CSIT 6000Q- Blockchain and Smart Contracts

Assigment #2
(Due: 11:59pm on Sun Day, Nov. 18, 2023)

October 26, 2023

1 Problem 1 20 points

Design a smart contract for Supply Chain Tracker using Solidity.

1.1 Problem statement

Implement a smart contract for tracking the supply chain of a product.

- The contract should allow participants to record each stage of the product's journey from producer to consumer.
- Only authorized participants can update the product's status.
- The contract should maintain a transparent and immutable history of the product's journey.

1.2 Example scenario

A company wants to create transparency in its supply chain by using a blockchain-based tracking system. The product goes through various stages, from production to distribution, and finally to the consumer. Different parties, including the manufacturer, distributor, and retailer, have permission to update the product's status. The contract keeps an immutable record of each stage of the product's journey, which can be accessed by anyone.

1.3 Contract interface

- constructor: `function SupplyChainTracker (address manufacturer, address distributor, address retailer).`
  - `manufacturer` is the address of the product manufacturer.
  - `distributor` is the address of the product distributor.
  - `retailer` is the address of the product retailer.
- updateStatus: `function updateStatus(string status)`
  - allows authorized participants to update the product's status.
- getHistory: `function getHistory() view returns (string[])`
  - retrieves the complete history of the product's journey as an array of status updates.

2 Problem 2 20 points

Design a smart contract for Token Exchange using Solidity.

2.1 Problem statement

Implement a decentralized token exchange that allows users to trade tokens.

- The contract should support multiple types of tokens (e.g., Ethereum and custom tokens).
- Users can deposit and withdraw tokens from the exchange.
- Users can place buy and sell orders for tokens.
- The contract should handle matching orders and executing trades.

2.2 Example scenario

A decentralized exchange platform aims to facilitate the trading of various tokens, including Ethereum and custom tokens created by different projects. Users can deposit tokens into the exchange, place buy and sell orders for specific tokens, and the contract should match these orders and execute trades according to market conditions.

2.3 Contract interface

- constructor: `function TokenExchange()`
  - creates a new TokenExchange contract.
  - deposit: `function deposit(address tokenAddress, uint256 amount)`
  - allows users to deposit tokens (Ether or custom tokens) into the exchange.
- withdraw: `function withdraw(address tokenAddress, uint256 amount)`
  - allows users to withdraw tokens from the exchange.
- placeOrder: `function placeOrder(address tokenAddress, uint256 amount, uint256 price, bool isBuyOrder)`
  - allows users to place buy or sell orders for tokens at a specified price.
- matchOrders: `function matchOrders(address tokenAddress)`
  - matches buy and sell orders for a specific token and executes trades.
- getOrderBook: `function getOrderBook(address tokenAddress) view returns (uint256[], uint256[])`
  - retrieves the buy and sell order books for a specific token.

3 Problem 3 20 points

Design a smart contract for Decentralized Identity Verification using Solidity.

3.1 Problem statement

Implement a smart contract for decentralized identity verification.

- The contract should allow users to verify their identity by providing personal information.
- Once verified, users can generate a unique, verifiable digital identity.
- The contract should protect the privacy and security of users' personal information.

3.2 Example scenario

A decentralized identity verification system aims to provide individuals with a secure way to verify their identity. Users can submit their personal information, and the contract should verify this information securely. Once verified, the system generates a unique digital identity that users can use for various online services, such as accessing restricted websites or financial transactions. It's essential to ensure that users' personal information is handled with the utmost privacy and security.

3.3 Contract interface

- constructor: `function IdentityVerification()`
  - creates a new IdentityVerification contract.
- verifyIdentity: `function verifyIdentity(string firstName, string lastName, uint256 birthDate, string documentNumber)`
  - allows users to submit personal information for identity verification.
- generateDigitalIdentity: `function generateDigitalIdentity() view returns (bytes32)`
  - generates a unique, verifiable digital identity for a verified user.

4 Problem 4 20 points

Design a smart contract for Blockchain-Based Inventory Management using Solidity.

4.1 Problem statement

Implement a smart contract for blockchain-based inventory management in an e-commerce warehouse.

- The contract should allow warehouse staff to add and track products in the inventory.
- The contract should provide real-time visibility into stock levels and product details.
- The contract should log all product movements and changes in stock status.

4.2 Example scenario

An e-commerce company operates a large warehouse with a vast inventory of products. They want to implement a blockchain-based inventory management system to ensure efficient tracking of products, reduce errors, and enhance transparency. Warehouse staff can use this system to add products to the inventory, update product details, and monitor real-time stock levels. All product movements and changes in stock status are logged in the blockchain, providing a reliable record of inventory history.

4.3 Contract interface

- constructor: `function InventoryManagement()`
  - creates a new InventoryManagement contract.
- addProduct: `function addProduct(uint256 productId, string productName, uint256 quantity, uint256 price)`
  - allows warehouse staff to add a product to the inventory with a specified quantity and price.
- updateProduct: `function updateProduct(uint256 productId, string productName uint256 quantity, uint256 price)`
  - enables updates to product details and stock quantities.
- getInventory: `function getInventory() view returns (uint256[], string[], uint256[], uint256[])`
  - retrieves the current inventory, including product IDs, names, quantities, and prices.
- getInventoryHistory: `function getInventoryHistory(uint256 productId) view returns (string[])`
  - retrieves the history of a specific product's inventory changes.

5 Problem 5 20 points

Design a smart contract for Order Fulfillment Tracking using Solidity.

5.1 Problem statement

Implement a smart contract for order fulfillment tracking in an e-commerce warehouse.

- The contract should allow customers to track the status of their orders in real-time.
- Warehouse staff can update the status of orders as they progress through the fulfillment process.
- The contract should maintain a secure and immutable record of order statuses.

5.2 Example scenario

An e-commerce company wants to enhance its customer experience by providing real-time order tracking. Customers can monitor the status of their orders as they move through the warehouse fulfillment process, from order placement to shipping. Warehouse staff use the blockchain-based contract to update order statuses as they pack, ship, and deliver the items. This ensures transparency and trust in the order fulfillment process.

5.3 Contract interface

- constructor: `function OrderFulfillmentTracking()`
  - creates a new OrderFulfillmentTracking contract.
- placeOrder: `function placeOrder(uint256 orderId, string customerName, string shippingAddress)`
  - allows customers to place orders and initiate the fulfillment process.
- updateOrderStatus: `function updateOrderStatus(uint256 orderId, string status)`
  - enables warehouse staff to update the status of orders as they progress through fulfillment.
- trackOrder: `function trackOrder(uint256 orderId) view returns (string)`
  - allows customers to track the real-time status of their orders.