CODTECH INTERNSHIP - TASK 3

# Decentralized Finance (DeFi) Application

Name: Vaahini D

Intern ID: CT04DG1604

## Overview

This document contains a sample DeFi smart contract written in Solidity for implementing a basic lending and borrowing mechanism with dynamic interest rate calculation.

## Smart Contract Code

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";  
  
contract SimpleLending {  
 IERC20 public token;  
 address public owner;  
 uint256 public interestRate = 5; // 5% annual interest (for demo)  
  
 struct Loan {  
 uint256 amount;  
 uint256 timestamp;  
 }  
  
 mapping(address => uint256) public deposits;  
 mapping(address => Loan) public loans;  
  
 constructor(IERC20 \_token) {  
 token = \_token;  
 owner = msg.sender;  
 }  
  
 function deposit(uint256 amount) public {  
 require(amount > 0, "Amount must be > 0");  
 token.transferFrom(msg.sender, address(this), amount);  
 deposits[msg.sender] += amount;  
 }  
  
 function borrow(uint256 amount) public {  
 require(amount > 0, "Amount must be > 0");  
 require(token.balanceOf(address(this)) >= amount, "Not enough liquidity");  
 loans[msg.sender] = Loan(amount, block.timestamp);  
 token.transfer(msg.sender, amount);  
 }  
  
 function repay() public {  
 Loan memory loan = loans[msg.sender];  
 require(loan.amount > 0, "No loan found");  
  
 uint256 interest = calculateInterest(loan.amount, loan.timestamp);  
 uint256 totalRepayment = loan.amount + interest;  
  
 token.transferFrom(msg.sender, address(this), totalRepayment);  
 delete loans[msg.sender];  
 }  
  
 function calculateInterest(uint256 amount, uint256 startTime) public view returns (uint256) {  
 uint256 timeElapsed = block.timestamp - startTime;  
 uint256 interest = (amount \* interestRate \* timeElapsed) / (365 days \* 100);  
 return interest;  
 }  
  
 function withdraw(uint256 amount) public {  
 require(deposits[msg.sender] >= amount, "Not enough deposit");  
 deposits[msg.sender] -= amount;  
 token.transfer(msg.sender, amount);  
 }  
}

## Expected Output & Testing

- After deployment, users can deposit tokens into the contract.

- Users can borrow tokens if enough liquidity is available.

- Interest is calculated based on the time elapsed since borrowing.

- Upon repayment, total tokens (including interest) are returned.

- Lenders can withdraw their funds anytime if not borrowed.