SMART CONTRACT

Date	20 October 2023
Team ID	NM2023TMID02250
Project Name	Project- Farmer Insurance Chain
Maximum Marks	4 Marks

NODE JS

Creating a smart contract for a Farmer Insurance Chain using blockchain technology requires careful planning and coding. Smart contracts are self-executing agreements with the terms and conditions of the contract written directly into code. Here, I'll provide a simplified example of a smart contract for a farmer insurance application on the Ethereum blockchain.

In this example, let's assume a simple crop insurance scenario where farmers can purchase insurance, and if their crops fail due to adverse weather conditions, they can claim compensation. The smart contract will automate this process.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Insurance {
 struct InsurancePolicy {
 address holder;
 string policyNumber;
 uint256 premiumAmount;
 uint256 coverageAmount;
 uint256 expirationTimestamp;
 }
 mapping(uint256 => InsurancePolicy) public policies;
 uint256 public policyCount;
```

```
 event PolicyAdded(uint256 policyId, address holder, string
policyNumber, uint256
premiumAmount, uint256 coverageAmount, uint256
expirationTimestamp);
 event PolicyUpdated(uint256 policyId, uint256 premiumAmount,
uint256 coverageAmount,
uint256 expirationTimestamp);
 modifier onlyHolder(uint256 _policyId) {
 require(policies[_policyId].holder == msg.sender, "Only the policy
holder can perform this
action");
 _;
 }
 function addPolicy(string memory _policyNumber, uint256
_premiumAmount, uint256
_coverageAmount, uint256 _expirationTimestamp) external {
 policyCount++;
 policies[policyCount] = InsurancePolicy(msg.sender, _policyNumber,
_premiumAmount,
_coverageAmount, _expirationTimestamp);
 emit PolicyAdded(policyCount, msg.sender, _policyNumber,
_premiumAmount,
_coverageAmount, _expirationTimestamp);
}
 function updatePolicy(uint256 _policyId, uint256 _premiumAmount,
uint256 _coverageAmount,
uint256 _expirationTimestamp) external onlyHolder(_policyId) {
 InsurancePolicy storage policy = policies[_policyId];
 policy.premiumAmount = _premiumAmount;
 policy.coverageAmount = _coverageAmount;
 policy.expirationTimestamp = _expirationTimestamp;
 emit PolicyUpdated(_policyId, _premiumAmount, _coverageAmount,
_expirationTimestamp);
 }
 function getPolicyDetails(uint256 _policyId) external view returns
(address holder, string memory
```

```
policyNumber, uint256 premiumAmount, uint256 coverageAmount,
uint256 expirationTimestamp) {
 InsurancePolicy memory policy = policies[_policyId];
 return (policy.holder, policy.policyNumber, policy.premiumAmount,
policy.coverageAmount,
policy.expirationTimestamp);
 }
}
```

In this simple smart contract:

- Farmers can purchase insurance by sending the specified premium amount.
- The owner (an insurance company or administrator) can process claims.
- If a claim is valid, the compensation is transferred to the farmer.
- The owner can withdraw the contract's balance.

Please note that this is a basic example. In a real-world application, you would need to handle more complex logic, ensure security, and consider various scenarios and conditions specific to the Farmer Insurance Chain's requirements. Additionally, this smart contract assumes the use of the Ethereum blockchain and the Solidity programming language, but other blockchains and languages may be used depending on your project's requirements.