A

PROJECT REPORT

ON

BANKING SYSTEM

By

CHAUHAN RIKIN DIPAKKUMAR (CE-022) (19CEUOS106)
GANDEVIA KEVAL DHARMESHBHAI (CE-045) (19CEUEG017)
GANDHI JENIL JIGNESHBHAI (CE-046) (19CEUON133)

B.Tech CE Semester-V
Subject: Advanced Technologies (AT) (CE - 515)

Guided by:
Prof. Prashant M. Jadhav Associate Professor
Prof. Siddharth P. Shah Assistant Professor
Dept. of Computer Engg.

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of **Advanced Technologies (CE - 515)** and recorded in this journal is the bonafide work of

CHAUHAN RIKIN DIPAKKUMAR (CE-022) (19CEUOS106)
GANDEVIA KEVAL DHARMESHBHAI (CE-045) (19CEUEG017)
GANDHI JENIL JIGNESHBHAI (CE-046) (19CEUON133)

of B.Tech Semester **V** in the branch of **Computer Engineering** during the academic year **2021-2022**.

Project Guide:

Prof. Prashant M. Jadhav
Associate Professor,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Prof. Siddharth P. Shah
Assistant Professor,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Head of the Department
Dr. C. K. Bhensdadia,
Professor & Head,
Dept. of Computer Engg.,
Faculty of Technology
Dharmsinh Desai University, Nadiad

Table of Contents

1. Abstract

- When every day is a race against time in our busy lives, we are looking at saving time everywhere possible.

- When it comes to daily errands, online banking has made the visits to bank a rare occurrence.

- Thus “ONLINE BANKING” system provides a number of benefits like: Security, No Access Problem, No Hidden Fees, Convenience, Monitoring your Account and a lot more.

2. Introduction

2.1 Brief Introduction

- In this fast-moving world it is necessary to develop online banking system as it allows a user to conduct financial transactions via the internet.
- Online banking is also known as *Internet Banking* or *Web Banking*.
- Online banking offers customers every service traditionally available through a local branch including facilities like deposits, transfers, loans, and online bill payments.
- In online banking mode users aren't required to visit a bank branch in order to complete most of their basic banking transactions.
- A user only needs a device, an internet connection, and a bank card to register. Once registered, the user sets up a password to begin using the services.

2.2 Tools/Technologies Used

Technologies:

- Node.JS
- React.JS
- JavaScript
- jQuery
- CSS
- MongoDB
- Express.JS
- BootStrap

Tools:

- VS Code
- Git
- NPM

Platform:

- Local development server
- MongoDB Atlas database server

3. Software Requirement Specifications

3.1 Product Scope

This system is designed to provide the user online banking services.

3.2 System Functional Requirements

.

1. Admin Management:

R.1.1 Create user account:

- Description: Admin can create user accounts.
- Input: User details.
- Output: If correct details are provided, the user account would be created and pin for accessing the user account.

R.1.2 Block user account:

- Description: If any kind of fraudulent transactions are found or get requests from the user, then the user would be blocked.
- Input: Request from user or fraudulent transactions.
- Output: user account is blocked.

R.1.3 Transfer cash:

- Description: Admin has its own account from which cash is debited and cash from the user is credited to the admin's account.
- Input: Cash from the user.
- Output: Cash is transferred.

R.1.4 Approve loans:

- Description: Admin can approve loans that are requested from the user.
- Input: Loan details.

- Output: Loan is approved depending upon the situations.

R.1.5 Make user debit cards:

- Description: Based on the requests of the user, admin can make user debit cards.
- Input: Request of the user.
- Output: User debit cards would be created.

2. Loan Management:

R.2.1 Calculate loan amount:

- Description: Provided the collateral value and account number it calculates your allocable loan amount.
- Input: Collateral Value, Account number.
- Output: if successful it gives the loan amount that could be sanctioned else it gives -1.

R.2.2 Apply for loan:

- Description: This converts your loan application request to the actual loan.
- Input: Loan request id.
- Output: loan is granted or not. if loan is granted it returns the loan amount sanctioned and if not granted it returns -1.

R.2.3 View loan details:

- Description: The user can view his/her existing loans.
- Input: User selection.
- Output: list of all existing loans.

R.2.4 Refinance loan:

- Description: If the user wants to transfer the loan to some other company then he needs to refinance it.
- Input: user's request.
- Output: request logged in the customer handler admin of the user.

R.2.5 Redefine loan agreement:

- Description: If the user wishes to change any of the loan terms or mortgage a bigger collateral than the current one. He can do so.
- Input: user's request.
- Output: Loan might be redefined.

3. Fixed Deposit Management:

R.3.1 Make fixed deposit:

- Description: Users can apply for the fixed deposit or recurring deposit.
- Input: Deposit details.
- Output: Fixed deposit would be created.

R.3.2 View fixed deposit:

- Description: Users can view details of the existing fixed deposits.
- Input: User selection.
- Output: Details of the fixed deposits.

R.3.3 Close fixed deposit:

- Description: The fixed deposit could be closed based upon user's request.
- Input: User selection.
- Output: Fixed deposit is deleted and the money is transferred to the respected user.

R.3.4 Calculate fixed deposit amount:

- Description: Provided the deposit amount, deposit type and deposit tenure it calculates your maturity amount of your deposited amount.
- Input: Deposit amount, deposit type and deposit tenure.
- Output: If successful it gives the calculated maturity amount of your deposited amount.

4. Account Management:

R.4.1 View all accounts:

- Description: User can see the details of the current accounts.
- Input: User selection.
- Output: All user accounts are shown.

R.4.2 See transactions:

- Description: User can see transactions of all his accounts.
- Input: User selection.
- Output: Users can see transaction history of all his accounts also he could sort them by money or date.

R.4.3 Get statistics of all accounts:

- Description: Users can get statistics of all their accounts.
- Input: User selection.
- Output: Users can see statistics of their account.

5. Card Management:

R.5.1 Card transfer:

- Description: User can transfer money / make payment from his / her account using their card.
- Input: Transfer details.
- Output: If transfer is valid, a success message is displayed.

R.5.2 Apply for cards:

- Description: User can apply for an application of card on a particular account number.
- Input: Account number.
- Output: Request might be granted depending on the funds of the user.

R.5.3 Change pin of the card:

- Description: Users can change the pin of the card.
- Input: User selection.
- Output: Pin would be changed.

R.5.4 Renew Card:

- Description: Once card validity is over, users can renew the card.
- Input: User selection.
- Output: Card would be renewed.

R.5.5 Close card:

- Description: Users can close cards.
- Input: Request to the admin.
- Output: Card would be closed.

6. Fund-Transfer Management:

R.6.1 RTGS transfer:

- Description: Transfers money in RTGS mode to other users.
- Input: Beneficiary name, Beneficiary Account Number, Amount to be transferred, IFSC code of the bank.
- Output: Money transactions settled through RTGS procedure.

R.6.2 NEFT transfer:

- Description: Transfers money through NEFT mode to other users.
- Input: Beneficiary name, beneficiary Account Number, Amount to be transferred, IFSC code of the bank.
- Output: Money transactions settled through NEFT procedure.

R.6.3 Card transfer:

- Description: User can transfer money / make payment from his / her account using their card.
- Input: Transfer details.
- Output: If transfer is valid, a success message is displayed.

3.3 Other Non-functional Requirements

1. Performance

The system must be interactive and must not involve long delays. Though in case of opening the website components or loading the page the system shows the delays less than 2 seconds.

2. Safety

The user's data is highly personal. The system has authorization to avoid any un-authorized access to user's private data.

3. Reliability

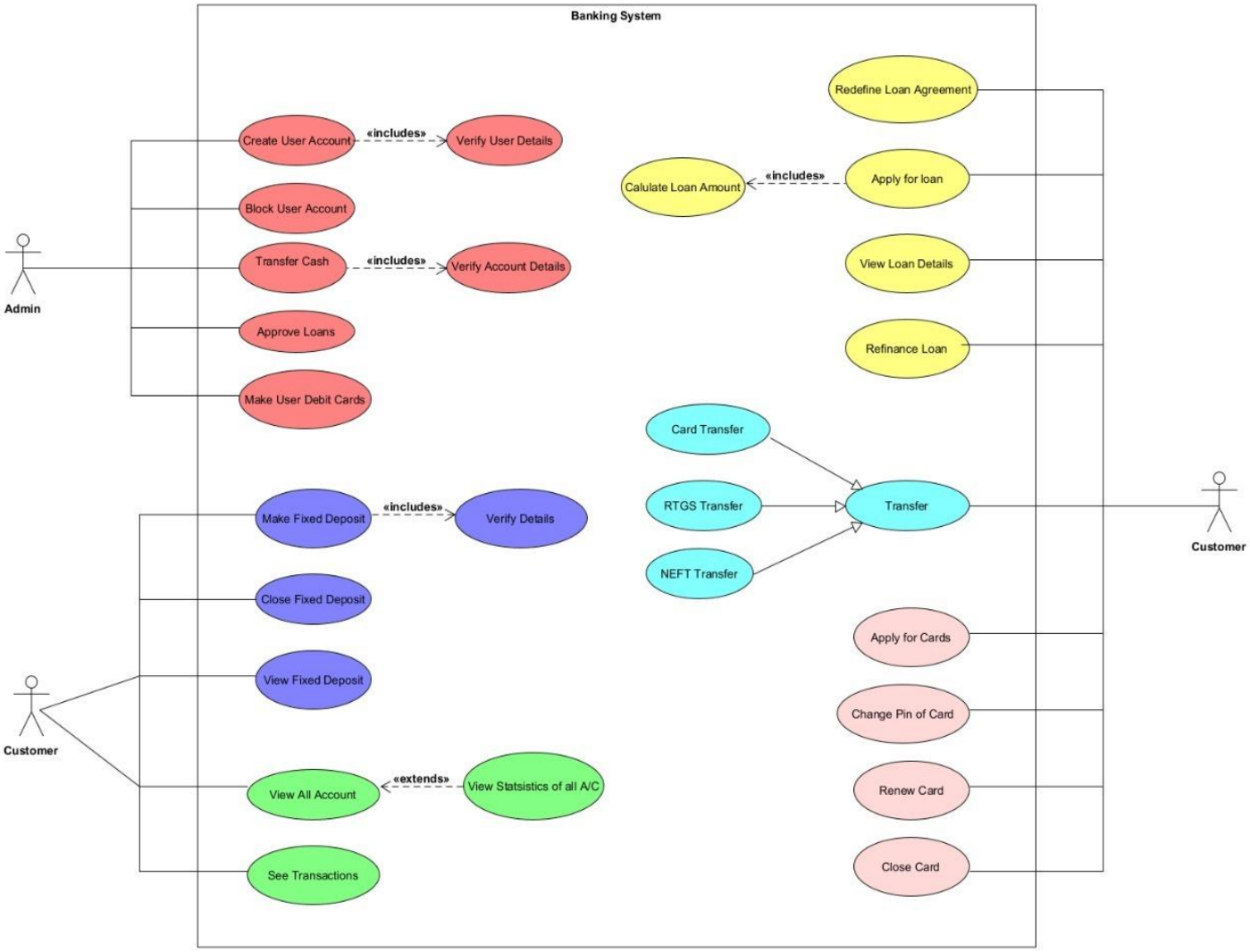
As the system has personal data, its reliability is the major factor for consideration.

4. Database

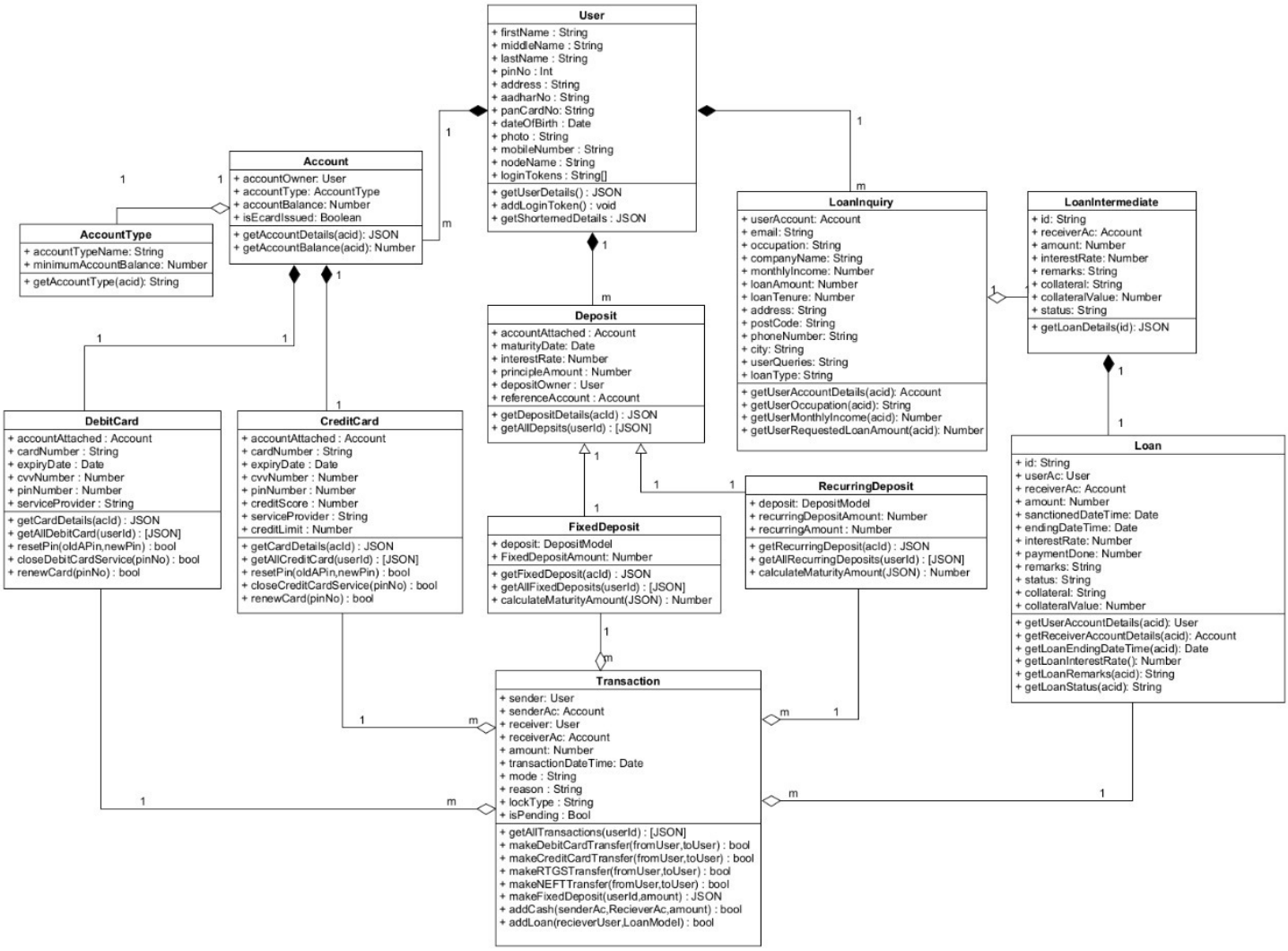
System requires to access user profile data fast to maintain the performance.

4. Design

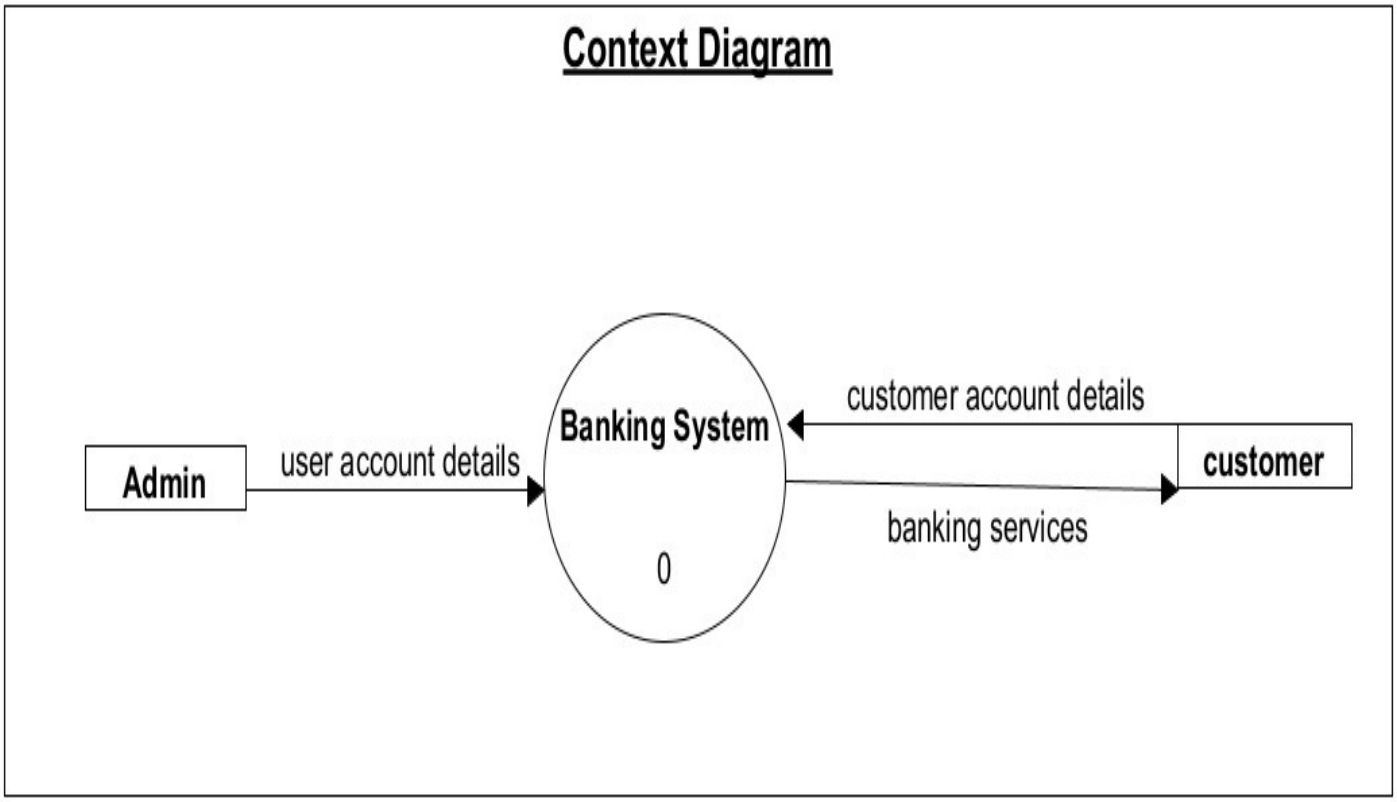
4.1 Use Case Diagram



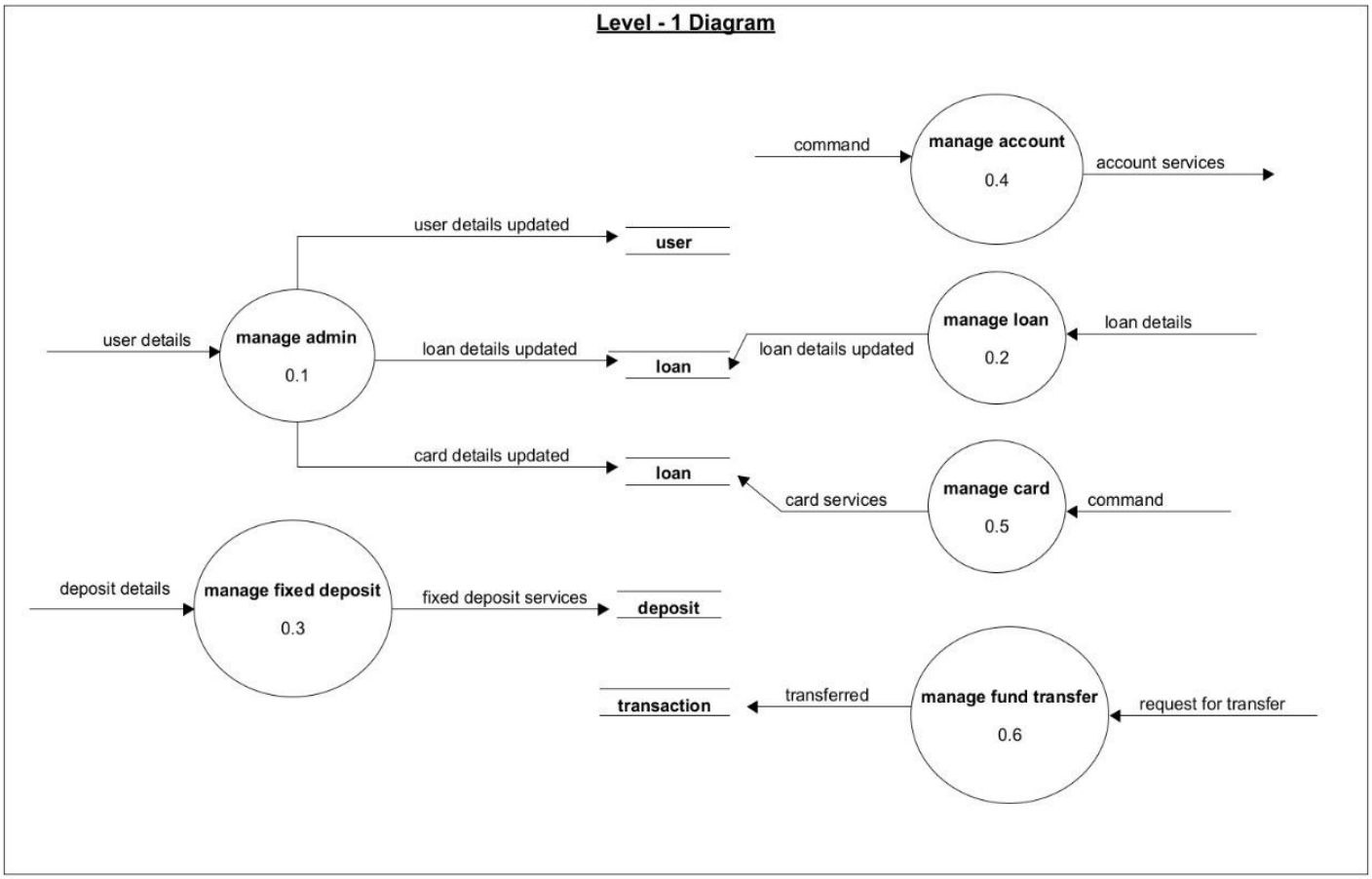
4.2 Class Diagram



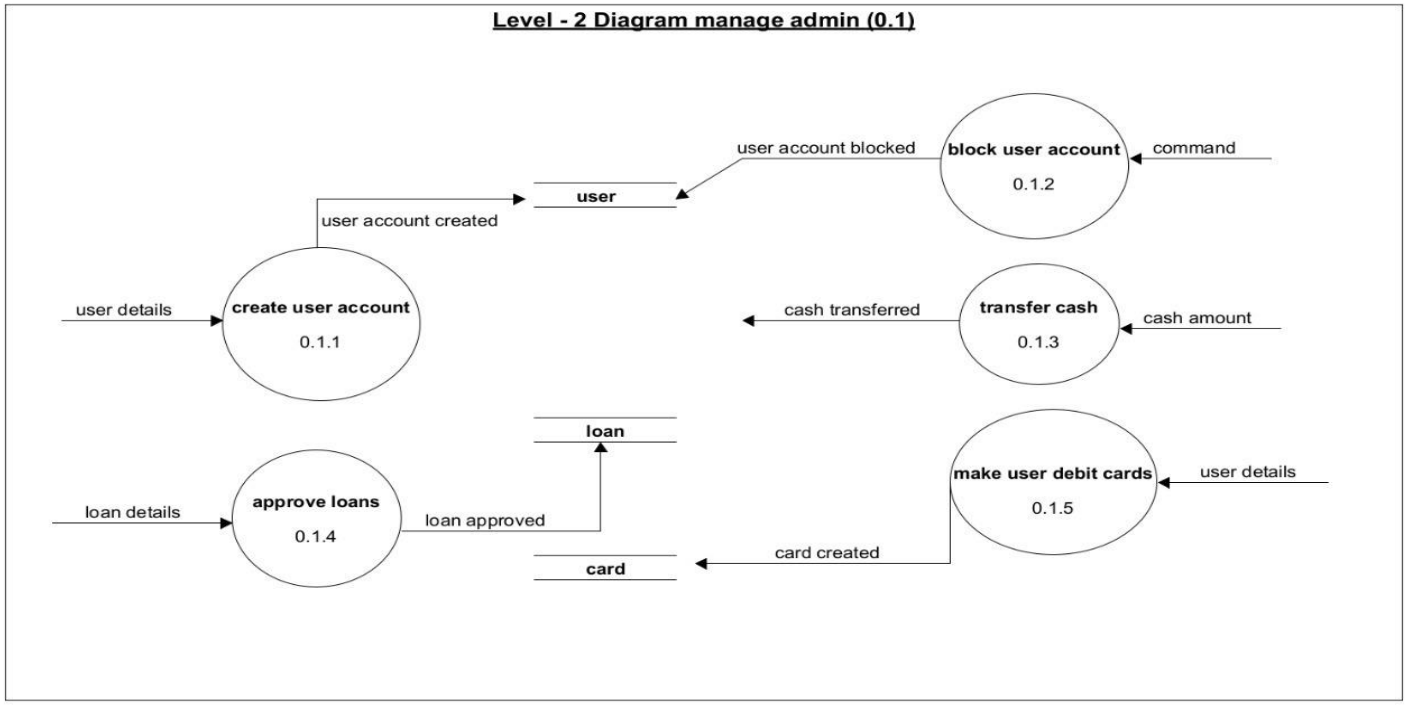
4.3 DFD Diagram



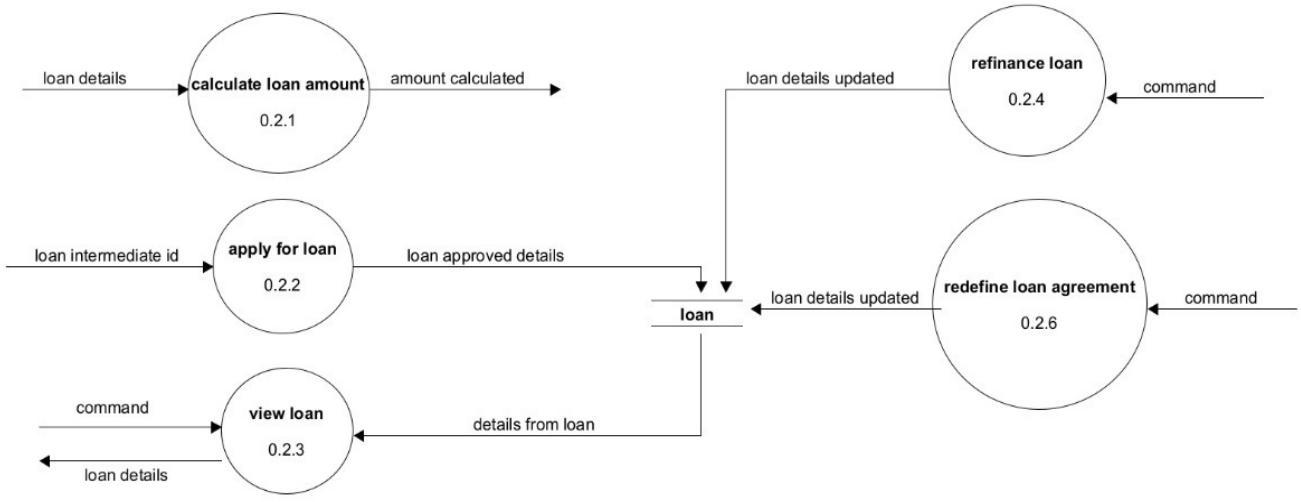
Level - 1 Diagram



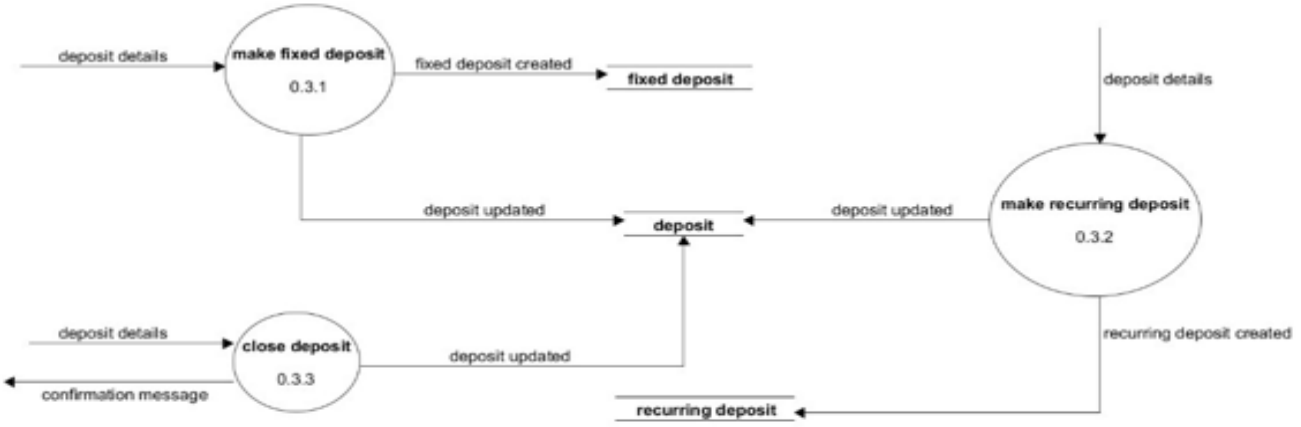
Level - 2 Diagram manage admin (0.1)



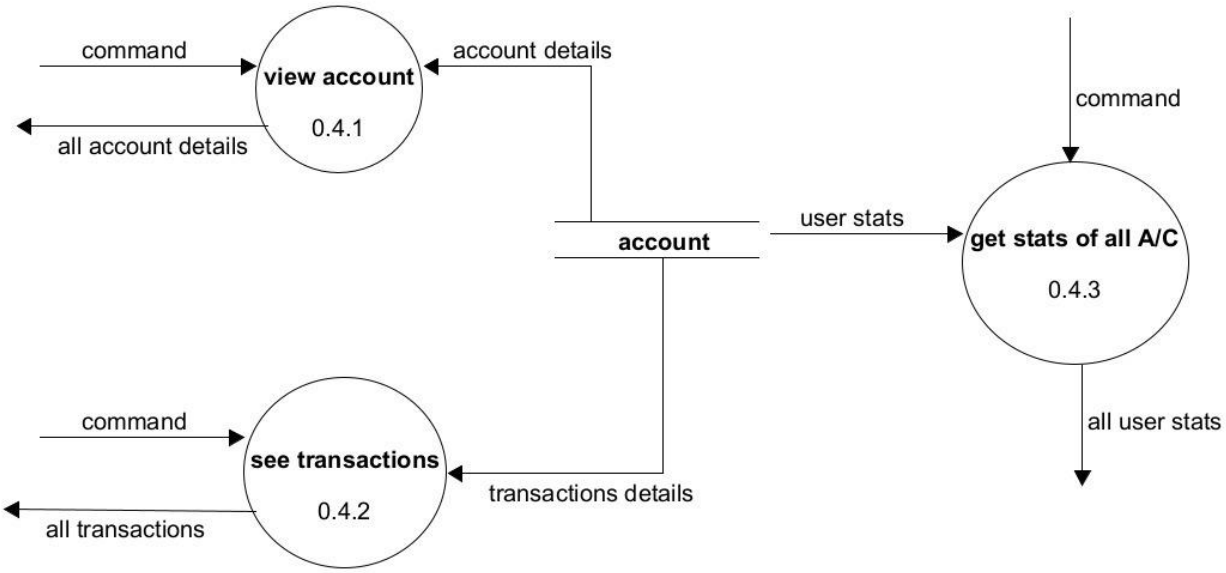
Level - 2 Diagram manage loan (0.2)



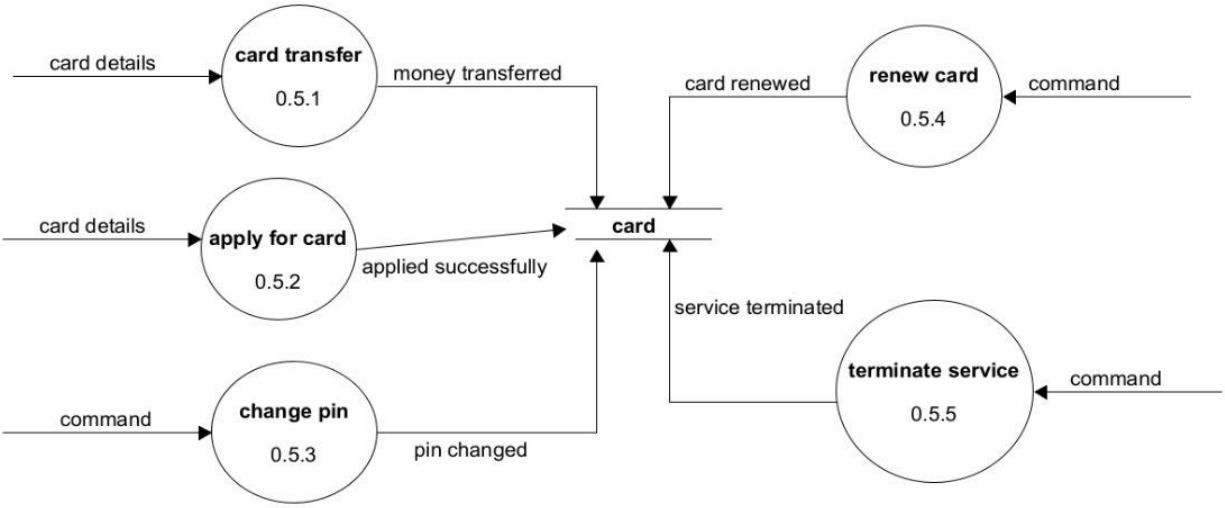
Level - 2 Diagram manage fixed deposit (0.3)

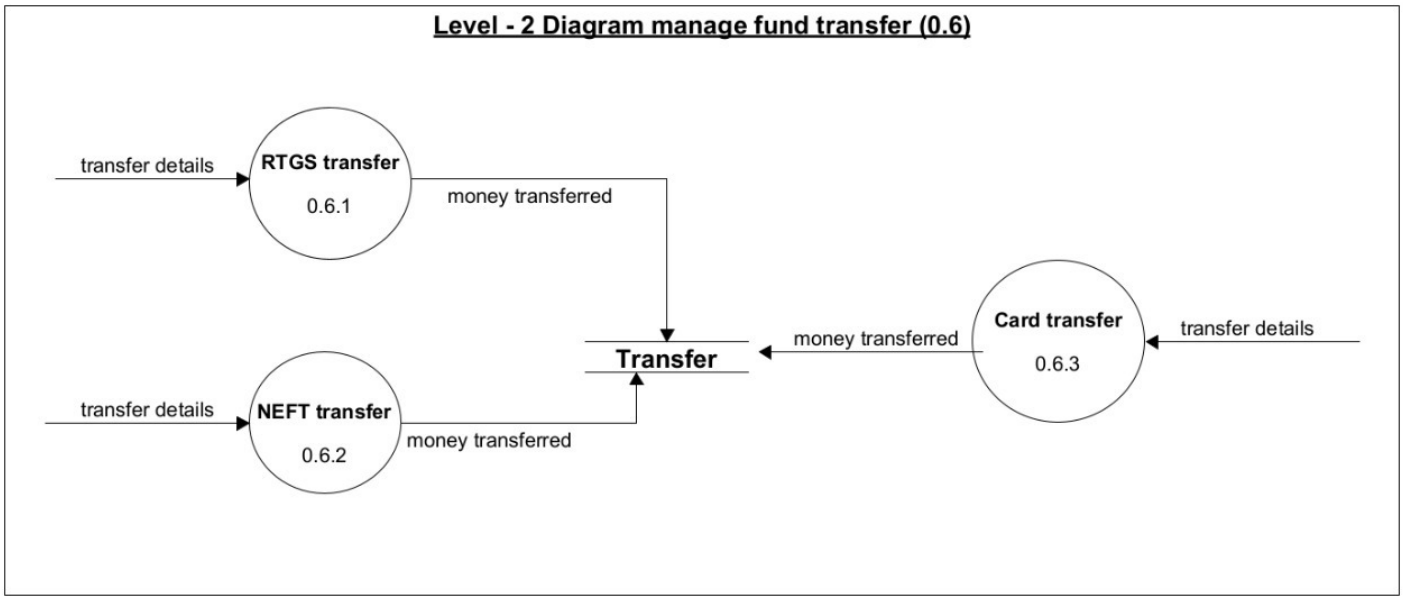


Level - 2 Diagram manage account (0.4)

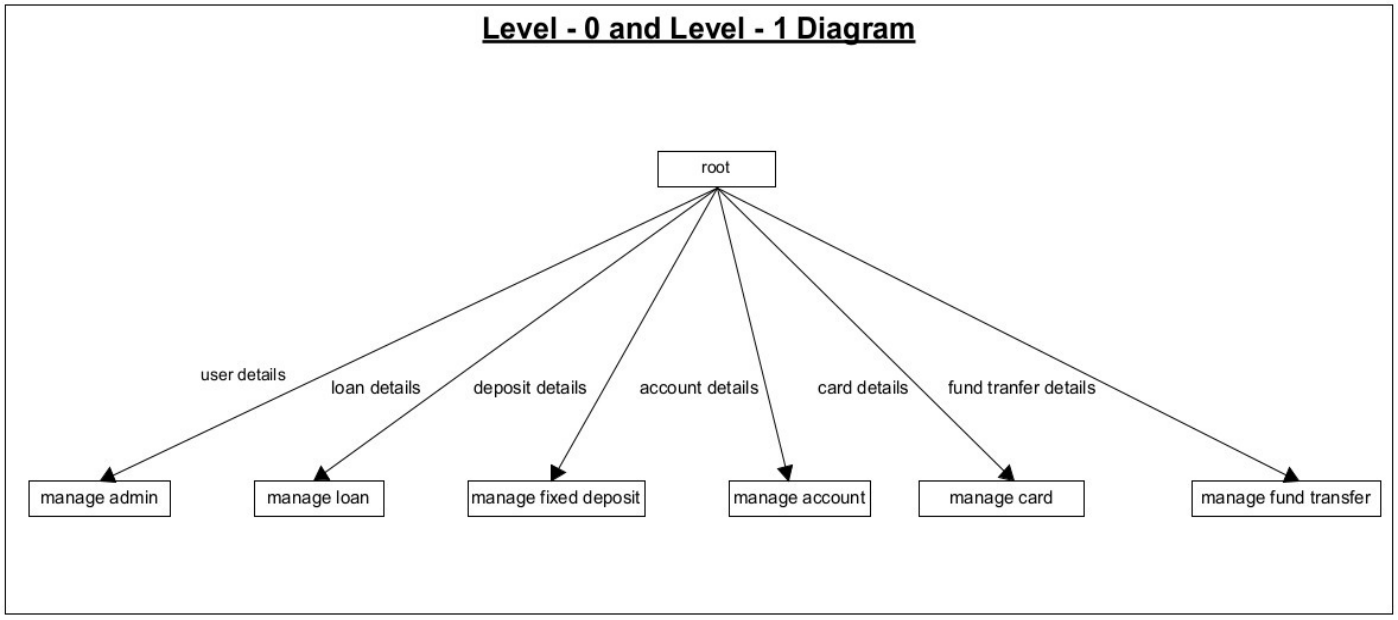


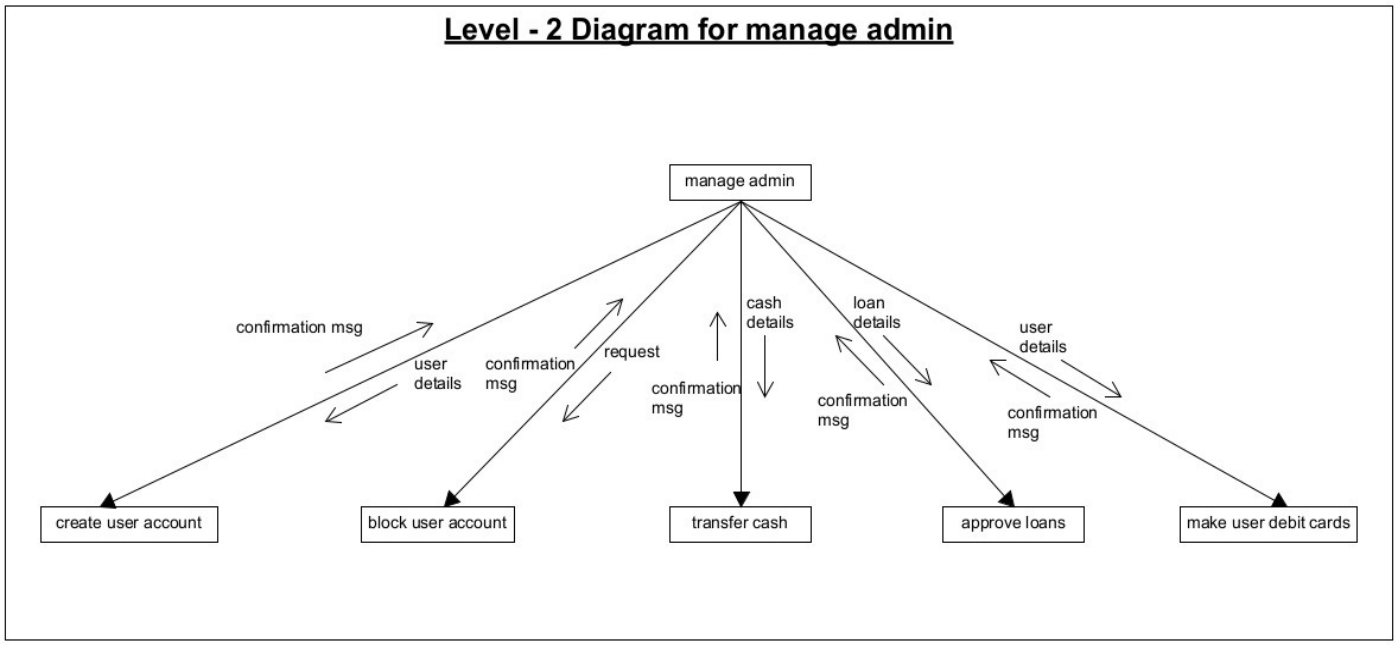
Level - 2 Diagram manage card (0.5)



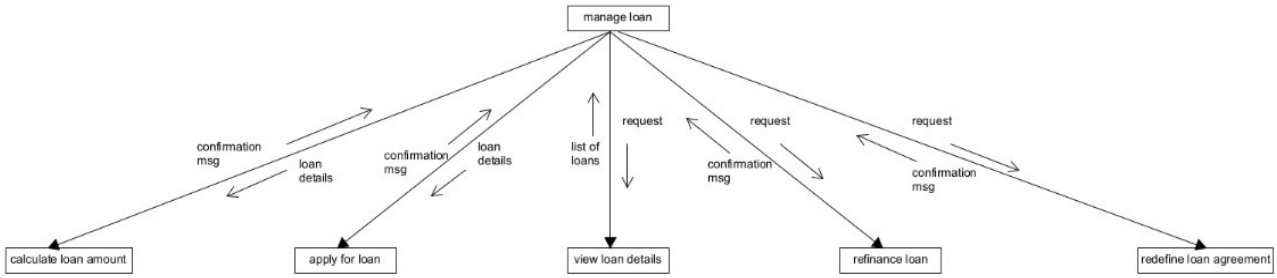

Level - 2 Diagram manage fund transfer (0.6)
transfer details
RTGS transfer
0.6.1
money transferred
Transfer
money transferred
Card transfer
0.6.3
transfer details
transfer details
NEFT transfer
0.6.2
money transferred

4.4 Structure Chart

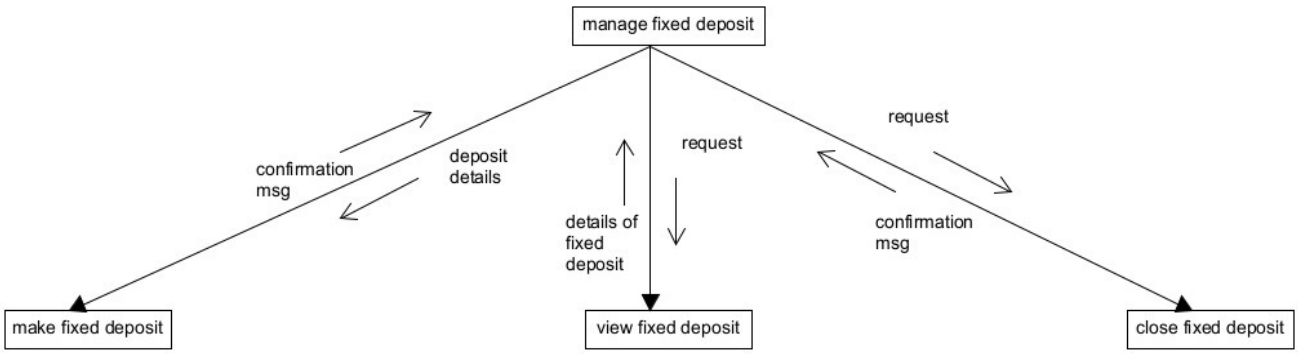




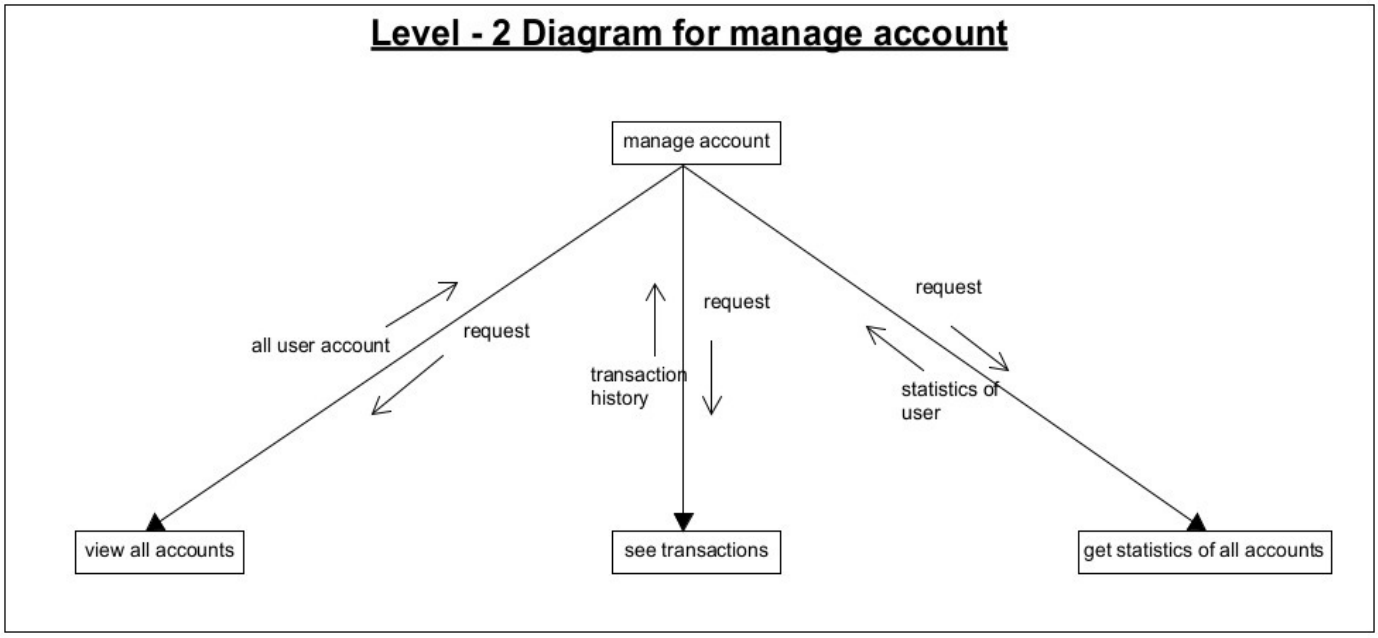
Level - 2 Diagram for manage loan



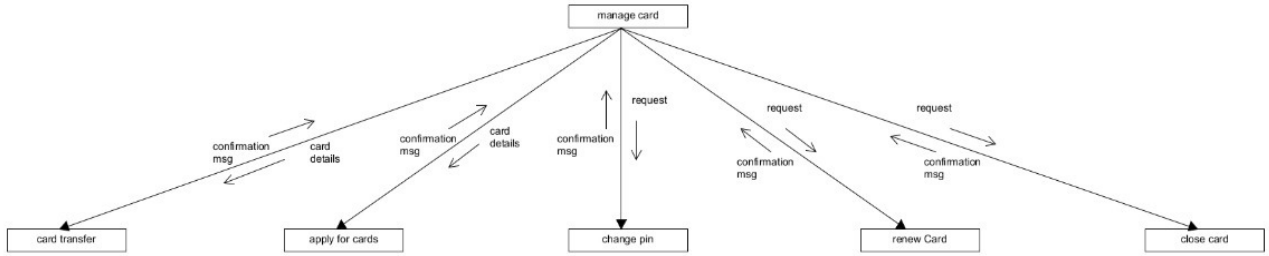
Level - 2 Diagram for manage fixed deposit



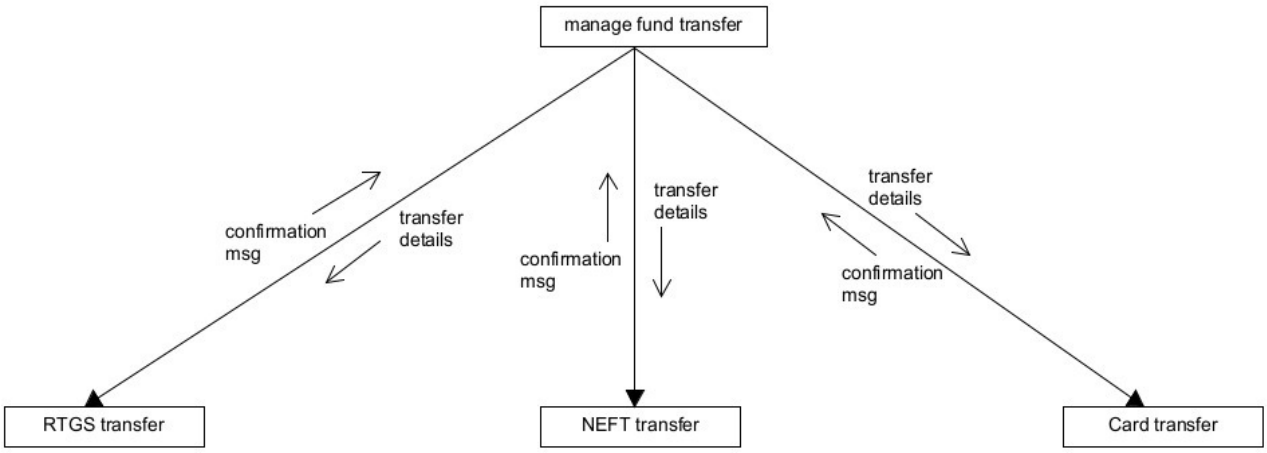
Level - 2 Diagram for manage account



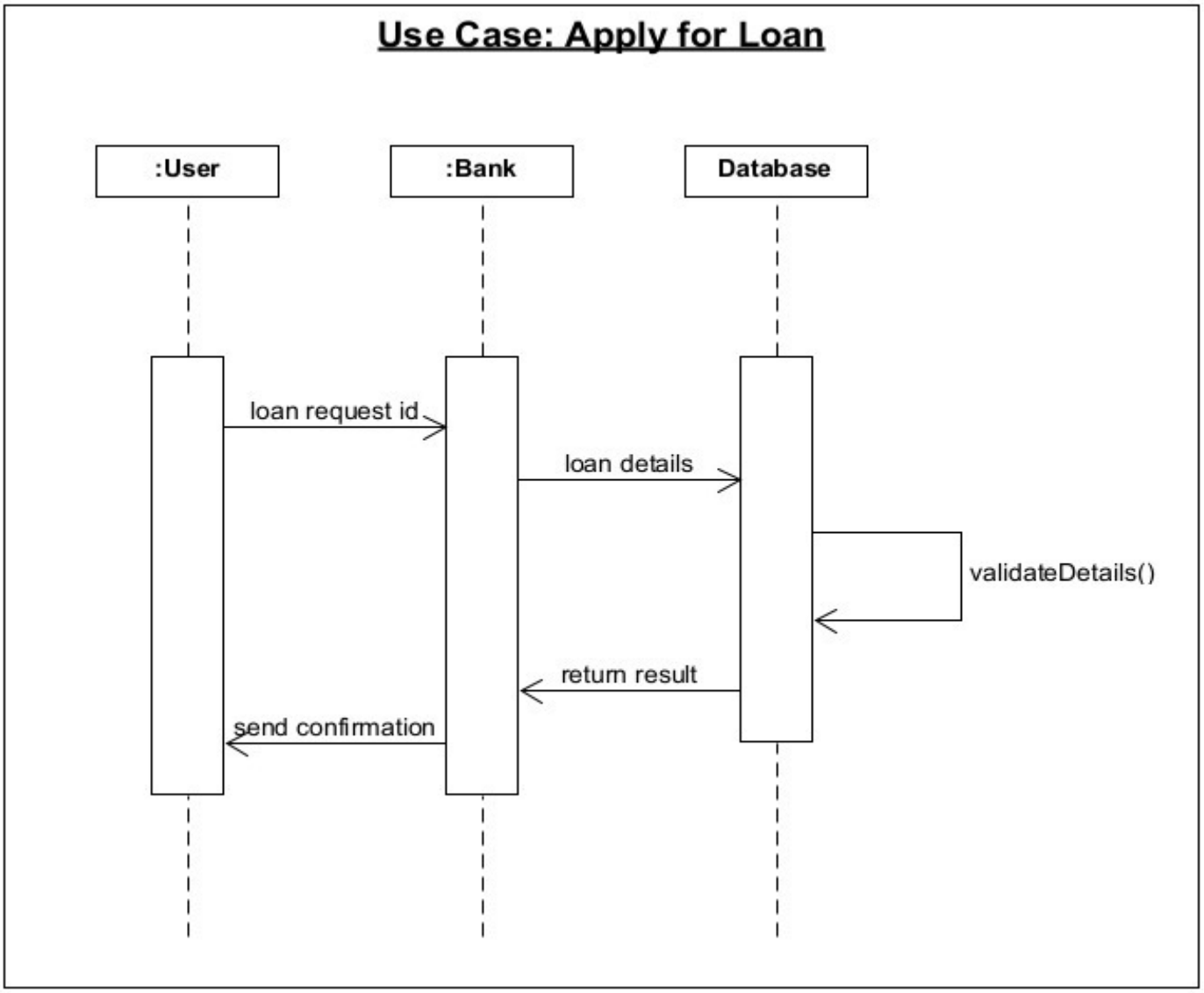
Level - 2 Diagram for manage card

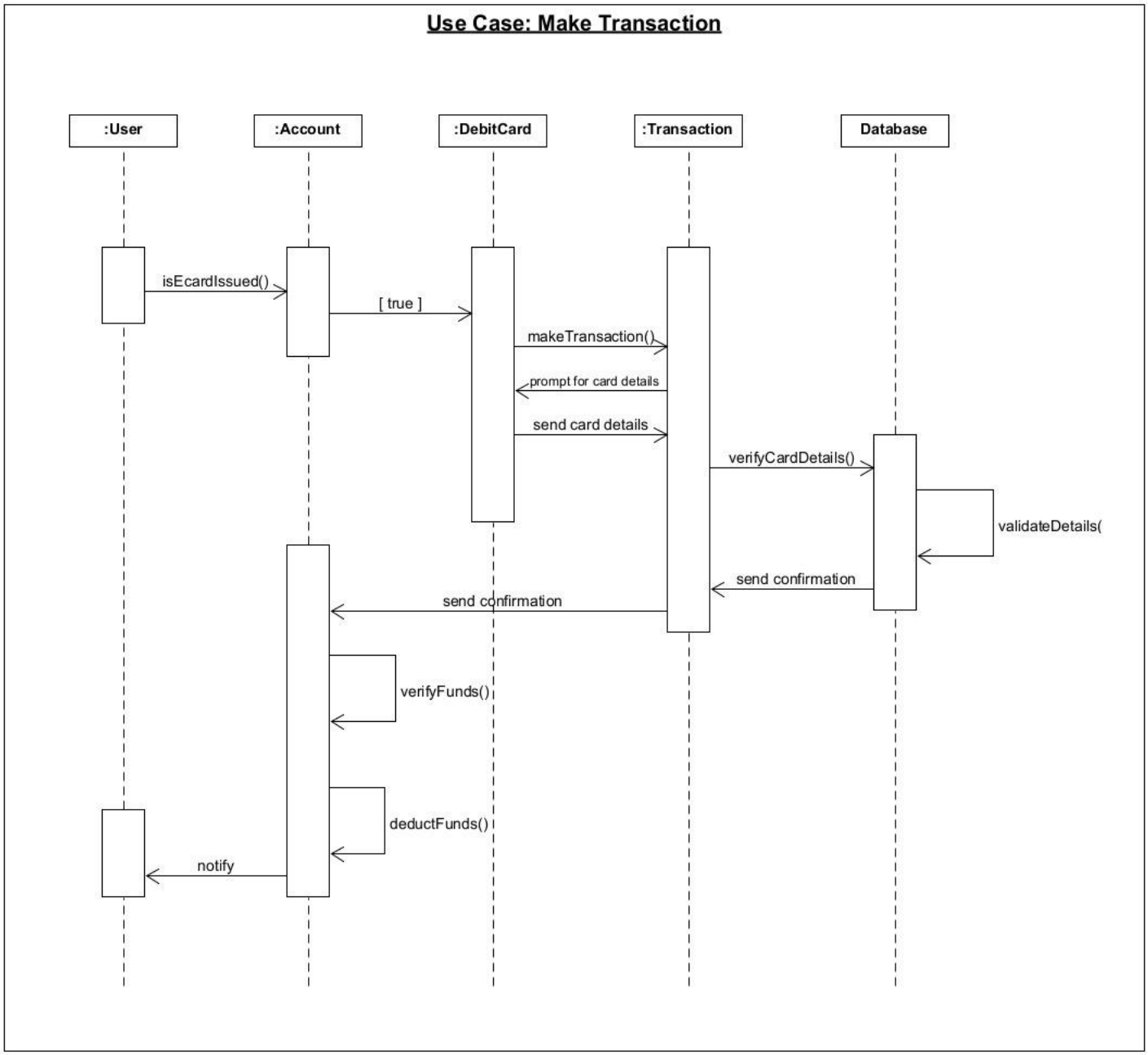


Level - 2 Diagram for manage fund transfer

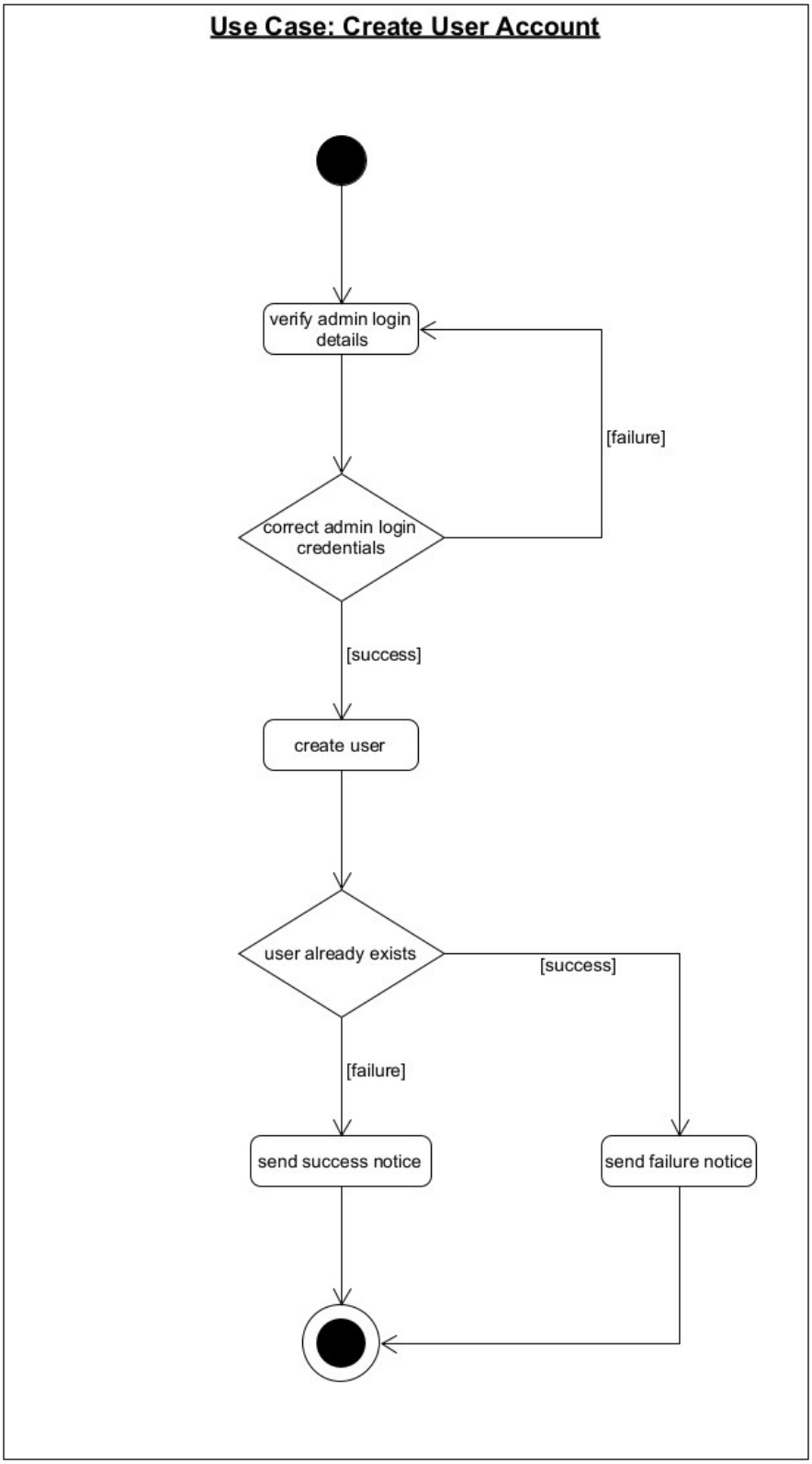


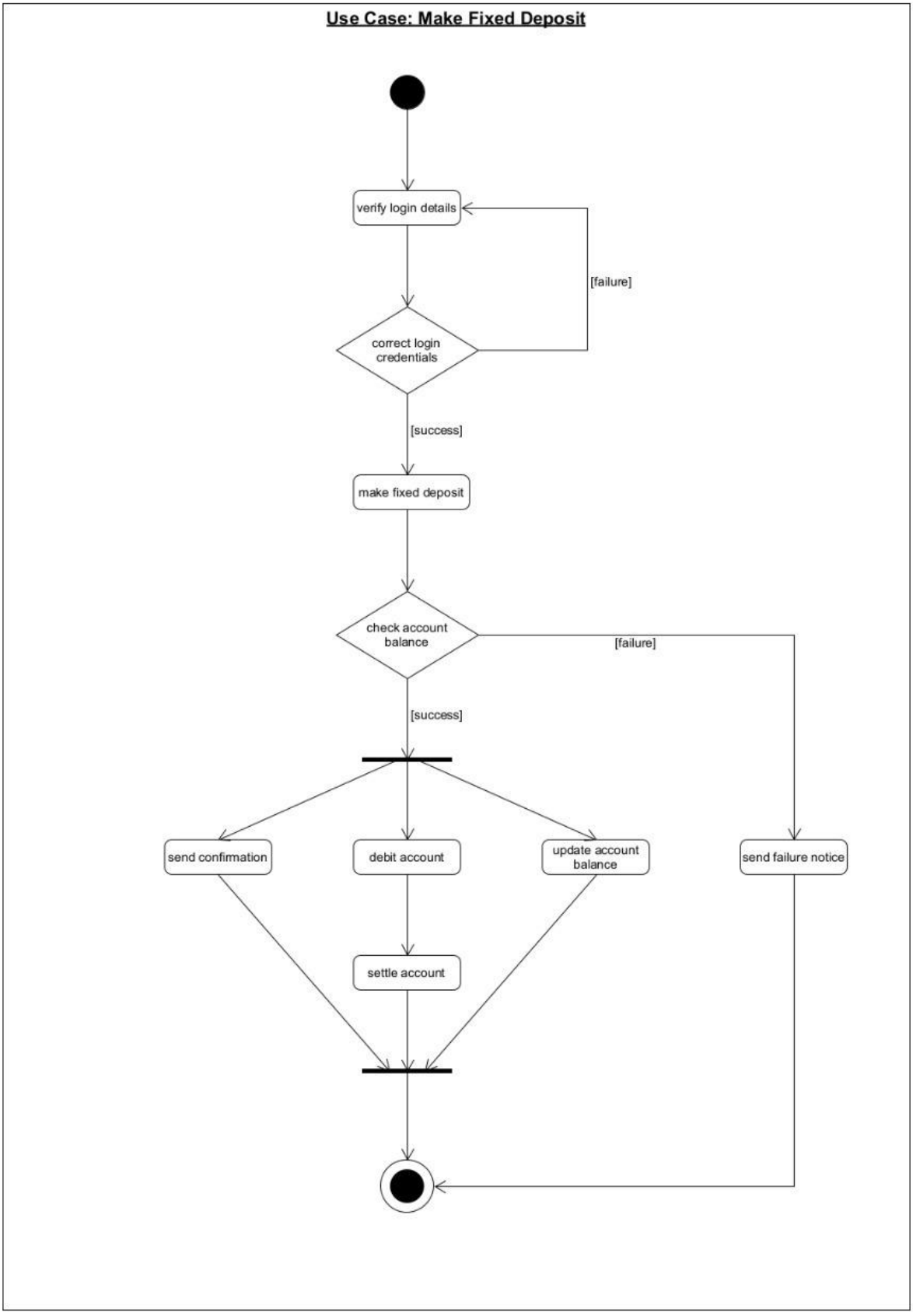
4.5 Sequence Diagram




Use Case: Make Transaction
:User
:Account
:DebitCard
:Transaction
Database
isEcardIssued()
[true]
makeTransaction()
prompt for card details
send card details
verifyCardDetails()
validateDetails(
send confirmation
send confirmation
verifyFunds()
deductFunds()
notify

4.6 Activity Diagram




Use Case: Make Fixed Deposit
verify login details
[failure]
correct login credentials
[success]
make fixed deposit
check account balance
[failure]
[success]
send confirmation
debit account
update account balance
send failure notice
settle account

4.7 Data Dictionary

Account Model

```
accountOwner: { type: [Function], ref: 'UserModel', required: true },
accountType: { type: [Function], ref: 'AccountTypeModel', required: true },
accountBalance: { type: [Function: Number], required: true },
isEcardIssued: { type: [Function: Boolean], required: true },
```

Account Type Model

```
accountTypeName: { type: [Function: String], required: true },
minimumAccountBalance: { type: [Function: Number], required: true },
```

Admin Model

```
firstName: { type: [Function: String], required: true },
middleName: { type: [Function: String], required: true },
lastName: { type: [Function: String], required: true },
pinNo: { type: [Function: String], required: true },
loginTokens: [ { token: [Object] } ],
```

Credit Card Model

```
accountAttached: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
cardNumber: { type: [Function: String], required: true },
expiryDate: { type: [Function: Date], required: true },
cvvNumber: { type: [Function: Number], required: true },
pinNumber: { type: [Function: Number], required: true },
creditScore: { type: [Function: Number], required: true },
serviceProvider: { type: [Function: String], required: true },
creditLimit: { type: [Function: Number], required: true },
```

Debit Card Model

```
accountAttached: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
cardNumber: { type: [Function: String], required: true },
expiryDate: { type: [Function: Date], required: true },
cvvNumber: { type: [Function: Number], required: true },
pinNumber: { type: [Function: Number], required: true },
serviceProvider: { type: [Function: String], required: true },
```

Deposit Model

```
dateOfIssue: { type: [Function: Date], required: true },
maturityDate: { type: [Function: Date], required: true },
interestRate: { type: [Function: Number], required: true },
principleAmount: { type: [Function: Number], required: true },
depositOwner: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'UserModel',
  required: true
},
referenceAccount: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
```

Fixed Deposit Model

```
deposit: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'DepositModel',
  required: true
},
FixedDepositAmount: { type: [Function: Number], required: true },
_id: { auto: true, type: 'ObjectId' },
__v: [Function: Number],
id: VirtualType {
  path: 'id',
  getters: [ [Function: idGetter] ],
  setters: [],
  options: {}
}
```

Loan Inquiry Model

```
userAccount: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
email: { type: [Function: String], required: true },
occupation: { type: [Function: String], required: true },
companyName: { type: [Function: String], required: true },
monthlyIncome: { type: [Function: Number], required: true },
loanAmount: { type: [Function: Number], required: true },
loanTenure: { type: [Function: Number], required: true },
address: { type: [Function: String], required: true },
postCode: { type: [Function: Number], required: true },
phoneNumber: { type: [Function: String], required: true },
city: { type: [Function: String], required: true },
userQueries: { type: [Function: String], required: true },
loanType: { type: [Function: String], required: true },
```

Loan Intermediate Model

```
receiverAc: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
amount: { type: [Function: Number], required: true },
interestRate: { type: [Function: Number], required: true },
remarks: { type: [Function: String], required: true },
collateral: { type: [Function: String], required: true },
collateralValue: { type: [Function: Number], required: true },
status: { type: [Function: String], required: true },
```

Loan Model

```
userAc: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'UserModel',
  required: true
},
receiverAc: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
amount: { type: [Function: Number], required: true },
sanctionedDateTime: { type: [Function: Date], required: true },
endingDateTime: { type: [Function: Date], required: true },
interestRate: { type: [Function: Number], required: true },
paymentDone: { type: [Function: Number], required: true },
remarks: { type: [Function: String], required: true },
status: { type: [Function: String], required: true },
collateral: { type: [Function: String], required: true },
collateralValue: { type: [Function: Number], required: true },
```

OTP Model

```
transaction: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'TransactionModel',
  required: true
},
OTP: { type: [Function: Number], required: true },
```

Recurring Deposit Model

```
deposit: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'DepositModel',
  required: true
},
recurringDepositAmount: { type: [Function: Number], required: true },
```

Transaction Model

```
sender: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'UserModel',
  required: true
},
senderAc: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
receiver: {
  type: [Function: ObjectId] {
    schemaName: 'ObjectId',
    defaultOptions: {},
    get: [Function (anonymous)],
    set: [Function: set],
    _checkRequired: [Function (anonymous)],
    _cast: [Function: castObjectId],
    cast: [Function: cast],
    _defaultCaster: [Function (anonymous)],
    checkRequired: [Function (anonymous)]
  },
  ref: 'AccountModel',
  required: true
},
```

```
amount: { type: [Function: Number], required: true },
transactionDateTime: { type: [Function: Date], required: true },
mode: { type: [Function: String], required: true },
reason: { type: [Function: String], required: true },
lockType: { type: [Function: String], required: false },
isPending: { type: [Function: Boolean], required: true },
```

User Model

```
firstName: { type: [Function: String], required: true },
middleName: { type: [Function: String], required: true },
lastName: { type: [Function: String], required: true },
pinNo: { type: [Function: String], required: true },
address: { type: [Function: String], required: true },
aadharNo: { type: [Function: String], required: true },
panCardNo: { type: [Function: String], required: true },
DOB: { type: [Function: Date], required: true },
mobileNo: { type: [Function: String], required: true },
emailId: { type: [Function: String], required: true },
photo: { type: [Function: String], required: false },
nodeName: { type: [Function: String], required: true },
loginTokens: [ { token: [Object] } ],
```

5. Implementation Details

The system consists of 6 basic modules namely

1. Admin Module
2. Loan Module
3. Fixed-Deposit Module
4. Account Module
5. Card Module
6. Fund-Transfer Module

Each module consists of several methods to implement the required functionality. Implementation is done using JavaScript, NodeJS, jQuery, Express. Database used in these modules is MongoDB Atlas.

5.1 Admin Module

This module is the base for authentication and authorization to ensure the security aspect of the user. It also includes user account creation, block user account, transfer cash, approve loans, and make user debit card if required.

5.2 Loan Module

This module handles the functions related to Loan management. It allows user to calculate loan amount which they need to pay if they get loan approval from the bank. It allows user to apply for loan and if they get loan approval they can also view loan details. It also allows user to refinance loan and also can redefine their loan agreement if they wishes too.

5.3 Fixed Deposit Module

This module handles the functions related to Fixed-Deposit management. It allows user to calculate fixed-deposit amount which they will get if they apply for it in the bank. It allows user to apply for fixed-deposit and after the fixed-deposit is created they can also view fixed-deposit details. It also allows user to close their fixed-deposit if they wishes too.

5.4 Account Module

This module handles the functions related to Account management. It allows user to view all their accounts in which they are holder. It allows user to see all their transactions which they made using that account. It also allows user to get statistics of all his accounts.

5.5 Card Module

This module handles the functions related to Card management. It allows user to transfer amount / make any payment via a specific card (debit/credit). It allows user to apply for cards if they haven't any debit/credit card. It also allows user to change pin of the card (debit/credit) if they wishes too. It provides the functionality to user to renew their card (debit/credit) if they are going to expire and also close their card (debit/credit) if they are not using it at all.

5.6 Fund-Transfer Module

This module handles the functions related to Fund-Transfer management. It provides the functionality to user to transfer money / make payment via RTGS / NEFT / Card transfer mode of payment.

5.7 Function Prototypes

Make Debit Card

This functionality provides the user to get debit card after appropriate validation.

```
// The request object requires a pin number to be passed.
router.post("/cards/makeDebitCards/:acNum", [authenticate, debitCardAuthenticate], async
(req, res) => {
    try {

        // checking whether user is authenticated or not and also checking their debit
card status
        if (req.is_authenticated && req.makeDebitStatus) {

            // getting current ac
            const ac = req.ac

            // checking pinNumber and current account
            if (ac && req.body.hasOwnProperty("pinNo")) {
                ac.isEcardissued = true
                // create debitcard object.
                const cardDetails = await generateDebitCardDetails()
                const curDate = new Date()
                const expiryDate = new Date(
                    curDate.getFullYear() + 5,
                    curDate.getMonth(),
                    curDate.getDate(),
                    curDate.getHours(),
                    curDate.getMinutes(),
                    curDate.getSeconds()
                );

                const pinNumber = req.body.pinNo
                if (pinNumber.toString().length == 4) {

                    const debitCard = await DebitCard({
                        accountAttached: ac,
                        cardNumber: cardDetails.cardNum,
                        cvvNumber: cardDetails.cvv,
                        pinNumber: pinNumber,
                        serviceProvider: "American Express",
                        expiryDate: expiryDate
```

```
                    })
                    const status = debitCard.save()
                    const acStatus = ac.save()
                    if (status && acStatus) {
                        return res.json({ "Success:": "Saved your card successfully" })
                    }
                    else {
                        return res.json({ "Error:": "Failed saving debit card" })
                    }
                }
                else {
                    throw "Pin Number must be of 4 digit Long"
                }

            }
            else {
                return res.json({ "Error:": "No accounts found" })
            }


        }
        else {
            if (!req.makeDebitStatus) {
                throw "Debit card already exists!"
            }
            return res.json({ "Error:": "Sorry you must be authenticated!", "redirect":
true })
        }
    }
    catch (e) {
        return res.json({ "Error: ": e.toString() })
    }
})
```

Fixed Deposit

This functionality provides the user to apply for fixed deposit after appropriate validation.

```
// used to add new fixed deposit
router.post("/fd/addNewFD", [authenticate, verifyDeposit], async (req, res) => {
  try {
    // checking if user is authenticated and checking if user is applying for new fixed
deposit from the account logged in
    if (req.is_authenticated && req.accountAttchedWithUser) {
      console.log("from deposit router");
      const {
        principleAmount,
        maturity,
        acNumber,
        depositType,
        recurringAmount,
      } = req.body;

      // checking all field are empty or not
      if (!principleAmount || !maturity || !acNumber || !depositType) {
        throw "All fields are required!!";
      }

      const curDate = new Date();
      const maturityDate = new Date(
        curDate.getFullYear() + Number(maturity),
        curDate.getMonth(),
        curDate.getDate(),
        curDate.getHours(),
        curDate.getMinutes(),
        curDate.getSeconds()
      );

      // check validation related to recurring amount
      if (depositType === "recurringDeposit") {
        if (!recurringAmount) {
          throw "Please provide recurring amount";
        }
      }

      const depositobj = await Deposit({
        dateOfIssue: new Date(),
        maturityDate: maturityDate,
```

```
      interestRate: 8,
      principleAmount: principleAmount,
      depositOwner: req.current_user,
      referenceAccount: req.current_ac,
    });

    const depositStatus = await depositobj.save();

    if (!depositStatus) {
      throw "Error while creating Deposit Object";
    }
    else {
      req.current_ac.accountBalance -= principleAmount;
      req.current_ac.save();
      const trxObj = await Transaction({
        sender: req.current_user,
        senderAc: req.current_ac,
        receiverAc: req.admin_account,
        receiver: req.admin_user,
        amount: principleAmount,
        transactionDateTime: Date.now(),
        mode: "FD",
        reason: "FD:" + String(depositobj._id).substr(21, 24),
        isPending: false,
      });

      const trxStatus = await trxObj.save();

      if (!trxStatus) {
        throw "Error while saving data!!";
      }
    }

    // create RecurringDeposit object
    if (depositType === "recurringDeposit") {

      const recurObj = await RecurringDeposit({
        deposit: depositobj,
        recurringDepositAmount: recurringAmount,
      });

      const recurringStatus = await recurObj.save();

      if (!recurringStatus) {
        throw "Error while creating RecurringObject!!";
      }
```

```
      }
      // create FixedDeposit object
      else {
        const fixedObj = await FixedDeposit({
          deposit: depositobj,
          FixedDepositAmount: principleAmount,
        });

        const fixedStatus = await fixedObj.save();

        if (!fixedStatus) {
          throw "Error while creating FixedDeposit Object!!";
        }
      }
      logger.add_log("New FD added!" + fixedObj._id, "SUCCESS");
      return res.json({ "Success:": true });
    }
  }
  catch (e) {
    logger.add_log("/fd/addnewfd " + e.toString(), "ERROR");
    console.log(e.toString());
    return res.json({ Error: e.toString(), "Success:": false });
  }
});
```

New User Account

This functionality provides the admin to create new user account.

```
// use to create new user by admin
router.post("/admin/createUserAccount", [authenticate], async (req, res) => {
  try {
    const { username, balance } = req.body;
    const usernames = String(username).split(" ");
    const firstName = usernames[0];
    const middleName = usernames[1];
    const lastName = usernames[2];
    console.log(firstName, middleName, lastName);
    const user = await User.findOne({ firstName, middleName, lastName });
    if (!user) {
      throw "user does not exists!!";
    }
    else {
      const acType = await AccountType.findOne({
        accountTypeName: "joint",
```

```
      });
      const ac = await Account({
        accountOwner: user,
        accountType: acType,
        accountBalance: balance,
        isEcardIssued: false,
      }).save();
      if (!ac) {
        throw "Error saving user!";
      }
      logger.add_log(
        "/create admin ac account would be now visible in users dashboard",
        "SUCCESS"
      );
      return res.json({
        "Success:": true,
        message: "account would be now visible in users dashboard!",
      });
    }
  }
  catch (e) {
    logger.add_log("/create admin ac " + e.toString(), "ERROR");
    return res.json({ "Error:": e.toString() });
  }
});
```

User Login

This functionality provides the user to login to the system.

```
// used to login user which is registered already
router.post("/user/login_user", async (req, res) => {
  try {
    const { username, pinNo } = req.body;
    console.log(req.body);
    if (!username || !pinNo) {
      return res.json({ Error: "Fields cannot be empty!" });
    }
    else {
      // your username is firstname_middlename_lastname
      const user_names = username.split("_");
      console.log(req.body.pinNo);
      console.log(user_names);
      const user = await User.findOne({
```

```
        firstName: user_names[0],
        middleName: user_names[1],
        lastName: user_names[2],
        pinNo: req.body.pinNo,
      });
      if (!user) {
        logger.add_log("Username: " + username + " not found", "ERROR");
        return res.json({ "Error:": "No user found" });
      }
      else {
        console.log(
          "Current user:" +
            user.firstName +
            " " +
            user.middleName +
            " " +
            user.lastName
        );
        const token = await user.generateAuthToken();

        // Saving current users session
        const savedCookie = await res.cookie("LoginToken", token, {
          expires: new Date(Date.now() + 2589200000),
          httpOnly: true,
        });
        const userCookie = await res.cookie("UserId", user._id, {
          expires: new Date(Date.now() + 2589200000),
          httpOnly: true,
        });
        // End of saving session.post
        logger.add_log("User logged in name: " + username, "INFO");
        res.send("Success");
      }
    }
  }
  catch (e) {
    logger.add_log(e.toString(), "ERROR");
    console.log("Error:" + e.toString());
  }
});
```

Add Cash to User Account

This functionality provides the admin to add cash to user account.

```
// use to add cash to user account by admin
router.post("/admin/addCashToUser", [authenticate], async (req, res) => {
  try {
    const { acNum, amount, pinNo } = req.body;

    // check account is exist or not
    const accountExist = await Account.findOne({
      _id: acNum,
    });

    if (!accountExist) {
      throw "Account does not exist!!";
    }

    if (req.current_admin.pinNo == pinNo) {
      accountExist.accountBalance =
        accountExist.accountBalance + parseInt(amount);
      req.admin_account.accountBalance =
        req.admin_account.accountBalance - parseInt(amount);
      let st1 = await accountExist.save();
      let st2 = await req.admin_account.save();

      const trxObj = await Transaction({
        sender: req.admin_user,
        senderAc: req.admin_account,
        receiverAc: accountExist,
        receiver: accountExist.accountOwner,
        amount: amount,
        transactionDateTime: Date.now(),
        mode: "CASH",
        reason: "Cash Transfer",
        isPending: false,
      });

      const st3 = await trxObj.save();

      if (!st1 || !st2 || !st3) {
        throw "Error saving your data";
      }
      logger.add_log("/admin/addcashtouser added cash" + amount, "SUCCESS");
      return res.json({ Success: true });
    }
```

```
  }
  catch (e) {
    console.log(e);
    logger.add_log("/admin/addcashtouser " + e.toString(), "ERROR");
    return res.json({ "Error:": e.toString() });
  }
});
```

RTGS Transfer

This functionality provides the user to transfer cash to other user account.

```
const sendMail = function (from, to, otp) {
  console.log("Transaporter");
  var transporter = nodemailer.createTransport({
    service: "gmail",
    auth: {
      user: "abcxyz1814@gmail.com",
      pass: "abcxyzpass",
    },
  });
  console.log("mailOptions");
  var mailOptions = {
    from: from,
    to: to,
    subject: "Your OTP for Banker Transaction",
    html: "<h1>" + String(otp) + "</h1>",
  };
  console.log("sending mail");
  transporter.sendMail(mailOptions, function (error, info) {
    if (error) {
      console.log(error);
    } else {
      console.log("Email sent: " + info.response);
    }
  });
  logger.add_log("Sent mail from:" + from + " to:" + to);
  return "Success";
};
router.post(
  "/verifyRTGS/:acNum" ,
  [authenticate, verifyRTGSDetails],
```

```
async (req, res) => {
  try {
    const { beneficiaryName, beneficiaryAcNum, ifscCode, amount, reason } =
      req.body;

    if (
      !beneficiaryName ||
      !beneficiaryAcNum ||
      !ifscCode ||
      !amount ||
      !reason
    ) {
      logger.add_log("RTGS all fields are required", "ERROR");
      return res.status(200).json({ Error: "All fields are required!!" });
    }

    if (req.current_ac.accountBalance < amount) {
      logger.add_log("RTGS Not enough amount to do RTGS!!", "ERROR");
      return res.status(200).json({ Error: "Not enough amount to do RTGS!!" });
    }
    else if (amount < 0 || amount == 0) {
      logger.add_log("RTGS Please enter valid amount to do RTGS!!", "ERROR");
      return res.status(200).json({ Error: "Please enter valid amount to do RTGS!!" });
    }
    else {
      const beneficiaryUser = await User.findOne({
        _id: req.beneficiaryAc.accountOwner,
      });

      req.beneficiaryAc.accountBalance =
        req.beneficiaryAc.accountBalance + parseInt(amount);
      req.current_ac.accountBalance =
        req.current_ac.accountBalance - parseInt(amount);

      let st1 = await req.beneficiaryAc.save();
      let st2 = await req.current_ac.save();

      const trxObj = await Transaction({
        sender: req.current_user,
        senderAc: req.current_ac,
        receiverAc: req.beneficiaryAc,
        receiver: beneficiaryUser,
        amount: amount,
        transactionDateTime: Date.now(),
        mode: "RTGS",
        reason: reason,
```

```
          isPending: true ,
        });
        const otpNumber = Math.floor(100000 + Math.random() * 900000);
        const st4 = OTP({
          transaction: trxObj,
          OTP: otpNumber,
        }).save();
        let st3 = await trxObj.save();
        sendMail(
          "abcxyz1814@gmail.com",
          String(req.current_user.emailId),
          otpNumber
        );
        if (!st1 || !st2 || !st3 || !st4) {
          throw "Error saving your data";
        }
        logger.add_log("RTGS transaction initated otp sent", "SUCCESS");
        return res.json({ "Success:": true, data: trxObj });
      }
    }
    catch (e) {
      logger.add_log(e.toString(), "ERROR");
      console.log(e.toString());
      return res.json({ "Success:": false });
    }
  }
);
```

Loan

This functionality provides the user to apply for loan online.

```
// The verifyLoan Middleware is a middleware that decides that the loan is to be
// granted or not and this is to be done by a composite score parameter.
// The composite score takes in considerations your past transactions
// and your current account balance and the collateral you have.
router.post(
  "/loan/addUserLoan",
  [authenticate, verifyLoan],
  async (req, res) => {
    if (req.grantLoan) {
      const userAc = aw ait AccountModel.findOne({ _id: req.body.userAccount });
      if (userAc) {
        const userLoanQuery = await LoanIntermediateModel({
          receiverAc: userAc,
```

```
          amount: req.LoanValue,
          interestRate: req.interestRate,
          remarks: "None",
          collateral: "House",
          collateralValue: req.body.colVal,
          status: "pending",
        }).save();
        if (!userLoanQuery) {
          return res.json({ "Error:": "Failed saving your details" });
        }
      }

      return res.json({
        grant: req.grantLoan,
        loanValue: req.LoanValue,
        interestRate: req.interestRate,
      });
    }
    else {
      return res.json({
        grant: req.grantLoan,
        reason: req.reason,
        compositeScore: req.compositeScore,
      });
    }
  }
);
```

6. Testing

Manual testing was performed in order to find and fix the bugs in development process.

Testing Method: Manual Testing

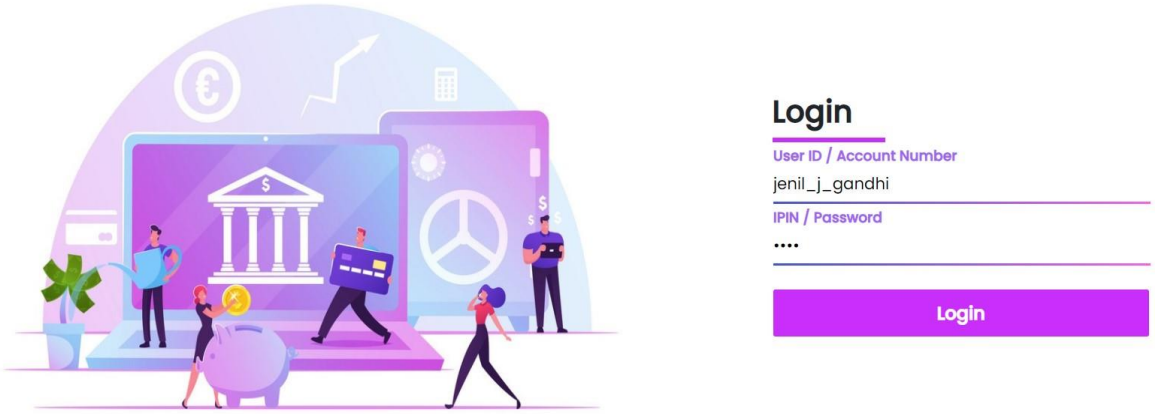
Sr. No.	Test Scenario	Expected Result	Actual Result	Status
1	Login with incorrect credentials	User should not be able to log-in and shown an error message	The system shows the message 'Invalid Credentials' and redirected to login page	Success
2	Login with correct credentials	User should be ableto log-in and redirected to home page	User is logged in and redirected to home page	Success
3	Card transfer with incorrect details	User should not be able to transfer money and shown an error message	The system shows the message 'invalid details' and money transfer is rejected	Success
4	Card transfer with correct details	User should not be able to transfer money and shown an success message	The system shows the message 'Transaction Successful' and money transferred	Success

5	Make fixed deposit with insufficient balance	User should not be able to make fixed deposit and shown an error message	The system shows the message 'Insufficient Account Balance' and redirected to same page	Success
6	Make fixed deposit with sufficient balance	User should able to make fixed deposit and shown an success message	The system shows the message 'Fixed Deposit created'	Success
7	View Fixed Deposit when it has been created	User should be able to view fixed deposit	The system shows all the fixed deposit that are created under the current logged user.	Success
8	View Fixed Deposit when it has not been created	User should not be able to view fixed deposit page.	The system does not show view fixed deposit page.	Success
9	Apply for loan if all criteria are satisfied	Loan is approved	The system approves the loan and shows success message	Success
10	Apply for loan if all criteria are not satisfied	Loan is not approved	The system does not approves the loan and shows error message	Success

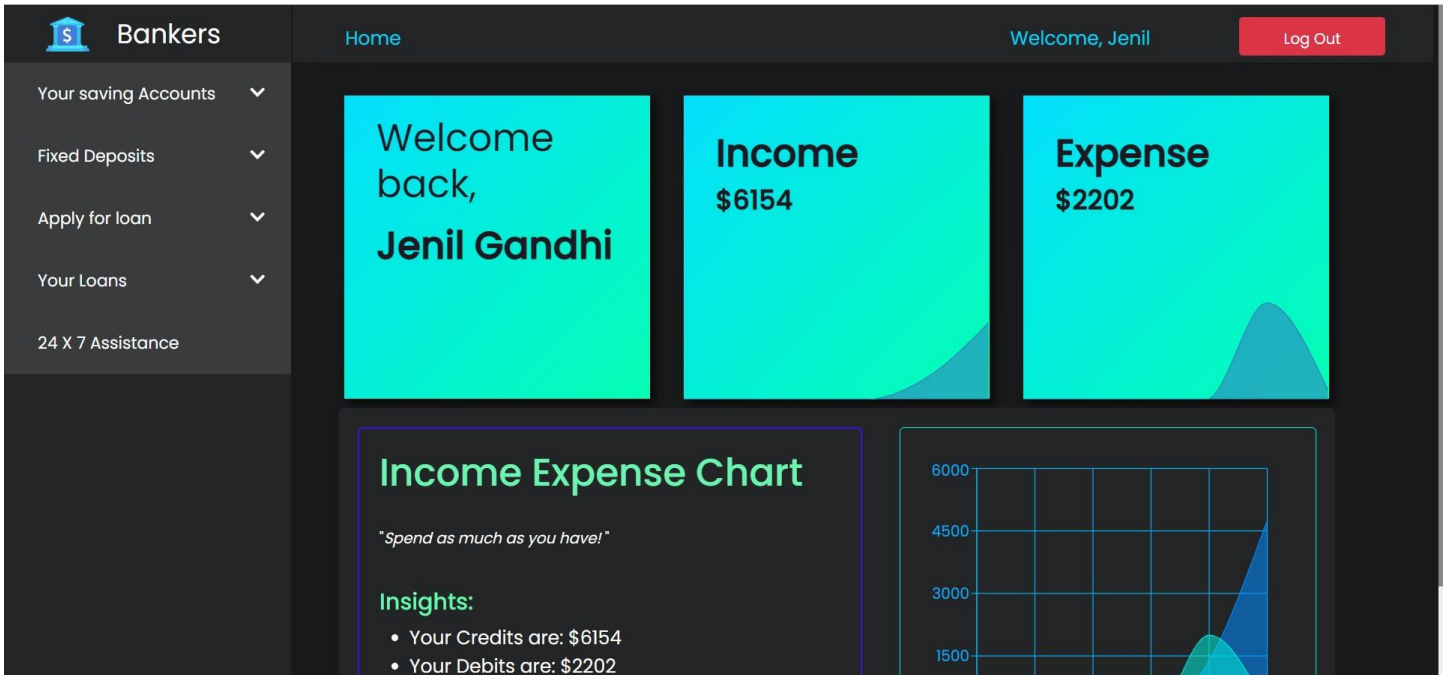
11	View Transaction if user have made any	Transaction table with detailed view is shown	The system shows the detailed transaction table	Success
12	View Transaction if user haven't made any	Transaction table is shown but with zero entries.	The system shows the transaction table but with zero entires.	Success
13	RTGS Transfer with correct details	User should not be able to transfer money and shown an success message	The system shows the message 'Transaction Successful' and money transferred	Success
14	RTGS Transfer with incorrect details	User should not be able to transfer money and shown an error message	The system shows the message 'invalid details' and money transfer is rejected	Success
15	NEFT Transfer with correct details	User should not be able to transfer money and shown an success message	The system shows the message 'Transaction Successful' and money transferred	Success
16	NEFT Transfer with incorrect details	User should not be able to transfer money and shown an error message	The system shows the message 'invalid details' and money transfer is rejected	Success

7. Screenshots

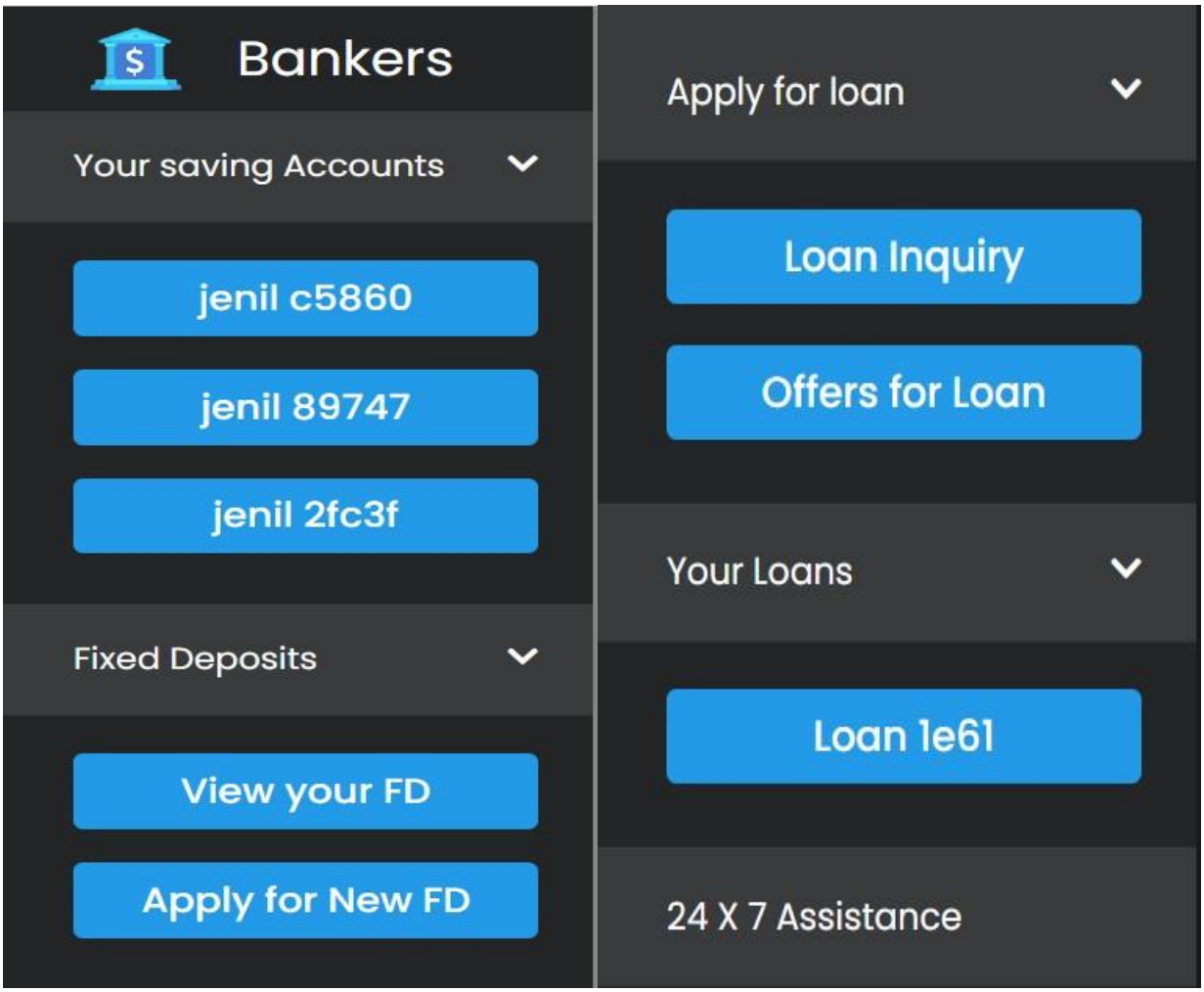
Login



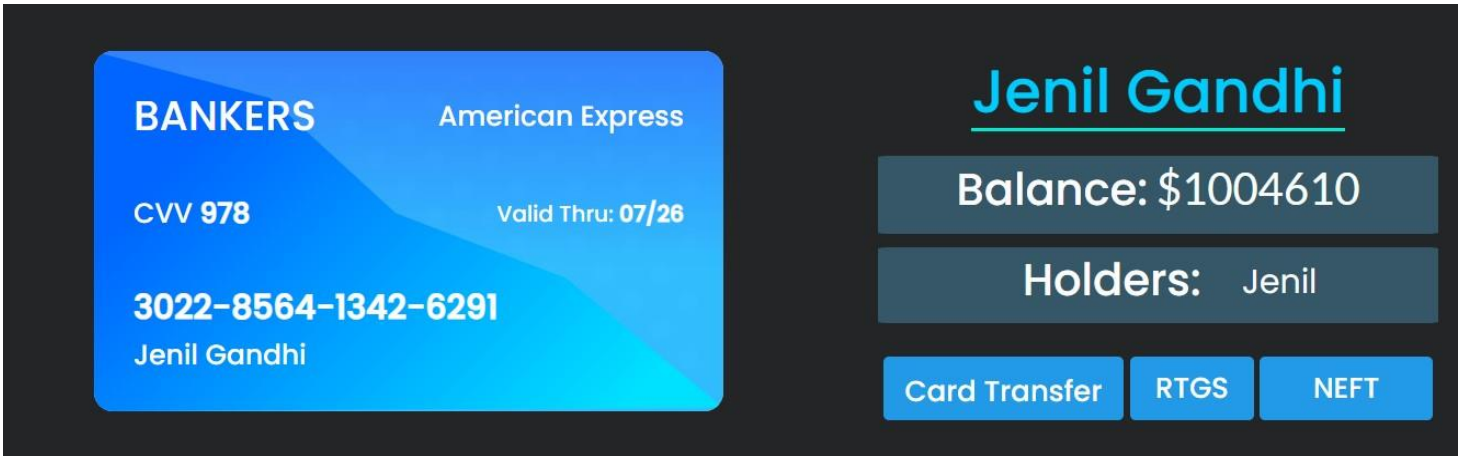
Home



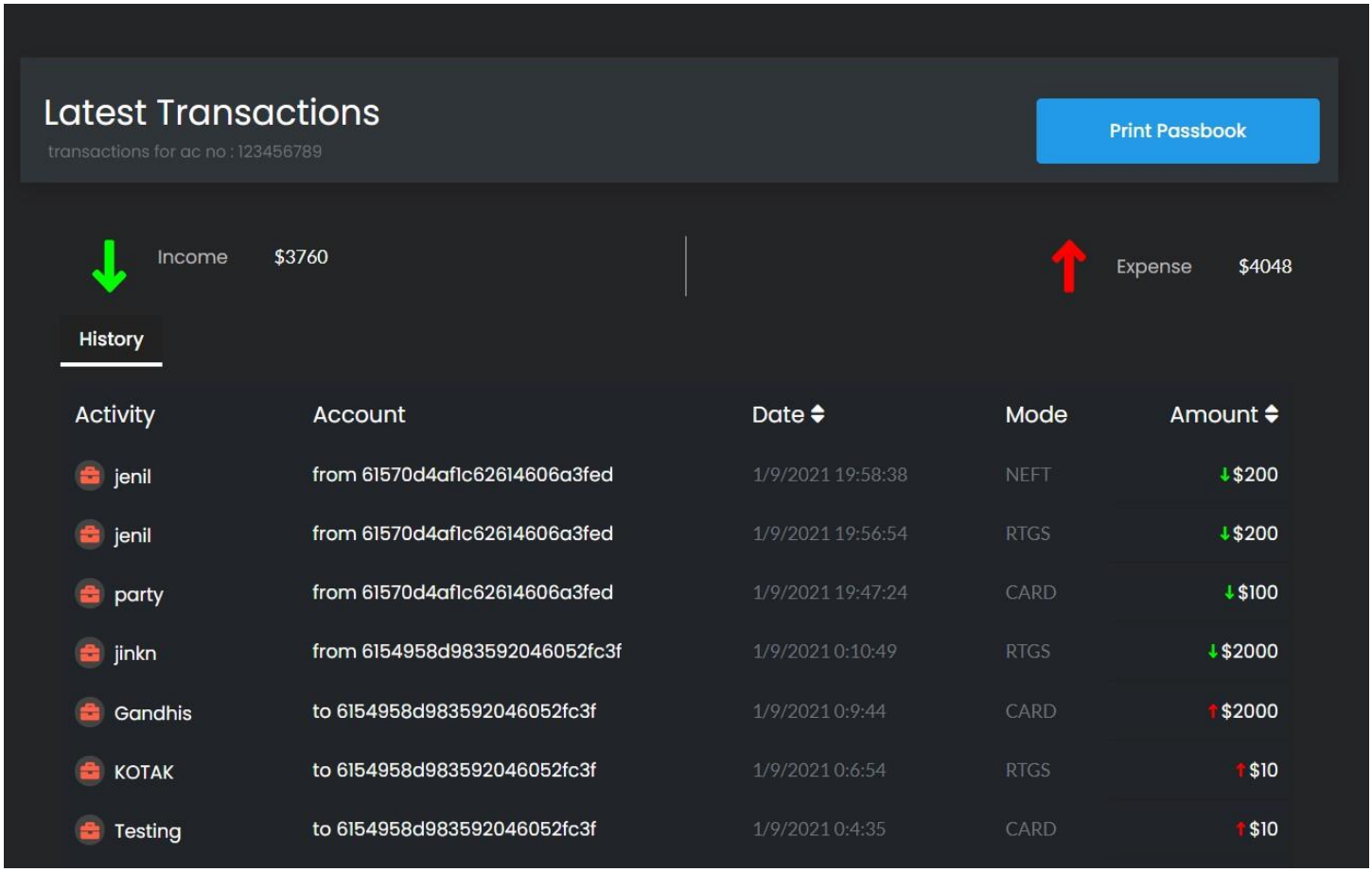
SideNavbar



Account Card

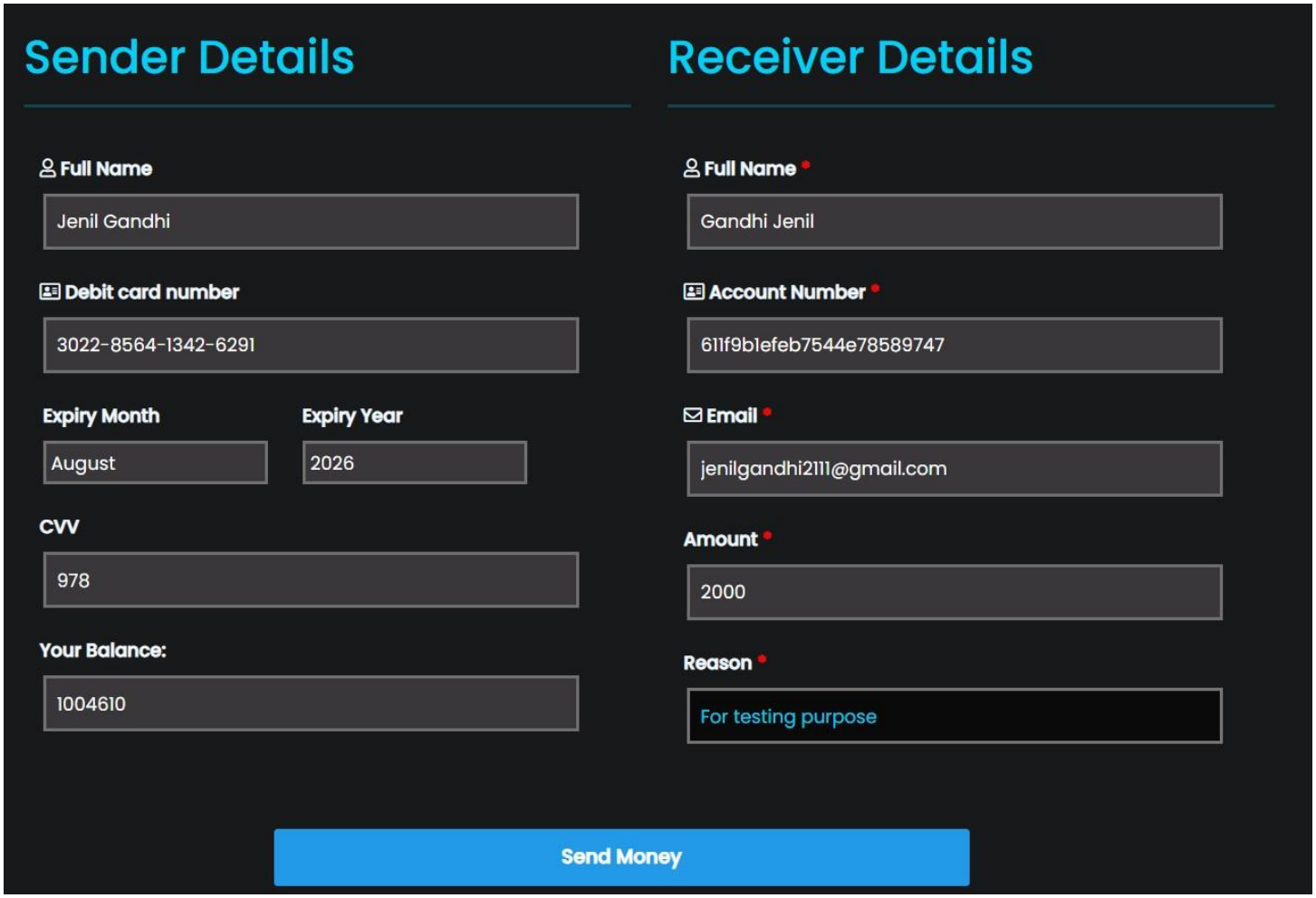


Account Transactions



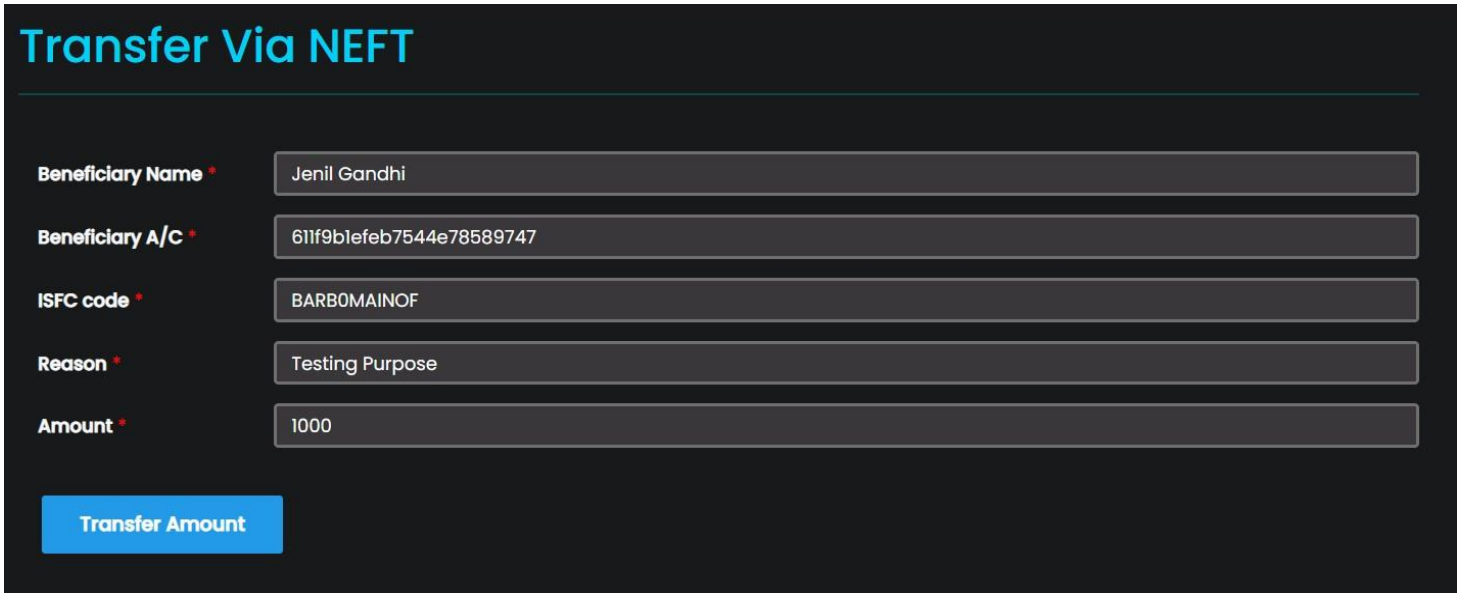
Account Chart

Card Transfer

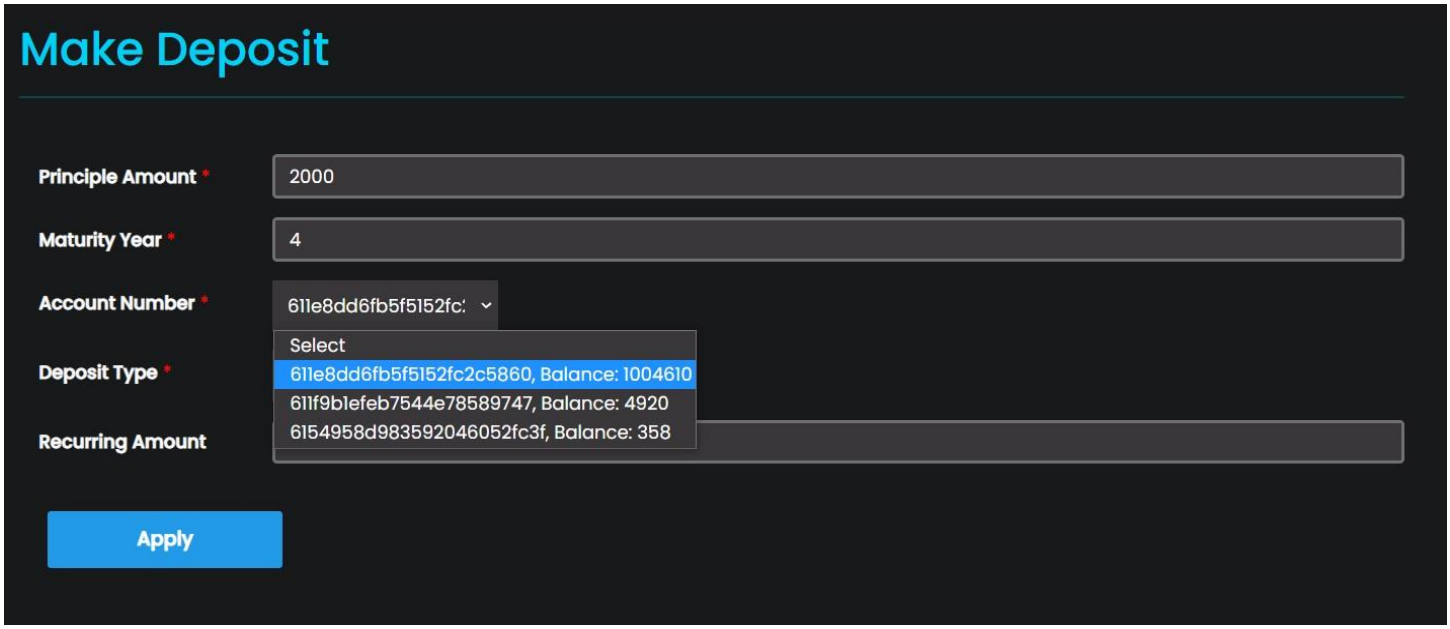


RTGS Transfer

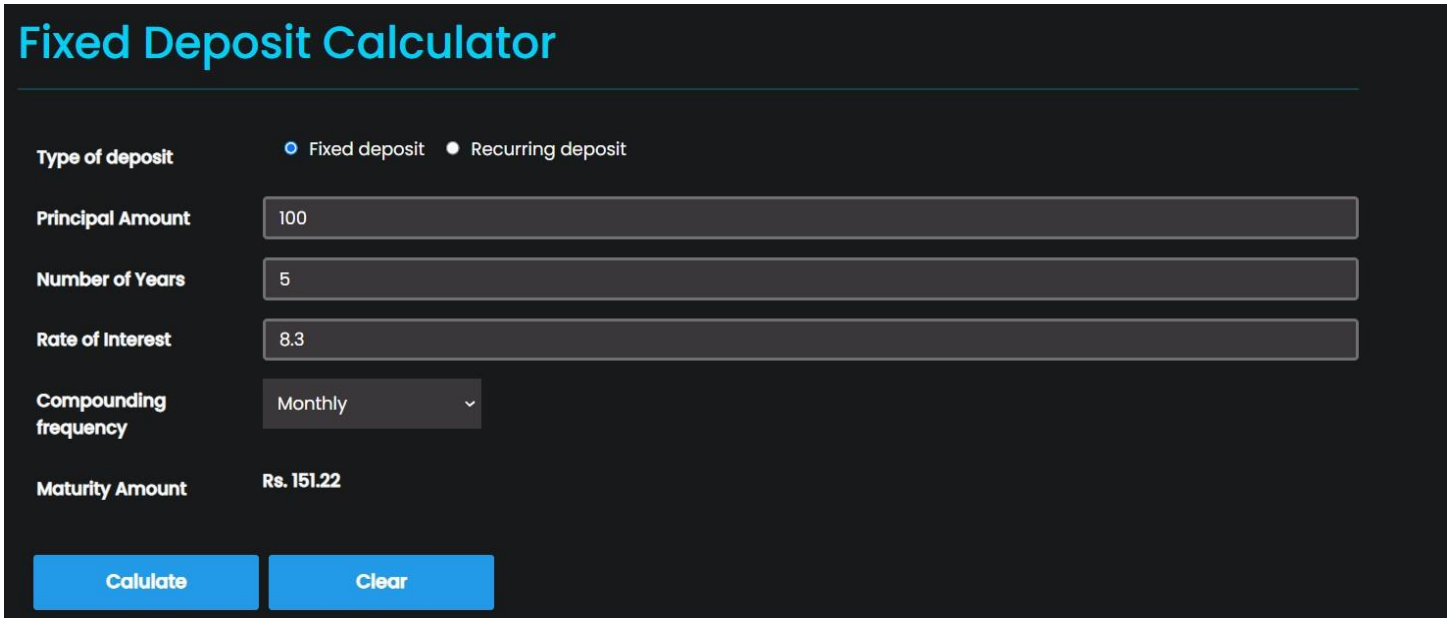
NEFT Transfer



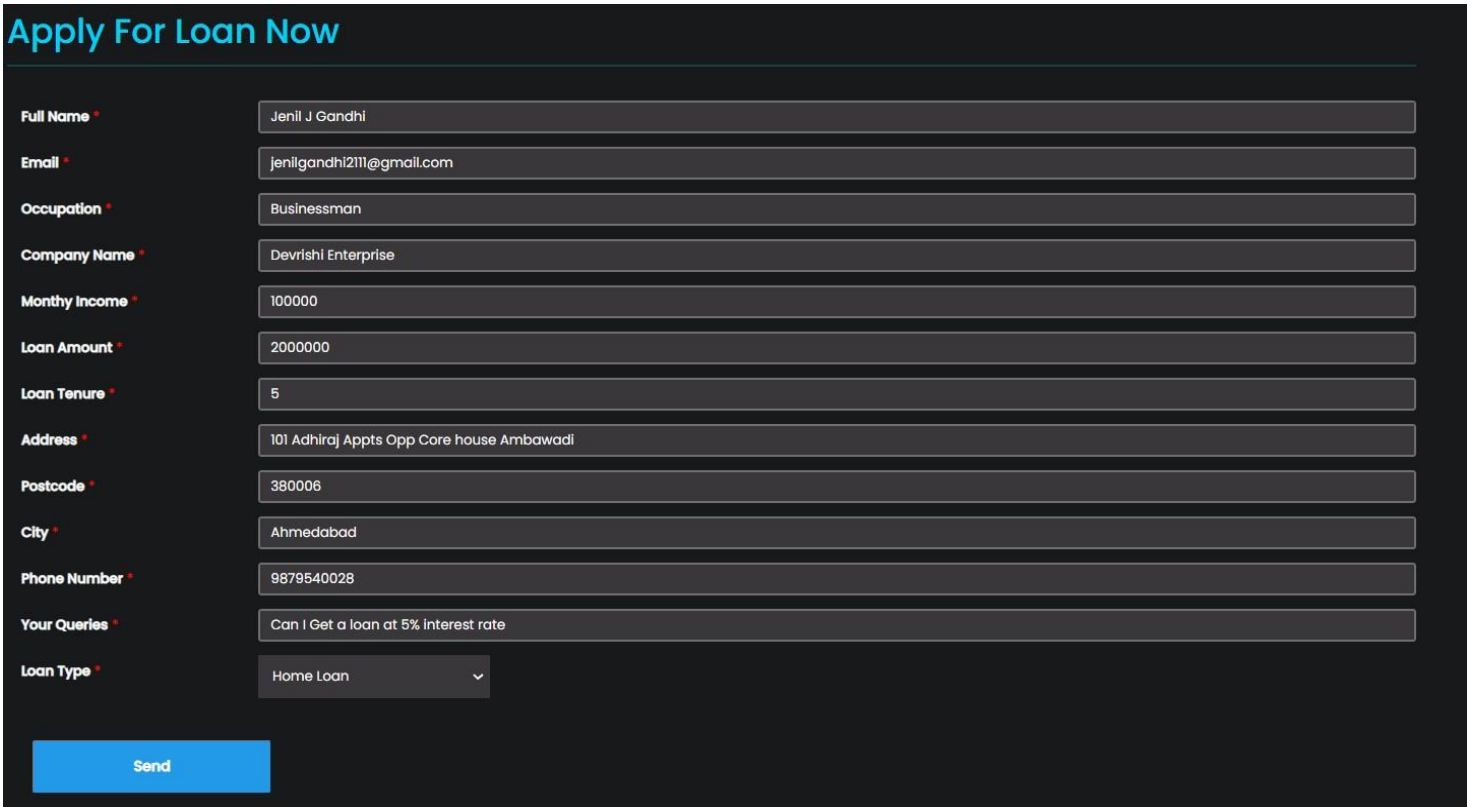
Make Fixed Deposit



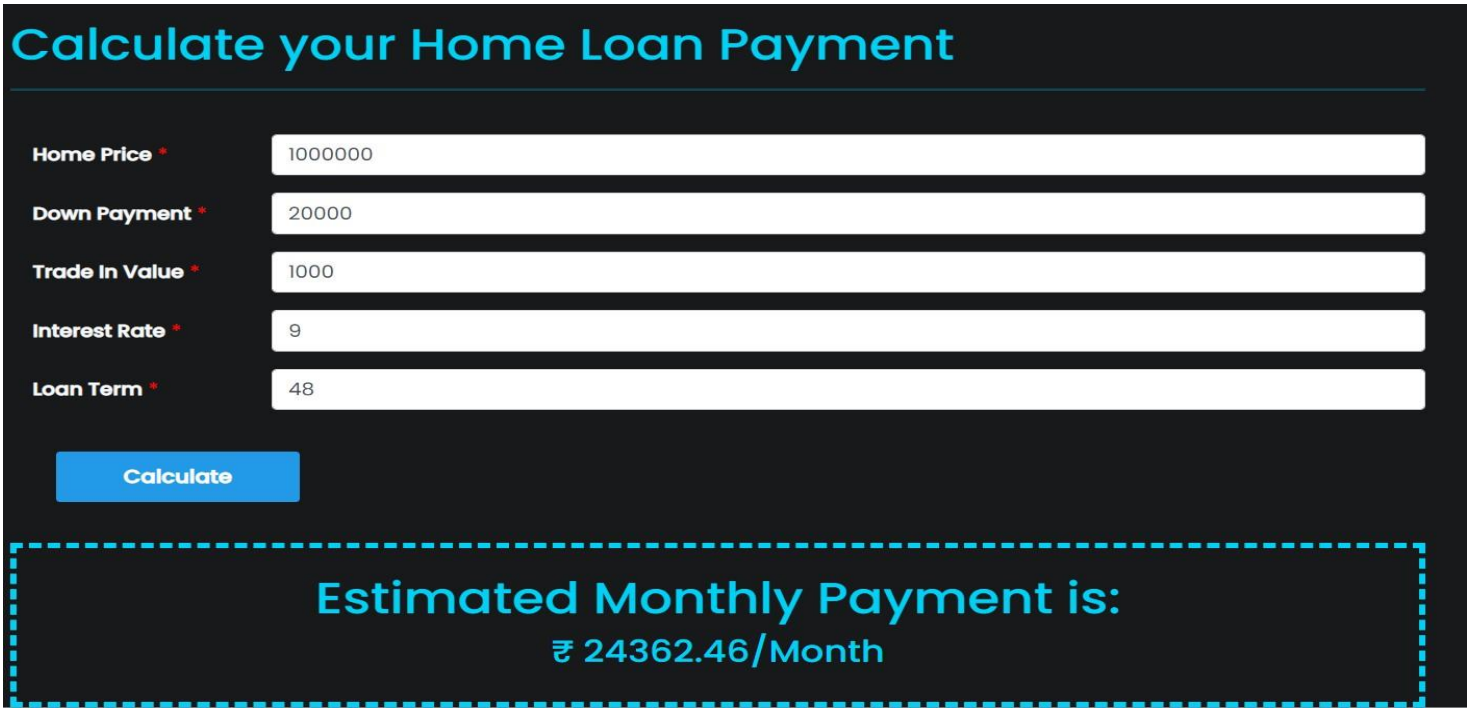
Fixed Deposit Calculator



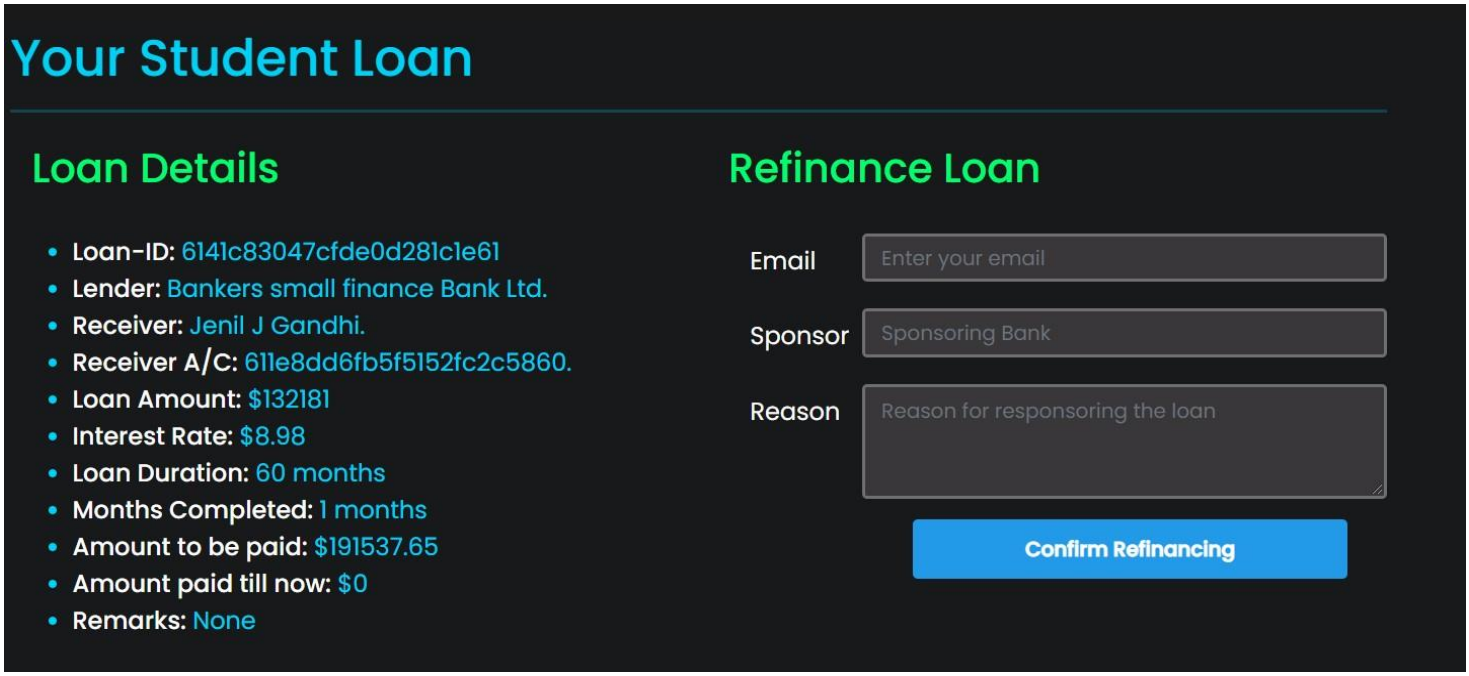
Apply for Loan



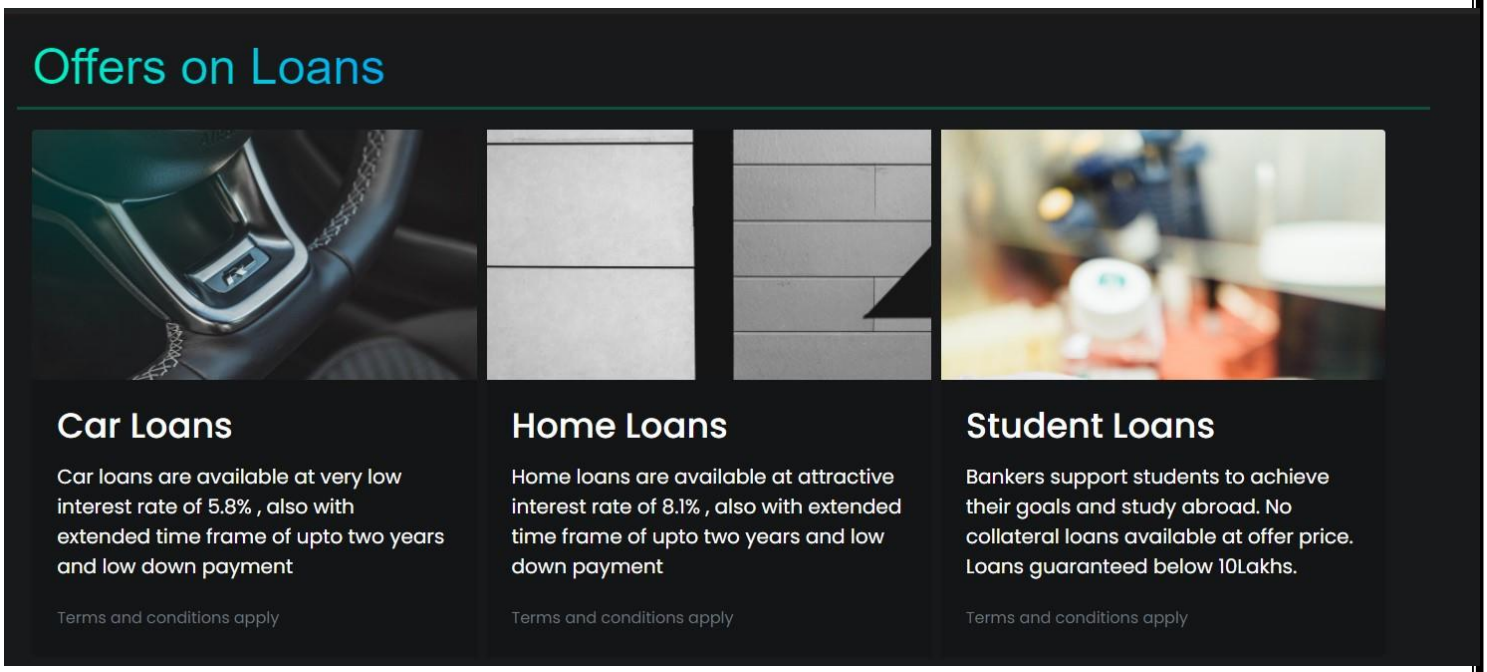
Loan Calculator



Loan Details



Loan Offer



8. Conclusion

The functionalities are implemented in system after understanding all the system modules according to the requirements. Functionalities that are successfully implemented in the system are:

- ❖ Login
- ❖ Logout
- ❖ Create User Account
- ❖ Block User Account
- ❖ Transfer Cash
- ❖ Approve Loans
- ❖ Make User Debit Cards
- ❖ Calculate Loan Amount
- ❖ Apply for Loan
- ❖ Refinance Loan
- ❖ Redefine Loan Agreement
- ❖ Calculate Deposit Amount
- ❖ Make Fixed Deposit
- ❖ View Fixed Deposit
- ❖ Close Fixed Deposit
- ❖ View all Accounts

- ❖ See Transactions
- ❖ Get Statistics of all Accounts
- ❖ Card Transfer
- ❖ Apply for Card
- ❖ Change Pin of Card
- ❖ Renew Card
- ❖ Close Card
- ❖ RTGS Transfer
- ❖ NEFT Transfer
- ❖ Card Transfer

After the implementation and coding of system, comprehensive testing was performed on the system to determine the errors and possible flaws in the system.

9. Limitations and Future Enhancements

9.1 Limitations

- We are able to implement most of the functionality of all modules.
- We aim to complete all the functionality of all modules and make this product ready to be used practically in all scenarios.
- Currently, the project runs completely fine if all the inputs / selections are given within proper criteria but it doesn't cover all the corner cases.

9.2 Future Enhacements

- User interface will be improved to provide better interaction of user with the system.
- Background processes and thread execution will be efficient and process load will be reduced.
- The project can be extended to run robust.
- We can also extend it to run on mobile devices.
- We can add a feature of in-browser test running of the code.
- Further extensions involve integrating it with Machine Learning / Artificial Intelligence recommendation chatbot models for suggesting / solving queries of user.
- We can also add Map Feature which tells the user the nearest located bank branch for emergency access.

10. Reference / Bibliography

Following links and websites were referred during the development of this project:

- **https://docs.npmjs.com/**
- **https://reactjs.org/docs/getting-started.html#learn-react**
- **https://docs.mongodb.com/**
- **https://expressjs.com/en/5x/api.html**
- **https://getbootstrap.com/docs/5.0/getting-started/introduction/**
- **https://www.w3schools.com/REACT/DEFAULT.ASP**
- **https://developer.mozilla.org/en-US/docs/Web/CSS/Reference**
- **https://developer.mozilla.org/en-US/docs/Web/JavaScript**
- **https://cryptojs.gitbook.io/docs/?cacheBust=1634646991657**
- **https://jwt.io/introduction**
- **https://stackoverflow.com/**
- **https://www.npmjs.com/package/react-toastify**
- **https://mycolor.space/**
- **https://cssgradient.io/**
- **https://colorhunt.co/**
- **https://www.npmjs.com/package/cookie-parser**
- **https://mongoosejs.com/docs/api.html**
- **https://reactjs.org/docs/jsx-in-depth.html**
- **https://api.jquery.com/**