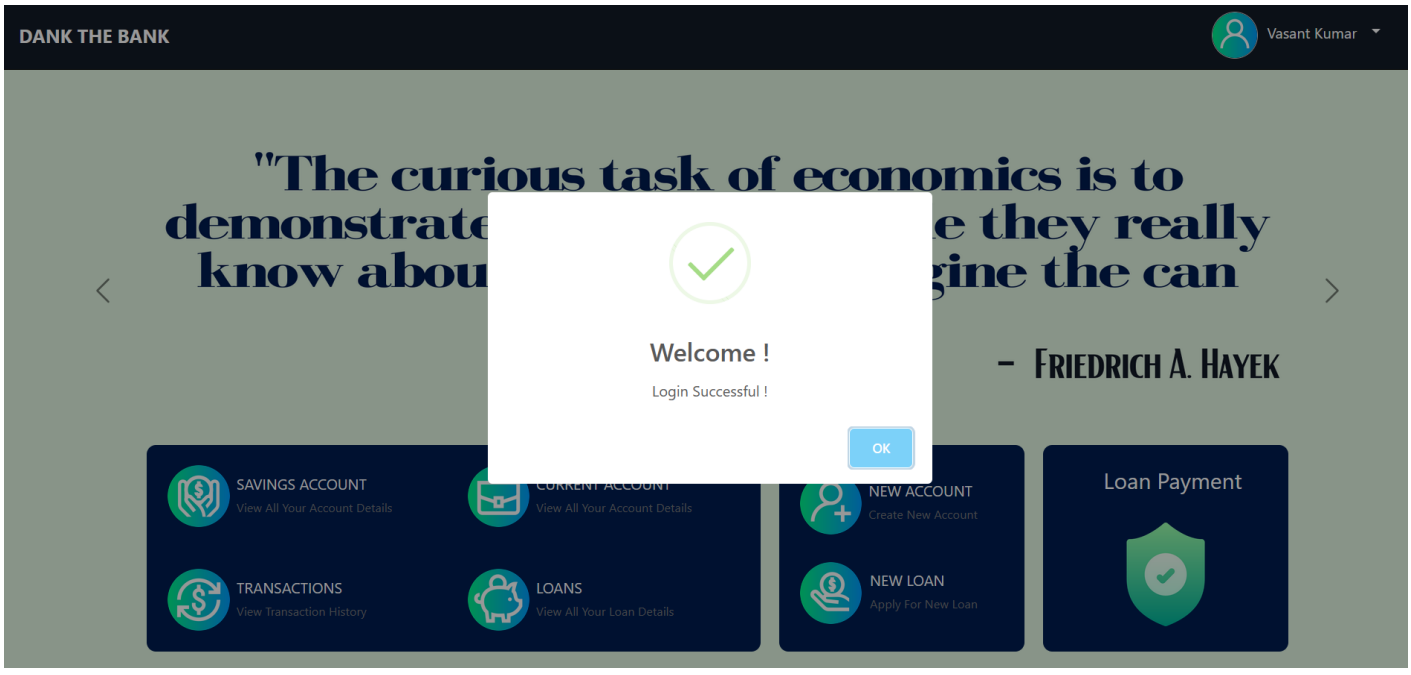
TRIGGERS

1. Updating balance in savings and current accounts after transactions.
2. Updating Overdraft limit in current accounts (0 for Balance < 3000)
3. Updating Interest Rate in savings accounts (0% for Balance < 3000)
4. Updating age of customer
5. Preventing a user with “defaulter” and “pending” status to take loans
6. Preventing a user with “defaulter” status to make transactions

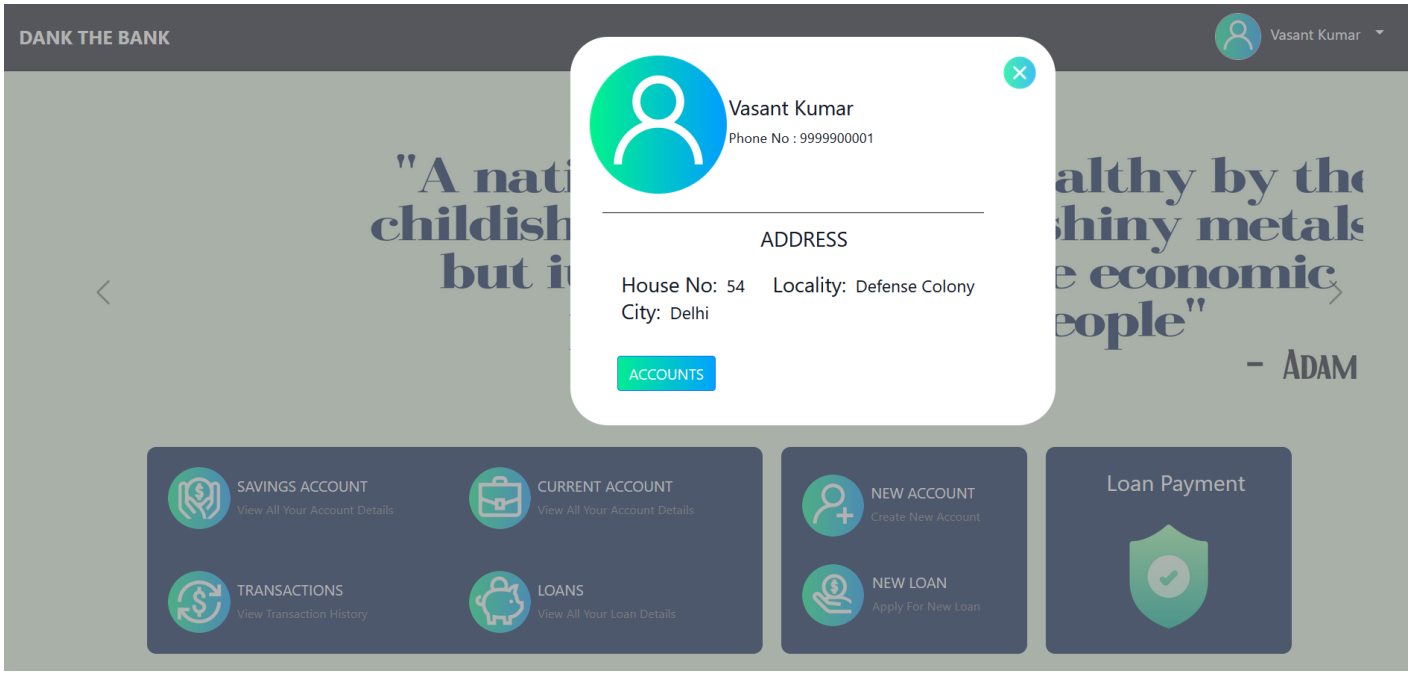
WEB-DEVELOPMENT

User/Admin Login Page

Successful Login

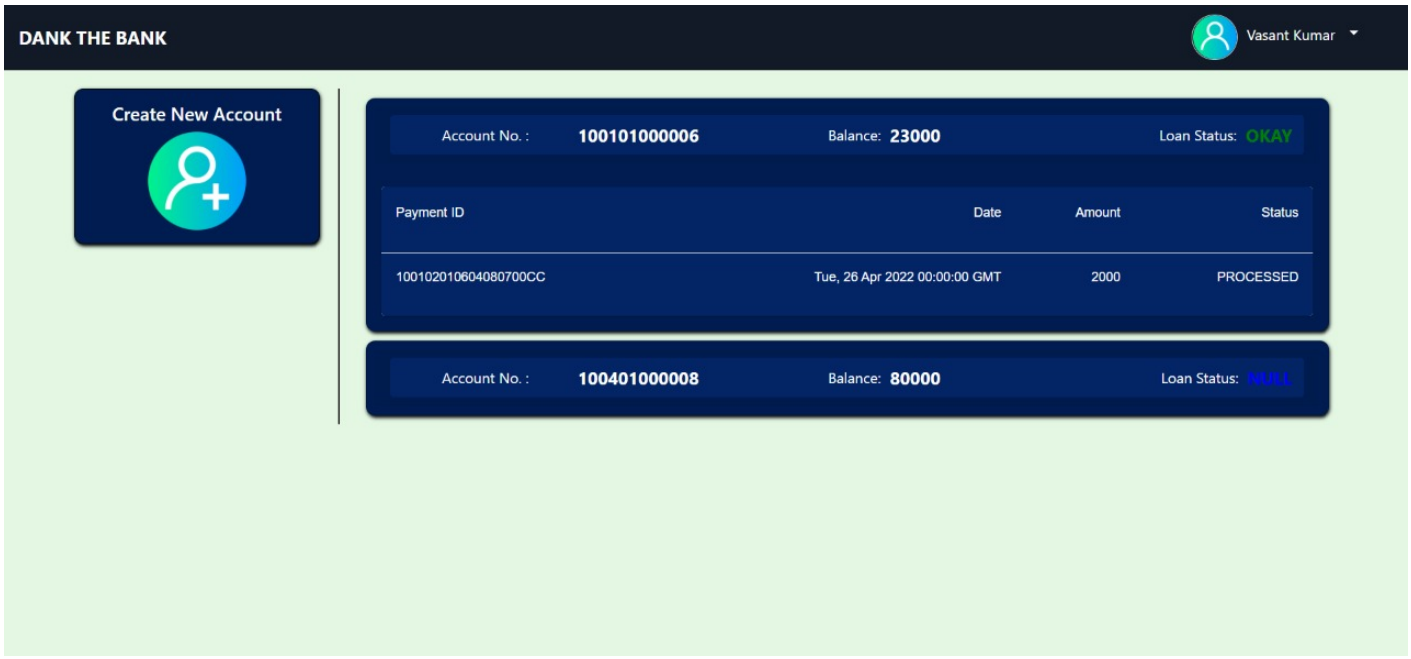


User Profile

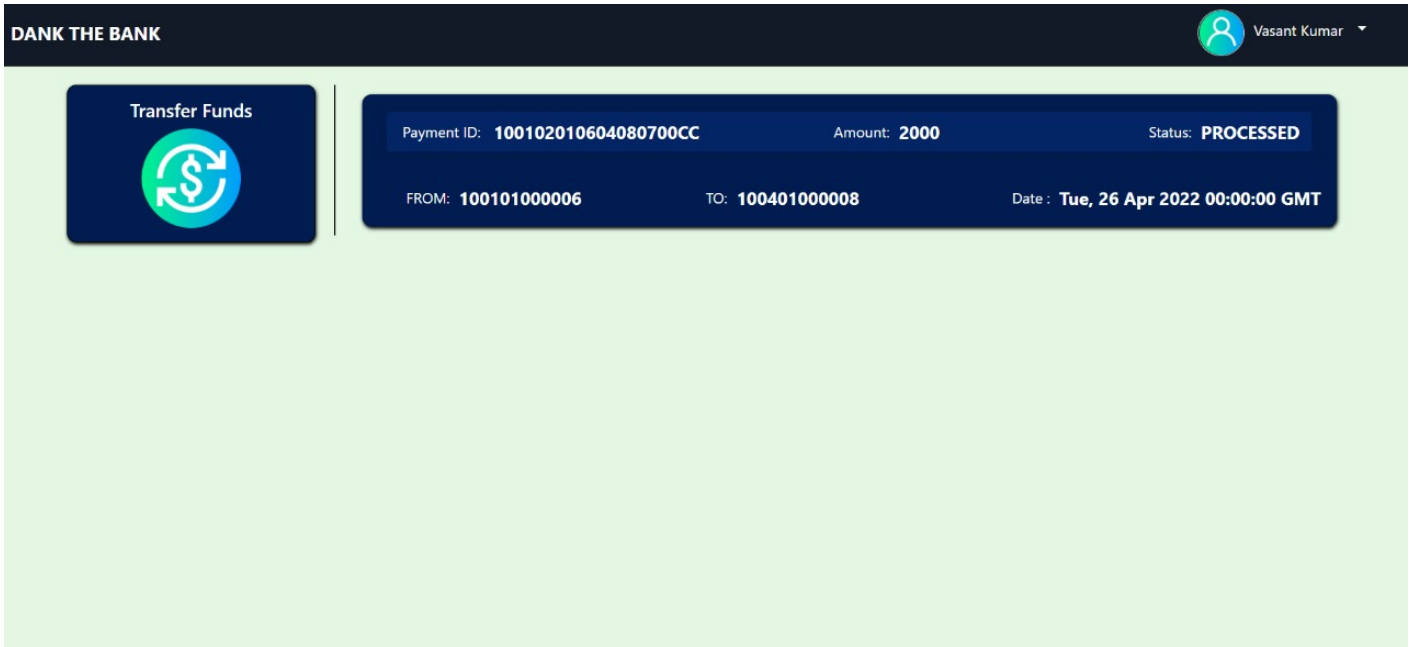


User Dashboard

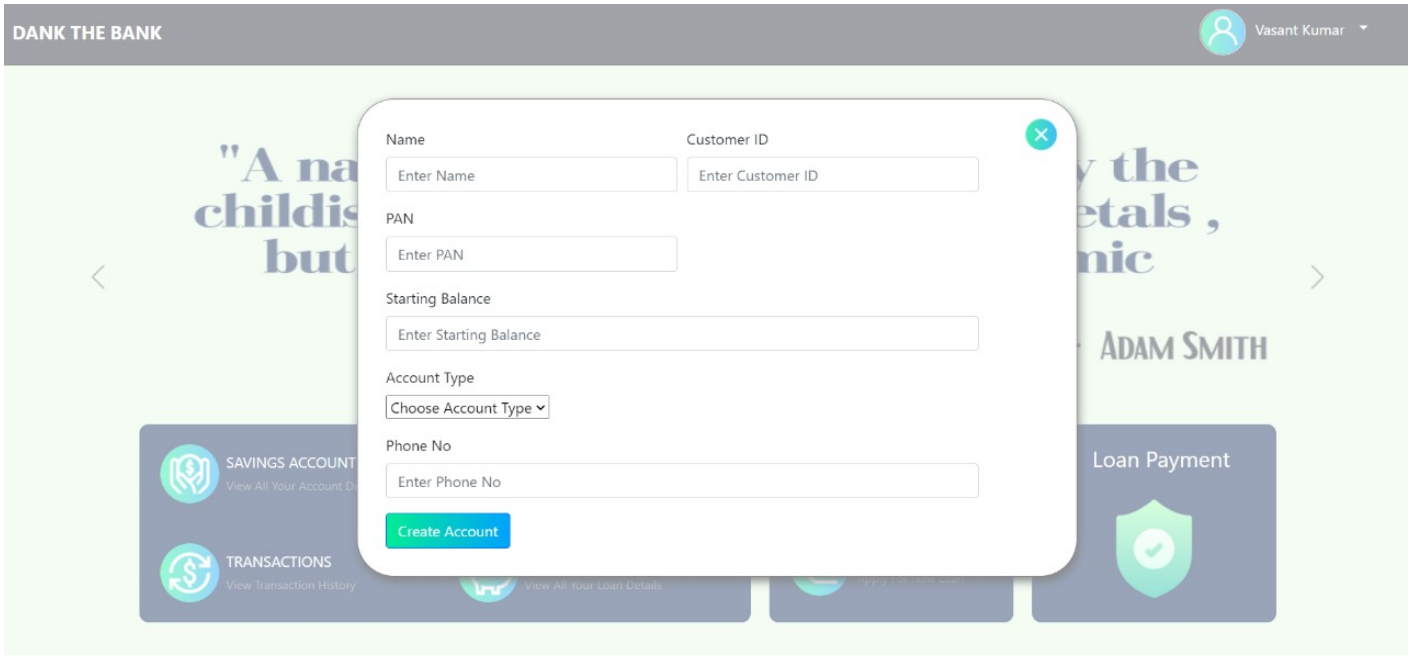
User Savings Account List



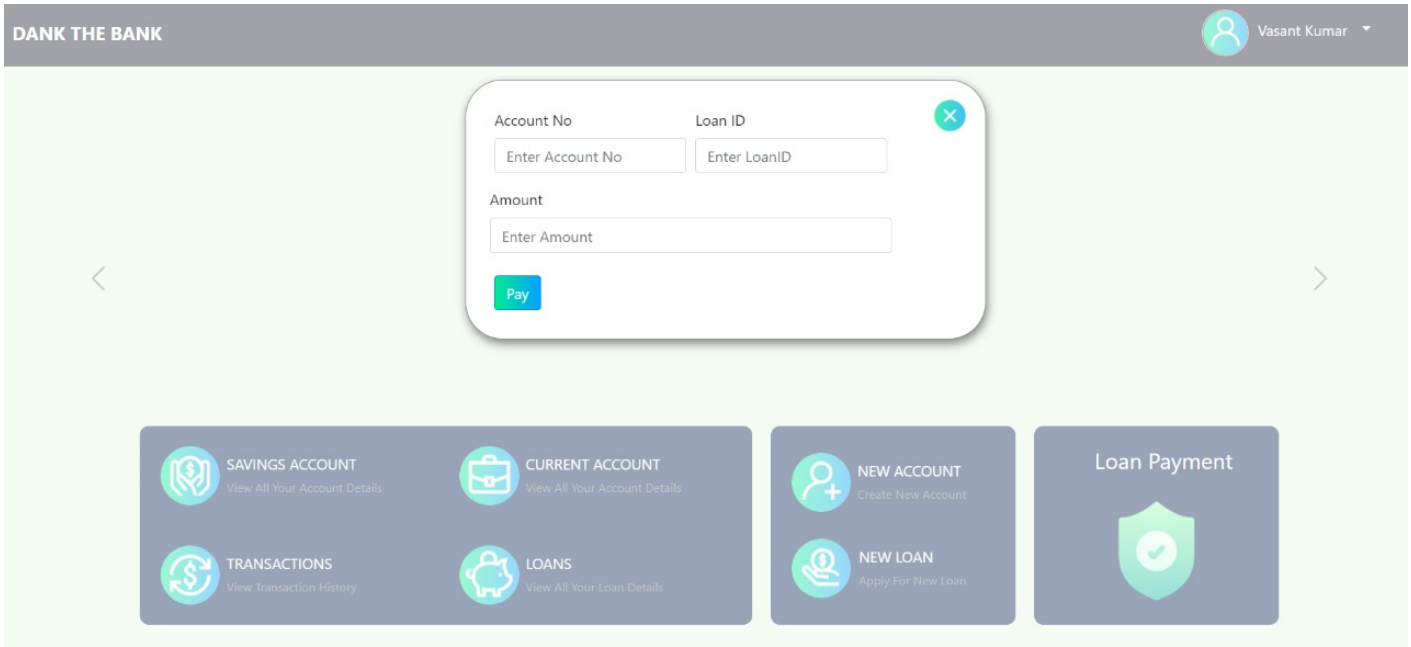
User Transactions



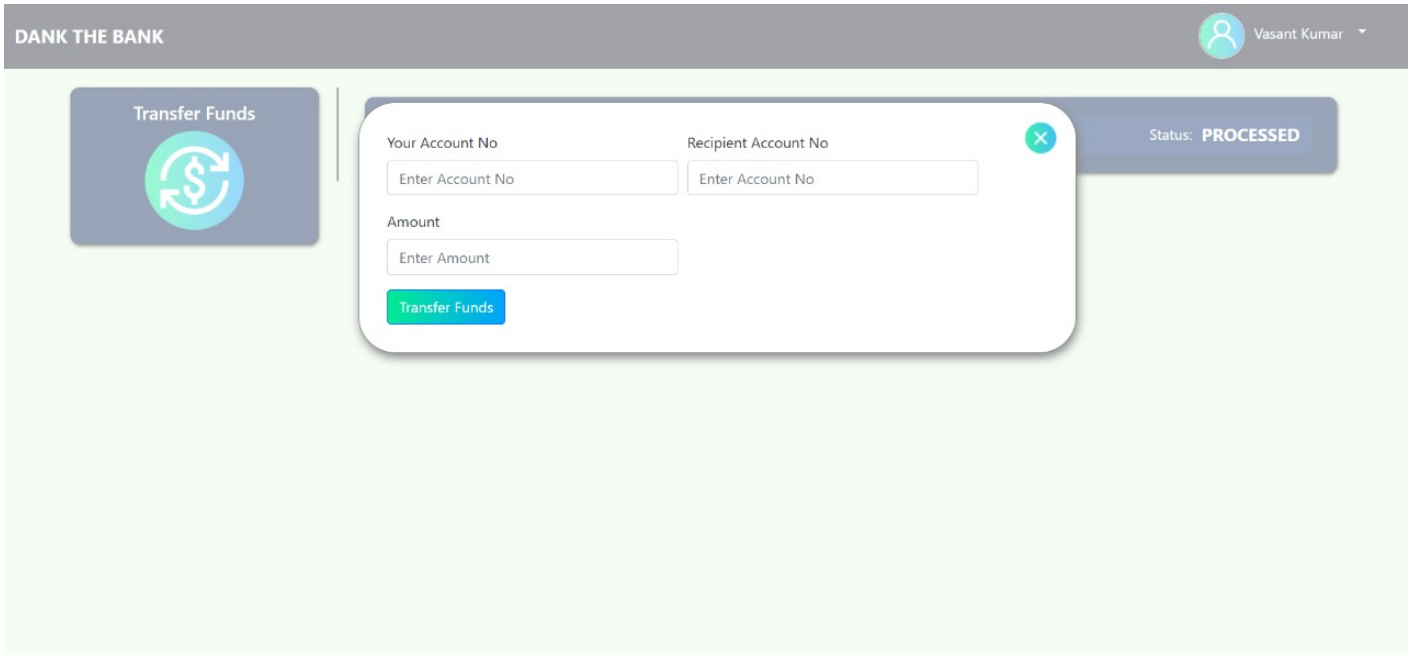
User New Account Form



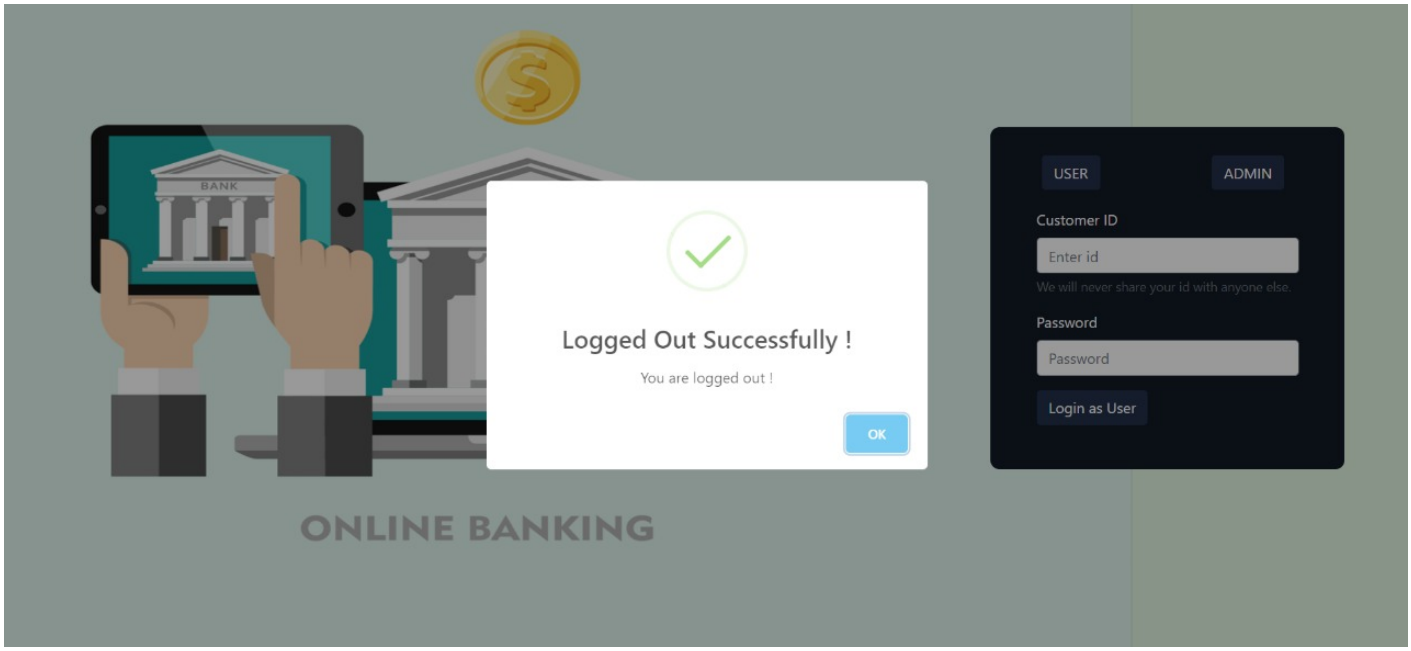
User Loan Payment Form



User Transaction Form



User Logout

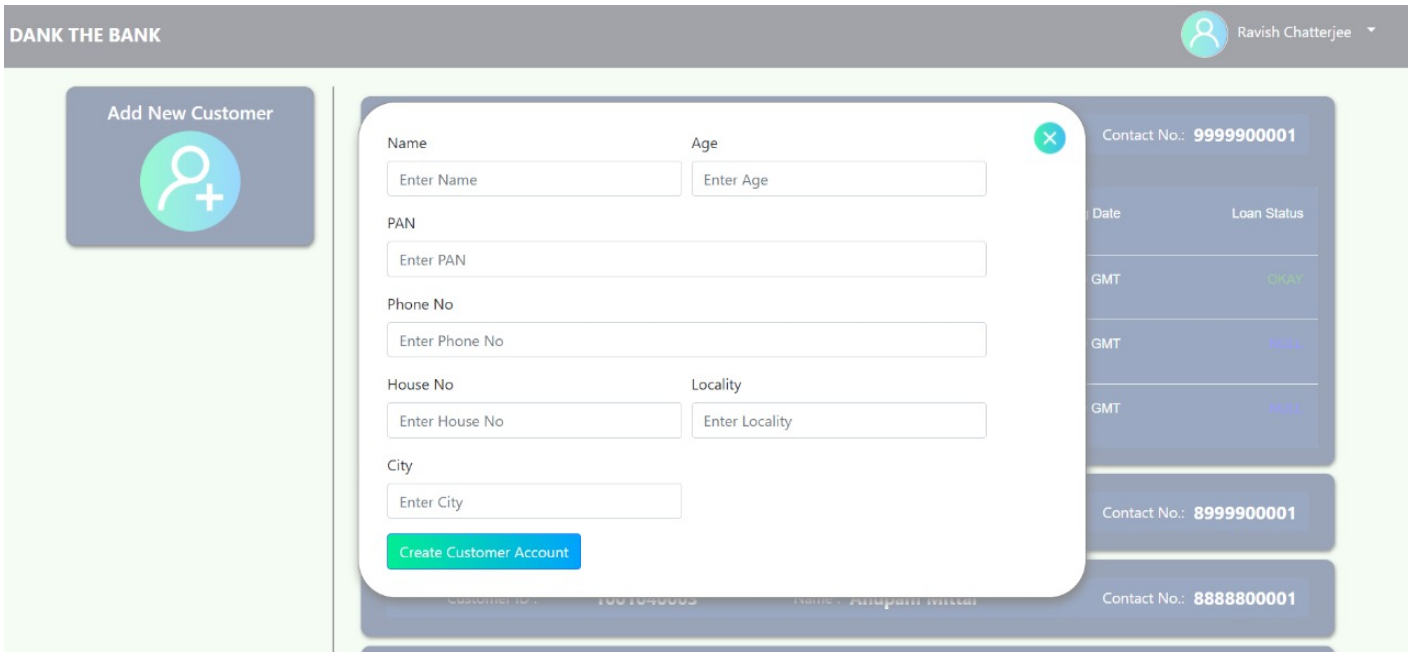


Admin Profile

Admin Dashboard

Branch's Customer List
(with their accounts)

Admin New Customer Form



Branch's Transaction History

Branch's Employee List