Software Engineering

ATM Simulator

Final Report

Contributed by:-

1) Tushar J (PES1UG20CS472)

2)Prem Sagar J S (PES1UG20CS825)

3)Vaibhav Vijay (PES1UG20CS479)

4)Thushar M (PES1UG20CS471)

Index

SL No	Contents	Page No
1	Synopsis	2-3
2	SRS Document	3-8
3	Planning	9-11
4	Architecture	12-13
5	User Stories	13-14
6	Code	15-16
7	Testing	16-30

Synopsis

Proposed project description:-

An ATM Simulator,

What is ATM Automated teller machines are electronic banking outlets that allow people to complete transactions without going into a branch of their bank.

An ATM Simulator is a Web Application which is very easy to use. Talking about the system, it contains various functions which include Account Statement, Withdrawing, Depositing amount and changing the pin.

Here, at first the user has to enter an existing username, when the username matches the system proceed toward the next procedure i.e asking pin number.

When a user passes all these sign-in procedures, he/she can use all those features. It is too easy to use, he/she can check their respective account statements.

Actions that users can perform :

- User Login / Authentication
- Account Statement
- Withdrawing Amount
- Depositing Amount
- Changing The Pin
- Account Balance
- Change Phone Number
- Block The ATM Card
- Etc..

Plan of work and product ownership:

Our Team is going initiate this project with the SRS document preparation.

Then Prepare a scheduling chart using Gantt.

With help of best software development model we will follow the Steps to build our Application.

Using Github we will be able to work together as a team and Start, manage, Make changes to a file and push them to GitHub as commits, Open and merge a pull request and Track changes in our code across versions.

We will be able to deliver or submit the project within the given time.

Qualitative properties :

Response time will be less and application will work smoothly.

As we are Using MERN stack framework for Development it will be faster as we use Asynchronous execution of functions

User Credentials are Encrypted.

MongoDB Database is Used.

Software Requirements Specification

1. Introduction

1.1 Purpose :- The software ATM version1.0 is to be developed for Automated Teller Machines (ATM).An automated teller machine (ATM) is computerized telecommunications device that provides a financial institution's customers a secure method of performing financial transactions, in a public space without the need for a human bank teller. Through ATM, customers interact with a user friendly interface that enables them to access their bank accounts and perform various transactions.

1.2 Intended Audience :- This SRS defines External Interface, Performance and Software System Attributes requirements of ATM version1.0. This document is intended for the following group of people:- Developers for the purpose of maintenance and new releases of the software. Management of the bank. Documentation writers. Testers.

1.3 Product Scope :- This document applies to Automated Teller Machine software ATMversion1.0. This software facilitates the user to perform various transactions in his account without going to bank. This software offers benefits such cash withdrawals, balance transfers, deposits, inquiries, credit card advances and other banking related operations for customers. It also allows the administrator to fix the tariffs and rules as and when required. The software takes as input the login Id and the bank account number of the user for login purposes. The outputs then comprise of an interactive display that lets the user select the desirable function that he wants to perform.

1.4 References :- We have used Google Search for research purpose, Other Google search Images for the use case diagrams, and other websites for the info related to our project. The references for the above software are as follows:- i. www.google.co.in ii. www.wikipedia.com

1.5 Overview :- Section 1.0 discusses the purpose and scope of the software. Section 2.0 describes the overall functionalities and constraints of the software and user characteristics. Section 3.0 details all the requirements needed to design the software

2. Overall Description

2.1 Product Perspective :-

- The ATM is a single functional unit consisting of various subcomponents.
- This software allows the user to access their bank accounts remotely through an ATM without any aid of human bank teller.
- This software also allows to perform various other functions apart from just accessing his bank account such as mobile bill clearings etc.
- The ATM communicates with the bank's central server through a dial-up communication link.

2.2 Product Functions :-

The major functions that ATM performs are described as follows:-

Language Selection:- After the user has logged in, the display provides him with a list of languages from which he can select any one in order to interact with the machine throughout that session. After the language selection the user is prompted with an option that whether he wants the selected language to be fixed for future use so that he is not offered with the language selection menu in future thus making the transaction a bit faster. User also has the freedom to switch to a different language mentioned in the list in between that session.

Account Maintenance:-

The various functions that a user can perform with his account areas follows:-

- **Account Type:-**The user has the freedom to select his account type to which all the transactions are made, i.e. he can select whether the account is current account or savings account etc.
- **Withdrawal/Deposit:** The software allows the user to select the kind of operation to be performed i.e. whether he wants to withdraw or deposit the money.
- **Amount:-** The amount to be withdrawn or deposited is then mentioned by the user.
- **Denominations:-** The user is also provided with the facility to mention the required denominations. Once he enters his requirements the machine goes through its calculations on the basis of current resources to check whether it is possible or not. If yes, the amount is given to the user otherwise other possible alternatives are displayed.
- **Money Deposition:-** Money deposition shall be done with an envelope. After typing the amount to be deposited and verification of the same, the customer must insert the envelope in the depositary.

- **Balance Transfer:-** Balance transfer shall be facilitated between any two accounts linked to the card for example saving and checking account.
- **Balance Enquiry:-** Balance enquiry for any account linked to the card shall be facilitated.

2.3 User Classes and Characteristics :-

There are different kinds of users that will be interacting with the system. The intended users of the software are as follows:-

- **User A:** A novice ATM customer. This user has little or no experience with electronic means of account management and is not a frequent user of the product. User A will find the product easy to use due to simple explanatory screens for each ATM function. He is also assisted by an interactive teaching mechanism at every step of the transaction, both with the help of visual and audio help sessions.
- **User B:** An experienced customer. This user has used an ATM on several occasions before and does most of his account management through the ATM. There is only a little help session that too at the beginning of the session thus making the transaction procedure more faster.
- **Maintenance Personnel:** A bank employee. This user is familiar with the functioning of the ATM. This user is in charge of storing cash into the ATM vault and repairing the ATM in case of malfunction. This user is presented with a different display when he logs in with the administrator's password and is provided with options different from that of normal user. He has the authority to change or restrict various features provided by the software in situations of repairing.

2.4 Operating Environment :-

Since this Software is going to be a ATM simulator this application will operated on the Any Desktop Computers with a stable internet connection.

2.5 Design and Implementation Constraints :-

This project will be implement using the Web Development Framework MERN Stack and due to this User will be able to use the product using Web browser only. other than the above mentioned points there no constraints for this project. The major constraints that the project has are as follows:-

- The ATM must service at most one person at a time.
- The number of invalid pin entries attempted must not exceed three. After three unsuccessful login attempts, the card is seized/blocked and need to be unlocked by the bank.
- The simultaneous access to an account through both, the ATM and the bank is not supported.
- The minimum amount of money a user can withdraw is Rs 100/- and the maximum amount of money a user can withdraw in a session is Rs.10,000/- and the maximum amount he can withdraw in a day is Rs 20,000/-

- Before the transaction is carried out, a check is performed by the machine to ensure that a minimum amount of Rs 1000/- is left in the user's account after the withdrawal failing which the withdrawal is denied.

2.6 Assumptions and Dependencies The requirements stated in the SRS could be affected by the following factors:

- One major dependency that the project might face is the changes that need to be incorporated with the changes in the bank policies regarding different services. As the policies changes the system needs to be updated with the same immediately. Adelay in doing the same will result to tremendous loss to the bank. So this should be changed as and when required by the developer.
- The project could be largely affected if some amount is withdrawn from the user's account from the bank at the same time when someone is accessing that account through the ATM machine. Such a condition shall be taken care of.
- At this stage no quantitative measures are imposed on the software in terms of speed and memory although it is implied that all functions will be optimized with respect to speed and memory

3. External Interface Requirements User Interfaces :-

The interface provided to the user should be a very user-friendly one and it should provide an optional interactive help for each of the service listed. The interface provided is a menu driven one and the following screens will be provided:-

- A login screen is provided in the beginning for entering the required username/pin no. and account number.
- An unsuccessful login leads to a reattempt (maximum three) screen for again entering the same information. The successful login leads to a screen displaying a list of supported languages from which a user can select any one.
- In case of administrator, a screen will be shown having options to reboot system, shut down system, block system, disable any service.
- In case of reboot/ shut down, a screen is displayed to confirm the user's will to reboot and also allow the user to take any backup if needed. In case of blocking system, a screen is provided asking for the card no. By entering the card no of a particular user, system access can be blocked for him.
- Administrator is also provided with a screen that enables him to block any service provided to the user by entering the name of the service or by selecting it from the list displayed.
- After the login, a screen with a number of options is then shown to the user. It contains all the options along with their brief description to enable the user to understand their functioning and select the proper option.
- A screen will be provided for user to check his account balance.
- A screen will be provided that displays the location of all other ATMs of same bank elsewhere in the city.
- A screen will be provided for the user to perform various transactions in his account.

Software Interfaces :-

This project is built using the MERN Stack framework and as we are using MongoDB for database, express for the server, React.js for the UI and Node for the Server environment

Communications Interfaces :-

Since this is a web based software application we are going to use some network protocols such as https and socket programming for communicating with server and getting the data that user needs.

The machine needs to communicate with the main branch for each session for various functions such as login verification, account access etc. so the following are the various communication interface requirements that are needed to be fulfilled in order to run the software successfully:-

The communication protocol used shall be TCP/IP

4. System Features

5. Other Non functional Requirements

5.1 Performance Requirements :-

The following list provides a brief summary of the performance requirements for the software:

5.1.1 Capacity

The ATM shall provide customers a 24 hour service.

5.1.2 Dynamic requirements

- The card verification time must not exceed 0.8 sec. under normal server workload and 1 sec. under peak server workload.
- The pin number verification time must not exceed 0.3 sec. under normal server workload and 0.5 sec. under peak server workload.
- Account balance display time must not exceed 2 sec. under normal server workload and 3 sec. under peak server workload.
- Account balance transfer time must not exceed 3 sec. under normal server workload and 4 sec. under peak server workload.
- Cash withdrawal transaction time must not exceed 4 sec. under normal server workload and 5 sec. under peak server workload.
- Deposit transaction time after insertion of the deposit envelope must not exceed5sec.under normal server workload and 6 sec. under peak server workload.
- Receipt printing time after must not exceed 3 sec. under normal server and peak server workload.

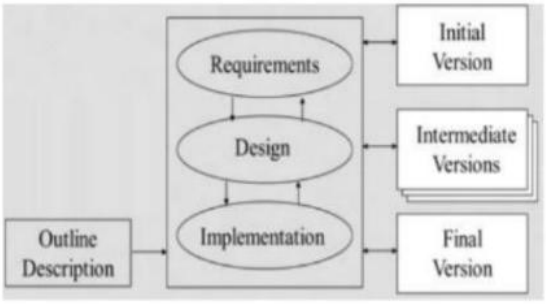
Security Requirements :- The system shall have two levels of security i.e. ATM card and pin verification both authenticated. The Encryption standard used during pin transmission shall be triple DES. The password shall be 6-14 characters long. Passwords shall not contain name of customers as they are easy to be hacked. Passwords can contain digit, hyphen and underscore. User should be provided with only three attempts for login failing which his card needs to be blocked.

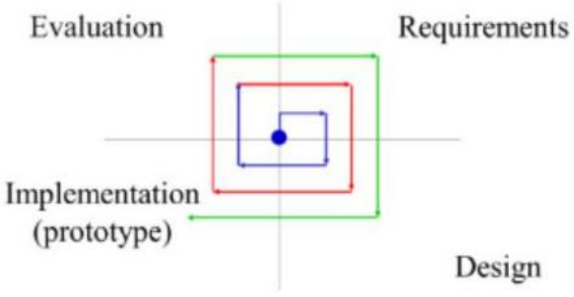
Software Quality Attributes :-

- **Reliability :** Software will reliable has we are adding some extreme measures and encryptions during the transaction to maintain the integrity.
- **Availability:** Software will be up and running as we are going to deploy the software sever in the best possible way to reduce any distruptions.
- **Security :** Software will be secure with all the authentication process and all the data transacted will be encrypted in the best possible way.
- **Maintainability :** Software is easily maintained and we can patch up any issues by adding new features.
- **Business Rules:-** The business rules for the software are as follows: The Administrator has the authority to fix the rules and regulations and to set or update the policies as and when required. The staff at the bank performs the following:
    a. Making the entries in the system regarding all the details of the bank account of the user.
    b. Keeping the bank account of the user updated as soon as changes are encountered so that the data is in consistent state.
    c. Blocking or seizing of the account of user on discovery of any illegal transaction.
    d. Unblocking of ATM card that got blocked due to more than three unsuccessful login attempt.
    e. Blocking of a lost/stolen ATM card on complaint of the user, only if he presents his verification and a FIR filed for that case.
    f. Constantly monitor all the ATMs in the city to check whether any one of them is encountering any fault.
    g. Immediately correct any fault discovered in any of the ATM.

Project Plan

Lifecycle to be followed for the execution of our project is Legacy Software Development Life Cycle - Iterative model We will start with simple implementation of a subset of software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, we will be designing and modifying and adding new functional capabilities





Tools which we want to use it throughout the lifecycle –

Planning Tool - Jira

Design Tool - Jira

Version Control - Github

Development Tool – Visual Studio Code

Bug Tracking – Github and Visual Studio Code

All the deliverables are build components

Deliverables –

User Login/Authentication

Account Type

Withdrawing Amount

Depositing Amount

Denominations Account

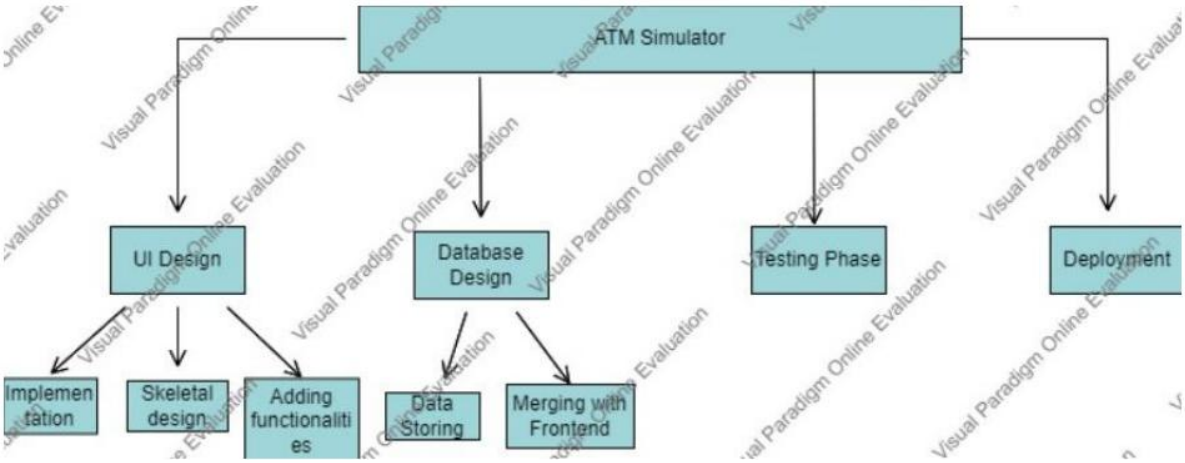
Balance

Changing Pin

Change Phone Number

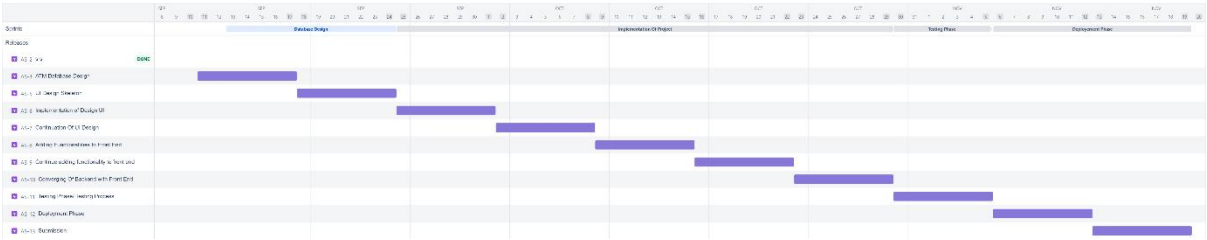
Block ATM Card

WBS for entire functionalities of ATM Simulator

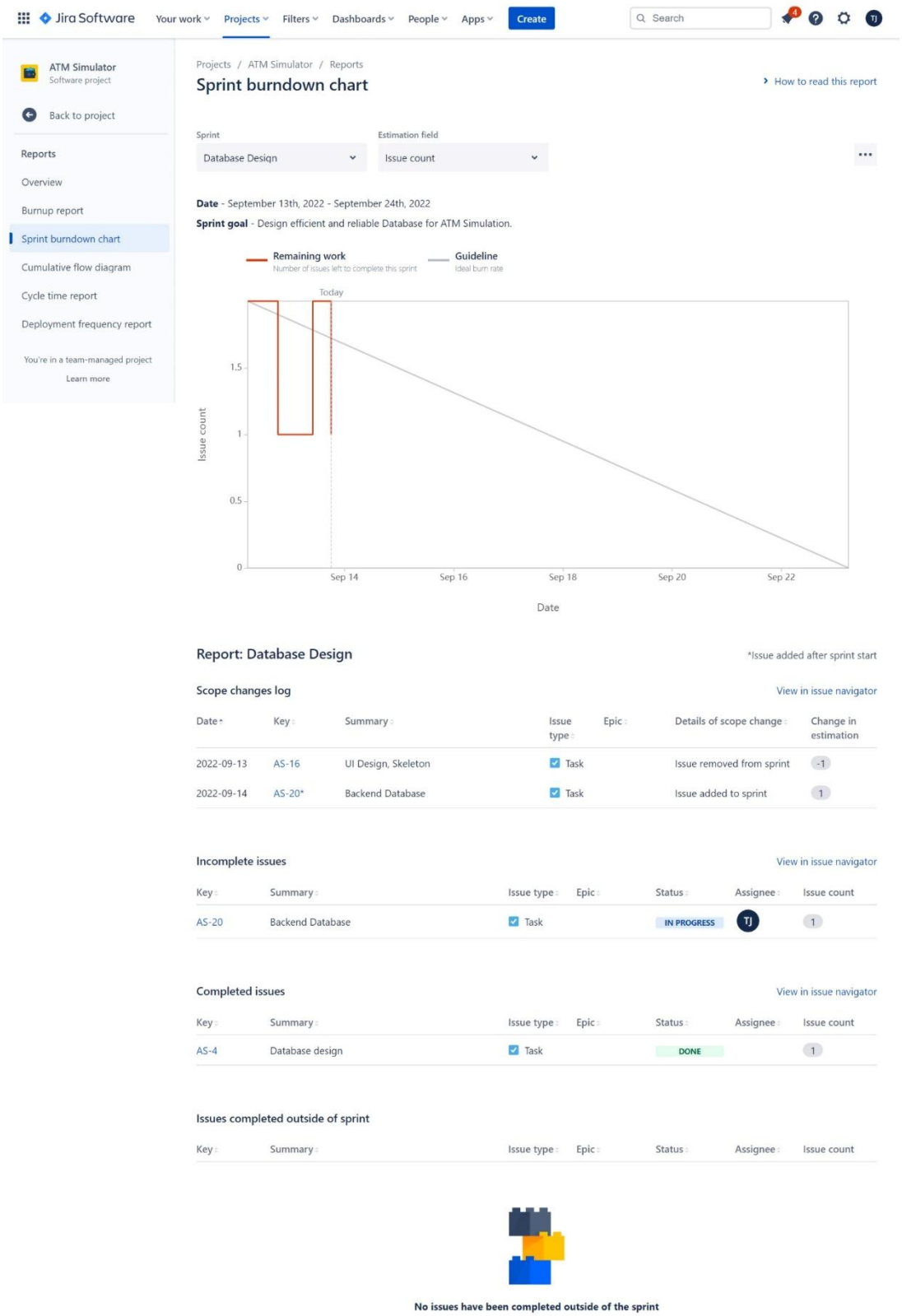


5K lines of code and rough estimate is 13 person months

Gantt Chart



Burndown



Chart

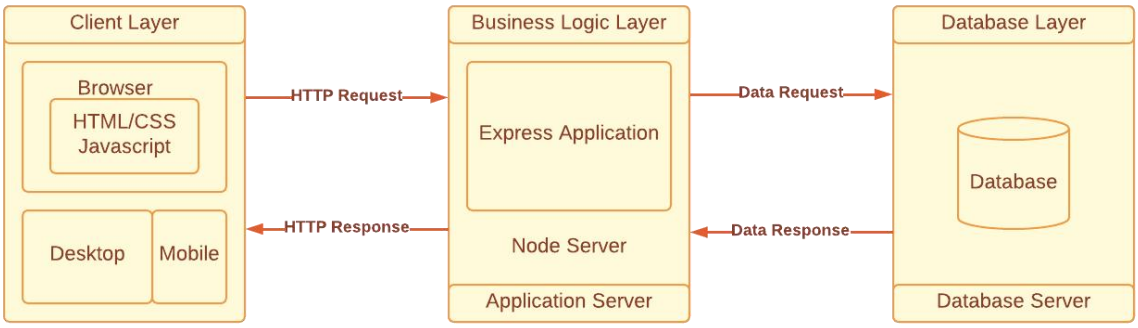
Software Architecture

The software architecture of a system depicts the system's organization or structure, and provides an explanation of how it behaves. A system represents the collection of components that accomplish a specific function or set of functions.

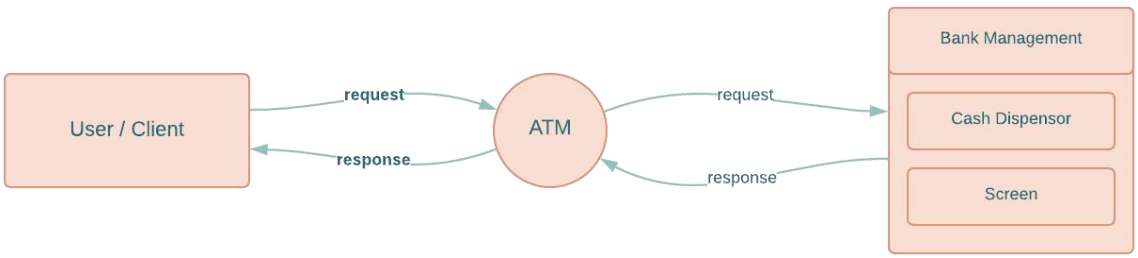
Architecture Used : Three Tier Architecture

A 3-tier architecture is a web app architecture that is widely used around the world. It basically contains 3 tiers: Client, Server, and Database. For our project ATM Simulation we are using Three Tier Architecture Based On MERN (MongoDB, Express, React, Node) Stack technology.

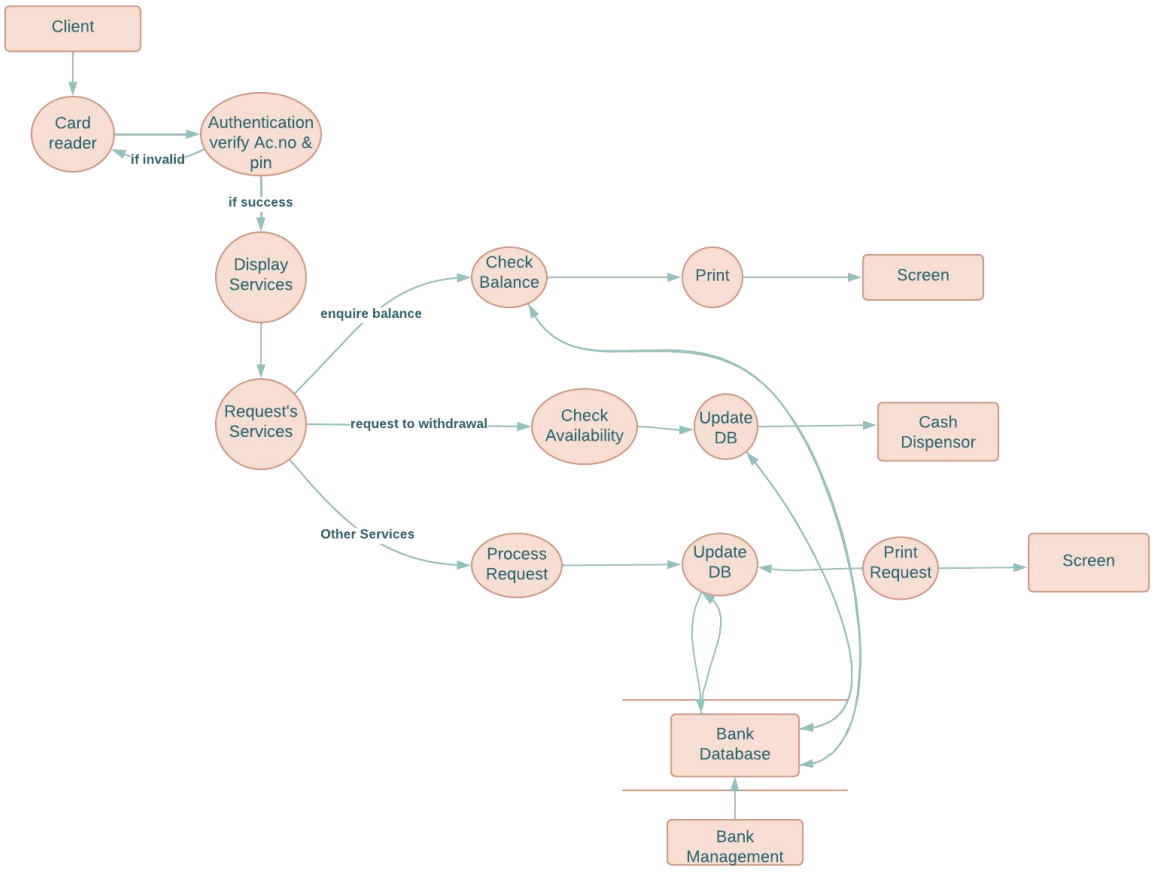
Project Architecture :-



Top level Data Flow diagram:-



Top Level -2 Data Flow Diagram:-



User Stories

1. As a Customer I want to Login to my account using card and PIN code So that I can perform the transactions.
   Acceptance Criteria –
   - System must validate the card and pin code
   - In case Customer enters wrong Pin code three times then the system locks the card.
2. As a Customer I want to to check the balance of my bank account So that I can perform transactions.
   Acceptance Criteria –
   - Customer needs to be logged in before checking balance.
   - Balances is displayed.
3. As a Customer I want to to deposit cash in my bank account through ATM So that I may save my time and perform transactions later.
   Acceptance Criteria –
   - Customer needs to be logged in before depositing cash.
   - System should verify the amount of cash deposited by checking with the user.
   - If the user doesn't agree then the system ejects back the cash.

- If Ok the account balance is updated and displayed.

4. As a Customer I want to to deposit check in my bank account through ATM So that I may save my time and perform transactions later.
   Acceptance Criteria –
   - Customer needs to be logged in before depositing check.
   - System should verify the amount written on the deposited check by asking the user.
   - If the user doesn't agree then the system ejects back the check.
   - If Ok the account balance is updated and displayed.
5. As a Customer I want to withdraw cash from my bank account through ATM So that I may save my time.
   Acceptance Criteria –
   - Customer needs to be logged in before withdrawing cash.
   - System checks to see if the request amount exceeds the balance
   - If so the system displays the balance and asks the user to enter a new amount.
   - If amount entered is less than the account balance cash is dispensed and the new balance is displayed.
6. As a Customer I want to logout from my bank account through ATM So that I may end up my ATM session.
   Acceptance Criteria –
   - System asks user if the user wants session report and receipt for the entire session.
   - If yes then the receipt is dispensed
   - User is logged off from the account
7. As a Customer I want to transfer money from my account to another bank account through ATM So that I may save my time.
   Acceptance Criteria –
   - Customer needs to be logged in before transferring amount.
   - System should check the receivers account number and validate it prior to performing the transactions.
   - If Ok the local account balance is updated and displayed.
   - System should update both accounts concurrently

Snippet of Tested Code's

1) After user has entered card number and pin this function searches for information in database, tells if user doesn't exists, after account information has been fetched successfully then 'account information fetching is completed' message is sent

```
exports.accountInfo = async (req, res) => {
  const {cardno} = req.body; // fetching in DB
  const user = await User.findOne({
    cardno
  });
  if(!user) throw "User Does Not Exist !";
  res.json({ message: "Account Info has been fetched",
    account: user // this user is JSON Object  which contains user name, pin, balance, card no, account type, phone number can be extracted in front end
  });
};
```

2) This function allows user to withdraw money, tell's if user doesn't exist if user, tries to withdraw money more than his/her balance amount then 'insufficient funds' message is sent, if user has successfully withdrawn money then balance is updated and 'transaction completed' message is sent.

```
exports.withdrawMoney = async (req, res) => {
  const {cardno,amount} = req.body;
  const user = await User.findOne({
    cardno
  });
  if(!user) throw "User Does Not Exist !";
    if(Number(amount)>Number(user.balance)){
      res.json({
        message: "Insufficient Funds !"
      });
    }
```

```
    else{
        balance = Number(user.balance)
        balance = balance-Number(amount)
        var myquery = { cardno: cardno };
        var newvalues = { $set: {balance: balance} };
        User.updateOne(myquery, newvalues,(err, res)=>{
        if (err) throw err;
        console.log("1 document updated");
    });
        res.json({
        message: "Transaction Successfull !",
    });
    };
  };
```

Testing

Unit Testing:-

Unit description : when user gives card number and pin , this API will fetches/queries the DB for the user info which is available on the DB and fetches and displays in the resposnse page.

```
exports.AccountInfo = async (req, res) => {
    const {cardno , pin} = req.body;
    // fetching in DB
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });
    if(!user) throw "Invalid Pin !";
    res.json({
        message: "Account Info has fetched",
```

```
        account: user, // this user is JSON Object  which contains user name,
pin, balance, card no, account type, phone number can be extracted in front
end

    });

};
```

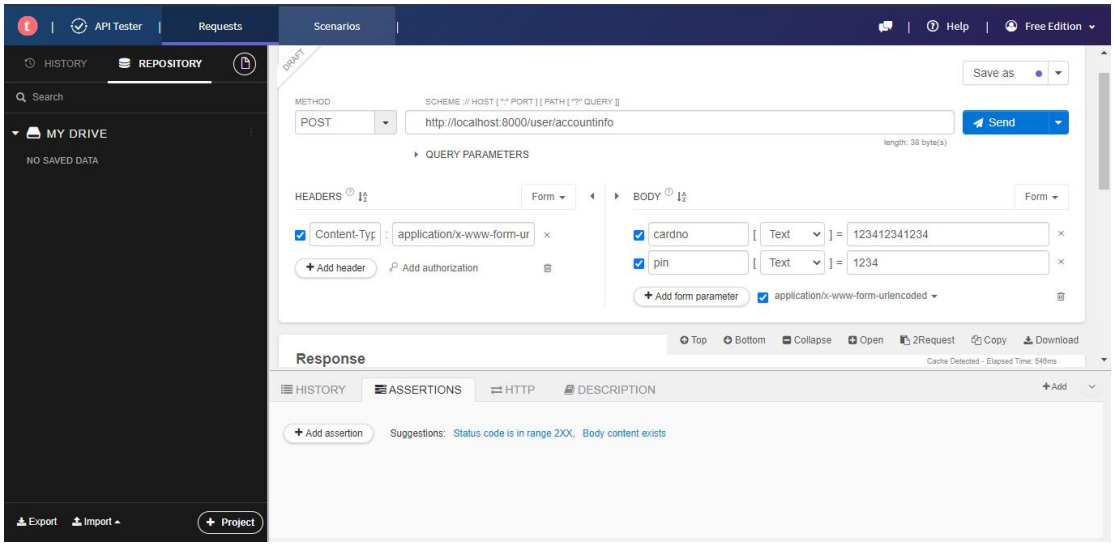
Available data in the DB:-

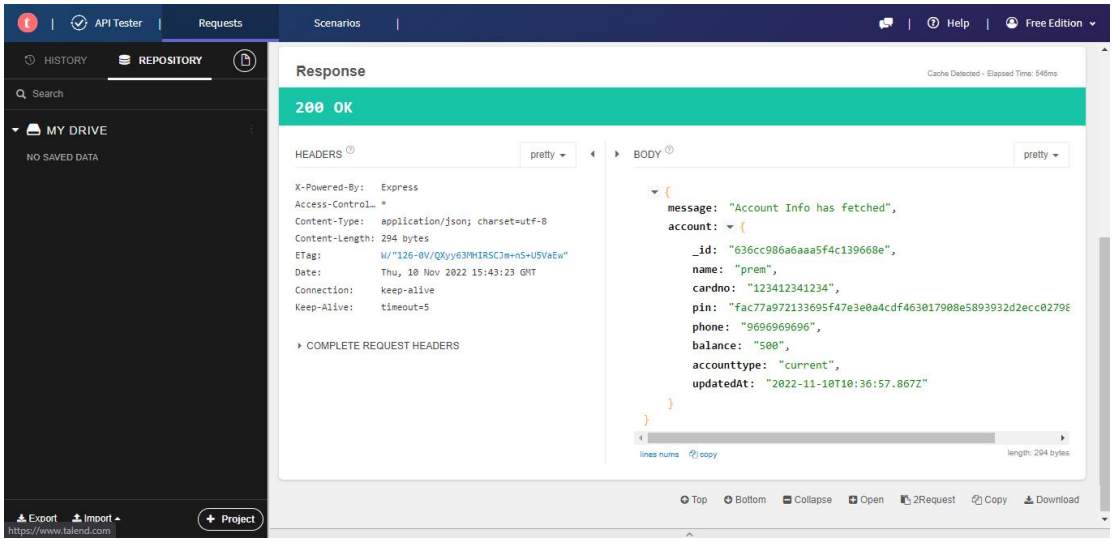
QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('636cc986a6aaa5f4c139668e')
name: "prem"
cardno: "123412341234"
pin: "fac77a972133695f47e3e0a4cdf463017908e5893932d2ecc0279818779cec71"
phone: "9696969696"
balance: "500"
accounttype: "current"
updatedAt: 2022-11-10T10:36:57.867+00:00
```

Above is expected data to be fetched by the API

We are using TELEND API TESTER Tool for testing our API's





Dynamic Testing:-

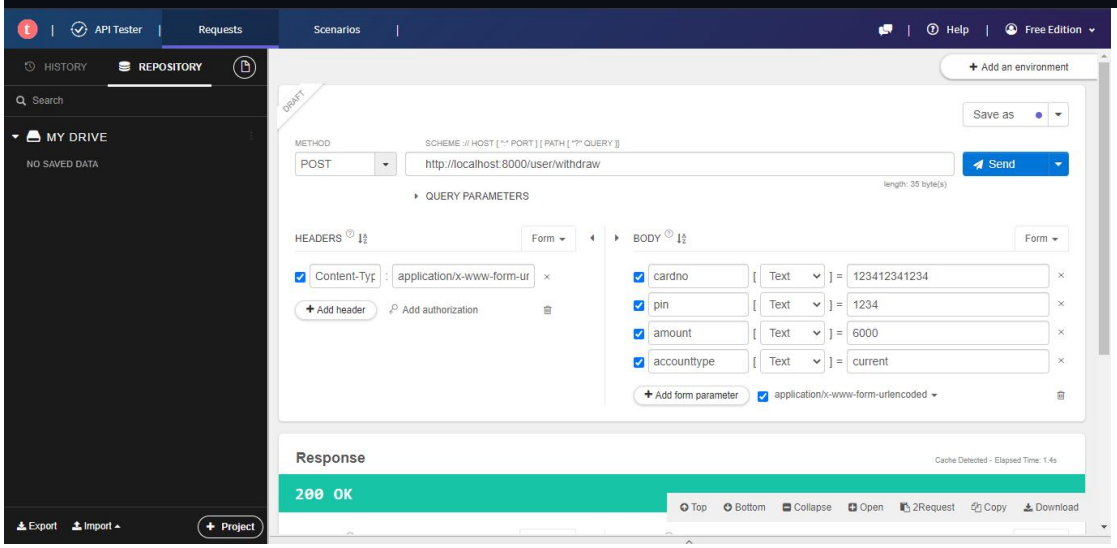
2.a) Boundary Value Analysis:-

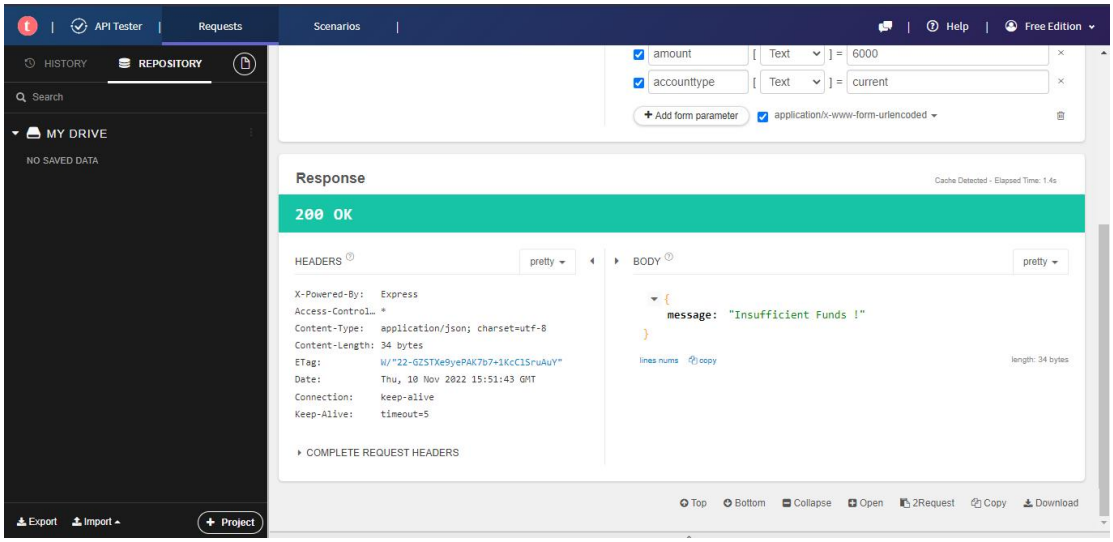
This is a technique where we analyze the behavior of the application with test data residing at the boundary values of the equivalence classes.

By using the test data residing at the boundaries, there is a higher chance of finding errors in the software application.

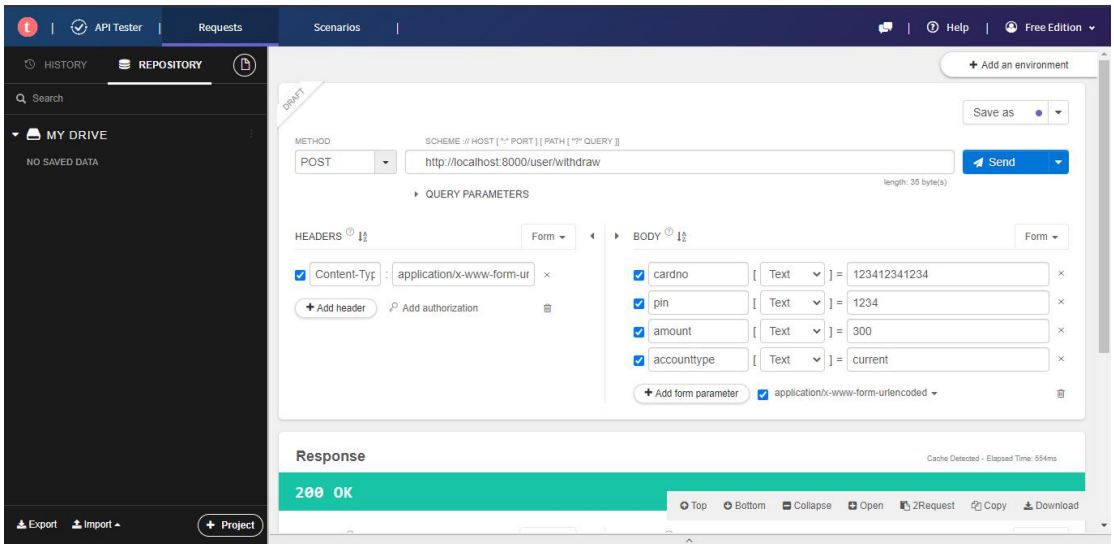
```
exports.withdrawMoney = async (req, res) => {
    const {cardno , pin , amount, accounttype} = req.body;
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });
    if(!user) throw "Invalid Pin";
    if(accounttype==user.accounttype){
        if(amount>Number(user.balance)){
            res.json({
                message: "Insufficient Funds !"
            });
        }
        else{
```

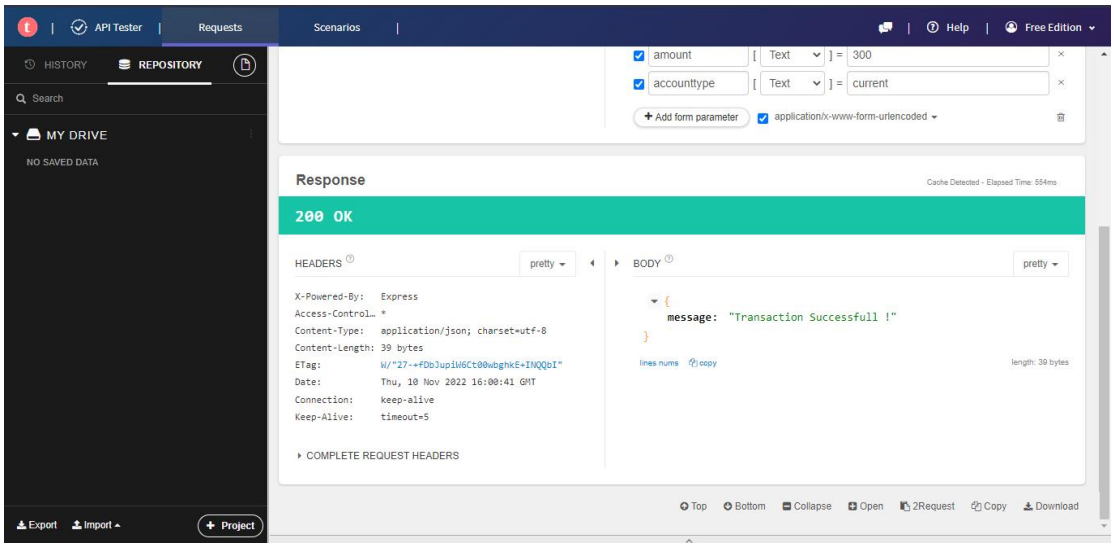
```
            balance = Number(user.balance)
            balance = balance-amount
            balance = String(balance)
            var myquery = { cardno: cardno };
            var newvalues = { $set: {balance: balance} };
            User.updateOne(myquery, newvalues,(err, res)=>{
            if (err) throw err;
            console.log("1 document updated");
        });
            res.json({
            message: "Transaction Successfull !",
        });
        };
    }
```




API Tester
Requests
Scenarios
Help
Free Edition
HISTORY
REPOSITORY
Search
MY DRIVE
NO SAVED DATA
amount [Text] = 6000
accounttype [Text] = current
Add form parameter
application/x-www-form-urlencoded
Response
200 OK
HEADERS
pretty
BODY
X-Powered-By: Express
Access-Control… *
Content-Type: application/json; charset=utf-8
Content-Length: 34 bytes
ETag: W/"22-GZSTXe9yePAK7b7+1KcC1SruAuY"
Date: Thu, 10 Nov 2022 15:51:43 GMT
Connection: keep-alive
Keep-Alive: timeout=5
message: "Insufficient Funds !"
COMPLETE REQUEST HEADERS
Top
Bottom
Collapse
Open
2Request
Copy
Download
Export
Import
Project

Valid :


API Tester
Requests
Scenarios
Help
Free Edition
HISTORY
REPOSITORY
Search
MY DRIVE
NO SAVED DATA
Add an environment
Save as
METHOD
POST
http://localhost:8000/user/withdraw
Send
QUERY PARAMETERS
HEADERS
Form
BODY
Content-Type : application/x-www-form-ur
Add header
Add authorization
cardno [Text] = 123412341234
pin [Text] = 1234
amount [Text] = 300
accounttype [Text] = current
Add form parameter
application/x-www-form-urlencoded
Response
200 OK
Top
Bottom
Collapse
Open
2Request
Copy
Download
Export
Import
Project


API Tester
Requests
Scenarios
Help
Free Edition
HISTORY
REPOSITORY
Search
MY DRIVE
NO SAVED DATA
amount [Text] = 300
accounttype [Text] = current
Add form parameter
application/x-www-form-urlencoded
Response
200 OK
HEADERS
pretty
BODY
X-Powered-By: Express
Access-Control… *
Content-Type: application/json; charset=utf-8
Content-Length: 39 bytes
ETag: W/"27-+fDbJupiW6Ct00wbghkE+INQQbI"
Date: Thu, 10 Nov 2022 16:00:41 GMT
Connection: keep-alive
Keep-Alive: timeout=5
message: "Transaction Successfull !"
COMPLETE REQUEST HEADERS
Top
Bottom
Collapse
Open
2Request
Copy
Download
Export
Import
Project

2.b) Mutation Testing:-

Here we have taken three mutation of withdrawMoney function.

Mutantion-1:-

In this mutant, user is able to proceed with transaction and also get access to thing inside even after pin he entered is wrong because here we are not checking for pin at all. So lets us not to remove it from original copy of code.// thus making it one boundary condition.

```
exports.withdrawMoney = async (req, res) => {
    const {cardno , pin , amount, accounttype} = req.body;
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });

    if(accounttype==user.accounttype){
        if(amount>Number(user.balance)){
            res.json({
                message: "Insufficient Funds !"
            });
        }
        else{
            balance = Number(user.balance)
            balance = balance-amount
            balance = String(balance)
            var myquery = { cardno: cardno };
            var newvalues = { $set: {balance: balance} };
            User.updateOne(myquery, newvalues,(err, res)=>{
            if (err) throw err;
            console.log("1 document updated");
        });
            res.json({
            message: "Transaction Successfull !",
```

```
        });
        };
    }
```

Mutation-2:-

Here in this mutation we have taken down one of our if statement which used to check for users account type, which ensures user account type match for further transaction to happen. For example: a user having savings bank account can't withdraw from business bank account.

Which make it to be there as a one of the condition.

```
exports.withdrawMoney = async (req, res) => {
    const {cardno , pin , amount, accounttype} = req.body;
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });
    if(!user) throw "Invalid Pin";

        if(amount>Number(user.balance)){
            res.json({
                message: "Insufficient Funds !"
            });
        }
        else{
            balance = Number(user.balance)
            balance = balance-amount
            balance = String(balance)
            var myquery = { cardno: cardno };
            var newvalues = { $set: {balance: balance} };
            User.updateOne(myquery, newvalues,(err, res)=>{
            if (err) throw err;
            console.log("1 document updated");
        });
            res.json({
```

```
            message: "Transaction Successfull !",
        });
    }
```

Mutation-3:-

Now think if some user trying to withdraw money more then his savings , at this scenario he must get message stating insufficient balance instead of providing money without boundary value checking or analysis on his account savings which leads us to put another if statement to check for that.

```
exports.withdrawMoney = async (req, res) => {
    const {cardno , pin , amount, accounttype} = req.body;
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });
    if(!user) throw "Invalid Pin";
    if(accounttype==user.accounttype){
        if(amount>Number(user.balance)){
            res.json({
                message: "Insufficient Funds !"
            });
        }
        else{
            balance = Number(user.balance)
            balance = balance-amount
            balance = String(balance)
            var myquery = { cardno: cardno };
            var newvalues = { $set: {balance: balance} };
            User.updateOne(myquery, newvalues,(err, res)=>{
            if (err) throw err;
            console.log("1 document updated");
        });
```

```
            res.json({
            message: "Transaction Successfull !",
        });
        };
    }
```

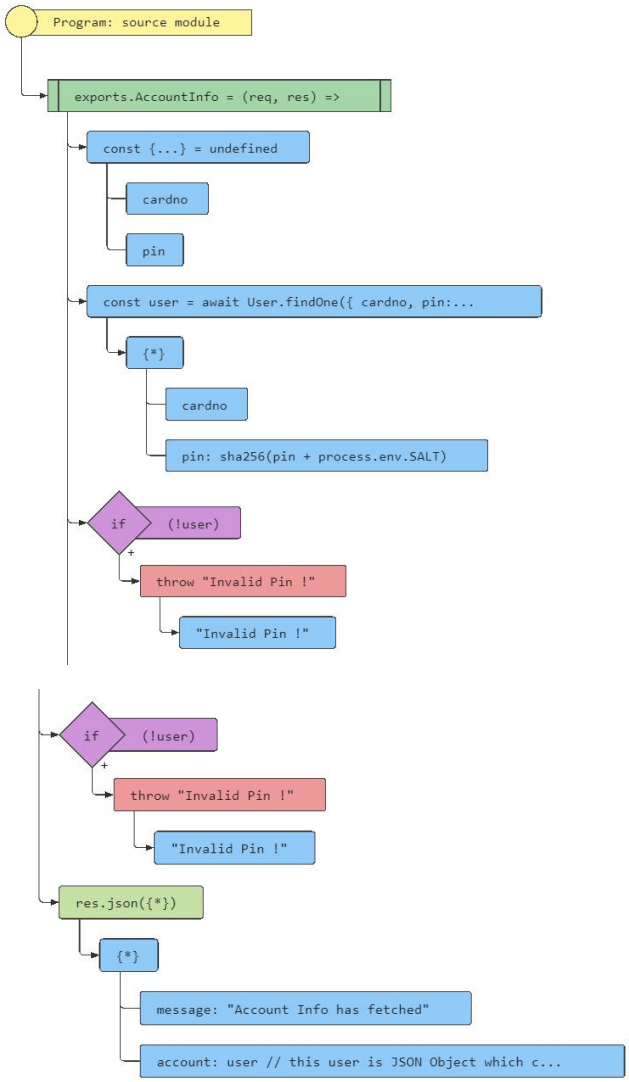
Static Testing:-

Unit description : when user gives card number and pin , this API will fetches/queries the DB for the user info which is available on the DB and fetches and displays in the resposnse page

Code :

```
exports.AccountInfo = async (req, res) => {
    const {cardno , pin} = req.body;
    // fetching in DB
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });

    if(!user) throw "Invalid Pin !";

    res.json({
        message: "Account Info has fetched",
        account: user, // this user is JSON Object  which contains user name, pin, balance, card no, account type, phone number can be extracted in front end

    });
};
```

Control Flow Graph:-



Cyclomatic Complexity=2

2.

```
exports.withdrawMoney = async (req, res) => {
```

```
    const {cardno , pin , amount, accounttype} = req.body;
    const user = await User.findOne({
        cardno,
        pin: sha256(pin+process.env.SALT),
    });
```

```
    if(!user) throw "Invalid Pin";
```

```
    if(accounttype==user.accounttype){
```

```
        if(amount>Number(user.balance)){
            res.json({
                message: "Insufficient Funds !"
            });
        }
        else{
```

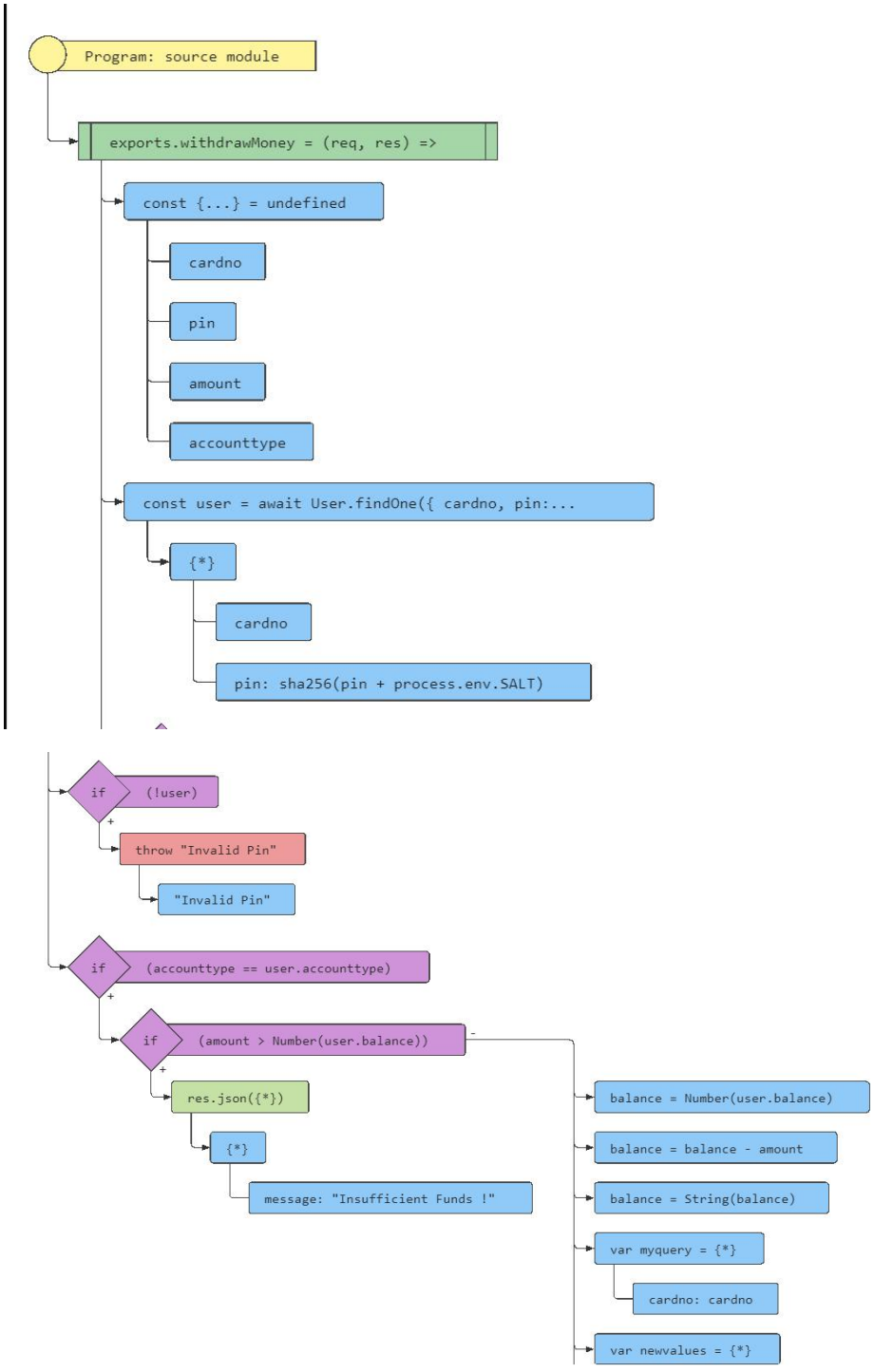
```
            balance = Number(user.balance)
            balance = balance-amount
            balance = String(balance)
            var myquery = { cardno: cardno };
            var newvalues = { $set: {balance: balance} };
```

```
            User.updateOne(myquery, newvalues,(err, res)=>{
            if (err) throw err;
            console.log("1 document updated");

        });
```

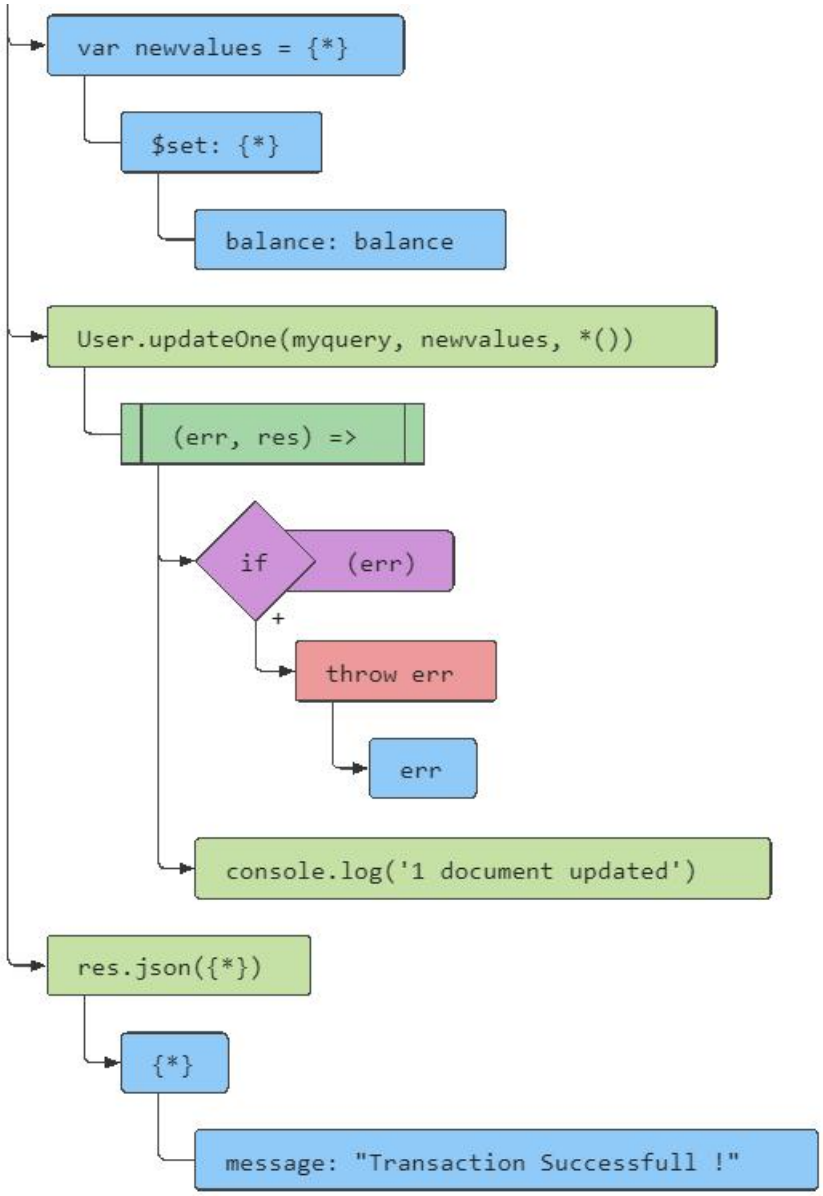
```
            res.json({
            message: "Transaction Successfull !",
        });
```

```
        };

    }
```

ATM SIMULATOR



ATM SIMULATOR



Test	Name of	Test case	Pre-	Test Steps	Test data	Expected	Actual	Test

ATM SIMULATOR

Case ID	Module	description	conditions			Results	Result	Result
UT-01	login	Login using wrong number of card digits	Working internet and access to browser	Try logging in with 15 digit card number instead of 16	CardNo=123412341234123 Pin=1234	Message saying "Card No should contain 16 digits"	Message saying "Card No should contain 16 digits"	Pass
UT-02	login	Login with wrong pin	Working internet and access to browser	Try logging in with wrong pin	CardNo=1234123412341234 Pin=4567	Message saying "Invalid card and pin no"	Message saying "Invalid card and pin no"	Pass
UT-03	Account Info	when user gives card number and pin , this API will fetches/queries the DB for the user info which is available on the DB and fetches and displays in the response page	Working internet and access to browser	Login and click on account info	CardNo=1234123412341234 Pin=1234	Account details	Account details	Pass
DT-01	withdrawMoney	Boundary Value Analysis	Working internet and access to browser	Withdraw funds more than balance	CardNo=1234123412341234 Pin=1234 Amount=3000	"insufficient funds"	"insufficient funds"	Pass
DT-02	withdrawMoney	Boundary Value Analysis	Working internet and access to browser	Withdraw funds less than balance	CardNo=1234123412341234 Pin=1234 Amount=2999	"transaction successful"	"transaction successful"	Pass
DT-03	withdrawMoney	Mutation testing	Working internet and access to browser	In this mutant, user is able to proceed with transaction and also get access to thing inside even after pin he entered is wrong because here we are not checking for pin at all	CardNo=1234123412341234 Amount=2999	transaction successful"	"transaction successful"	Pass
ST-01	accountInfo	Cyclomatic Complexity testing	Working internet and access to browser	Cyclomatic Complexity testing using the module's CFG	The module's CFG	CC=2	CC=2	Pass
ST-	withdrawMoney	Cyclomatic	Working	Cyclomatic	The module's CFG	CC=5	CC=5	Pass

02		Complexity testing	internet and access to browser	Complexity testing using the module's CFG				