GITAM DEEMED TO BE UNIVERSITY

# GITAM SCHOOL OF TECHNOLOGY

A project report on

**Credit Card Management System**

BY

| | |
|---|---|
| B. Sreelatha rani | BU22CSEN0500309 |
| M. Sumanth | BU22CSEN0500271 |
| C.H. Srikanth Reddy | BU22CSEN0500230 |
| G. Avinash Reddy | BU22CSEN0500349 |

Guided by:

Mr. Kerenalli Sudarshana

Department of Computer Science and Engineering

Signature:

Date:

# PROJECT TITLE

# CREDIT CARD MANAGEMENT SYSTEM.

# TABLE OF CONTENTS:

# ABSTRACT

In today's fast-paced world, ensuring the security of financial transactions is more important than ever. This project introduces a Credit Card Management System designed to make managing credit card transactions safer and easier. The system focuses on key security features, such as PIN validation, multi-factor authentication (MFA) with OTP, and fraud detection to protect users from unauthorized access and fraudulent activities.

In addition, the system allows users to set online and hard cash transaction limits, providing them more control over their spending. Users can also manage EMI payments, ensuring their monthly installments are tracked and handled effectively. The system automatically blocks the card for 24 hours if incorrect authentication (PIN or OTP) is attempted multiple times, adding an extra layer of protection. It also sends notifications and balance sheets via email, helping users keep track of their financial activities.

The project is built using C programming, incorporating advanced features like time-based operations, error handling, and modular functions to ensure a smooth and secure transaction experience. The end goal is to provide users with a reliable, easy-to-use, and secure credit card management system that offers flexibility, security, and control over their transactions.

## Problem Statement

Develop a Credit Card Management System that handles various operations like validating PIN, managing transactions, applying transaction limits, and EMI handling. Implement security features like card blocking after failed attempts and real-time transaction limits, ensuring user authentication with OTP and CVV validation.

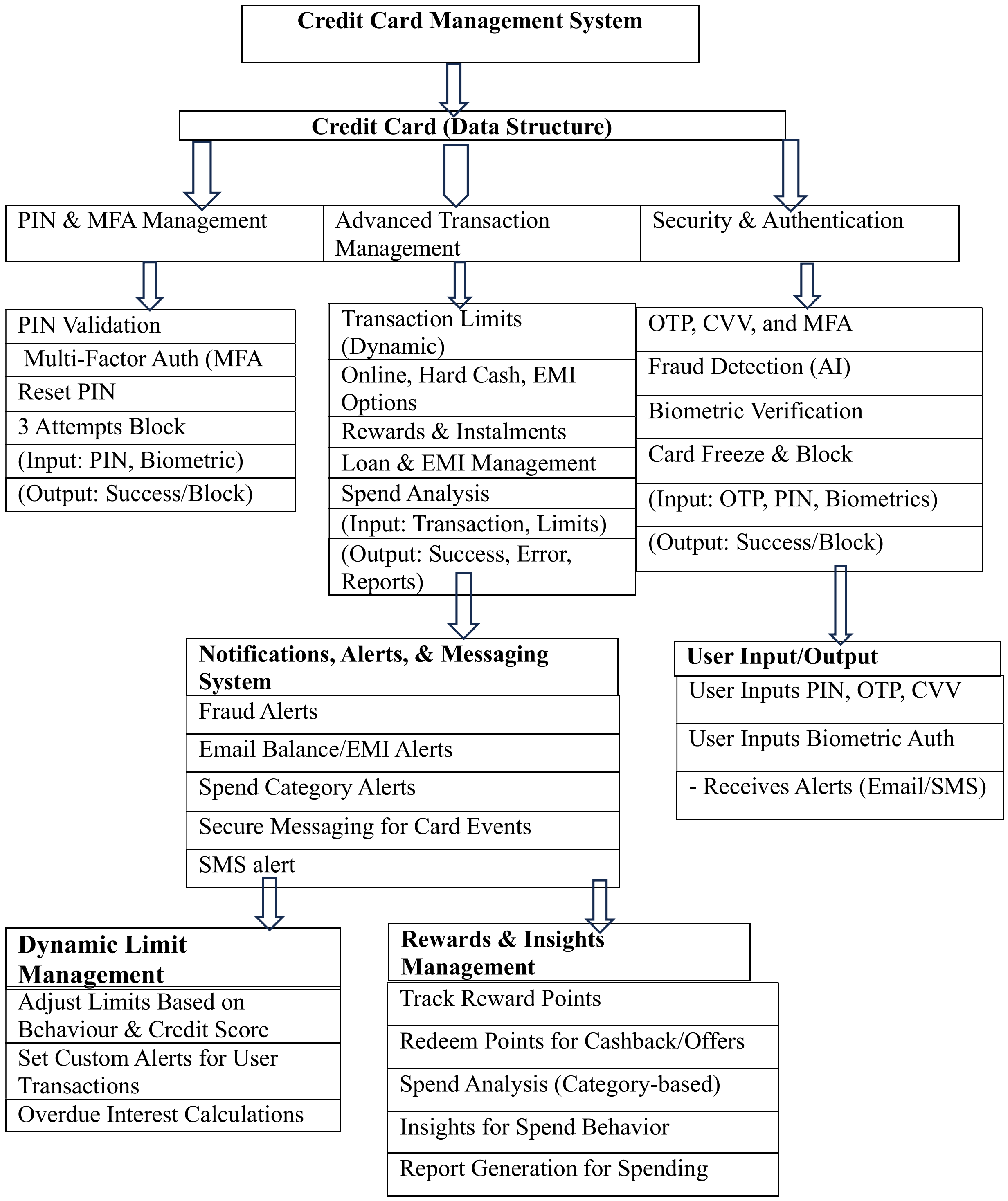
## Introduction:

In an increasingly digital world, credit cards have become a staple for many consumers, providing convenience and flexibility in financial transactions. However, the rise in credit card usage also brings significant challenges related to security and management.

As cyber threats continue to evolve, ensuring the safety of financial transactions has never been more critical. This Credit Card Management System project addresses these challenges by offering a comprehensive solution that simplifies the management of credit cards while prioritizing user security.

The significance of this project lies in its ability to empower users to take control of their credit card transactions. By implementing features such as PIN validation, transaction limits, and EMI management, the system allows users to make informed decisions about their spending. Furthermore, the inclusion of security measures—like blocking a card for 24 hours after incorrect PIN or OTP attempts—demonstrates a commitment to protecting users from unauthorized access.

# FLOWCHART


Credit Card Management System
Credit Card (Data Structure)
PIN & MFA Management
Advanced Transaction Management
Security & Authentication
PIN Validation
Multi-Factor Auth (MFA
Reset PIN
3 Attempts Block
(Input: PIN, Biometric)
(Output: Success/Block)
Transaction Limits (Dynamic)
Online, Hard Cash, EMI Options
Rewards & Instalments
Loan & EMI Management
Spend Analysis
(Input: Transaction, Limits)
(Output: Success, Error, Reports)
OTP, CVV, and MFA
Fraud Detection (AI)
Biometric Verification
Card Freeze & Block
(Input: OTP, PIN, Biometrics)
(Output: Success/Block)
Notifications, Alerts, & Messaging System
Fraud Alerts
Email Balance/EMI Alerts
Spend Category Alerts
Secure Messaging for Card Events
SMS alert
User Input/Output
User Inputs PIN, OTP, CVV
User Inputs Biometric Auth
- Receives Alerts (Email/SMS)
Dynamic Limit Management
Adjust Limits Based on Behaviour & Credit Score
Set Custom Alerts for User Transactions
Overdue Interest Calculations
Rewards & Insights Management
Track Reward Points
Redeem Points for Cashback/Offers
Spend Analysis (Category-based)
Insights for Spend Behavior
Report Generation for Spending

## Source code:

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>

#define OTP_BLOCK_TIME 86400   // 24 hours in seconds
#define CVV_BLOCK_TIME 86400   // 24 hours in seconds

// Structure for Credit Card details
struct CreditCard {
    char cardNumber[17];  // Credit card number (16 digits)
    char cardholderName[50];
    char expiryDate[6];   // Format MM/YY
    int cvv;              // CVV for online transactions
    float creditLimit;    // Custom allowed credit limit
    float moneySpent;     // Total money spent so far
    float currentBalance;  // Current available balance after spending
(credit limit - money spent)
    int pin;              // PIN for accessing card
    bool isBlocked;
    int failedOTPAttempts;
    int failedCVVAttempts;
    time_t blockStartTime;
    float rewardPoints;    // Reward points for transactions
};

// Function prototypes
void registerCard(struct CreditCard *cc);
bool validatePin(const struct CreditCard *cc);
void displayCardDetails(const struct CreditCard *cc);
void makeTransaction(struct CreditCard *cc, float amount, bool
isOnline, const char *category);
void makePayment(struct CreditCard *cc, float amount);
void checkBalance(const struct CreditCard *cc);
void sendEmail(const struct CreditCard *cc, const char *message);
void sendSMS(const struct CreditCard *cc, const char *message);
int generateOTP();
bool validateCVV(struct CreditCard *cc, int inputCVV);
void blockCard(struct CreditCard *cc);
bool isCardBlocked(struct CreditCard *cc);
void exitIfBlocked(struct CreditCard *cc);

// New feature function prototypes
void sendSpendCategoryAlert(const char *category, float amount);
void adjustCreditLimit(struct CreditCard *cc);
void trackRewardPoints(struct CreditCard *cc, float amount);
void showRewardsAndInsights(const struct CreditCard *cc);

int main() {
    struct CreditCard cc;
    int choice;
    float amount;
    char category[50];
    int attempts = 0;

    // Register a new card
    registerCard(&cc);

    // PIN validation loop (3 attempts)
    while (!validatePin(&cc)) {
        attempts++;
        if (attempts == 3) {
            printf("Too many wrong PIN attempts! Access blocked.\n");
            return 0; // Exit program
        }
    }

    // Menu loop
    do {
        exitIfBlocked(&cc);  // Check if card is blocked before every
operation
```

```
        printf("\n--- Credit Card Management System ---\n");
        printf("1. Display Card Details\n");
        printf("2. Make an Online Transaction\n");
        printf("3. Make a Payment (EMI)\n");
        printf("4. Check Balance\n");
        printf("5. Send Balance Sheet via Email\n");
        printf("6. Show Rewards & Insights\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                displayCardDetails(&cc);
                break;
            case 2:
                printf("Enter transaction amount: ");
                scanf("%f", &amount);
                printf("Enter transaction category (e.g., Shopping, Grocery, 
Entertainment): ");
                scanf("%s", category);
                makeTransaction(&cc, amount, true, category);  // Online 
transaction with category
                break;
            case 3:
                printf("Enter payment amount (EMI): ");
                scanf("%f", &amount);
                makePayment(&cc, amount);
                break;
            case 4:
                checkBalance(&cc);
                break;
            case 5:
                sendEmail(&cc, "Your updated balance sheet.");
                break;
            case 6:
                showRewardsAndInsights(&cc);
                break;
            case 7:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 7);

    return 0;
}

// Function to register a new credit card
void registerCard(struct CreditCard *cc) {
    printf("Enter credit card number (16 digits): ");
    scanf("%s", cc->cardNumber);

    printf("Enter cardholder name: ");
    scanf(" %[^\n]s", cc->cardholderName);

    printf("Enter expiry date (MM/YY): ");
    scanf("%s", cc->expiryDate);

    printf("Enter CVV (3 digits): ");
    scanf("%d", &cc->cvv);

    printf("Enter credit limit: ");
    scanf("%f", &cc->creditLimit);

    printf("Set a 4-digit PIN: ");
    scanf("%d", &cc->pin);

    cc->currentBalance = cc->creditLimit;     // Initialize current balance
to credit limit
    cc->moneySpent = 0;                      // Initialize money spent to 0
    cc->isBlocked = false;                  // Card initially unblocked
    cc->failedOTPAttempts = 0;              // Initialize failed OTP attempts
to zero
```

```
    cc->failedCVVAttempts = 0;              // Initialize failed CVV
attempts to zero
    cc->rewardPoints = 0;                   // Initialize reward points to zero

    printf("Card registered successfully with a credit limit of %.2f!\n", cc-
>creditLimit);
}

// Function to validate the card's PIN
bool validatePin(const struct CreditCard *cc) {
    int inputPin;
    printf("Enter your 4-digit PIN: ");
    scanf("%d", &inputPin);

    if (inputPin == cc->pin) {
        printf("PIN validated successfully!\n");
        return true;
    } else {
        printf("Incorrect PIN! Please try again.\n");
        return false;
    }
}

// Function to display card details
void displayCardDetails(const struct CreditCard *cc) {
    printf("\n--- Credit Card Details ---\n");
    printf("Card Number: %s\n", cc->cardNumber);
    printf("Cardholder Name: %s\n", cc->cardholderName);
    printf("Expiry Date: %s\n", cc->expiryDate);
    printf("Credit Limit: %.2f\n", cc->creditLimit);
    printf("Current Balance (Remaining Credit): %.2f\n", cc->creditLimit
- cc->moneySpent);
    printf("Reward Points: %.2f\n", cc->rewardPoints);
}

// Function to make an online transaction
void makeTransaction(struct CreditCard *cc, float amount, bool
isOnline, const char *category) {
    exitIfBlocked(cc);  // Check if card is blocked before making a
transaction

    if ((cc->creditLimit - cc->moneySpent) - amount < 0) {
        printf("Transaction declined! Exceeds available balance.\n");
        return;
    }

    if (isOnline) {
        int otp, cvv;
        printf("Enter CVV: ");
        scanf("%d", &cvv);

        // Validate CVV first
        if (!validateCVV(cc, cvv)) {
            printf("Transaction declined due to incorrect CVV.\n");
            return;  // Exit if CVV is incorrect
        }

        printf("CVV validated successfully. Now validating OTP...\n");

        // Generate and validate OTP
        int generatedOTP = generateOTP();
        printf("Generated OTP: %d (Simulated, for demonstration
purposes)\n", generatedOTP);

        // Allow 3 attempts for OTP within the same transaction
        for (int attempt = 0; attempt < 3; attempt++) {
            printf("Enter OTP: ");
            scanf("%d", &otp);

            if (otp == generatedOTP) {
                printf("OTP validated successfully. Proceeding with
transaction...\n");
                break;  // Exit loop if OTP is correct
            } else {
                printf("Incorrect OTP! You have %d attempts left.\n", 2 -
attempt);
```

```
            if (attempt == 2) {
                printf("Transaction declined due to incorrect OTP.\n");
                cc->failedOTPAttempts++;
                if (cc->failedOTPAttempts == 3) {
                    printf("Too many incorrect OTP attempts. Card blocked
for 24 hours!\n");
                    blockCard(cc);
                    exit(0);  // Exit the program after the card is blocked
                }
                return;  // Exit if OTP is incorrect after 3 attempts
            }
        }
    }
}

// Transaction proceeds if both CVV and OTP are correct
cc->moneySpent += amount;
cc->currentBalance = cc->creditLimit - cc->moneySpent;
printf("Transaction of %.2f in %s category successful. New remaining
balance: %.2f\n", amount, category, cc->currentBalance);

// Track reward points and insights for the transaction
trackRewardPoints(cc, amount);

// Send Spend Category Alerts
sendSpendCategoryAlert(category, amount);

// Adjust credit limit dynamically based on behavior (simulated)
adjustCreditLimit(cc);

// Send SMS with the new balance
char message[100];
sprintf(message, "Transaction successful! Your remaining balance is
%.2f.", cc->currentBalance);
sendSMS(cc, message);  // Send SMS with the updated balance

// Send an email with the updated balance sheet
sendEmail(cc, "Your updated balance sheet after transaction.");
```

```
}

// Function to make a payment (EMI)
void makePayment(struct CreditCard *cc, float amount) {
   exitIfBlocked(cc);  // Check if card is blocked before making a
payment

   if (amount > cc->moneySpent) {
      printf("Payment exceeds the amount spent. Transaction
declined!\n");
      return;
   }

   cc->moneySpent -= amount;  // Deduct the payment from money
spent
   cc->currentBalance = cc->creditLimit - cc->moneySpent;  // Update
current balance
   printf("Payment of %.2f successful! New remaining balance: %.2f\n",
amount, cc->currentBalance);
}

// Function to check current balance
void checkBalance(const struct CreditCard *cc) {
   printf("Current available balance: %.2f\n", cc->creditLimit - cc-
>moneySpent);
}

// Function to send balance sheet via email
void sendEmail(const struct CreditCard *cc, const char *message) {
   printf("Sending email to %s with message: %s\n", cc-
>cardholderName, message);
}

// Function to send SMS notification
void sendSMS(const struct CreditCard *cc, const char *message) {
   printf("Sending SMS to cardholder with message: %s\n", message);
}
```

```
// Function to generate a random OTP
int generateOTP() {
   srand(time(0));
   return 100000 + rand() % 900000;  // Generate a 6-digit OTP
}

// Function to validate CVV
bool validateCVV(struct CreditCard *cc, int inputCVV) {
   if (inputCVV == cc->cvv) {
      return true;
   } else {
      cc->failedCVVAttempts++;
      if (cc->failedCVVAttempts == 3) {
         printf("Too many incorrect CVV attempts. Card blocked for 24
hours!\n");
         blockCard(cc);
         exit(0);  // Exit program after the card is blocked
      }
      return false;
   }
}

// Function to block the card
void blockCard(struct CreditCard *cc) {
   cc->isBlocked = true;
   cc->blockStartTime = time(NULL);  // Record the block time
}

// Function to check if the card is blocked
bool isCardBlocked(struct CreditCard *cc) {
   if (cc->isBlocked) {
      time_t currentTime = time(NULL);
      if (difftime(currentTime, cc->blockStartTime) >=
OTP_BLOCK_TIME) {
         cc->isBlocked = false;  // Unblock the card after 24 hours
         cc->failedOTPAttempts = 0; // Reset failed attempts
         cc->failedCVVAttempts = 0; // Reset failed attempts
         printf("Card is now unblocked.\n");
```

```
        }
    }
    return cc->isBlocked;
}
// Function to check if the card is blocked before each transaction
void exitIfBlocked(struct CreditCard *cc) {
    if (isCardBlocked(cc)) {
        printf("Card is currently blocked. Please try again later.\n");
        exit(0);  // Exit the program if the card is blocked
    }
}
// New feature: Send spend category alerts
void sendSpendCategoryAlert(const char *category, float amount) {
    printf("Alert: You spent %.2f in the %s category.\n", amount,
category);
}
// New feature: Adjust credit limit dynamically (simulated)
void adjustCreditLimit(struct CreditCard *cc) {
    if (cc->moneySpent > cc->creditLimit * 0.8) {
        cc->creditLimit *= 0.9; // Decrease limit if over 80% spent
        printf("Credit limit adjusted down to %.2f due to high spending.\n",
cc->creditLimit);
    }
}

// New feature: Track reward points
void trackRewardPoints(struct CreditCard *cc, float amount) {
    float pointsEarned = amount * 0.01; // Earn 1 point for every 100
spent
    cc->rewardPoints += pointsEarned;
    printf("You earned %.2f reward points!\n", pointsEarned);
}
// New feature: Show rewards and insights
void showRewardsAndInsights(const struct CreditCard *cc) {
    printf("Total reward points: %.2f\n", cc->rewardPoints);
    printf("Total spent: %.2f\n", cc->moneySpent);
}
```

# Main C programming Concepts Used:

**Structures (struct):**
Defines a custom data type, CreditCard, to store card details like card number, PIN, CVV, and balance information in one place.

**Pointers:**
Used to pass the CreditCard structure by reference, allowing functions to modify the data within the structure directly.

**Standard Libraries:**
stdio.h: Manages input/output functions (printf, scanf) for interacting with the user.
string.h: Provides functions for string manipulation (e.g., strcpy, strcmp).
stdbool.h: Enables the use of Boolean data types (true and false).
time.h: Handles time functions for OTP and CVV blocking duration and timestamps.

**Control Structures:**
if, else, switch, for, and while loops control the program flow and user choices within the menu system.

**Functions:**
Used to break down functionality into manageable parts like registerCard(), makeTransaction(), and blockCard(), each with a specific responsibility for modular and readable code.

**Random Number Generation:**
rand() (from stdlib.h) generates a random OTP for secure online transactions, simulating real-world OTP functionality.

**Macros (#define):**
#define creates constants for block times to make code more readable and allow easy updates if the blocking time changes.

## Conclusion:

To wrap up, the credit card management system has come a long way hitting important targets while tackling problems head-on. The system aims to make things better for users by offering safe payment processing strong account management tools, and new features like tracking rewards and sending spending alerts.

From start to finish, we've stuck to the best ways of writing code and designing systems. This means our end product not does what we set out to do, but also matches what the industry expects in terms of safety and ease of use. The feedback we got from early tests has been helpful giving us ideas on how to make things even better.
As we go ahead, we'll keep our eyes on thorough testing and quality checks to make sure the system works well and doesn't let us down. We think the project will not do what users need but also change with future needs making it a solution that can grow in the world of financial tech.