// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract LoanAgreementContract {

address public lender;

// Constructor

constructor() {

lender = msg.sender; // Deployer is the lender

}

// Modifiers

modifier onlyLender() {

require(msg.sender == lender, "Only lender can call this function");

\_;

}

// Function: Change lender

function changeLender(address \_address) external onlyLender {

lender = \_address;

}

// Struct for Common and Fixed Interest Rate Details

struct CommonAndInterestDetails {

string emiPaymentStartDate; // e.g., January 1, 2025

uint256 principalAmount; // e.g., $250,000

uint256 fixedInterestRate; // e.g., 3.5% stored as basis points (350 = 3.5%)

uint256 monthlyPaymentAmount; // e.g., $1,200

string maturityDate; // e.g., December 1, 2055

uint8 emiPaymentDay; // e.g., 1 (1st day of the month)

string interestRateChangeDate; // e.g., June 1, 2026

string lenderName; // e.g., ABC Bank

string borrowerName; // e.g., Tony

}

// Struct for EMI Payment and Late Payment Details

struct PaymentAndLateDetails {

uint8 latePaymentGracePeriod; // Number of calendar days, e.g., 15

uint256 lateChargePercentage; // e.g., 5% stored as basis points (500 = 5%)

uint256 margin; // e.g., 2.0% stored as basis points (200 = 2.0%)

uint256 currentIndex; // e.g., 3.0% stored as basis points (300 = 3.0%)

uint256 maxInterestRateAtFirstChange; // e.g., 6.0% stored as basis points (600 = 6.0%)

uint256 minInterestRateAtFirstChange; // e.g., 2.0% stored as basis points (200 = 2.0%)

uint256 maxInterestRateAfterChange; // e.g., 6.0% stored as basis points (600 = 6.0%)

uint256 minInterestRateAfterChange; // e.g., 2.0% stored as basis points (200 = 2.0%)

}

// Struct for Adjustable Interest Rate Details

struct AdjustableInterestRateDetails {

string noteDate; // e.g., January 8, 2025

string city; // e.g., New York

string state; // e.g., NY

string propertyAddress; // e.g., 123 Main St, New York, NY

string paymentLocation; // e.g., 123 Bank St., New York, NY

}

// Mappings to store each type of data

mapping(string => CommonAndInterestDetails) public commonAndInterestDetailsMap;

mapping(string => PaymentAndLateDetails) public paymentAndLateDetailsMap;

mapping(string => AdjustableInterestRateDetails) public adjustableInterestRateDetailsMap;

// Array to store all loan IDs

string[] private loanIds;

// Events for tracking changes

event LoanCreated(string loanId, string propertyAddress);

event LoanUpdated(string loanId, string propertyAddress);

event LoanDeleted(string loanId);

event ContractFunded(address lender, uint256 amount);

// Function to set Common and Fixed Interest Rate Details

function setCommonAndInterestDetails(

string memory loanId,

string memory \_emiPaymentStartDate,

uint256 \_principalAmount,

uint256 \_fixedInterestRate,

uint256 \_monthlyPaymentAmount,

string memory \_maturityDate,

uint8 \_emiPaymentDay,

string memory \_interestRateChangeDate,

string memory \_lenderName,

string memory \_borrowerName

) public {

// Add loan ID to the array only if it's new

if (bytes(commonAndInterestDetailsMap[loanId].borrowerName).length == 0) {

loanIds.push(loanId);

}

commonAndInterestDetailsMap[loanId] = CommonAndInterestDetails(

\_emiPaymentStartDate,

\_principalAmount,

\_fixedInterestRate,

\_monthlyPaymentAmount,

\_maturityDate,

\_emiPaymentDay,

\_interestRateChangeDate,

\_lenderName,

\_borrowerName

);

emit LoanCreated(loanId, commonAndInterestDetailsMap[loanId].borrowerName);

}

// Function to set EMI Payment and Late Payment Details

function setPaymentAndLateDetails(

string memory loanId,

uint8 \_latePaymentGracePeriod,

uint256 \_lateChargePercentage,

uint256 \_margin,

uint256 \_currentIndex,

uint256 \_maxInterestRateAtFirstChange,

uint256 \_minInterestRateAtFirstChange,

uint256 \_maxInterestRateAfterChange,

uint256 \_minInterestRateAfterChange

) public {

paymentAndLateDetailsMap[loanId] = PaymentAndLateDetails(

\_latePaymentGracePeriod,

\_lateChargePercentage,

\_margin,

\_currentIndex,

\_maxInterestRateAtFirstChange,

\_minInterestRateAtFirstChange,

\_maxInterestRateAfterChange,

\_minInterestRateAfterChange

);

emit LoanUpdated(loanId, commonAndInterestDetailsMap[loanId].borrowerName);

}

// Function to set Adjustable Interest Rate Details

function setAdjustableInterestRateDetails(

string memory loanId,

string memory \_noteDate,

string memory \_city,

string memory \_state,

string memory \_propertyAddress,

string memory \_paymentLocation

) public {

adjustableInterestRateDetailsMap[loanId] = AdjustableInterestRateDetails(

\_noteDate,

\_city,

\_state,

\_propertyAddress,

\_paymentLocation

);

emit LoanUpdated(loanId, adjustableInterestRateDetailsMap[loanId].propertyAddress);

}

// Function to retrieve all details for a loanId

function getLoanDetails(string memory loanId)

public

view

returns (

CommonAndInterestDetails memory,

PaymentAndLateDetails memory,

AdjustableInterestRateDetails memory

)

{

return (

commonAndInterestDetailsMap[loanId],

paymentAndLateDetailsMap[loanId],

adjustableInterestRateDetailsMap[loanId]

);

}

// Function to delete all data for a specific loanId

function deleteLoan(string memory loanId) public {

// Delete data from all mappings

delete commonAndInterestDetailsMap[loanId];

delete paymentAndLateDetailsMap[loanId];

delete adjustableInterestRateDetailsMap[loanId];

// Remove loan ID from the loanIds array

for (uint256 i = 0; i < loanIds.length; i++) {

if (keccak256(bytes(loanIds[i])) == keccak256(bytes(loanId))) {

loanIds[i] = loanIds[loanIds.length - 1];

loanIds.pop();

break;

}

}

// Emit an event for deletion

emit LoanDeleted(loanId);

}

// Function to get all loan IDs

function getAllLoanIds() public view returns (string[] memory) {

return loanIds;

}

// Function: Fund the contract

function fundContract() external payable {

emit ContractFunded(lender, msg.value);

}

// Function: Take out the funds from contract

function takeOutContractFunds(uint256 \_amount) public onlyLender {

require(address(this).balance > 0, "No funds are available");

if (\_amount > 0) {

payable(lender).transfer(\_amount);

} else {

payable(lender).transfer(address(this).balance);

}

}

// Function: Check the balance of the lender

function getLenderBalance() public view onlyLender returns (uint256) {

return lender.balance;

}

// Function: Check the contract balance

function checkBalanceOfSmartContract() public view returns (uint256) {

return address(this).balance;

}

// Function: Check the balance of a specific borrower

function getBorrowerBalance(address \_borrower)

public

view

returns (uint256)

{

return \_borrower.balance;

}

}