Chapter 8

How to Write Smart Contracts with Solidity

Episode 8.01

Designing Your Supply Chain App

Supply Chain Review

- Supply chain app
  - Framework that connects producers to consumers
  - Manages products and services along that journey
- Blockchain advantages
  - Reduce costs
  - Offer transparency

Supply Chain Design – Process Needs

- Assets
  - The product to be bought by the consumer
- Participants
  - All supply chain participants
  - Manufacturers, suppliers, shippers, consumers

Supply Chain Design – Process Needs

- Ownership structure
  - Which participant currently owns the product
  - Tracks the product
- Payment token
  - Participants pay each other with tokens as ownership changes

Supply Chain Design - Capabilities

- Initialize tokens
  - Establish initial pool of payment tokens
- Transfer tokens
  - Move tokens between accounts as payment
- Authorize token payments
  - Allow token transfers on behalf on another

Supply Chain Design - Capabilities

- Add and update participants
- Move products along the supply chain
  - Transfer product ownership
- Add and update products
- Track an asset
  - Where a product is today
  - Find product provenance (ownership)

Episode 8.02

What are dAppps?

Developing dApps

- Advantages to decentralized apps (dApps)
  - Automatic history tracking
  - Built-in fault tolerance
  - Trusted data

Developing dApps

- Before developing
  - Know what your dApp does
  - Know your goals and how you plan to achieve them
  - Understand why the Ethereum blockchain environment is best

Developing dApps

- Allow users to access data stored on the blockchain
  - Unlike “normal” applications, you can’t bypass integrity protections

Developing dApps

- Understand cost of interacting with the blockchain
- Smart contracts provide the interface between users and the blockchain
  - Making dApps possible

Episode 8.03

Token Smart Contract Details

Payment Token Smart Contract Data Items

- `totSupply`
  - Total number of tokens in circulation
- `name`
  - Descriptive token name
- `decimals`
  - Number of decimals to use when displaying token amounts

- `symbol`
  - Short identifier
- `balances`
  - Current balance of each participating account, mapped to the account's address
- `allowed`
  - Number of tokens authorized to transfer between accounts, mapped to sender's address

Payment Token Smart Contract Functions

- `totalSupply()`
  - Returns current total number of tokens
- `balanceOf()`
  - Returns current balance of specified account
- `allowance()`
  - Returns remaining number of tokens allowed to be transferred from one specific account to another specific account

Payment Token Smart Contract Functions

- `transfer()`
  - Transfers tokens from the owner to specified target account
- `transferFrom()`
  - Transfers tokens from one specific account to another specific account
- `approve()`
  - Maximum allowed tokens that can be transferred from one specific account to another specific account

Episode 8.04

Supply Chain Smart Contract Details

Supply Chain Smart Contract

- Data and functionality to manage products, participants, ownership transfer data
- Everything else except payment

Supply Chain Smart Contract Data Structures

- Product structure
  - Ex: model number, part number, cost, etc.
  - Data that defines a unique product
- Participant structure
  - Data that defines a unique participant
  - Ex: username, password, Ethereum address, etc.

Supply Chain Smart Contract Data Structures

- Ownership structure
  - Data that records product ownership transfer information
  - Ex: product ID, owner ID, transaction time, etc.

Supply Chain Smart Contract Data Variables

- `product_id`
  - Unique product ID, mapped to product structure
- `participant_id`
  - Unique participant ID, mapped to participant structure
- `owner_id`
  - Unique owner, mapped to registration structure

Supply Chain Smart Contract Functions

- `addParticipant()`
  - Create new participant
- `getParticipant()`
  - Fetch information about a participant
- `addProduct()`
  - Create new product
- `getProduct()`
  - Fetch information about a particular product

Supply Chain Smart Contract Functions

- `newOwner()`
  - Transfer of ownership
- `getProvenance()`
  - Record of ownership
- `getOwnership()`
  - Owner of a product in a specific point in time
- `authenticateParticipant()`
  - Confirms participant is allowed to access certain data

Episode 8.05

Smart Contract Road Map

• Your Smart Contract Road Map

	Payment Token Smart Contract	Supply Chain Smart Contract

Data Items	• totSupply
	• name
	• decimals
	• symbol
	• balances
	• allowed
		• Product structure
		    ○ product_id
		• Participant structure
		    ○ participant_id
		• Ownership structure
		    ○ owner_id

Functions	• totalSupply() • balanceOf() • allowance() • transfer() • transferFrom() • approve()	• addParticipant() • getParticipant() • addProduct() • getProduct() • newOwner() • getProvenance() • getOwnership() • authenticateParticipant()

Episode 8.06

Token Smart Contract Data Lab, Part 1

• ERC-20 Token Interface Code

```
// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------------
pragma solidity >=0.4.21 <0.6.0;

contract ERC20Interface {
    uint256 public totSupply;

    function totalSupply() public view returns (uint);
    function balanceOf(address tokenOwner) public view returns (uint balance);
    function allowance(address tokenOwner, address spender) public view returns (uint
remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool
success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Episode 8.07

Token Smart Contract Data Lab, Part 2

Interface

- An agreement
- In order to base your smart contract on an interface, you must follow standards
- Minimum requirements to fit the standard

- ERC-20 Token Code, Part 1

```
// ----------------------------------------------------------------------------
///Implements EIP20 token standard: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-
20.md
// ----------------------------------------------------------------------------

pragma solidity >=0.4.21 <0.6.0;

import "./erc20Interface.sol";

contract ERC20Token is ERC20Interface {

    uint256 constant private MAX_UINT256 = 2**256 - 1;
    mapping (address => uint256) public balances;
    mapping (address => mapping (address => uint256)) public allowed;

    uint256 public totSupply;             // Total number of tokens
    string public name;                   // Descriptive name (i.e. For SupplyChainApp Token)
    uint8 public decimals;                // How many decimals to use when displaying amounts
    string public symbol;                 // Short identifier for token (i.e. FDT)
```

```
// Create the new token and assign initial values, including initial amount
   constructor(
       uint256 _initialAmount,
       string memory _tokenName,
       uint8 _decimalUnits,
       string memory _tokenSymbol
   ) public {
       balances[msg.sender] = _initialAmount;         // The creator owns all initial tokens
       totSupply = _initialAmount;                    // Update total token supply
       name = _tokenName;                             // Store the token name (used for display only)
       decimals = _decimalUnits;                      // Store the number of decimals (used for display only)
       symbol = _tokenSymbol;                         // Store the token symbol (used for display only)
   }
```

```
// Transfer tokens from msg.sender to a specified address
   function transfer(address _to, uint256 _value) public returns (bool success) {
       require(balances[msg.sender] >= _value,"Insufficient funds for transfer source.");
       balances[msg.sender] -= _value;
       balances[_to] += _value;
       emit Transfer(msg.sender, _to, _value); //solhint-disable-line indent, no-unused-vars
       return true;
   }

// Transfer tokens from one specified address to another specified address
   function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {
       uint256 allowance = allowed[_from][msg.sender];
       require(balances[_from] >= _value && allowance >= _value,"Insufficient allowed funds for transfer source.");
       balances[_to] += _value;
       balances[_from] -= _value;
       if (allowance < MAX_UINT256) {
           allowed[_from][msg.sender] -= _value;
       }
       emit Transfer(_from, _to, _value); //solhint-disable-line indent, no-unused-vars
       return true;
   }
```

- ERC-20 Token Code, Part 4

```
 // Return the current balance (in tokens) of a specified address
    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }

    // Set
    function approve(address _spender, uint256 _value) public returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value); //solhint-disable-line indent, no-
unused-vars
        return true;
    }

 // Return the
    function allowance(address _owner, address _spender) public view returns (uint256
remaining) {
        return allowed[_owner][_spender];
    }

    // Return the total number of tokens in circulation
    function totalSupply() public view returns (uint256 totSupp) {
        return totSupply;
    }
}
```

Episode 8.08

Supply Chain Smart Contract Data Lab, Part 1

```
pragma solidity >=0.4.21 <0.6.0;

contract supplyChain {
    uint32 public product_id = 0;   // Product ID
    uint32 public participant_id = 0;   // Participant ID
    uint32 public owner_id = 0;   // Ownership ID

    struct product {
        string modelNumber;
        string partNumber;
        string serialNumber;
        address productOwner;
        uint32 cost;
        uint32 mfgTimeStamp;
    }
```

```
    mapping(uint32 => product) public products;

    struct participant {
        string userName;
        string password;
        string participantType;
        address participantAddress;
    }
    mapping(uint32 => participant) public participants;

    struct ownership {
        uint32 productId;
        uint32 ownerId;
        uint32 trxTimeStamp;
        address productOwner;
    }
    mapping(uint32 => ownership) public ownerships; // ownerships by ownership ID (owner_id)
    mapping(uint32 => uint32[]) public productTrack;  // ownerships by Product ID (product_id) / Movement track for a
product
```

```
    event TransferOwnership(uint32 productId);

    function addParticipant(string memory _name, string memory _pass, address _pAdd, string memory _pType) public returns
(uint32){
        uint32 userId = participant_id++;
        participants[userId].userName = _name;
        participants[userId].password = _pass;
        participants[userId].participantAddress = _pAdd;
        participants[userId].participantType = _pType;

        return userId;
    }
    function getParticipant(uint32 _participant_id) public view returns (string memory,address,string memory) {
        return (participants[_participant_id].userName,
                participants[_participant_id].participantAddress,
                participants[_participant_id].participantType);
    }
```

```
function addProduct(uint32 _ownerId,
                    string memory _modelNumber,
                    string memory _partNumber,
                    string memory _serialNumber,
                    uint32 _productCost) public returns (uint32) {
    if(keccak256(abi.encodePacked(participants[_ownerId].participantType)) == keccak256("Manufacturer")) {
        uint32 productId = product_id++;

        products[productId].modelNumber = _modelNumber;
        products[productId].partNumber = _partNumber;
        products[productId].serialNumber = _serialNumber;
        products[productId].cost = _productCost;
        products[productId].productOwner = participants[_ownerId].participantAddress;
        products[productId].mfgTimeStamp = uint32(now);

        return productId;
    }
   return 0;
}
```

```
    modifier onlyOwner(uint32 _productId) {
         require(msg.sender == products[_productId].productOwner,"");
         _;

    }

    function getProduct(uint32 _productId) public view returns (string memory,string memory,string
memory,uint32,address,uint32){
        return (products[_productId].modelNumber,
                products[_productId].partNumber,
                products[_productId].serialNumber,
                products[_productId].cost,
                products[_productId].productOwner,
                products[_productId].mfgTimeStamp);
    }

    function newOwner(uint32 _user1Id,uint32 _user2Id, uint32 _prodId) onlyOwner(_prodId) public returns (bool) {
        participant memory p1 = participants[_user1Id];
        participant memory p2 = participants[_user2Id];
        uint32 ownership_id = owner_id++;
```

```
        if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Manufacturer")
            && keccak256(abi.encodePacked(p2.participantType))==keccak256("Supplier")){
            ownerships[ownership_id].productId = _prodId;
            ownerships[ownership_id].productOwner = p2.participantAddress;
            ownerships[ownership_id].ownerId = _user2Id;
            ownerships[ownership_id].trxTimeStamp = uint32(now);
            products[_prodId].productOwner = p2.participantAddress;
            productTrack[_prodId].push(ownership_id);
            emit TransferOwnership(_prodId);

            return (true);
        }
        else if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Supplier") &&
keccak256(abi.encodePacked(p2.participantType))==keccak256("Supplier")){
            ownerships[ownership_id].productId = _prodId;
            ownerships[ownership_id].productOwner = p2.participantAddress;
            ownerships[ownership_id].ownerId = _user2Id;
            ownerships[ownership_id].trxTimeStamp = uint32(now);
            products[_prodId].productOwner = p2.participantAddress;
            productTrack[_prodId].push(ownership_id);
            emit TransferOwnership(_prodId);

            return (true);
        }
```

```
        else if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Supplier") &&
keccak256(abi.encodePacked(p2.participantType))==keccak256("Consumer")){
            ownerships[ownership_id].productId = _prodId;
            ownerships[ownership_id].productOwner = p2.participantAddress;
            ownerships[ownership_id].ownerId = _user2Id;
            ownerships[ownership_id].trxTimeStamp = uint32(now);
            products[_prodId].productOwner = p2.participantAddress;
            productTrack[_prodId].push(ownership_id);
            emit TransferOwnership(_prodId);

            return (true);
        }

        return (false);
    }

   function getProvenance(uint32 _prodId) external view returns (uint32[] memory) {

       return productTrack[_prodId];
    }
```

```
    function getOwnership(uint32 _regId)  public view returns (uint32,uint32,address,uint32) {

        ownership memory r = ownerships[_regId];

         return (r.productId,r.ownerId,r.productOwner,r.trxTimeStamp);
    }

    function authenticateParticipant(uint32 _uid,
                                    string memory _uname,
                                    string memory _pass,
                                    string memory _utype) public view returns (bool){
        if(keccak256(abi.encodePacked(participants[_uid].participantType)) == keccak256(abi.encodePacked(_utype))) {
            if(keccak256(abi.encodePacked(participants[_uid].userName)) == keccak256(abi.encodePacked(_uname))) {
                if(keccak256(abi.encodePacked(participants[_uid].password)) == keccak256(abi.encodePacked(_pass))) {
                    return (true);
                }
            }
        }

        return (false);
    }
}
```

Episode 8.09

Supply Chain Smart Contract Data Lab, Part 2

- Supply Chain Smart Contract Data Code
  - Code can be found in episode 8.08 Supply Chain Smart Contract Data Lab, Part 1
  - Code can also be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > SupplyChain.sol

Episode 8.10

Token Smart Contract Functions Lab, Part 1

- Supply Chain Smart Contract Data Code
  - Code can be found in episode 8.07 Token Smart Contract Data Lab, Part 1
  - Code can also be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > erc20Interface.sol

Episode 8.11

Token Smart Contract Functions Lab, Part 2

- Supply Chain Smart Contract Data Code
  - Code can be found in episode 8.07 Token Smart Contract Data Lab, Part 1
  - Code can also be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > erc20Interface.sol

Episode 8.12

Supply Chain Smart Contract Functions Lab, Part 1

- Supply Chain Smart Contract Data Code
  - Code can be found in episode 8.08 Supply Chain Smart Contract Data Lab, Part 1
  - Code can also be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > SupplyChain.sol

Episode 8.13

Supply Chain Smart Contract Functions Lab,
Part 2

- Supply Chain Smart Contract Data Code
  - Code can be found in episode 8.08 Supply Chain Smart Contract Data Lab, Part 1
  - Code can also be downloaded from accompanying files, sourceCode > supplyChainApp > contracts > SupplyChain.sol

Episode 8.14

Using Events

Smart Contract Issues

- Smart contracts run on each EVM
- Smart contracts are essentially server-side code
  - Smart contracts execute on blockchain nodes, not on the client
  - Issue: communication with client
    - Smart contracts must be called by the client or another smart contract

Server-Side Code

- Server-side code simply listens for requests from clients
- When a request is received, the server-side code responds
- All requests originate with the client

Implementing Events in Solidity

1. Define the event
   - Give it a name
   - Define what happens when the event occurs
     - What type of data is going to be sent with the event
2. Trigger the events
   - Smart contract code that detects when an event occurs
   - Use `emit` statement
3. Client receives and responds to the events

Episode 8.15

Implementing Events

Step 1: Define the Events

- ERC-20 token interface
  - Transfer event
  - Approval event
- Supply chain smart contract
  - Transfer of ownership

Step 2: Trigger the Events

- Use **`emit`** in Solidity to trigger an event
  - Like calling a function
    - Tell Solidity what event to trigger, provide parameter values

Episode 8.16

More on Ownership

Ownership

- Every action has an owner
- Blockchain apps can be “anonymous”
  - Hard to associate account with real-world identity
- Every time you invoke smart contract functions, owner address is associated
  - Nonrepudiation – every action can be attributed to an account

Ownership

- Might want to provide functionality with higher authority and more capability
- Owner is the address of whoever called the function
  - msg.sender is the owner
- Can restrict functionality (or ability to call a function) based on ownership

Modifiers

- Help restrict access for external entities
  - Add-on functionality to make sure things occur

Episode 8.17

Designing for Security

Smart Contract Security

- Smart contracts can be as insecure as any other software
- Must consider security in data and functionality of your design
- Use multi-level security approach
  - Defense in depth

Common Security Mistakes

- Randomness
  - Smart contract code must run the same on all EVMs
  - If random number is generated locally, could create different output on different EVMs
  - Avoid random numbers, especially for blockchain-related data

Common Security Mistakes

- Re-entrancy
  - Call function forwards all received gas to the called function
  - Can cause multiple withdrawals
    - Like allowing someone to use one check to withdraw money from your account multiple times
  - Update state data before transferring control to another function

Common Security Mistakes

- Overflow
  - Variable that has incremented larger than what is allowed for the stated variable
  - Ex: you defined a uint8 but the value was larger than 8 bits (255 characters)
- Underflow
  - Variable that has decremented to a value not allowed by stated variable
  - Ex: a value that tries to decrement to a negative number, but is an unsigned integer (uint)

Common Security Mistakes

- `delegate-call` function
  - Allows one smart contract to execute a function from another smart contract
  - Causes confusion with who invoked a function
  - Don't allow public/external functions to modify state data
  - Call a local function if public/external functions need to modify state data

Episode 8.18

Implementing Minimal Functionality

Minimal Functionality

- These examples are for instruction
- Only implements basic functionality
- Fully-featured smart contracts are complex
- Coding starts with minimal functionality, then begin building more
- If you create a live smart contract, add more functionality