Code-Layout, Readability and Reusability

Date	06 November 2023
Team ID	NM2023TMID02201
Project Title	Project- Drug Traceability

Code-Layout:

```
 # Import necessary Ethereum and smart contract libraries
from web3 import Web3
from solcx import compile_standard
import json

# Define the contract source code (Solidity)
contract_source_code = """
pragma solidity ^0.8.0;

contract DrugTraceability {

    // Define data structures for drug traceability
    struct DrugProduct {
        uint productId;
        string productName;
        string batchNumber;
        string manufacturer;
        string owner;
        bool isVerified;
    }
}



    // Mapping to store drug products by their unique IDs
    mapping(uint => DrugProduct) public drugProducts;

    // Events to log key actions
    event ProductCreated(uint productId, string productName, string batchNumber,
string manufacturer);
    event ProductTransferred(uint productId, string fromOwner, string toOwner);
    event ProductVerified(uint productId);

    // Constructor to initialize the contract
    constructor() {
        // Initialize the contract
    }

    // Function to create a new drug product
    function createDrugProduct(uint _productId, string memory _productName, string
memory _batchNumber, string memory _manufacturer) public {
        // Check for permissions, authentication, and data validation
        // Create a new drug product and add it to the mapping
        // Emit ProductCreated event
    }

    // Function to transfer ownership of a drug product
    function transferOwnership(uint _productId, string memory _newOwner) public {
        // Check for permissions, authentication, and data validation
        // Update the owner of the drug product
        // Emit ProductTransferred event
    }

    // Function to verify the authenticity of a drug product
    function verifyProduct(uint _productId) public {
        // Check for permissions, authentication, and data validation
        // Mark the product as verified
        // Emit ProductVerified event
    }
}
"""

# Compile the Solidity contract
compiled_sol = compile_standard(
    {
        "language": "Solidity",
        "sources": {"DrugTraceability.sol": {"content": contract_source_code}},
        "settings": {
            "outputSelection": {
                "*": {
                    "*": ["abi", "evm.bytecode"],
                }
            }
        },
    }
)

# Extract the contract's ABI and bytecode
contract_interface =
compiled_sol["contracts"]["DrugTraceability.sol"]["DrugTraceability"]

# Define Ethereum network connection and contract deployment
w3 = Web3(Web3.HTTPProvider("http://localhost:8545"))  # Connect to a local
Ethereum node
```

```
# Create a contract instance
contract = w3.eth.contract(
    abi=contract_interface["abi"],
    bytecode=contract_interface["evm"]["bytecode"]["object"],
)

# Deploy the contract
transaction_hash = contract.constructor().transact()
transaction_receipt = w3.eth.waitForTransactionReceipt(transaction_hash)

# Interact with the deployed contract
contract_instance = w3.eth.contract(
    address=transaction_receipt.contractAddress,
    abi=contract_interface["abi"],
)
```

Readability and Reusability:

Readability and reusability are crucial aspects of code quality for a drug traceability blockchain system. Writing readable and reusable code ensures that your codebase is maintainable, understandable, and can be extended or repurposed for future needs. Here are some best practices to enhance readability and reusability in your blockchain code:

1. **Code Comments and Documentation:**

   - Provide clear and concise comments to explain the purpose of functions, variables, and complex logic.

   - Use well-structured documentation, such as Javadoc or Epytext, to describe the overall architecture, data structures, and important functions.

2. **Descriptive Variable and Function Names:**

   - Choose meaningful and descriptive names for variables and functions to make their purpose apparent.

   - Avoid cryptic or abbreviated names that may confuse other developers.

3. **Modular Code Structure:**

   - Organize your code into logical modules or files. Each module should have a specific responsibility, making it easier to locate and understand relevant code.

4. **Separation of Concerns:**

   - Implement a clear separation of concerns in your code. Keep blockchain-specific logic separate from application-specific logic.

   - Use design patterns, such as the Model-View-Controller (MVC) pattern, to separate concerns effectively.

5. **Code Formatting:**

   - Enforce a consistent code formatting style throughout the project. Tools like Prettier (for JavaScript) or Black (for Python) can help maintain a consistent style.

   - Use indentation, spacing, and line breaks to make the code more readable.

6. **Error Handling:**

   - Implement proper error handling throughout your code. Use meaningful error messages and handle exceptions gracefully to avoid unexpected crashes.

7. **Testing and Testable Code:**

   - Write unit tests to validate the functionality of individual components.

   - Develop testable code by minimizing dependencies, making it easier to write and maintain tests.

8. **Reuse Existing Libraries:**

   - Leverage existing blockchain libraries, smart contract templates, and open-source projects to avoid reinventing the wheel.

   - Follow the DRY (Don't Repeat Yourself) principle and encapsulate common functionality in reusable modules or libraries.

9. **Parameterization and Configurability:**

   - Make your code configurable by using parameters, configuration files, or environment variables. This flexibility allows you to adapt the code to different use cases without rewriting it.

10. **Design Patterns:**

    - Use design patterns like Singleton, Factory, or Strategy when they fit the problem domain. These patterns promote modular, reusable code.

11. **Code Reviews:**

    - Regularly conduct code reviews with team members to ensure that the codebase adheres to readability and reusability standards.

    - Address feedback and suggestions from code reviews to improve the codebase continuously.

12. **Version Control and Documentation:**

    - Use version control (e.g., Git) to track changes and maintain a detailed commit history.

    - Maintain a clear and up-to-date README file to provide instructions for setting up, configuring, and running the project.

13. **Code Style Guide:**

    - Develop and follow a code style guide for your project, specifying coding conventions and standards to ensure consistency.

14. **Refactoring:**

    - Periodically review and refactor the codebase to eliminate duplication, improve clarity, and enhance reusability.

15. **Consistent Naming Conventions:**

    - Follow consistent naming conventions for variables, functions, and files across the project.

16. **Testing Environment:**

    - Create a dedicated testing environment for running automated tests and integration testing to catch issues before deployment.

17. **Educate Team Members:**

    - Train team members and collaborators on coding standards, best practices, and the project's architecture to maintain a common understanding.