FINAL PROJECT REPORT

Expense Management with Lending App

For

Course: INFO 6210

Database Management and Database Design

Under the guidance of

Professor Chaiyaporn Mutsalklisana

SUBMITTED BY Team CK

Jui Prashant Ashinkar (001443824)
Vinay Kangokar (001401602)
Rakesh Reddy Birika (001897118)
Tushar Sharma (001494599)
Nishit Samir Ajmera (001818318)

Problem Statement:

Traditional credit cards use an outdated credit scoring model to assess the credit score for a customer. It uses only credit history of the person and ignores important factors such as income level, bill payments, monthly expenses, and age group. This credit limit might be right for some users but most of them are eligible for a higher credit limit. The major reason of the ignorance of these important factors might be unavailability of this data for most of the credit card users. We aim at gathering all this important data by offering our users a payment platform where they can get credit, pay bills, and keep a track of their expenses. Providing users with a dynamic credit line compared to a static credit line will fill the gap in current credit offerings and what customers really needs.

Abstract:

Our idea is to offer consumers with a micro-lending platform where they can also keep a track of their day-to-day expenses. This system will allow us to pay all bills for user during a month and users can pay us back anytime later. It will provide easy to use interface for managing and paying bills both online as well as offline. Users must provide their SSN when they register with us and they get pre-approved for a certain amount (say $5000) of credit on the app. Many users will need a further extension of credit limit and for that we plan to gather more information about the users such as their income level, age group, monthly expense history, credit history and criminal records to develop a model which will suggest what credit limit is right for a user. In case users are not able to pay us their monthly balances within a due date, we will charge them a nominal interest rate. If a user draws amount more than their approved credit limit they will be charged with an extra interest rate component, if they are not able to pay by due date.

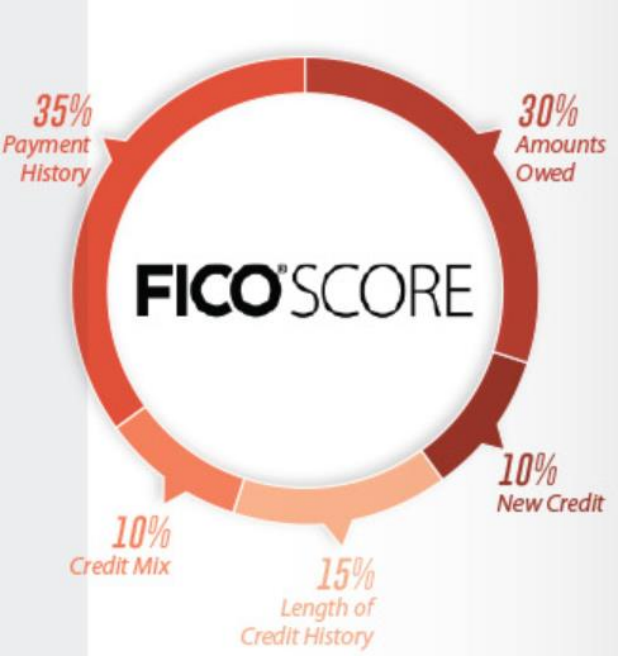
Our aim is to make users realize about their financial state and give them insights on how to spend more wisely. To achieve this, we will provide users with free monthly analytical report on spend patterns, give them insights on where they spent the most and where they spent less than previous months. This way our users can track expenses and stop worrying about the expenditures even at the end of month.

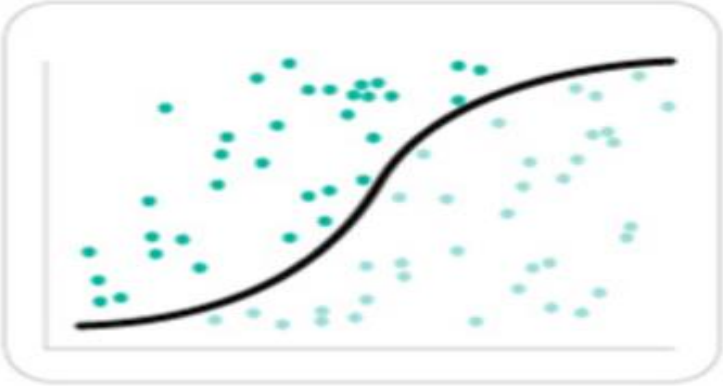
We plan on associating with investors for investments that we will use for lending to users when we pay their expenses. Lenders will be earning from part of profits we earn from interest. Interest earned is going to be our major source of revenue.

Objective and Scope:

Providing users with a better way to track their expenses along with lending option at cheaper than credit card rates is our major objective. Credit scoring should be a dynamic process and should change with factors such as monthly payments and income of a person. Many people are not able to get sufficient credit line because their traditional credit scores are not up to the mark.
According to Equifax traditional credit scores have some limitations

- It does not capture bigger picture (income levels, spend history and personal expenses)
- Since all payments are not made using credit cards and even if so, not the same bank credit card, most credit companies fail to capture all consumer's transactions in its entirety
- Losing on this information means losing on an opportunity to better assessing the consumer's creditworthiness
- Not being able to access true worthiness of consumers is a missed opportunity for lending companies
- Our expense manager fills the info gap
- We introduce our expense tracking app with a lending option in a hope to capture these missed opportunities
- Expense tracker will provide us personal info of consumers using which we can better assess their creditworthiness
- Knowing our consumers monthly finances will add a personal touch to the traditional credit score
- Personalized credit score will allow us to offer more credit to eligible users
- More money lent to credible users means more interest in return





Source: https://www.visualcapitalist.com/tech-changing-modern-credit-landscape/

The above image shows how traditional credit scores consider only Credit history and ignore all other important financial factors.

The major reason for the scores to be low is an outdated approach in generating these scores and there is very gradual increase if ever. The traditional scores only depend on credit the user has taken during their lives and not the earnings or monthly payments they have been making. Due to ignorance of these important factors this model is not able to capture the true financial status of the users.

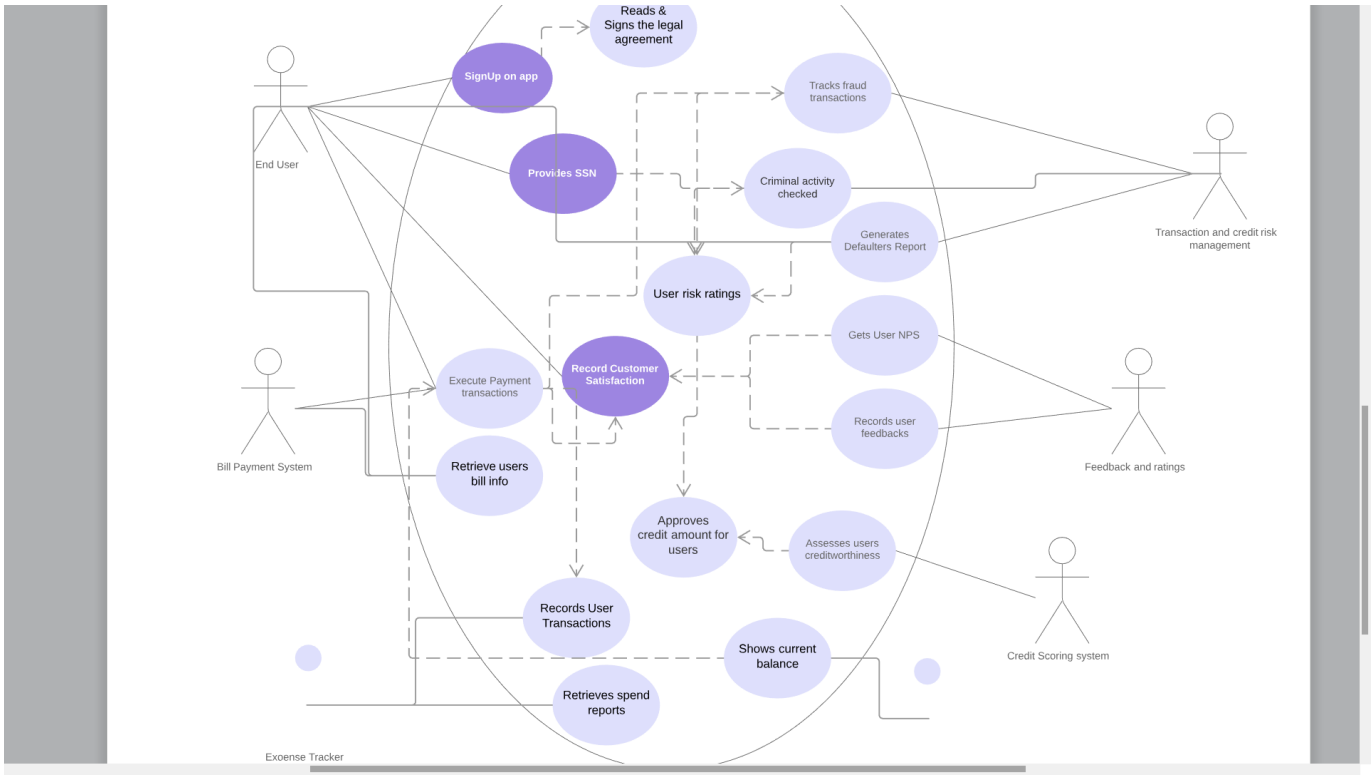
Our aim is also to make lending a safe and follow all the legal guidelines of Lending Laws. We would not want to lend a criminal or constant defaulter money by improving their scores. Therefore, we plan to include a risk assessment engine in our model. This will help us assess risk every time a new user signs up on our app and every time an existing

user makes a transaction. Getting user’s legal info like ssn and imei codes for devices will also help us to track fraud happening in our business.

We plan to collaborate with major merchants to accept our payment mode from their customers so that our acceptance among users can flourish. Along with merchants we will also get major utility providers for accepting our payment mode as an auto-payment mode for users’ bills and connections.

TOP LEVEL ACTIVITY DIAGRAM:

Before working on the UML diagrams in detail and in a proper manner we planned to create a rough level picture of what our business and database is going to look like. The same is explained in the following Top Level Activity Diagram:



SEQUENCE DIAGRAMS

1) Create Account:

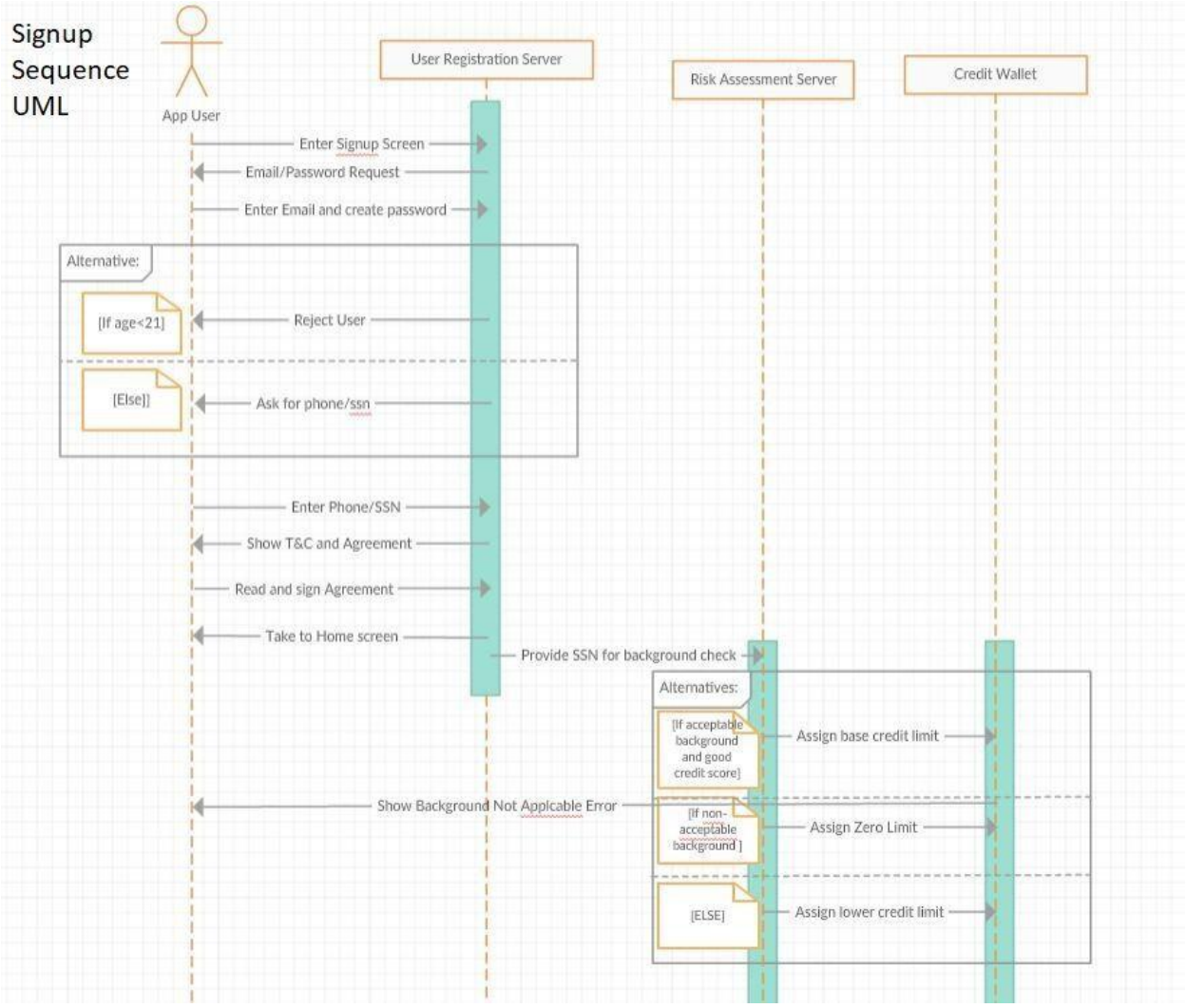
The sign up use case diagram shows the Registration process where the app user can be either a registered user or a new user

If it's a new user he/she will have to create a profile where they will have to provide personal information, read and sign agreement which will direct you to the homepage

If it's an existing user it may either login and direct you to the homepage or if the user enters the wrong credentials it will display "invalid details"

The user registration server will verify the users criminal record , email, phone number and credit score.

SIGNUP PROCESS SEQUENCE DIAGRAM



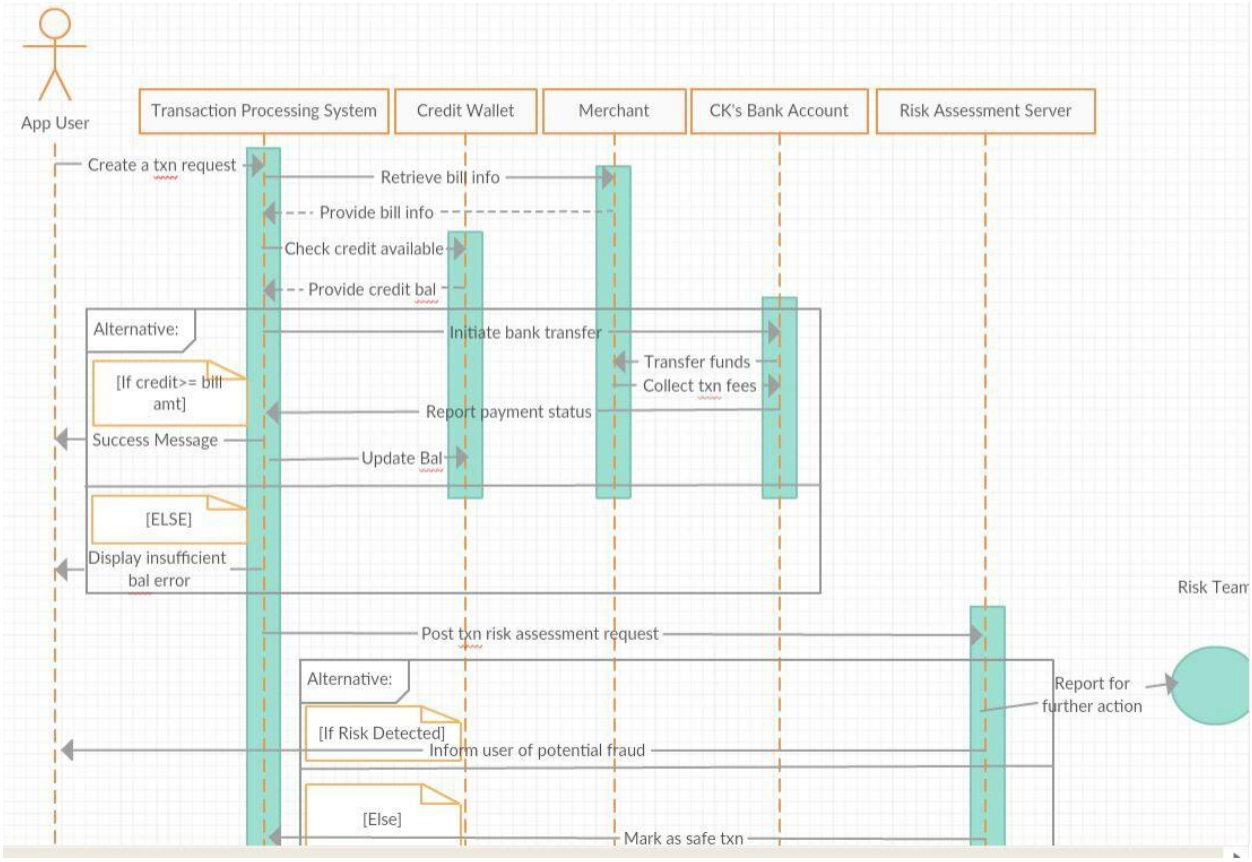
2) Payment Process:

The payment process use case diagram shows that the primary actor user will ask the transaction processing system to make a payment and he can also check his own credit wallet.

The transaction processing system will retrieve a bill amount and check the credit amount available if it is greater than the bill amount if yes he will approve the transaction and transfer the money to the merchant and in return collect transaction fees. But if credit amount is not sufficient it will display an error .

During this time the risk assessment system checks for fraud transactions and if they detect something they will report it to the Risk team if they find the transactions to be very spooky they might even confirm it with the user for verification.

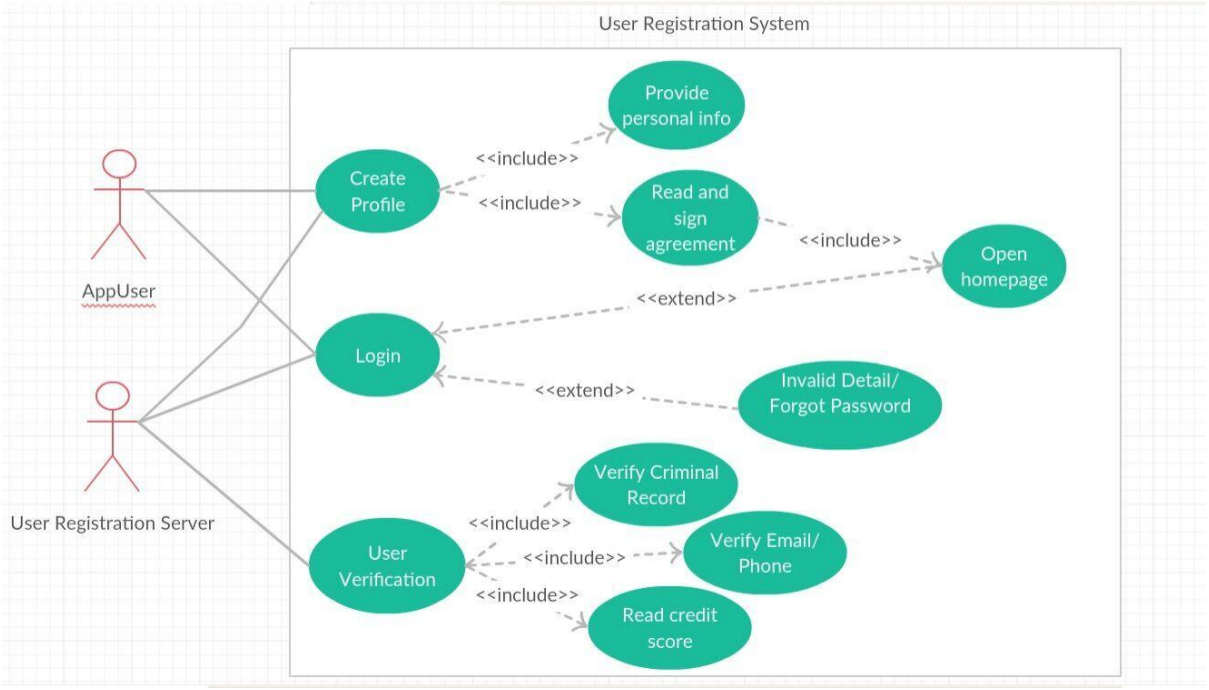
PAYMENT PROCESS SEQUENCE DIAGRAM



USE CASE DIAGRAMS

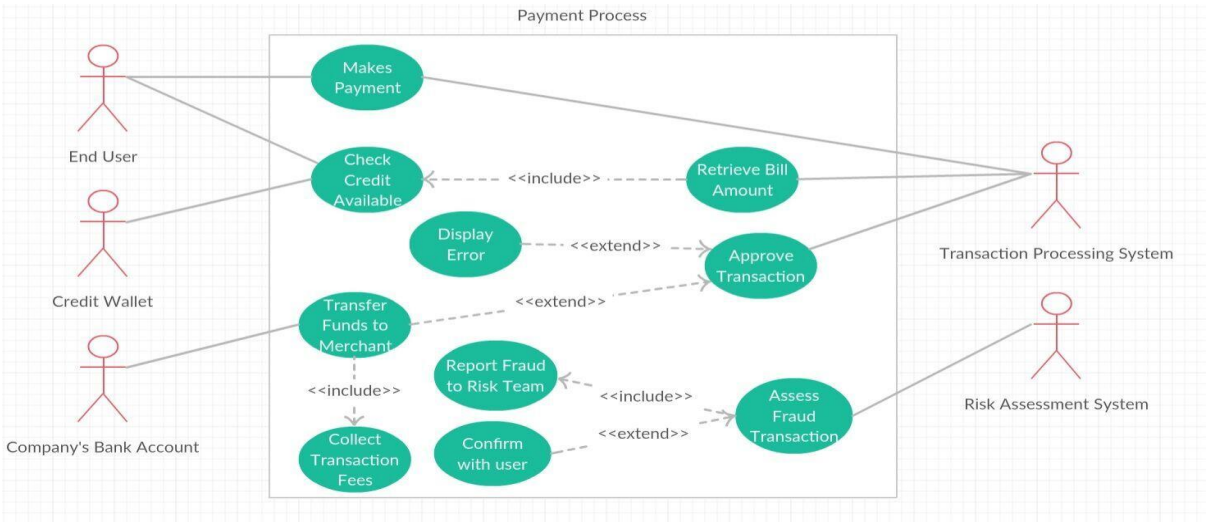
1) Signup Process:

SIGNUP PROCESS USE CASE DIAGRAM



2) Payment Process:

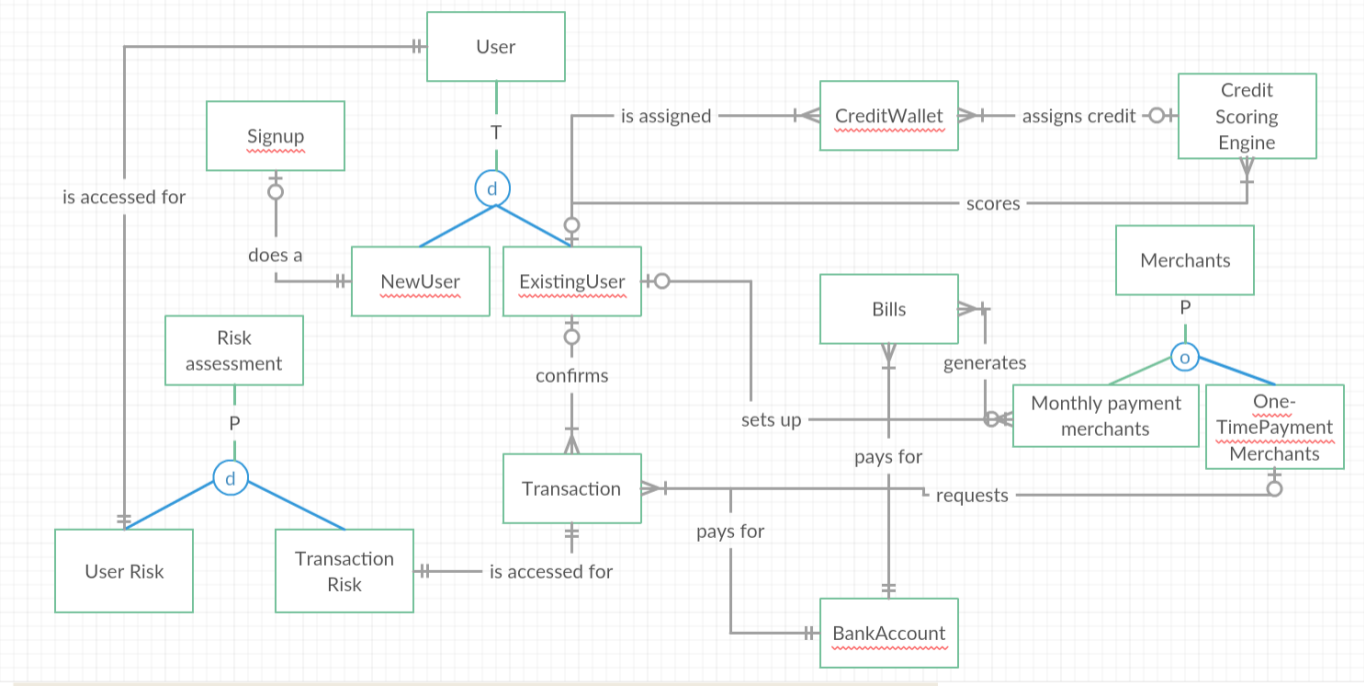
PAYMENT PROCESS USE CASE DIAGRAM



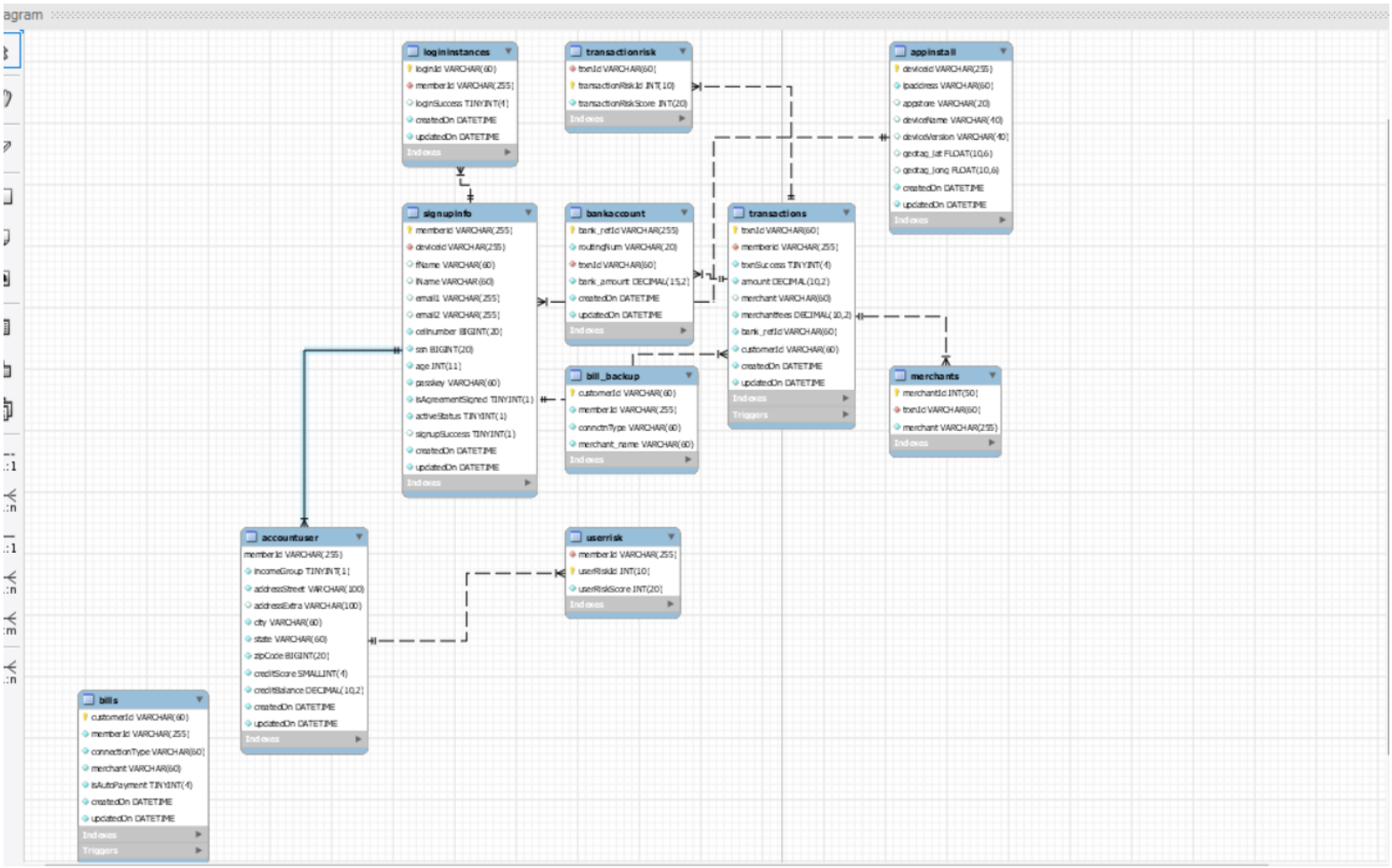
ENTITY RELATIONSHIP DIAGRAM

While creating ERD and EERD we took care of the normalization of databse including every step of normalization upto from 1NF to 3NF. We believe that creating a database with 3.5NF or BCNF form will affect the working of our webapp and data inserting and reading opeartions will become very slow. Since our webapp expects heavy traffic we have tried to create relations based on the processes included in our app. These processes are included in the EERD below.

Enhanced Entity Relationship Diagram



We also created another EERD with attributes using MySQL workbench reverse engineering option. Same is given below:


agram
loginInstances
loginId VARCHAR(60)
memberId VARCHAR(255)
loginSuccess TINYINT(4)
createdOn DATETIME
updatedOn DATETIME
Indexes
transactionrisk
txnId VARCHAR(60)
transactionRiskId INT(10)
transactionRiskScore INT(20)
Indexes
appinstall
deviceId VARCHAR(255)
ipaddress VARCHAR(60)
appstore VARCHAR(20)
deviceName VARCHAR(40)
deviceVersion VARCHAR(40)
geotag_lat FLOAT(10,6)
geotag_long FLOAT(10,6)
createdOn DATETIME
updatedOn DATETIME
Indexes
signupInfo
memberId VARCHAR(255)
deviceId VARCHAR(255)
fName VARCHAR(60)
lName VARCHAR(60)
email1 VARCHAR(255)
email2 VARCHAR(255)
cellnumber BIGINT(20)
ssn BIGINT(20)
age INT(11)
passkey VARCHAR(60)
isAgreementSigned TINYINT(1)
activeStatus TINYINT(1)
signupSuccess TINYINT(1)
createdOn DATETIME
updatedOn DATETIME
Indexes
bankaccount
bank_refId VARCHAR(255)
routingNum VARCHAR(20)
txnId VARCHAR(60)
bank_amount DECIMAL(15,2)
createdOn DATETIME
updatedOn DATETIME
Indexes
transactions
txnId VARCHAR(60)
memberId VARCHAR(255)
txnSuccess TINYINT(4)
amount DECIMAL(10,2)
merchant VARCHAR(60)
merchantFees DECIMAL(10,2)
bank_refId VARCHAR(60)
customerId VARCHAR(60)
createdOn DATETIME
updatedOn DATETIME
Indexes
Triggers
bill_backup
customerId VARCHAR(60)
memberId VARCHAR(255)
connctnType VARCHAR(60)
merchant_name VARCHAR(60)
Indexes
merchants
merchantId INT(50)
txnId VARCHAR(60)
merchant VARCHAR(255)
Indexes
accountuser
memberId VARCHAR(255)
incomeGroup TINYINT(1)
addressStreet VARCHAR(100)
addressExtra VARCHAR(100)
city VARCHAR(60)
state VARCHAR(60)
zipCode BIGINT(20)
creditScore SMALLINT(4)
creditBalance DECIMAL(10,2)
createdOn DATETIME
updatedOn DATETIME
Indexes
userrisk
memberId VARCHAR(255)
userRiskId INT(10)
userRiskScore INT(20)
Indexes
bills
customerId VARCHAR(60)
memberId VARCHAR(255)
connectionType VARCHAR(60)
merchant VARCHAR(60)
isAutoPayment TINYINT(4)
createdOn DATETIME
updatedOn DATETIME
Indexes
Triggers

CODE IMPLEMENTATION

Front End: HTML, PHP

Back End: MySQL

Tools used: MySQL 5.0, Apache server, MySQL 8.0 Workbench, Google Chrome Web Browser

We created 7 SQL tables, 6 Stored Procedures, 3 Functions, 1 Trigger, 1 View, 2 Users, 1 Backup Plan.

DATABASE CREATION SCRIPT:

```
DROP DATABASE IF EXISTS loanckr;

CREATE DATABASE IF NOT EXISTS loanckr;


USE loanCkr;


#Step 1 is to install app/open the webpage and record relevant data
```

1) **APPINSTALL:** It is the first table which will record every instance of app installed from various stores. Primary key is deviceid

```
DROP TABLE IF EXISTS appInstall;

CREATE TABLE IF NOT EXISTS appInstall(

deviceid VARCHAR(255) NOT NULL,

ipaddress VARCHAR(60) NOT NULL,

appstore VARCHAR(20),

deviceName VARCHAR(40),

deviceVersion VARCHAR(40),

geotag_lat FLOAT(10,6),

geotag_long FLOAT(10,6),
```

```
createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

PRIMARY KEY(deviceid),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

2) **SignupInfo:** Signup Table will record all the signup instances which will occur when a user enters signup information successfully

```
#Step 2 is to record user signup instance

DROP TABLE IF EXISTS signupInfo;

CREATE TABLE IF NOT EXISTS signupinfo(

memberid VARCHAR(255) NOT NULL,

deviceid VARCHAR(255) NOT NULL,

fName VARCHAR(60),

lName VARCHAR(60),

email1 VARCHAR(255),

email2 VARCHAR(255),

cellnumber BIGINT NOT NULL,

ssn BIGINT NOT NULL,

age INT NOT NULL,

passkey VARCHAR(60) NOT NULL DEFAULT '0000',

isAgreementSigned BOOLEAN NOT NULL DEFAULT false,

activeStatus BOOLEAN NOT NULL DEFAULT false,

signupSuccess BOOLEAN,

createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
```

```
PRIMARY KEY(memberid),

CONSTRAINT `fk_signupinfo_appinstall` FOREIGN KEY(deviceid) REFERENCES appInstall (deviceid),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

3) **AccountUser:** Once the user has signed up we take them to home after login and ask them to updated their account info such as income level, address etc. Here the idea is to collect some relevant information for credit scoring. Once the user submits this information we assign a credit score based on these factors and also a credit balance is allocated to the user.

```
DROP TABLE IF EXISTS accountUser;

CREATE TABLE IF NOT EXISTS accountUser(

memberId VARCHAR(255) NOT NULL,

incomeGroup TINYINT(1) NOT NULL DEFAULT 0,

addressStreet VARCHAR(100) NOT NULL,

addressExtra VARCHAR(100),

city VARCHAR(60) NOT NULL,

state VARCHAR(60) NOT NULL,

zipCode BIGINT NOT NULL,

creditScore SMALLINT(4) NOT NULL DEFAULT 0,

creditBalance DECIMAL(10,2) NOT NULL DEFAULT 0,

createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

PRIMARY KEY(memberId),
```

```
CONSTRAINT `fk_demographics_signupinfo` FOREIGN KEY(memberId) REFERENCES signupinfo (memberId),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

4) **Logininstances:** Whenever the user logs on the app a record is created to keep the session activity info. This info can be used by transaction risk assessment server to assess whether device used was same as signup, login was done before transaction or not. This info will help us to make our business safer.

```
#Step 3 is to record user login instances

DROP TABLE IF EXISTS loginInstances;

CREATE TABLE IF NOT EXISTS loginInstances(

loginId VARCHAR(60) NOT NULL,

memberId VARCHAR(255) NOT NULL,

loginSuccess tinyint,

createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

PRIMARY KEY(loginId),

CONSTRAINT `fk_loginInstances_signupinfo` FOREIGN KEY(memberId) REFERENCES signupinfo (memberId),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

5) **Transactions:** When a user receives a transaction request from a merchant all they have to do go to transactions page and accept/decline the request.

```
#Step 4 is to record user Transactions

DROP TABLE IF EXISTS transactions;

CREATE TABLE IF NOT EXISTS transactions(

trxnId VARCHAR(60) NOT NULL,

memberid VARCHAR(255) NOT NULL,

trxnSuccess tinyint NOT NULL DEFAULT false,

amount decimal(10,2) NOT NULL CHECK(amount> 0),

merchant VARCHAR(60),

merchantfees decimal(10,2) NOT NULL CHECK(amount> 0),

bank_refId VARCHAR(60) NOT NULL,

customerId VARCHAR(60) NOT NULL DEFAULT 'noBill',

createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

PRIMARY KEY(trxnId),

CONSTRAINT `fk_transactions_signupinfo` FOREIGN KEY(memberid)
REFERENCES signupInfo (memberid),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

6) **Bills:** A user can add auto payment bills info which will keep occurring regularly using on our system. Users can stop the autopayment and that will be recorded in this table as well.

```
#Step 4 is to get utility/subscriptions connection info

DROP TABLE IF EXISTS bills;

CREATE TABLE IF NOT EXISTS bills(

customerId VARCHAR(60) NOT NULL,

memberId VARCHAR(255) NOT NULL,
```

```
connectionType VARCHAR(60) NOT NULL,

merchant VARCHAR(60) NOT NULL,

isAutoPayment tinyint NOT NULL DEFAULT false, #loginid as autopayment

createdOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

updatedOn DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

PRIMARY KEY(customerId),

INDEX(createdOn),

INDEX(updatedOn)

)ENGINE=InnoDB;
```

7) **Merchants:** merchant names will be stored in this table

```
DROP TABLE IF EXISTS merchants;

CREATE TABLE IF NOT EXISTS merchants(

merchantId int(50) NOT NULL,

merchant VARCHAR(255) NOT NULL,

PRIMARY KEY(merchantId)

)ENGINE=InnoDB;
```

FUNCTION CREATION SCRIPT:

```
#FUNCTIONS

USE loanckr;
```

1) **F_randChar()** generates a random character from 26 alphabets which we used to create random IDs and other data.

```
DROP FUNCTION IF EXISTS F_randChar;

DROP FUNCTION IF EXISTS F_randNum;

DELIMITER $$

CREATE FUNCTION F_randChar(len int)

RETURNS VARCHAR(255)

DETERMINISTIC

BEGIN

        DECLARE randChar VARCHAR(255);

   SET randChar = substring('abcdefghijklmnopqrstuvwxyz',FLOOR(RAND()*26.5),1);

        WHILE (LENGTH(randChar) < len) DO

                SET randChar = CONCAT(randChar, substring('abcdefghijklmnopqrstuvwxyz',FLOOR(RAND()*26.5),1));

        END WHILE;

        RETURN randChar;

END$$
```

2) **F_randNum()** generates a random number of given length. This was also used to generate random data.

```
DELIMITER $$

CREATE FUNCTION F_randNum(len int)

RETURNS int

DETERMINISTIC
```

```
BEGIN
        DECLARE randNum VARCHAR(255);
    SET randNum = substring('0123456789',FLOOR(RAND()*10.5),1);
        WHILE (LENGTH(randNum) < len) DO
                SET randNum= CONCAT(randNum,
substring('0123456789',FLOOR(RAND()*10.5),1));
        END WHILE;
        RETURN CAST(randNum AS UNSIGNED);
END$$
```

3) **F_CheckExistence()** uses signupinfo table to check whether the user exists or not. We need to update many tables based on this condition if true.

```
DELIMITER $$
CREATE FUNCTION F_checkExistence(memberidp VARCHAR(255))
RETURNS int
DETERMINISTIC
BEGIN
        DECLARE userExists boolean;
    SET userExists = IF(EXISTS(SELECT memberid FROM loanckr.signupInfo
WHERE memberid = memberidp AND activeStatus = true), true, false);
        RETURN userExists;
END$$
```

STORED PROCEDURES SCRIPT:

APPINSTALL : This procedure updates the appinstall table.

```
USE loanckr;

DROP PROCEDURE IF EXISTS loanckr.SP_appInstall;

DELIMITER $$

CREATE PROCEDURE loanckr.SP_appInstall()

BEGIN

        DECLARE deviceid CHAR;

        DECLARE ipaddress CHAR;

        DECLARE appstore CHAR;

        DECLARE deviceName CHAR;

        DECLARE deviceVersion CHAR;

        DECLARE geotag_lat CHAR;

        DECLARE geotag_long CHAR;

        SET @deviceid = CONCAT(UPPER(F_randChar(10)), CAST(F_randNum(5) AS CHAR(5)), F_randChar(7));

        SET @ipaddress = CONCAT('1',F_randNum(2),'.',F_randNum(2),'.',F_randNum(1),'.',F_randNum(1));

        SET @appstore = ELT(CEIL((rand()*5)+0.1),'webapp','appstore','playstore','samsungapps','','');

        SET @deviceName = CASE WHEN @appstore = 'webapp' THEN ELT(CEIL((rand()*5)+0.1),'lenovo','samsung','apple','dell','','')

                                WHEN @appstore = 'appstore' THEN 'apple'

                                WHEN @appstore = 'playstore' THEN ELT(CEIL((rand()*5)+0.1),'google','samsung','lenovo','OnePlus','','')

                                WHEN @appstore = 'samsungapps' THEN 'samsung'

                                ELSE '' END;

        SET @deviceVersion = CONCAT(CAST(F_randNum(1) AS CHAR),'.',CAST(F_randNum(1) AS CHAR));
```

```
        SET @geotag_lat = F_randNum(2)+round(rand() * 0.99 + 0.01, 6);

        SET @geotag_long = F_randNum(2)+round(rand() * 0.99 + 0.01, 6);


    INSERT IGNORE INTO loanckr.appInstall
(deviceid,ipaddress,appstore,deviceName,deviceVersion,geotag_lat,geotag_long,cre
atedOn,updatedOn)

    VALUES
(@deviceid,@ipaddress,@appstore,@deviceName,@deviceVersion,@geotag_lat,@
geotag_long, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);


COMMIT;

END $$
```

SIGNUP PROCEDURE: This procedure records all signup instances and updates the signup table.

```
USE loanckr;

DROP PROCEDURE IF EXISTS loanckr.SP_signupInfo;

DELIMITER $$

CREATE PROCEDURE loanckr.SP_signupInfo (IN fNamep varchar(60), IN lNamep
varchar(60), IN cellNumberp BIGINT, IN ssnp BIGINT, IN memberidp
VARCHAR(255),IN passwordp1 VARCHAR(60), IN agep int, IN deviceidp
VARCHAR(255), OUT errmsg VARCHAR(255))

MODIFIES SQL DATA

BEGIN

        DECLARE inputdata int;


    SET @inputdata = 0;


    #error handling
```

```
SELECT CASE WHEN LENGTH(cellnumberp) <> 10 THEN 1
    WHEN LENGTH(ssnp) <> 9 THEN 2
    WHEN LENGTH(memberidp) = 0 THEN 3
    WHEN LENGTH(passwordp1) < 3 THEN 4
    WHEN F_checkExistence(memberidp) = true THEN 5
    ELSE 0 END INTO @inputdata;


     IF @inputdata = 0
THEN
            INSERT IGNORE INTO loanckr.signupInfo(memberid,deviceid,fName,lName,email1,email2,cellnumber,ssn,age,passkey,isAgreementSigned,activeStatus,signupSuccess,createdOn,updatedOn)
            VALUES (memberidp,deviceidp,fNamep,lNamep,'','',cellNumberp,ssnp,agep,passwordp1,true,true,true, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);
            #ON DUPLICATE KEY UPDATE ;
            SET errmsg  = '';
ELSEIF @inputdata = 1
THEN
            SET errmsg = 'Cellnumber needs to be 10 digit number';
ELSEIF @inputdata = 2
THEN
            SET errmsg = 'SSN needs to be 9 digit number';
     ELSEIF @inputdata = 3
THEN
            SET errmsg = 'Email cannot be empty';
     ELSEIF @inputdata = 4
THEN
```

```
            SET errmsg = 'Password should be more than 3 characters';
        ELSEIF @inputdata = 5
    THEN
            SET errmsg = 'This user already exists. Use a different email.';
        END IF;
COMMIT;
END$$
DELIMITER ;
```

LOGIN: This procedure records all login instances and information is saved in loginInstances table.

```
USE loanckr;
DROP PROCEDURE IF EXISTS loanckr.SP_login;
DELIMITER $$
CREATE PROCEDURE loanckr.SP_login (IN memberidp VARCHAR(255), IN passkeyp VARCHAR(60), OUT phpmsg VARCHAR(60))
BEGIN
        DECLARE loginidp VARCHAR(10);
    DECLARE conditionCheck boolean DEFAULT false;
    SET loginidp = CONCAT(F_randNum(4),F_randChar(2),F_randNum(1),F_randChar(1),F_randNum(2));
    SET @conditionCheck = IF((SELECT passkey FROM loanckr.signupInfo WHERE memberid = memberidp) = passkeyp, true, false);
    IF(@conditionCheck  = true AND F_checkExistence(memberidp)=true)
            THEN
                SET phpmsg = 'CKHome.php';
        INSERT IGNORE INTO loanckr.loginInstances(loginId, memberId, loginSuccess, createdOn, updatedOn)
```

```
        VALUES (loginidp, memberidp, true, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);


                ELSE

                        SET phpmsg = IF(F_checkExistence(memberidp),'Wrong Password! Go back to','You were not found. Click on ');

        END IF;

END$$

DELIMITER ;
```

SP_Useraccount: This procedure records all account update instances and also assigns a credit score and credit balance for the user.

```
# Create Account

USE loanckr;

DROP PROCEDURE IF EXISTS loanckr.SP_useraccount;

DELIMITER $$

CREATE PROCEDURE loanckr.SP_useraccount (IN memberidp VARCHAR(255), IN income decimal(10,2), IN addressStreetp VARCHAR (255), IN addressExtrap VARCHAR(255), IN zipCodep VARCHAR(5), IN cityp VARCHAR(60), IN statep VARCHAR(60))

BEGIN

                DECLARE incomeGroupp INT;

                DECLARE creditScorep SMALLINT(4);

                DECLARE credit_amt decimal(10,2);


                SET incomeGroupp = F_randNum(1);

        SET creditScorep = F_randNum(3);

        SET creditScorep = CASE WHEN creditScorep< 100 THEN creditScorep+100
```

```
                                                    ELSE creditScorep END;

        SET incomeGroupp = CASE WHEN income BETWEEN 0 AND 50000 THEN 1
                                                    WHEN income BETWEEN
50000 AND 75000 THEN 2
                        WHEN income BETWEEN 75000 AND 100000 THEN 3
                        WHEN income > 100000 THEN 4
                       ELSE 0 END;

          IF creditScorep < 300 then
                                SET credit_amt = 1000;
                         ELSEIF creditScorep BETWEEN 300 AND 500 then
                                SET credit_amt = 2000;
                         ELSEIF creditScorep BETWEEN 500 AND 700 then
                                SET credit_amt = 4000;
                         ELSEIF creditScorep BETWEEN 700 AND 900 then
                                SET credit_amt = 5000;
                         ELSE
                                SET credit_amt = 10000;
                         END IF;

        IF (F_checkExistence(memberIdp) = true)
                                THEN
                                INSERT IGNORE INTO loanckr.accountUser(memberId ,
incomeGroup, addressStreet, addressExtra, city, state, zipCode, creditScore,
creditBalance, createdOn, updatedOn)
                        VALUES (memberidp, incomeGroupp, addressStreetp, addressExtrap,
cityp, statep, zipCodep, creditscorep, credit_amt, CURRENT_TIMESTAMP(),
CURRENT_TIMESTAMP())
```

```
            ON DUPLICATE KEY UPDATE incomeGroup = incomeGroupp,
                                    addressStreet = addressStreetp,
                                    addressExtra = addressExtrap,
                                    city = cityp,
                                    state = statep,
                                    creditscore = creditscorep,
                                    updatedOn = CURRENT_TIMESTAMP;
    END IF;
END$$
DELIMITER ;
```

SP_maketransaction: This procedure records enables transactions and after checking all conditions records the info of the transaction and send message to php server that transaction can be made successfully.

```
USE loanckr;
DROP PROCEDURE IF EXISTS loanckr.SP_maketransaction;
DELIMITER $$
CREATE PROCEDURE loanckr.SP_maketransaction(IN memberidp VARCHAR(255), IN amountp decimal(10,2), IN merchantp VARCHAR(60), IN trxnSuccessp VARCHAR(10))
BEGIN
        DECLARE merchantfees decimal(10,2);
   DECLARE trxnidp VARCHAR(60);
   DECLARE bankrefidp VARCHAR(60);
   DECLARE txnStatus boolean;

   SET @trxnidp = CONCAT(F_randNum(3),F_randChar(3),F_randNum(1),F_randChar(2),F_randNum(1),F_randChar(1));
```

```
    SET @merchantfees  = ROUND(0.03*amountp,2);

    SET @bankrefidp = CONCAT(F_randNum(1), F_randChar(3), F_randNum(2),F_randChar(1),'_bankref');

        SET @txnStatus = IF(trxnSuccessp = 'Accept', true, false);


    IF(amountp <= (SELECT creditbalance FROM loanckr.accountUser WHERE memberid = memberidp))

                THEN

                    INSERT IGNORE INTO loanckr.transactions (trxnId,memberid, trxnSuccess,amount,merchant,merchantfees,bank_refId,createdOn,updatedOn)

         VALUES (@trxnidp,memberidp,@txnStatus,amountp,merchantp,@merchantfees,@bankrefidp ,CURRENT_TIMESTAMP(),CURRENT_TIMESTAMP());

        END IF;

END $$

DELIMITER ;
```

SP_addAutoBill: This procedure is called when user wants to add an auto bill payment.

```
USE loanckr;

DROP PROCEDURE IF EXISTS loanckr.SP_addAutoBill;

DELIMITER $$

CREATE PROCEDURE loanckr.SP_addAutoBill(IN memberidp VARCHAR(255))

BEGIN

        DECLARE customerIdp VARCHAR(10);

    DECLARE connectionType VARCHAR(10);

    DECLARE merchantp VARCHAR(60);


    SET @customerIdp = CONCAT(F_randNum(3),F_randChar(5),F_randNum(4));
```

```
SET @merchantp = (SELECT merchant FROM ((SELECT CAST(CONCAT(1,F_randNum(1)) AS CHAR) AS id) A JOIN merchants B ON A.id = B.merchantId));


IF (F_checkExistence(memberidp)=true)

THEN

INSERT IGNORE INTO loanckr.bills (customerId,memberId,connectionType,merchant,isAutoPayment,createdOn,updatedOn)

VALUES (@customerIdp,memberIdp,@merchantp,@merchantp,1, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);

END IF;


COMMIT;

END $$

DELIMITER ;
```

T_updateCreditBalance: This trigger is called after insert on transactions table updates the balance in accountUser table.

```
## TRIGGER TO UPDATE CREDIT WALLET BALANCE AFTER EVERY TRANSACTION

USE loanckr;

DROP TRIGGER IF EXISTS loanckr.T_updateCreditBalance;

DELIMITER $$

CREATE TRIGGER loanckr.T_updateCreditBalance

AFTER INSERT

ON loanckr.transactions FOR EACH ROW

BEGIN

IF (NEW.trxnSuccess= 1)

THEN
```

```
        UPDATE accountUser A
    SET A.creditBalance= A.creditBalance-NEW.amount
    WHERE A.memberid = NEW.memberid;
END IF;
END $$
DELIMITER ;
```

USER PRIVILEGES SCRIPT:

```
###### WEBAPP USER FOR INSERT, SELECT AND UPDATE (DELETE NOT PROVIDED) #####
DROP USER IF EXISTS 'webapp'@'localhost';
CREATE USER 'webapp'@'localhost' IDENTIFIED BY '1413';

SELECT * FROM mysql.user;

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'webapp'@'localhost';

GRANT       INSERT,
            UPDATE,
    SELECT,
    EXECUTE
ON *.*
TO 'webapp'@'localhost';

SHOW GRANTS FOR 'webapp'@'localhost';
```

```
###### Account Team USER FOR Only View #####
DROP USER IF EXISTS 'account'@'localhost';
CREATE USER 'account'@'localhost' IDENTIFIED BY '1413';

SELECT * FROM mysql.user;

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'account'@'localhost';

GRANT SELECT
ON loanckr.accountsTeamView
TO 'account'@'localhost';

SHOW GRANTS FOR 'account'@'localhost';
```

CREATE VIEWS SCRIPT:

```
#Views
####### Transactions and Users View for Accounts Team
DROP VIEW IF EXISTS loanckr.accountsTeamView;
CREATE VIEW loanckr.accountsTeamView
AS
        SELECT      A.*,
                    B.creditScore,
         B.creditBalance
   FROM loanckr.transactions A
        JOIN  loanckr.accountUser B
        ON A.memberid = B.memberid;
```

```
SELECT  *

FROM loanckr.accountsTeamView;
```

BACKUP PLAN:

WE PLAN TO TAKE INCREMENTAL BACKUP EVERY NIGHT AT 03:00AM and FULL BACKUP ONCE EVERY MONDAY MORNING;

1. Since our app is supposed to be busy entire day we plan late night daily incremental backup.

2. We expect our app to be a little less busy on Monday early Mornings so we can take FULL backup on monday 03:00AM

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Full	Incremental	Incremental	Full	Incremental	Incremental	Incremental
Backup	Backup	Backup	Incremental	Backup	Backup	Backup
(2-7AM)	(3-4 AM)	(3-4 AM)	(3-5 AM)	(3-4 AM)	(3-4 AM)	(3-4 AM)

SNAPSHOTS OF FRONTEND:

Signup Page:

Login Page:

CK Login

Email:

Password:

Log In

Not a member yet? Sign up

Home Page:

CK Home

My Account
Credit Wallet
Make a Transaction
Add an Auto Bill Payment(COMING SOON)
View Expense Report (COMING SOON)
Pay CK credit bill (COMING SOON)

Account Update:

Account Update

Go to HOME

Annual Income($):

Street Address:

Address Line 2(Optional):

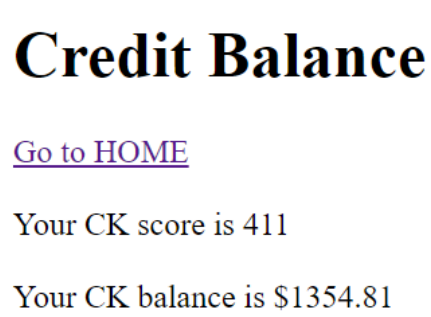
City:

State:

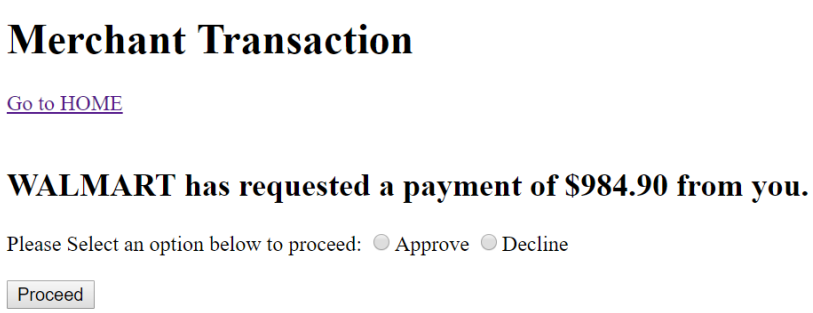
ZIP Code:

Update Info

Credit Wallet:



Transaction Page:



REFERENCES USED:

1) http://www.visualcapitalist.com/tech-changing-modern-credit-landscape/
2) https://stackoverflow.com/
3) https://www.investopedia.com/
4) https://superuser.com/questions/748117/how-to-manually-install-apache-php-and-mysql-on-windows
5) Course Content provided for INFO6210