COS30049 - Computing Technology Innovation Project

Assignment 3 for Group Set 1
Project Report

Project Team: Group 1-3

Year: 2023

World Count: 3273

Table of Contents

1: Project information

A: Project name

The Foodie smart contract project is named FoodieChain. FoodieChain is a way for people who love food, cooks, places to eat, and farmers to connect with each other. FoodieChain wants to make a food system that is honest, fair, and good for the environment. It wants to give recognition and rewards to food that is tasty, unique, and helps people. FoodieChain lets people find new types of food, get food delivered, reserve tables at restaurants, give ratings and reviews of dishes, join cooking competitions, and help local food makers. FoodieChain uses a special kind of technology called blockchain to make sure that all the transactions and interactions on the platform are safe, can be traced, and the people involved can be held responsible. FoodieChain also has its own special token called FOODIE. You can use FOODIE to pay for things, unlock special features, and get rewards for helping out.

B: Project Description

FoodieChain is a technology that connects people who make food, deliver it, and eat it in a clear and helpful way. The project aims to bring food to people that is safe and has good quality. It also makes sure that we can see where the food comes from and not waste any of it, preventing from adding to pollution. The main jobs of the FoodieChain smart contract are:

1. Managing food transactions

2. Tracking the food supply chain

3. Ensuring transparent and secure transactions.

- Food producers can put their products on the blockchain with details like its producing source, ingredients, and its expiration.

- Checking food: Food sellers and buyers can check if the food is real and good by scanning the QR code or NFC tag on the package.

- Food token: FOODIE token is the special type of currency used on the FoodieChain platform to buy food, get extra features, and for other more functionalities.

C: Team information:

Member name	Responsibilities
Tran Khai Kiet	Write the project information 2, deal with the first prototype of smart contract and in charge of login and register page
Nguyen Manh Duc	Write the Inheritance Graph Discussion 5 (part a, b and c), in charge of the database and minting the NFTs
Tran Quoc Dung	Write the Executive Summary 2 and ABI Illustration 4, and the main pages like transaction, home, cart, etc.

Do Tuan Dat	Write the Call Graph Discussion 5 and Inheritance Graph Discussion 5 (part d and e), create the wallet page, along with developing smart contracts

D: Version Information:

1. The current version of the smart contract is 0.5.16 which has been bundled with Truffle
2. The FoodieChain smart contract is a computer program that lets people make, buy, and sell food on the Ethereum blockchain. The smart contract uses different tools to make sure it works well, is safe and can be used with other things. Some of the main libraries and frameworks that are used are:

    - OpenZeppelin Contracts: A set of safe and reusable pieces for Solidity smart contracts. The FoodieChain smart contract uses the ERC721, Ownable, and Pausable contracts from OpenZeppelin. This helps to create non-fungible tokens, manage ownership, and stop the contract in emergencies.
    - The FoodieChain uses Truffle to create, put out and test the smart contract code on different networks like Ganache, Rinkeby and Mainnet.
    - Web3js is a tool that helps web applications talk to Ethereum using JSON-RPC. The FoodieChain smart contract connects to the MetaMask browser extension using Web3. js This allows users to access their Ethereum accounts and approve transactions.
    - IPFS is a way for people to share and save files together on the internet. The FoodieChain smart contract uses IPFS to save information and pictures of the food items. Each food item has its own unique ID in the smart contract.

2. Executive Summary

A smart contract project is an approach to making and executing computerized contracts that are stored on a blockchain and run naturally when certain circumstances are met. The most common way of fostering a smart contract project incorporates these following steps:

- Define and analyze the terms and logic of the contract between the components that are involved.
- Using a programming language to translate the contract term into coding lines. In this particular project, we are using Solidity, which is designed for developing smart contracts on the Ethereum blockchain.
- Deploying the code to the private blockchain network after developing the full-stack website, where it is replicated and verified.
- Interact with the smart contract by sending transactions that trigger its functions and update its state. This is also a compulsory requirement in this project, since transactions are required to be displayed.

To develop smart contracts in this project, our team uses Truffle, a suite of tools for developing, testing, debugging, and deploying smart contracts on the Ethereum blockchain. It provides a command-line interface, a framework, a personal blockchain, a browser extension, and a VS Code extension to help developers create and manage their smart contract projects.

A portion of the significant discoveries of smart contract projects are magnificent. To begin with, Smart contracts can provide transparency, security, efficiency, and automation for various applications, such as supply chain management, voting systems, crowdfunding, and decentralized finance. Moreover, smart contracts are immutable and irreversible, which means that any errors or bugs in the code can have serious consequences and cannot be easily fixed or updated, which explains its high privacy. Nevertheless, smart contracts are subject to legal and regulatory uncertainties, as different jurisdictions may have different rules and interpretations of how smart contracts should be enforced and validated.

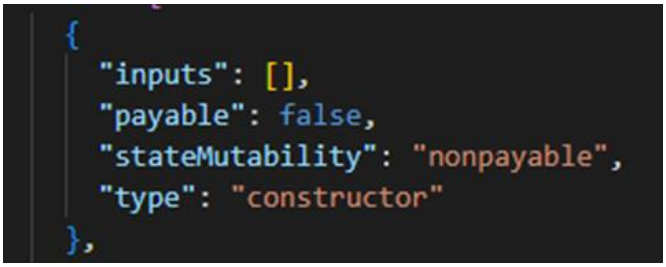
The requirements of the smart contract project should be understood correctly to grasp the processes. First and foremost, the smart contract in this project will act as escrow during the trading process. The smart contract will import the data of NFT products from our website, and online users' implementations to confirm trading procedures. Secondly, the smart contract includes two main functions, which is first creating the currency unit of our webpage (FOODIE), and displaying transactions to users.

3. ABI Illustration

Throughout the process of developing smart contracts, we are required to migrate smart contracts into our group's website using Truffle. After that, the file named ***"Color.json"*** is utilized to interact with the deployed contract on a private blockchain system. The file "***Color.json"*** contains the contract's ABI, which is a list of methods and events that the contract supports. The contract follows the ERC-721 standard for non-fungible tokens, which means that each token is unique and has its own metadata. The contract also implements the ERC-165 standard for interface detection, which means that it can tell if it supports a given interface ID. Corresponding ABI has been written in this file with numerous supported functionalities. The file "***Color.json"*** *also* contains the contract's address, which is the location of the contract on the blockchain. We need the address to connect to the contract and send transactions to it.

Our ABI implementation includes various functions with their unique logic and functionalities:

- *constructor:* This is a special function that is executed only once when the contract is deployed. It does not have any inputs or outputs.



- *approve:* This function allows the owner of a token (FOODIE) to approve another address to transfer it. It takes two inputs: the address of the recipient and the token ID. It does not return outputs.

```
{
  "constant": false,
  "inputs": [
    {
      "internalType": "address",
      "name": "to",
      "type": "address"
    },
    {
      "internalType": "uint256",
      "name": "tokenId",
      "type": "uint256"
    }
  ],
  "name": "approve",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
```


- *balanceOf:* This function returns the number of tokens owned by a given address. It takes one input: the address of the owner. It returns one output: the number of tokens (FOODIE).

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "address",
      "name": "owner",
      "type": "address"
    }
  ],
  "name": "balanceOf",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *colors:* This function returns the color of a token (FOODIE) by its ID. It takes one input: the token ID. It returns one output: the color as a string.

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "colors",
  "outputs": [
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```

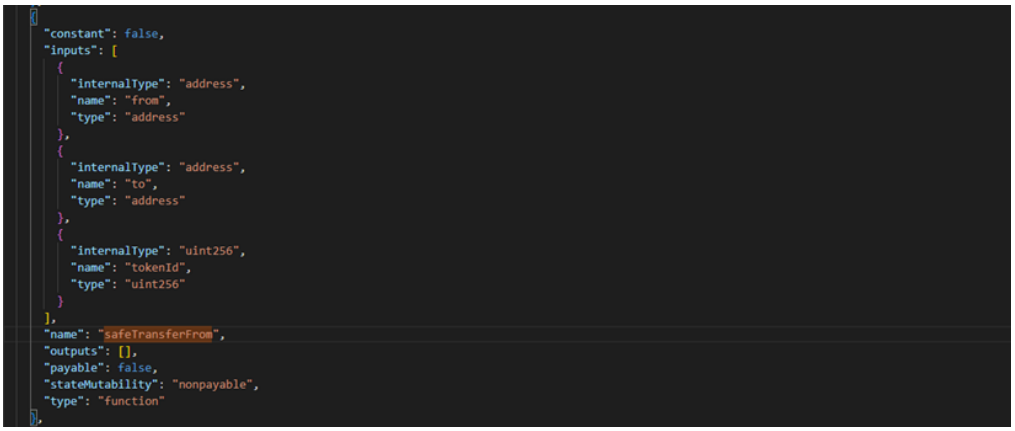
- *getApproved:* This function returns the address that is approved to transfer a token by its ID. It takes one input: the token ID. It returns one output: the address of the approved transferrer.

- *isApprovedForAll:* This function returns whether an operator is approved to transfer all tokens of a given owner. It takes two inputs: the address of the owner and the address of the operator. It returns one output: a boolean value indicating the approval status.

- *mint:* This function creates a new token with a given color and assigns it to the caller. It takes one input: the color as a string. It does not have any outputs.

- *ownerOf:* This function returns the address of the owner of a token by its ID. It takes one input: the token ID. It returns one output: the address of the owner.

- *safeTransferFrom:* This function transfers a token from one address to another, checking if the recipient is a smart contract and if so, calling a function on it. It takes three inputs: the address of the sender, the address of the recipient, and the token ID. It does not have any outputs. This function will be utilized in our project.



- *setApprovalForAll:* This function approves or revokes an operator to transfer all tokens of the caller. It takes two inputs: the address of the operator and a boolean value indicating the approval status. It does not have any outputs.

- *supportsInterface:* This function returns whether the contract supports a given interface ID. It takes one input: the interface ID as a bytes4 value. It returns one output: a boolean value indicating the support status.

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "bytes4",
      "name": "interfaceId",
      "type": "bytes4"
    }
  ],
  "name": "supportsInterface",
  "outputs": [
    {
      "internalType": "bool",
      "name": "",
      "type": "bool"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *tokenByIndex:* This function returns the token ID at a given index in the total supply. It takes one input: the index as a uint256 value. It returns one output: the token ID as a uint256 value.

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "uint256",
      "name": "index",
      "type": "uint256"
    }
  ],
  "name": "tokenByIndex",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *tokenOfOwnerByIndex:* This function returns the token ID at a given index in the balance of a given owner. It takes two inputs: the address of the owner and the index as a uint256 value. It returns one output: the token ID as a uint256 value.

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "address",
      "name": "owner",
      "type": "address"
    },
    {
      "internalType": "uint256",
      "name": "index",
      "type": "uint256"
    }
  ],
  "name": "tokenOfOwnerByIndex",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *tokenURI:* This function returns the URI for a given token ID. It takes one input: the token ID as a uint256 value. It returns one output: the URI as a string.

```
{
  "constant": true,
  "inputs": [
    {
      "internalType": "uint256",
      "name": "tokenId",
      "type": "uint256"
    }
  ],
  "name": "tokenURI",
  "outputs": [
    {
      "internalType": "string",
      "name": "",
      "type": "string"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *totalSupply:* This function returns the total number of tokens in existence. It does not have any inputs or outputs.

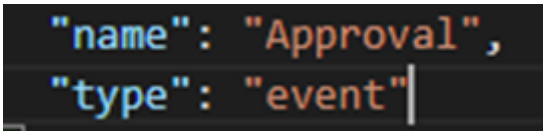
```
{
  "constant": true,
  "inputs": [],
  "name": "totalSupply",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
},
```


- *burn:* This function destroys a token by its ID and removes it from the total supply. It takes one input: the token ID as a uint256 value. It does not have any outputs. It can only be called by the owner of the token or an approved operator.

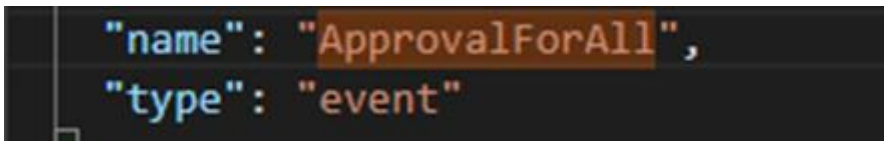
```
{
  "constant": false,
  "inputs": [
    {
      "internalType": "uint256",
      "name": "_tokenId",
      "type": "uint256"
    }
  ],
  "name": "burn",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
}
```


Moreover, the ABI in this code file also contains 3 main events to implement the foregoing functions:

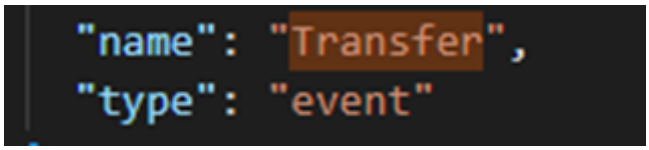
- *Approval*: This event is emitted when the owner of a token approves another address to transfer it. It has three indexed inputs: the address of the owner, the address of the approved transferrer, and the token ID.



- *ApprovalForAll*: This event is emitted when an owner approves or revokes an operator to transfer all their tokens. It has three indexed inputs: the address of the owner, the address of the operator, and a Boolean value indicating the approval status.



- *Transfer*: This event is emitted when a token is transferred from one address to another. It has three indexed inputs: the address of the sender, the address of the recipient, and the token ID.



To conclude, the ABI functions in the code file are a way of defining the interface of a smart contract for interacting with it. They specify the name, type, inputs, outputs, and other properties of each function and event that the contract supports. The ABI functions in the code file are related to the creation, transfer, approval, and destruction of tokens that represent colors on the blockchain. We can use the ABI with a library like ethers.js or web3.js to create a contract instance and call its functions, or listen to its events.

4.Call Graph

Before we talk about the call graph for a contract project including both, let's assess the roles, interactions, and links between the "Migrations" and "Color" contracts.

4.1. Call Graph Synopsis:

Function interactions between the 'Migrations' and 'Color' contracts are shown in the call graph. The edges show the control flow between functions, and each node represents a function.

4.2 Migration Contracts:

Fucntions:

- "constructor": Initializes the "last_completed_migration" and "owner" variables.

-"restricted": A change that restricts the functionality available to the contract owner.

-'setCompleted': This function provides the contract owner with exclusive access to the 'last_completed_migration' variable.

Relationships:

- The "constructor" establishes the contract's initial state upon deployment.

- Only the contract owner has access to the "setCompleted" function when the "restricted" modifier is used.

4.3. Color Contracts:

Function:

- {constructor}: Takes over from {ERC721Full}, initializing the ERC721 token with the prefix "COLOR" and the name "Color."

- {mint`: Enables users to create new tokens with distinctive hues.

Token owners can destroy their tokens by using the `burn` command.

Relationships:

- To produce new tokens, the `mint` function collaborates with the `_mint` function inherited from {ERC721Full`.

- To manage the burning process, the `burn` function communicates with the `_burn` function from {ERC721Full}.

4.4. Relationships between Core Features:

The Principal Elements of Color Contract:

- Existence of Color and Minting:

- Only distinct colors are minted as tokens thanks to the interaction between the `mint` function and the `_colorExists` mapping.

- Burning of tokens and user authorization:

An important access control technique is demonstrated by the `burn` function, which demands that the caller be the token's owner or authorized user in order to burn the token.

The call graph shows how the contracts are modularly designed. The `Color` contract, which extends `ERC721Full}, controls the production and destruction of color tokens, while the `Migrations` contract takes care of deployment-related duties.

Important Notes:

- Access control and deployment-related capabilities are the main topics of the `Migrations` contract.

- To implement ERC-721 token features, the `Color` contract connects with the `ERC721Full` library.

- Minting distinct colors, controlling color existence, and guaranteeing user consent for token burning are essential components of the {Color} contract.

This analysis aids in comprehending the interdependence, control flow, and structure of the contract while it is being executed. It also emphasizes how each contract's essential components are integrated with libraries and what makes each contract unique.

5.Inheritance Graph

5.1 Inheritance Relationships:

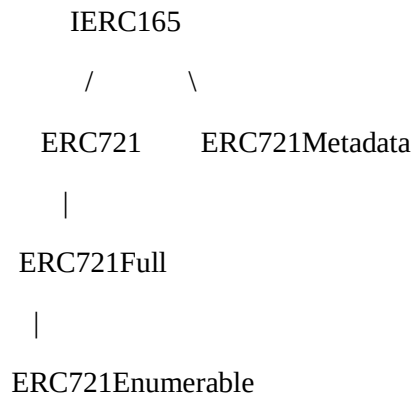
IERC165: IERC165 defines a uniform method for contract introspection and acts as an interface inside the given smart contract ecosystem. Contracts can indicate their support for specific standardized interfaces by adhering to this interface, which improves interoperability and standardization compliance.

ERC721: A commonly used protocol for non-fungible tokens (NFTs), the ERC-721 contract is a basic implementation of the ERC-721 standard. The Ethereum blockchain's non-fungible asset structure is established under ERC721.

ERC721Enumerable: Adding enumerable features to the fundamental functionality, ERC721Enumerable builds upon ERC721. This addon offers improved functionality for handling and dealing with NFTs by adding the ability to enumerate and iterate through the token collection.
ERC721Full: ERC721Full is an enhanced implementation of ERC-721 with additional features. ERC721Full encompasses a whole range of capabilities by combining both enumerable features and metadata, providing a more feature-rich and adaptable NFT architecture.

ERC 721 Metadata: ERC721Metadata focuses on establishing metadata additions for NFTs within the framework of the ERC-721 standard. With the help of this upgrade, each token may now have more descriptive information associated with it, making the representation of individual assets richer and more insightful.

Inheritance Diagram:



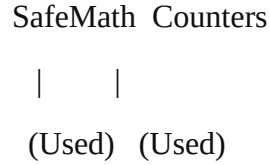
The presented hierarchy illustrates the interrelationship between contracts, portraying a clear lineage wherein ERC721Full extends ERC721, and ERC721Enumerable inherits from ERC721Full. Meanwhile, ERC721Metadata extends ERC721, establishing a comprehensive and structured inheritance model.

5.2 Code Reusability:

SafeMath: To reduce typical vulnerabilities related to arithmetic operations, the SafeMath utility library is a fundamental component. It is a reusable and indispensable tool for a variety of contracts because of its function in guaranteeing safe mathematical computations.

Counters: In a similar vein, the Counters utility library emphasizes code reuse by offering counter management features. This tool makes safe incrementing and decrementing operations easier and provides a reusable counter management solution for contracts.

Code Reusability Diagram:



The diagram visually represents the integration of SafeMath and Counters into different contracts, underscoring their status as shared and utilized libraries within the broader ecosystem.

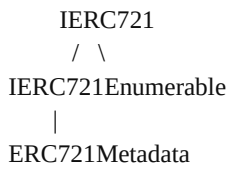
5.3 Function and State Variable Overriding:

ERC721Metadata: To include metadata features, ERC721Metadata may override some methods from its parent contract, ERC721, within the ERC-721 framework. Additional descriptive information related to certain tokens can be included with this change.

IERC721: Acting as an interface, IERC721 specifies the standard features that any contract that complies with the ERC-721 standard has to have. These features are required in contracts that adhere to IERC721, guaranteeing a uniform and compatible use of the NFT standard.

IERC721Enumerable: IERC721Enumerable expands on IERC721 by pointing out possible changes or additions to functions pertaining to token enumeration. Numerous features are added by this addition, enhancing the capabilities of contracts that follow this interface.

Function and State Variable Overriding Diagram:



The hierarchy highlights the opportunity for customization and feature expansion by outlining the relationships between the interfaces. While ERC721Metadata extends ERC721, implying modification relating to metadata features, IERC721Enumerable extends IERC721, indicating an emphasis on improving enumeration features.

Conclusively, the in-depth examination and graphical depiction of inheritance connections, code reuse, and function/state variable substitution offer a thorough comprehension of the structural and behavioral facets of the shown smart contract ecosystem. This scholarly method makes it easier to examine the links and design decisions made in the given contracts and interfaces in a more sophisticated manner.

6.Conclusion

FoodieChain is a blockchain-based project connecting food producers, sellers, and consumers. Using smart contracts, it ensures transparent and secure food transactions. The team, divided into specific roles, is currently on version 0.5.16. Key tools include Truffle, OpenZeppelin Contracts, Web3js, and IPFS.

The executive summary highlights smart contract development steps and emphasizes benefits like transparency and security. The ABI illustration details functions and events crucial for the platform, while the call graph analyzes interactions between 'Migrations' and 'Color' contracts.
The inheritance graph illustrates relationships between interfaces and contracts, showcasing code reusability. Overall, FoodieChain aims to revolutionize the food industry with a transparent and efficient blockchain solution.

7.Refencre

1. Antonopoulos, A. M. 2018. Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media.
2. Truffle Suite Documentation. 2023. [Online] Available at: https://www.trufflesuite.com/docs/truffle/overview [Accessed: November 15, 2023].
3. Open Zeppelin Contracts Documentation. [Online] Available at: https://docs.openzeppelin.com/contracts/ [Accessed: November 15, 2023].
4. Web3.js Documentation. [Online] Available at: https://web3js.readthedocs.io/ [Accessed: November 15, 2023].
5. IPFS Documentation. [Online] Available at: https://docs.ipfs.io/ [Accessed: November 15, 2023].
6. EIP-721: Non-Fungible Token Standard. [Online] Available at: https://eips.ethereum.org/EIPS/eip-721 [Accessed: November 15, 2023].
7. Solidity Documentation. [Online] Available at: https://docs.soliditylang.org/ [Accessed: November 15, 2023].
8. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. 2016. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.
9. Mougayar, W. 2016. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. John Wiley & Sons.
10. Pournaghi, S., & Ramachandran, M. 2020. Blockchain for the Food Industry. Springer.
11. Delmolino, K., Arnett, M., Kosba, A., Miller, A., & Shi, E. 2016. "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab." In International Conference on Financial Cryptography and Data Security (pp. 79-94). Springer.
12. Casey, M. J., & Wong, P. 2018. The Age of Cryptocurrency: How Bitcoin and Digital Money are Challenging the Global Economic Order. St. Martin's Publishing Group.