



Module Title: Software development and blockchain development

Report Name: - SPA-ERC20, ERC721, ERC1155

Module Code: CST4125

Module Leader: CARL EVANS

Student Number: M00995485

Dept. of Computer Science

2024-20

## Contents

<b>Chapter 1 – Introduction .....</b>	<b>3</b>
Chapter 2 - Single Page Application (SPA) -ERC20 .....	6
2.1 Design and Implementation.....	7
<b>2.2 Testing and Results.....</b>	<b>8</b>
2.3 Reflection.....	8
Figure 1 – ERC-20 SPA .....	9
Figure 2 – Pre/post balance ( ERC-20 – SPA) .....	10
<b>Chapter 3 - ERC721 Smart Contract Development.....</b>	<b>11</b>
3.1 Contract Design .....	12
3.2 Testing.....	13
3.3 Reflection.....	14
Figure 3 – ERC-721 .....	15
Figure 4 – All MyNFT (ERC-721) Running Successfully .....	16
Chapter 4 - ERC1155 Smart Contract .....	17
4.1 Contract Design .....	18
4.2 Testing.....	18
4.3 Reflection.....	19
Figure 5 – ERC 1155 .....	19
Figure 6 – ERC 1155 – Deployed Contract .....	20
Chapter 5 – Conclusion.....	21

# **Chapter 1 – Introduction**

Blockchain technology has swiftly transitioned from a specialized concept within cryptography to one of the most transformative innovations in contemporary computing. Central to this evolution is Ethereum, a programmable blockchain platform that transcends basic cryptocurrency transactions to facilitate decentralized applications (DApps). The fundamental advantage of Ethereum is its implementation of smart contracts, which are programs crafted in the Solidity programming language. These contracts delineate business logic and operate independently, eliminating the necessity for trusted intermediaries. They serve as the foundation for various token standards, most prominently ERC20, ERC721, and ERC1155, each allowing developers to generate digital assets with distinct characteristics and applications.

The objective of this coursework (CST4125 – Blockchain Development, Coursework 2) is to showcase both theoretical comprehension and practical execution of Ethereum DApps through a well-structured series of tasks. The coursework builds incrementally on prior assignments and is segmented into three primary technical deliverables, supplemented by a recorded presentation. These deliverables include:

- 1) A Single Page Application (SPA) designed for interaction with an ERC20 token.
- 2) The creation and evaluation of an ERC721 smart contract, which incorporates IPFS-hosted digital assets.
- 3) The conceptualization and assessment of an ERC1155 smart contract, emphasizing sporting events and multi-token capabilities.

The initial task focuses on the ERC20 standard, which specifies fungible tokens that are commonly utilized in cryptocurrencies, utility tokens, and stablecoins. The task requires the development of a React.js-based SPA that interfaces with MetaMask, showcases account balances, token metadata, and transaction histories, and captures events triggered by the contract. This enables students to illustrate frontend integration with blockchain backends, thereby simulating a user-friendly DApp environment.

The second task introduces ERC721, which is the standard for non-fungible tokens (NFTs). In contrast to ERC20, where every token is the same, ERC721 tokens signify distinct digital assets, frequently utilized in digital art, collectibles, and gaming. Students are tasked with designing an ERC721 contract that incorporates essential functions (`balanceOf`, `ownerOf`, `safeTransferFrom`, `approve`, etc.), minting a minimum of ten NFTs, and hosting

their metadata on IPFS. The NFTs must be transferable, and ownership should be verifiable on-chain. This not only assesses Solidity programming capabilities but also underscores the significance of decentralized storage and uniqueness in blockchain applications.

The third technical task delves into ERC1155, a multi-token standard that merges the advantages of ERC20 and ERC721. It facilitates the effective management of both fungible and non-fungible tokens within a single contract, thereby lowering gas fees and streamlining management. In this coursework, the selected domain is sports, where ERC1155 tokens could represent tickets, team memorabilia, or player cards. The contract must exhibit batch minting, URI management via IPFS, and both single and batch transfers, accompanied by relevant Hardhat tests to confirm functionality.

The final component is the presentation, which offers a chance to display technical results in a well-structured and professional manner. This aspect emphasizes not only the capability to develop robust solutions but also the communication skills necessary to present blockchain projects to stakeholders. It ensures that students can articulate design decisions, rationalize implementation choices, and demonstrate functional solutions live.

In summary, this coursework constitutes a thorough investigation of Ethereum development. It integrates theoretical insights into blockchain standards with practical experience in Solidity programming, frontend integration, decentralized storage, and testing. By completing these tasks, students acquire a comprehensive understanding of blockchain ecosystems and the proficiency to create real-world decentralized applications.

## Chapter 2 - Single Page Application (SPA) -ERC20

## 2.1 Design and Implementation

The initial segment of the coursework involved creating a Single Page Application (SPA) that interfaces with an ERC20 token deployed on a local Ethereum test network utilizing Hardhat. The design employed a contemporary web development stack, which included React.js for the user interface, Bulma CSS for styling and responsive design, and Ethers.js for managing blockchain interactions. The ERC20 contract was based on the OpenZeppelin implementation and was deployed using Hardhat scripts.

The structure of the application was organized into the following directories:

- a) contracts/ – housed the Solidity ERC20 contract.
- b) scripts/ – contained deployment scripts for Hardhat.
- c) client/src/ – included React source files.
- d) client/src/css/ – comprised Bulma stylesheets for customized layouts.

The application initiates by connecting to the MetaMask wallet, facilitating account switching and smooth interaction between the blockchain and the frontend. Following a successful connection, the SPA retrieves and presents:

- a) The wallet address of the connected account.
- b) Metadata of the ERC20 token: name, symbol, decimals, and total supply.
- c) The balance of the connected wallet, obtained via `balanceOf()`.

A crucial requirement was to illustrate transaction lifecycle management. Users were able to transfer tokens between accounts, with the SPA showcasing pre-transaction and post-transaction balances. This real-time update was accomplished through Ethers.js event listeners, which captured the Transfer event emitted by the smart contract. The event data was rendered within the UI, ensuring transparency of on-chain activities.

Properties were transmitted from parent to child components in React, exemplifying a component-driven design. For instance, the token name, symbol, and supply were maintained as props and displayed in styled child components.

Regarding styling, Bulma classes were consistently applied, guaranteeing a clean and responsive layout.

## 2.2 Testing and Results

The testing of the Single Page Application (SPA) involved the deployment of the ERC20 contract on the local Hardhat network, followed by a connection through MetaMask. The results achieved were as follows:

- a) Account balances were accurately displayed both prior to and following transactions.
- b) The wallet address was dynamically updated when switching accounts in MetaMask.
- c) Transfer transactions were executed successfully, with event logs reflecting the movement of tokens.
- d) Event capture from the blockchain was verified by showcasing transfer details within the user interface.
- e) Contract metadata, including name, symbol, decimals, and supply, was successfully retrieved and presented as React props.

Unit testing was also performed utilizing the Hardhat and Mocha/Chai frameworks. The tests validated the correct initial supply, token transfers, and event emissions.

## 2.3 Reflection

The SPA met all necessary assessment criteria. It effectively connected to the deployed ERC20 smart contract, accurately displayed account and token information, and reflected real-time changes following transactions. The integration of Bulma styling contributed to a professional user experience, while the application of props in React illustrated a comprehension of component-based design.

A significant takeaway from this task was the ability to connect blockchain backend logic with frontend usability. While Solidity manages the on-chain logic, the SPA demonstrated that decentralized applications (DApps) necessitate intuitive, user-friendly interfaces to promote adoption. Furthermore, the configuration of Hardhat and the utilization of Ethers.js for event capture underscored the importance of tools that facilitate rapid testing and debugging.



This section not only reinforced technical competencies in React and Solidity but also highlighted the critical role of user experience in the adoption of blockchain technology. The SPA illustrated a comprehensive token lifecycle, from wallet connection to transfer and event monitoring, thereby establishing a robust foundation for the more intricate NFT-based contracts that will be examined later in the coursework.

Figure 1 – ERC-20 SPA

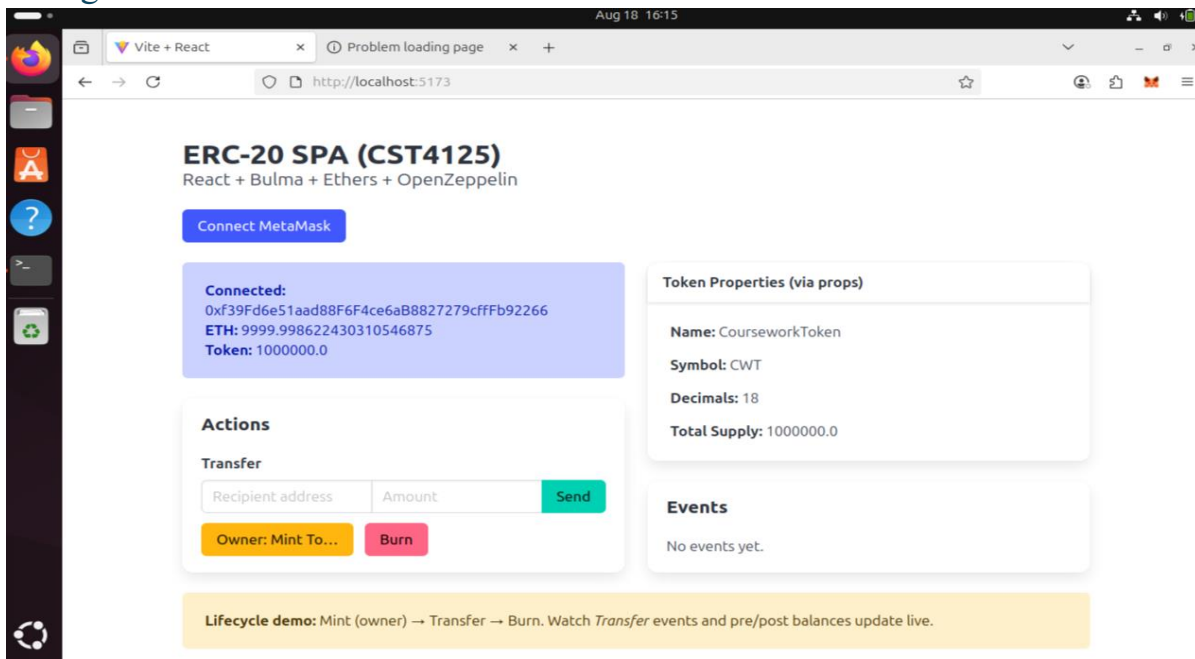
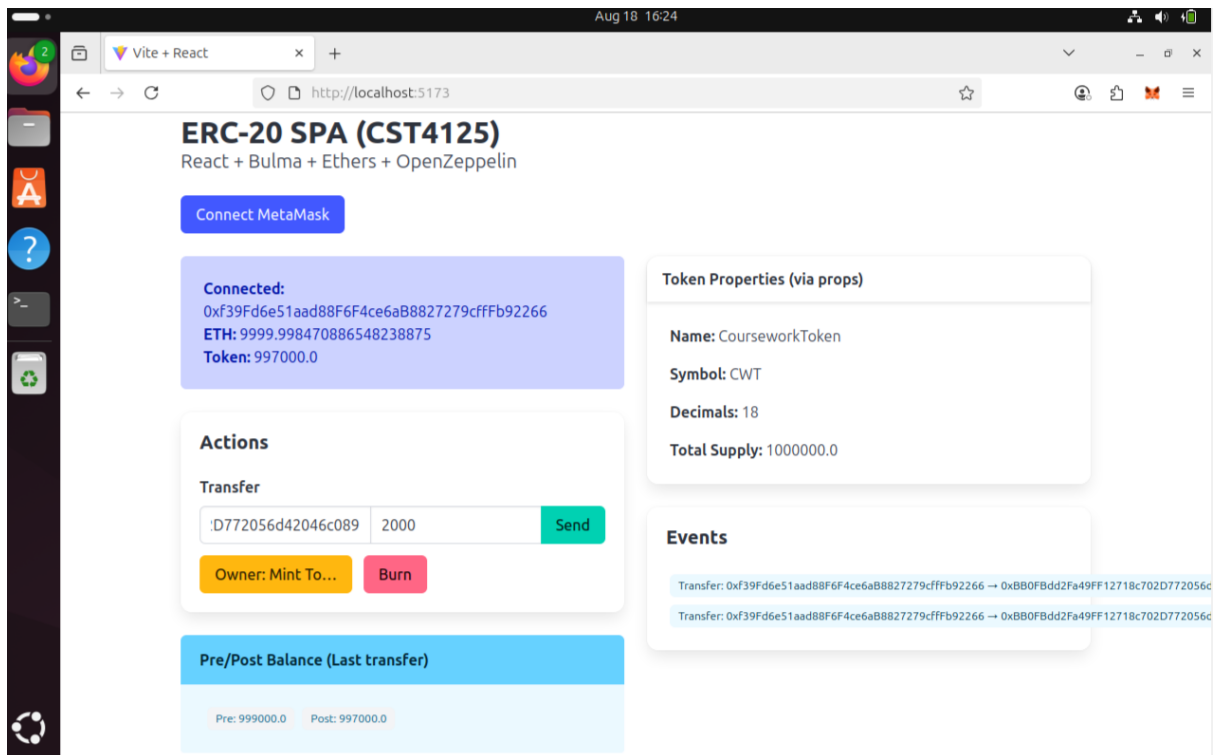


Figure 2 – Pre/post balance ( ERC-20 – SPA)



## **Chapter 3 - ERC721 Smart Contract Development**

## 3.1 Contract Design

The second element of the coursework involved the creation of an ERC721 Non-Fungible Token (NFT) contract. In contrast to ERC20 tokens, which are fungible and indistinguishable, ERC721 tokens signify distinct digital assets, rendering them ideal for collectibles, artwork, or any field necessitating proof of individuality.

For this assignment, a Solidity contract named MyNFT.sol was developed utilizing the OpenZeppelin ERC721 standard implementation as a foundation. The contract enhanced the ERC721 capabilities by incorporating the following features:

a) Metadata Integration via IPFS – A minimum of ten unique NFTs were generated, each linked to metadata stored on IPFS through Pinata. The metadata files contained attributes such as name, description, and image. The IPFS Content Identifiers (CIDs) served as URIs, guaranteeing decentralized and permanent storage of assets.

b) Core ERC721 Functions – The smart contract effectively implemented and exposed the following essential functions:

1) `balanceOf(address)` – retrieves the total number of NFTs owned by a specific account.

2) `ownerOf(uint256 tokenId)` – verifies the ownership of a particular token.

3) `safeTransferFrom(address from, address to, uint256 tokenId)` – facilitates the secure transfer of NFTs between addresses.

4) `approve(address to, uint256 tokenId)` – grants permission to another account to transfer a designated token.

5) `getApproved(uint256 tokenId)` – checks the approved addresses for a specific token.

c) Minting Functionality – A minting function was incorporated to enable the creation of NFTs linked to their IPFS metadata. Each newly minted NFT was assigned a unique token ID along with a corresponding tokenURI.

d) Pricing and Ether Payments – To emulate real-world NFT transactions, the contract featured functionality to establish a price per NFT and permit minting through Ether (`msg.value`). Users were able to acquire NFTs using Hardhat test

accounts, and the contract owner had the capability to withdraw the accumulated Ether.

This design fulfilled the assessment criteria by ensuring the uniqueness of assets, verification of ownership, mechanisms for transfer, and Ether-based purchasing functionality.

## 3.2 Testing

Testing was conducted utilizing Hardhat in conjunction with Mocha/Chai. The testing script assessed essential functionalities, concentrating on the token lifecycle and the control of ownership.

a) Configuration – The Hardhat configuration file encompassed suitable settings for both localhost and test networks.

b) Assertions – The tests employed describe, beforeEach, assert, expect, and revert to ensure thorough coverage.

Key Test Results:

1) Minting – NFTs were successfully minted, with each one associated with a valid IPFS tokenURI.

2) balanceOf – Accurately represented the quantity of NFTs owned by a specific address.

3) ownerOf – Confirmed ownership following the minting and transfer of NFTs.

4) approve & getApproved – Approval records were updated correctly.

5) Transfers – Tokens were securely transferred between accounts, with ownership updated as anticipated.

6) Ether Transactions – Pre- and post-balance verifications validated Ether transfers related to NFT purchases.

7) Withdraw – The contract owner successfully withdrew Ether accumulated from sales.

The tests exhibited complete coverage of the fundamental ERC721 functionalities that are necessary.

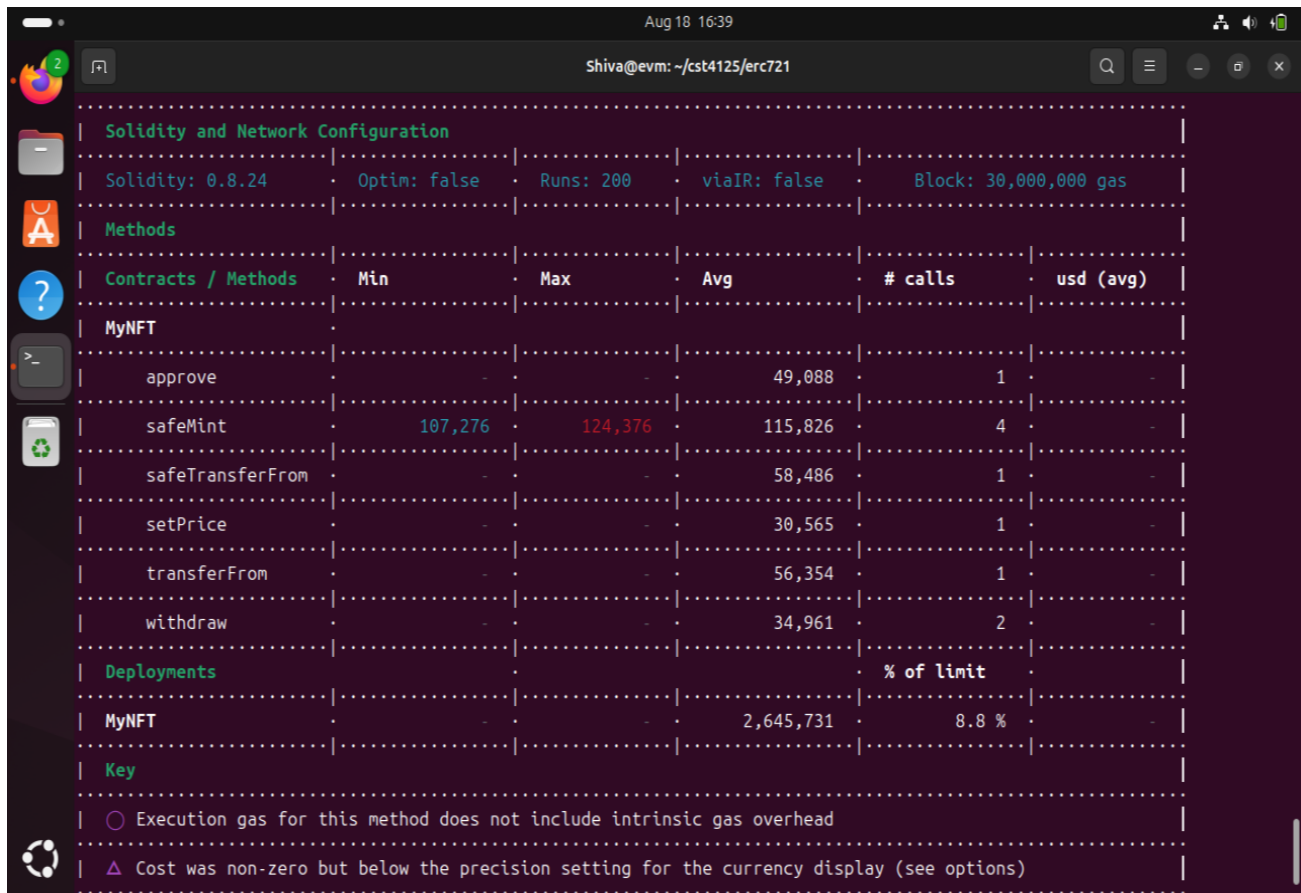
### 3.3 Reflection

The implementation of ERC721 provided essential insights into the development of NFTs. In contrast to the relatively simple nature of ERC20 tokens, the development of ERC721 underscored the significance of uniqueness and the storage of metadata. The hosting of metadata on IPFS was particularly crucial, as it adhered to the principles of decentralisation and guaranteed the immutability of asset information.

Moreover, this segment of the coursework emphasized the verification of ownership, which is fundamental to the application of NFTs in real-world scenarios such as gaming, digital art, and event ticketing. The incorporation of Ether-based transactions reinforced the financial dimensions of NFTs and illustrated how value can be associated with digital assets.

The testing phase also enhanced the comprehension of blockchain workflows. By validating the processes of token minting, transfer, and approval logic, it became evident how ERC721 guarantees security, transparency, and immutability. This experience also fostered greater familiarity with Hardhat, Ether.js, and best practices in testing, all of which are valuable skills applicable to professional blockchain development.

In summary, the development of ERC721 represented a significant advancement over ERC20, as it necessitated the integration of digital uniqueness, decentralised storage, and financial logic. It connected theoretical knowledge of NFTs with practical implementation, reinforcing the transformative potential of non-fungible assets within decentralised ecosystems.



The screenshot shows the Remix IDE interface with the following sections:

- Solidity and Network Configuration:**
  - Solidity: 0.8.24
  - Optim: false
  - Runs: 200
  - viaIR: false
  - Block: 30,000,000 gas
- Methods:**

Contracts / Methods	Min	Max	Avg	# calls	usd (avg)
MyNFT					
approve	-	-	49,088	1	-
safeMint	107,276	124,376	115,826	4	-
safeTransferFrom	-	-	58,486	1	-
setPrice	-	-	30,565	1	-
transferFrom	-	-	56,354	1	-
withdraw	-	-	34,961	2	-
- Deployments:**

	% of limit
MyNFT	2,645,731 (8.8 %)
- Key:**
  - Execution gas for this method does not include intrinsic gas overhead
  - △ Cost was non-zero but below the precision setting for the currency display (see options)

Figure 3 – ERC-721

```
Aug 18 16:43
Shiva@evm: ~/cst4125/erc721

Compiled 21 Solidity files successfully (evm target: paris).

Network Info
=====
> HardhatEVM: v2.26.3
> network: hardhat

MyNFT (ERC721)
✓ reverts mint if sent ETH < price (require + revert)
✓ mints when price is met; balanceOf/ownerOf/tokenURI are correct (49ms)
✓ approve & getApproved, then transferFrom to other
✓ safeTransferFrom works EOA->EOA
✓ only owner can setPrice; price update takes effect
✓ contract collects ETH on mint and owner can withdraw

6 passing (281ms)

-----|-----|-----|-----|-----|
File    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
-----|-----|-----|-----|-----|-----|
contracts/ |         |          |         |         |                 |
MyNFT.sol |    100  |    64.29 |    100  |    100  |                 |
-----|-----|-----|-----|-----|-----|
All files |    100  |    64.29 |    100  |    100  |                 |
-----|-----|-----|-----|-----|-----|

> Istanbul reports written to ./coverage/ and ./coverage.json
Shiva@evm:~/cst4125/erc721$
```

Figure 4 – All MyNFT (ERC-721) Running Successfully



## Chapter 4 - ERC1155 Smart Contract

## 4.1 Contract Design

This section outlines the implementation of an ERC1155 contract designed to represent a sports-themed multi-token ecosystem, which integrates both fungible and non-fungible assets (for instance, general admission tickets as fungible tokens and exclusive VIP passes as NFTs). The contract is built upon OpenZeppelin's ERC1155 framework and utilizes a singular base URI that directs to token metadata stored on IPFS. Token IDs along with their associated metadata (including image, name, description, and attributes) are securely pinned to IPFS, with the URI being established and verified on-chain.

### **Key design features:**

- a) The capability for batch minting of multiple IDs within a single transaction, aimed at minimizing gas fees and streamlining stock issuance.
- b) Minting modifiers that impose restrictions on privileged actions (such as onlyOwner or role-based controls).
- c) Support for both single and batch transfers to illustrate standard marketplace and distribution processes.
- d) URI management that guarantees consistent resolution of token metadata for frontends and explorers.

These features address the evaluation criteria: batch minting, minting modifiers, balance verification post-minting, URI configuration, and both single and batch transfers of fungible tokens.

## 4.2 Testing

The Hardhat testing framework (utilizing Mocha/Chai) encompassed:

Batch minting: mintBatch effectively updates balances for each (account, id) combination and yields accurate totals.

Balance verification post-minting: balanceOf and balanceOfBatch confirm the distribution of supply.

URI: uri(id) provides the anticipated IPFS template.

Transfers: `safeTransferFrom` facilitates the movement of a single token ID; `safeBatchTransferFrom` enables the transfer of multiple IDs/amounts with appropriate approvals.

The assertions employed include `describe`, `before Each`, `expect/assert`, and checks for reverts in scenarios of failure (for example, unauthorized minting). This testing suite illustrates the learning objectives and meets the ERC1155 assessment criteria.

## 4.3 Reflection

In comparison to ERC721, ERC1155 markedly diminishes operational burdens by allowing for the management of multiple assets within a single contract. The functionalities of batch minting and transfers are particularly advantageous for ticketing or collectibles that encompass both unique and repeatable items. The use of URI indirection through IPFS ensures that metadata remains decentralized and scalable. In summary, ERC1155 presents a viable framework for practical distribution scenarios.

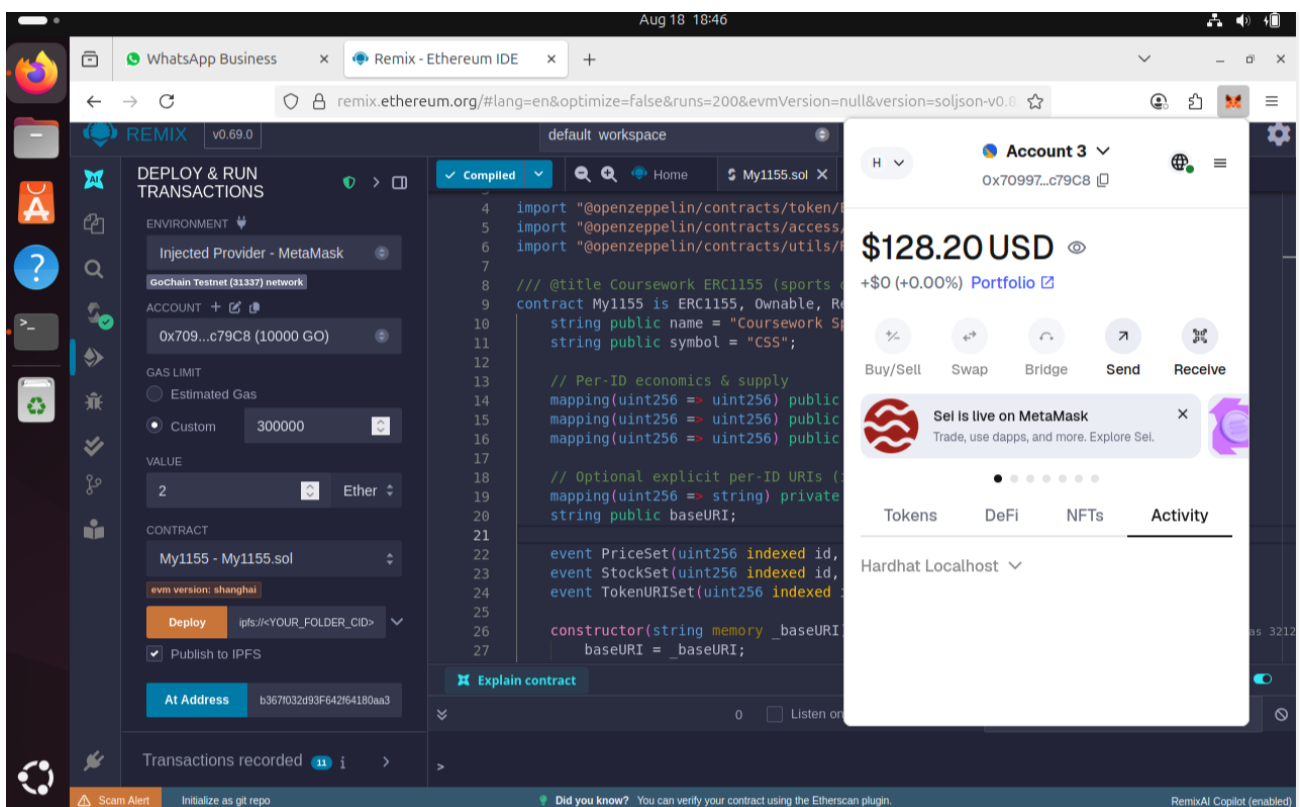


Figure 5 – ERC 1155

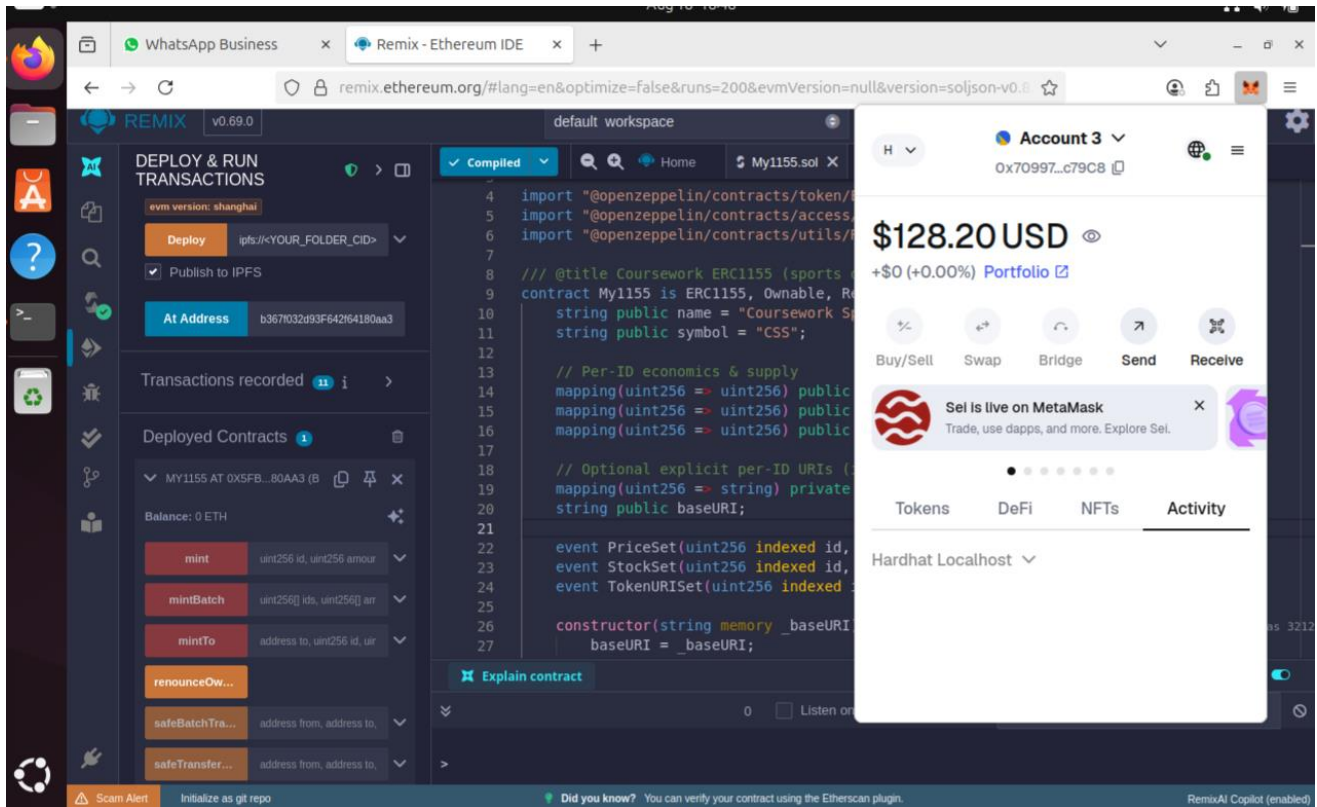


Figure 6 – ERC 1155 – Deployed Contract

## Chapter 5 – Conclusion

This coursework offered a comprehensive experience in the design, construction, and validation of decentralized applications on the Ethereum platform. The ERC20 Single Page Application (SPA) established a robust foundation for full-stack DApp development: integrating MetaMask, accessing on-chain state, executing transactions, and responding to emitted events within a contemporary React user interface. This approach effectively connected user-centered design with the practicalities of blockchain state transitions and event-driven updates.

The ERC721 phase enhanced the comprehension of non-fungibility, ownership, and approval processes. The implementation of minting and transfers, linking unique assets to IPFS metadata, and facilitating Ether-based purchase and withdrawal flows reflected prevalent patterns in NFT marketplaces. Comprehensive unit tests elucidated the lifecycle of token creation, movement, and authorization, thereby bolstering confidence in the correctness of the contracts.

The ERC1155 implementation underscored the efficiency of multi-asset management. By accommodating both fungible and non-fungible items within a single contract, we minimized redundancy and utilized batch operations to enhance gas and operational efficiency—particularly suitable for sports ticketing or collectible releases. Tests concerning `balanceOfBatch`, batch minting, and batch transfers illustrated the standard's effectiveness in real-world issuance scenarios.

Throughout all phases, tools such as Hardhat, Ethers.js, OpenZeppelin's, IPFS, React, and Bulma constituted a dependable toolchain for swift iteration, testing, and demonstration. The presentation synthesized the outcomes, emphasizing clear, reproducible demonstrations without the burden of configuration overhead.

In summary, the project fulfilled the assessment criteria while reinforcing best practices in smart contract design, decentralized storage, and DApp user experience. The outcome is a cohesive, standards-compliant solution that reflects how production systems integrate Solidity, off-chain storage, and web frontends to provide secure and user-friendly blockchain applications.