

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH



Report:

Bài tập 1

Lớp: 19_22

Môn: Blockchain và ứng dụng

Niên khóa: 2022-2023

Mục lục

I.	Thông tin sinh viên và phân công thực hiện	2
II.	Nội dung tìm hiểu	2
	A. Lý thuyết:	3
	1. Tìm hiểu và trình bày về khái niệm Finality trong Blockchain.	3
	2. Tìm hiểu, so sánh 2 công cụ hỗ trợ lập trình smart contract	4
	B. Thực hành	13
III.	Tài liệu tham khảo	19

I. Thông tin sinh viên và phân công thực hiện

Tên nhóm: Gaming House.

Danh sách thành viên:

STT	Họ tên	MSSV	Phân chia công việc	Hoàn thành
1	Nguyễn Quang Định	19120492	Thực hành (NFT)	100%
2	Phạm Đức Huy	19120534	Lý thuyết – Hardhat + Report	100%
3	Lê Nhật Khánh Hưng	19120494	Lý thuyết - Finality	100%
4	Lê Thành Lộc	19120534	Lý thuyết – Remix	100%
5	Nguyễn Hữu Phương	19120625	Thực hành (NFT)	100%

II. Nội dung tìm hiểu

A. Lý thuyết:

1. Tìm hiểu và trình bày về khái niệm Finality trong Blockchain.

1.1 Khái niệm Finality

Finality là quá trình xử lý để hệ thống chấp nhận giao dịch là hợp lệ. Quá trình này để đảm bảo rằng giao dịch sẽ không thể bị thay đổi bởi bất kỳ thực thể nào. Finality chính xác là quá trình xử lý mà bạn phải chờ đợi khi thực hiện giao dịch, qua đó có thể đo lường độ trễ (latency) của giao dịch.

1.2 Các loại Finality

- *Probabilistic Finality*: Với một số lượng khối nhất định sau khi một khối được ghi lại, có thể giả định rằng khối đó sẽ là vĩnh viễn và các giao dịch của nó là cuối cùng.

- *Economic Finality*: Một người sáng tạo đề xuất một khối và các nút bỏ phiếu cho tính đúng đắn của nó. Để phá vỡ một chuỗi trong mô hình này, kẻ tấn công cần có đủ phiếu bầu để ngăn chặn việc bao gồm một khối tốt (incorrect vote) hoặc cho phép tạo một chuỗi song song, với các khối khác nhau ở cùng một độ cao (equivocation).

- *Absolute Finality*: Theo định kỳ, một giao thức phân tán chuyên dụng xác định số lượng khối cuối cùng được hoàn thiện. Các khối sau cấp độ này vẫn có thể được tổ chức lại và có các giao dịch khác nhau trong một chuỗi thay thế, nhưng những khối trước đó là bất biến mãi mãi.

1.3 Phân tích, so sánh tính Finality của ít nhất 2 thuật toán đồng thuận

- *Khái quát thuật toán đồng thuận*: Thuật toán đồng thuận của blockchain là cơ chế đảm bảo các giao dịch được tạo ra trên blockchain là đúng đắn, trung thực và minh bạch. Về bản chất, blockchain bao gồm nhiều node kết hợp lại tạo ra một mạng lưới. Để một giao dịch được ghi lại trên blockchain, nó phải được đồng ý đồng thời bởi tất cả các node trên mạng lưới. Nếu trong mạng lưới có một block bị thay đổi, dữ liệu này được so sánh với các dữ liệu của các khối khác. Nếu có sự khác biệt thì nó sẽ không cho phép dữ liệu ấy được ghi vào bên trong Blockchain. Đó là cách blockchain được thiết kế để chống lại sự thay đổi dữ liệu.

	Proof of Work (PoW)	Proof of Stake (PoS)
Ai có thể khai	Sức mạnh tính toán càng cao,	Càng stake nhiều tiền, bạn

thác / xác thực các khối?	xác suất khai thác một khối càng cao.	càng có nhiều khả năng được xác thực một khối mới.
Làm thế nào một khối được khai thác / xác thực?	Các thợ đào cạnh tranh để giải các câu đố toán học phức tạp bằng cách sử dụng tài nguyên tính toán của mình.	Thông thường, thuật toán sẽ xác định người chiến thắng một cách ngẫu nhiên hoặc dựa trên số lượng tiền đã stake.
Thiết bị khai thác	Phần cứng khai thác chuyên nghiệp, chẳng hạn như ASIC, CPU và GPU.	Mọi máy tính hoặc thiết bị di động có kết nối internet.
Phần thưởng sẽ được phân bổ như thế nào?	Người đầu tiên khai thác khối sẽ nhận được phần thưởng khối.	Người xác thực có thể nhận được một phần phí giao dịch thu được từ khối mà họ đã xác thực.
Mạng được bảo mật như thế nào?	Hàm băm càng lớn, mạng càng an toàn.	Cơ chế khóa tiền mã hóa trên blockchain để bảo mật mạng.

2. Tìm hiểu, so sánh 2 công cụ hỗ trợ lập trình smart contract

2.1 Tìm hiểu công cụ HardHat.

2.1.1 Định nghĩa

- **Hardhat** là một môi trường phát triển giúp các nhà phát triển thử nghiệm, biên dịch, triển khai và gỡ lỗi các dApp trên chuỗi khối Ethereum. Nó đóng một vai trò quan trọng trong việc hỗ trợ các lập trình viên và nhà phát triển quản lý các tác vụ, vốn rất quan trọng đối với sự phát triển hợp đồng thông minh và dApp.

- Môi trường phát triển hợp đồng thông minh Hardhat cung cấp các công cụ phù hợp cho các nhà phát triển để quản lý quy trình phát triển. Ngoài ra, Hardhat cũng đảm bảo hiệu quả bằng cách giới thiệu tự động hóa trong các bước cụ thể cùng với cơ sở của các chức năng mới và hiệu quả. Hardhat có một mạng Ethereum cục bộ được xây dựng trước được điều chỉnh cho mục tiêu cốt lõi của sự phát triển. Phương trình Hardhat Solidity thể hiện rõ ràng trong khuôn khổ tập trung vào gỡ lỗi Solidity. Đồng thời, Hardhat cũng có tính năng cơ sở của dấu vết ngăn xếp và thông báo cho lỗi

dApp. Do đó, các nhà phát triển có thể có sẵn các công cụ cần thiết để xác định phiên bản và nguyên nhân gây ra lỗi ứng dụng. Nó cũng có thể đưa ra câu trả lời cần thiết để giải quyết các vấn đề cơ bản của lỗi ứng dụng.

- Một điểm nổi bật quan trọng khác của **Hardhat** đề cập đến các plugin, cung cấp một loạt các tiện ích. Các plugin đảm bảo rằng các nhà phát triển có thể chọn những plugin mà họ muốn thêm vào quá trình phát triển. Ngay cả khi bạn có thể tìm thấy các plugin mặc định trong bản dựng, bạn có thể ghi đè chúng theo nhu cầu của mình. Về cơ bản, Hardhat không quan tâm đến các công cụ hoặc chức năng mà các nhà phát triển muốn có trong chương trình của họ.

2.1.2 Hardhat Runner là gì?

- Yêu cầu quan trọng nhất đối với các hoạt động triển khai Hardhat là đề cập đến **Hardhat Runner**. Đây là giao diện dòng lệnh hoặc lệnh *CLI* được sử dụng cho các tương tác với Hardhat. **Hardhat Runner** giống như một người chạy nhiệm vụ, cung cấp nhiều giải pháp thay thế và tùy chọn cho người dùng. Các nhà phát triển phải lưu ý rằng một phần chính của thiết kế cho trình chạy tác vụ nhấn mạnh vào các nhiệm vụ và plugin. Thiết kế của trình chạy tác vụ cũng ngụ ý rằng các nhà phát triển chạy một tác vụ mỗi khi họ chạy Hardhat từ CLI. Lệnh ví dụ để chạy Hardhat từ CLI sẽ như sau:

npx hardhat compile

- Khi bạn chạy lệnh trong chuỗi khối Hardhat, nó sẽ bắt đầu tác vụ biên dịch tích hợp sẵn. Sau đó, một nhiệm vụ có thể gọi một nhiệm vụ khác, do đó dẫn đến việc phát triển và định nghĩa các quy trình công việc phức tạp dễ dàng hơn. Ngoài ra, các nhà phát triển cũng có thể sử dụng Hardhat để ghi đè các tác vụ hiện có, do đó cho phép các tùy chọn để mở rộng hoặc tùy chỉnh quy trình công việc.

2.1.3 Mạng Hardhat là gì?

- Điểm nhấn quan trọng tiếp theo trong hướng dẫn Hardhat, ta đề cập đến mạng Hardhat (**Hardhat Network**). Theo định nghĩa của Hardhat, khung phát triển hợp đồng thông minh có một mạng Ethereum cục bộ được xây dựng trước. Mạng Ethereum cục bộ giúp hỗ trợ các nhiệm vụ phát triển trên Hardhat. Đây là một yêu cầu không thể thiếu đối với các nhà phát triển để làm việc trên thử nghiệm, chạy, gỡ lỗi và

triển khai mã ứng dụng phi tập trung và các hợp đồng thông minh.

2.1.4 Hoạt động của mạng Hardhat

- Hoạt động của mạng Hardhat tạo ra một điểm nhấn đầy hứa hẹn khác trong việc tìm hiểu Hardhat Solidity development framework. Một trong những chi tiết đầu tiên về hoạt động của mạng Hardhat chỉ ra việc khai thác một khối cho mọi giao dịch mà không gặp phải bất kỳ sự chậm trễ nào. Ngoài ra, mạng cũng dựa vào sự hỗ trợ của việc triển khai EVM, giống như mạng được sử dụng trên các nền tảng như Ganache , Remix và Ethereum Studio. Ngoài ra, mạng này cũng cung cấp hỗ trợ cho nhiều hard fork như Petersburg, Byzantium và Constantinople.

2.1.5 Sử dụng mạng Hardhat.

- Mạng Hardhat đi kèm với các hành vi mặc định nhất định. Một trong những ví dụ phổ biến đề cập đến việc khuôn khổ bắt đầu một phiên bản khi mạng trống hoặc giá trị “defaultNetwork” đã được định cấu hình là “hardhat”. Các nhà phát triển có thể sử dụng mạng để chạy các tập lệnh, thử nghiệm và các tác vụ khác nhau. Khung phát triển hợp đồng thông minh Hardhat cũng cho phép bổ sung nhiều plugin hiệu quả. Ví dụ: bạn có thể thêm một số plugin phổ biến như Truffle , ethers.js và Waffle, trong số các bổ sung khác. Điều thú vị là tất cả các plugin đi kèm với sự đảm bảo của kết nối trực tiếp với nhà cung cấp. Cũng cần lưu ý rằng các ứng dụng khách bên ngoài như Metamask cũng có thể kết nối với mạng Hardhat, do đó cải thiện chức năng của nó.

2.1.6 Solidity Stacks Traces là gì?

- Ta sẽ làm sáng tỏ tầm quan trọng của **Solidity Stacks Traces**. Hardhat hỗ trợ Solidity theo tiêu chuẩn được công nhận rộng rãi và mạng có ý tưởng rõ ràng về các hợp đồng chạy trên mạng. Ngoài ra, mạng cũng biết về các mục tiêu dự kiến của các hợp đồng thông minh Solidity. Đồng thời, mạng Hardhat cũng xác định lý do đằng sau sự thất bại của các hợp đồng thông minh một cách hiệu quả. Trong trường hợp cuộc gọi hoặc giao dịch không thành công, mạng sẽ cung cấp một ngoại lệ cho người dùng.

- Ngoại lệ sẽ phục vụ theo dõi ngăn xếp JavaScript và Solidity kết hợp, do đó thiết lập tác động qua lại của Hardhat Solidity . Nó gợi ý rõ ràng rằng dấu vết ngăn xếp sẽ bắt đầu bằng JavaScript và tiếp tục ở định dạng tương tự cho đến khi có hợp

đồng. Sau đó, dấu vết ngăn xếp bao gồm một ngăn xếp cuộc gọi Solidity hoàn chỉnh. Do đó, các nhà phát triển có thể sử dụng các dấu vết ngăn xếp của mạng để xác định lý do chính xác cho sự thất bại của hợp đồng hoặc giao dịch.

2.1.7 Thông báo lỗi tự động?

- Mạng Hardhat có khả năng tìm ra trường hợp thất bại của một cuộc gọi hoặc giao dịch. Ngoài ra, nó cũng có thể xác định chính xác lý do thất bại của giao dịch hoặc cuộc gọi. Do đó, các nhà phát triển có thể sử dụng **Blockchain Hardhat Development Framework** để gỡ lỗi dễ dàng và hiệu quả hơn. Khi giao dịch thất bại mà không có bất kỳ lý do xác định rõ ràng nào, mạng lưới cung cấp sự rõ ràng bằng cách đưa ra lời giải thích chi tiết về lỗi. Các thông báo lỗi tự động đóng một vai trò quan trọng trong quá trình gỡ lỗi và có thể gặp phải các tình huống khác nhau. Một số trường hợp phổ biến trong đó mạng sẽ hiển thị thông báo lỗi tự động bao gồm những trường hợp sau:

- Khi người dùng gọi một hàm mà không chỉ định các tham số chính xác.
- Sử dụng tài khoản phi hợp đồng để gọi một chức năng bên ngoài.
- Lệnh gọi không chính xác cho một hợp đồng được biên dịch trước.
- Gửi Ether (ETH) đến một hợp đồng không có chức năng nhận hoặc dự phòng phải trả.
- Triển khai các hợp đồng trên mạng Hardhat vượt qua giới hạn kích thước mã bit.
- Gửi không đủ số lượng Ether hoặc ETH.
- Hiệu quả của Hardhat trong việc xác định các thông báo lỗi tự động giúp đảm bảo các cải tiến có giá trị sâu sắc cho các nhà phát triển. Quan trọng nhất, các nhà phát triển có thể có lợi thế về tốc độ tốt hơn trong quá trình gỡ lỗi của họ trước khi triển khai mã.

2.1.8 Cài đặt Hardhat

- Điểm nổi bật cơ bản nhất trong hướng dẫn về Hardhat sẽ đề cập đến các phương pháp cài đặt hardhats vì nó là điểm khởi đầu trong hành trình của một Developer. Điều thú vị là bạn không phải trải qua các bước phức tạp và nhiều hướng dẫn để cài đặt Hardhat. Việc cài đặt Hardhat để phát triển Ethereum là một công việc khá dễ

dàng, bạn có thể tự quản lý và hoàn thành trong vòng vài phút. Trước hết, bạn phải nhớ rằng Hardhat được sử dụng thông qua cài đặt cục bộ trong các dự án của một nhà phát triển. Lý do chính cho điều tương tự phản ánh nhu cầu tái tạo môi trường để tránh bất kỳ xung đột phiên bản nào trong tương lai. Bạn có thể sử dụng lệnh sau để cài đặt Hardhat.

`npm install --save-dev hardhat`

- Phải lưu ý rằng bước đầu tiên trong quá trình cài đặt nhấn mạnh vào việc tạo dự án npm. Người dùng có thể đạt được điều tương tự bằng cách xác định một thư mục trống và chạy lệnh *`“npm init”`*, sau đó tuân thủ các hướng dẫn đã chỉ định. Khi bạn đã chuẩn bị hoàn thiện một dự án và bắt đầu phát triển, bạn phải sử dụng lệnh cài đặt Hardhat. Trong quá trình cài đặt, bạn phải ghi nhớ rằng bạn phải sử dụng cài đặt cục bộ. Cách tiếp cận được đề xuất để sử dụng cài đặt cục bộ là lệnh *`“npx”`*. Bạn có thể nghĩ về một ví dụ như *`“npx hardhat”`* để sử dụng cài đặt cục bộ.

2.1.9 Sử dụng Hardhat căn bản

- Điểm nổi bật lớn nhất trong bất kỳ hướng dẫn sử dụng Hardhat nào rõ ràng sẽ là thu hút cuộc thảo luận về các phương pháp thực tế để sử dụng Hardhat. Người mới bắt đầu có thể khám phá các khả năng với Hardhat và các chức năng của nó bằng cách sử dụng một dự án mẫu làm tài liệu tham khảo. Bạn có thể bắt đầu sử dụng Hardhat sau khi hoàn tất quá trình cài đặt biến thể barebone của môi trường phát triển. Sau khi hoàn tất cài đặt môi trường phát triển, người dùng có thể chạy các tác vụ và thử nghiệm khác nhau cũng như biên dịch mã và sử dụng mạng Hardhat. Nó cũng có thể giúp các nhà phát triển triển khai các hợp đồng mà không cần cài đặt bất kỳ plugin nào.

- Nếu bạn muốn bắt đầu một dự án phát triển *Hardhat Solidity*, thì bạn phải bắt đầu bằng cách tạo một dự án. Các nhà phát triển có thể sử dụng lệnh *`“npx hardhat”`* trong thư mục dự án để bắt đầu một dự án. Khi bạn đã hoàn thành nhiệm vụ, bạn có thể tạo một dự án mẫu cơ bản để khám phá các phương pháp hay nhất để bắt đầu sử dụng Hardhat. Hãy nhớ rằng dự án mẫu sẽ yêu cầu các nhà phát triển cài đặt *`“hardhat-ethers”`* và *`“hardhat-waffle”`* để hỗ trợ các bài kiểm tra Waffle.

2.1.10 Nhiệm vụ chính

- Công việc của các nhà phát triển trong việc sử dụng blockchain Hardhat chủ yếu sẽ xoay quanh các task. Do đó, điều quan trọng là phải tìm kiếm các tác vụ có sẵn và bạn có thể làm như vậy bằng cách thực hiện lệnh “`npx hardhat`”. Bằng cách sử dụng một biến thể cơ bản của Hardhat mà không cần tận dụng bất kỳ plugin nào, bạn có thể tìm thấy một số tác vụ được tích hợp sẵn. Mặt khác, bạn có thể tìm thấy nhiều tác vụ hơn khi tiếp tục thêm các plugin. Một trong những ví dụ về nhiệm vụ sẽ đề cập đến nhiệm vụ “tài khoản”. Bạn có thể chạy tác vụ với lệnh “`npx tài khoản hardhat`” làm đầu vào và kết quả sẽ cung cấp danh sách các tài khoản được liên kết với dự án.

2.1.11 Tổng kết lại

- Điểm nổi bật đáng chú ý khác trong các trường hợp sử dụng phát triển hợp đồng thông minh Hardhat sẽ đề cập đến việc biên dịch mã. Khả năng biên dịch mã là một điểm nhấn cần thiết cho sự phát triển của Ethereum. Trong trường hợp này, bạn có thể xem qua các hợp đồng thông minh của dự án bằng cách sử dụng thư mục “contract/”. Tất cả những gì bạn cần làm là sử dụng lệnh sau để biên dịch mã:

`compleie hardhat npx`

2.1.12 Testing và deploying contracts

- Danh sách các chức năng trong Hardhat cũng thu hút sự chú ý đến việc thử nghiệm và triển khai các hợp đồng. Bạn có thể thêm thư viện từ Waffle tests và Ethers.js để thử nghiệm và thêm các thư viện khác nếu cần. Lệnh kiểm tra các hợp đồng trong Hardhat sẽ như thế này:

`npx hardhat test`

- Sau khi hoàn thành các bài kiểm tra, bạn có thể tìm kiếm quy trình triển khai Hardhat bằng cách tận dụng một tập lệnh. Mẫu-script.js có thể giúp bạn triển khai hợp đồng bằng cách sử dụng lệnh sau.

`npx hardhat run scripts/sample-script.js`

2.2 Tìm hiểu công cụ Remix.

2.2.1 Định nghĩa

- **Remix** là một ứng dụng mã nguồn mở hỗ trợ cho lập trình smart contract cho

người dùng ở mọi cấp độ hiểu biết. Nó không yêu cầu thiết lập gì cả, nhờ đó chúng ta có thể lập trình Smart Contract ngay lập tức với một bộ bổ sung phong phú với GUI vô cùng trực quan.

Remix được sử dụng để viết, thử nghiệm và triển khai các Smart Contract, bạn có thể sử dụng nó như một sân chơi để học hỏi, giảng dạy về phát triển blockchain.

2.2.2 Sự khác nhau giữa “Remix IDE” và “Remix”

- Mặc dù thực tế có “Remix IDE” và “Remix” thường được sử dụng thay thế cho nhau nhưng chúng lại không hoàn toàn giống nhau. **Remix IDE** thực chất chỉ là một phần của bức tranh lớn hơn, đó là **Remix Project** hay **Remix**. Sau này là một nền tảng cho các công cụ phát triển hoạt động dựa trên kiến trúc plugin. Hơn nữa, *Remix Project* bao gồm một số dự án phụ, tất nhiên, *Remix IDE* là một trong số đó. Hai dự án phụ khác là Remix Plugin Engine và Remix Libs.

2.2.3 Tổng quan về các thuộc tính và năng của Remix

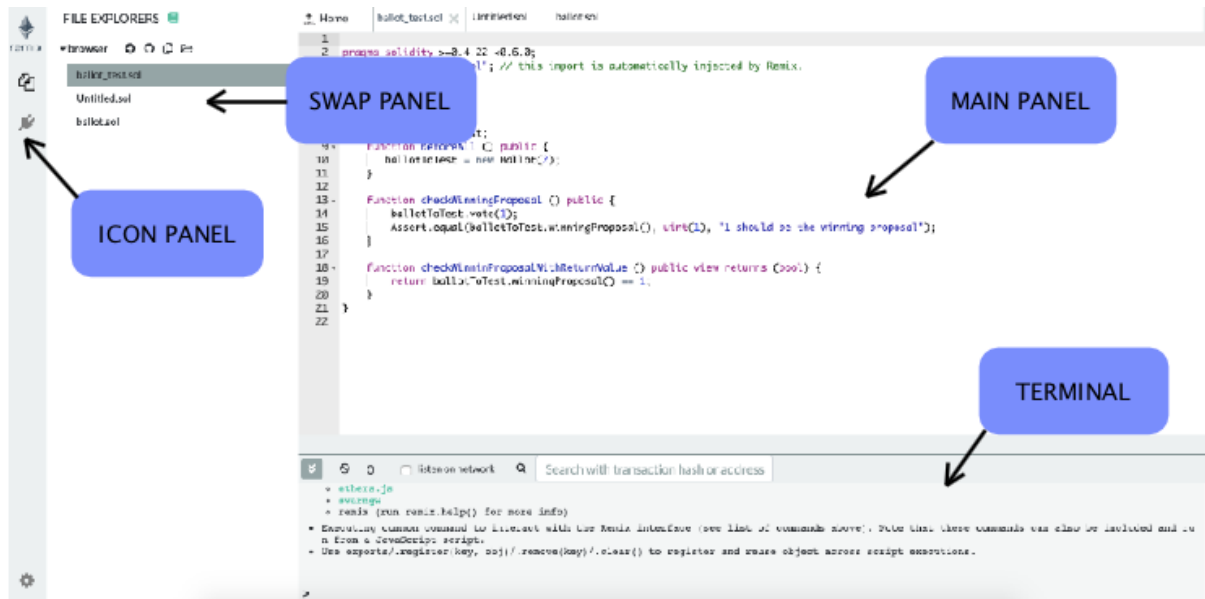
- Một tính năng khác biệt và tương đối thực tế của Remix IDE là có thể lập trình trong trình duyệt tại <https://remix.ethereum.org/>. Vâng, công cụ mã nguồn mở này giúp bạn dễ dàng lập trình các Smart Contract Solidity ngay từ trình duyệt. Tuy nhiên, cũng có một phiên bản dành cho máy tính để bàn nếu bạn muốn chạy nó cục bộ.

- Theo cấu trúc của hầu hết các ngôn ngữ lập trình phổ biến, Remix IDE có một số mô-đun. Ba mô-đun phổ biến nhất bao gồm: mô-đun để testing, debugging và deploy các Smart Contract. Hơn nữa, Remix cũng cung cấp một số thư viện để giúp phát triển nhanh hơn

- Bộ cục Remix IDE: với bản cập nhật gần nhất, Remix IDE có 4 cục gồm 4 phần chính:

- **Icon panel:** Một khu vực mà bạn nhấp để thay đổi plugin nào xuất hiện trong side panel (bảng điều khiển bên).
- **Swap panel:** Một khu vực mà hầu hết (nhưng không phải tất cả) plugin sẽ hiển thị GUI của chúng.
- **Main panel:** nó hiện cung cấp các tab nơi có thể biên dịch các plugin hoặc tệp cho IDE.
- **Terminal:** khu vực nơi bạn sẽ thấy kết quả tương tác của mình với GUI.

Bạn cũng có thể chạy các tập lệnh của mình tại đây.



- Nói chi tiết hơn:

- Ở bên trái, chúng ta có **Icon Panel**. Trong đó, bạn có thể chọn một plugin mà bạn muốn xem trong **Swap Panel**. Dưới đây là một số plugin phổ biến nhất:

Tên	Mục đích
Compiler	Biên dịch các Contract
Run & deploy	Gửi đi 1 giao dịch
Debugging	Gỡ lỗi 1 giao dịch
Analysis	Trình bày dữ liệu của phần tổng hợp cuối cùng

- **Control Panel** cho phép bạn xem và chỉnh sửa tệp trong nhiều tab. Nó làm nổi bật các từ khóa Solidity, giúp bạn dễ dàng nắm bắt cú pháp hơn. Bảng điều khiển sẽ biên dịch lại mã sau mỗi lần thay đổi và lưu nó vài giây sau khi lần cuối cùng hoàn thành.

- Bạn có thể xem kết quả và chạy mã của mình trong **Terminal** . Nó hiển thị tất cả các hoạt động và giao dịch quan trọng được khai thác hiện tại. Bạn cũng có thể tìm kiếm dữ liệu cụ thể và nhật ký rõ ràng trong **Terminal**.

2.2.4 Các Remix IDE mô-đun

- Một vài các IDE mô-đun mặc định:

- File Explorers Module.
- Plugin Manager.
- Settings.
- Solidity Editor.
- Terminal.

- Các IDE mô-đun điển hình:

- Compiler (Solidity).
- Deploy & Run.
- Debugger.
- Solidity Static Analysis.

2.2.4 Các thư viện Remix

- **Remix Analyzer** – Nó giúp bạn thực hiện phân tích tĩnh trên các hợp đồng thông minh Solidity để kiểm tra các lỗ hổng bảo mật và các hoạt động phát triển xấu.

- **Remix ASTWalker** – Cung cấp một cách dễ dàng để đọc AST của một hợp đồng thông minh được viết bằng Solidity.

- **Remix Debug** – Cung cấp cho bạn tất cả các công cụ cơ bản để thêm các tính năng gỡ lỗi cho các hợp đồng thông minh.

- **Remix Solidity** – Chỉ đơn giản là một trình bao bọc xung quanh trình biên dịch Solidity.

- **Remix Lib** – Một điều phổ biến cho các thư viện được sử dụng trên nhiều mô-đun.

- **Remix Tests** – Cho phép bạn thêm thử nghiệm đơn vị Solidity vào tích hợp liên tục hoặc các công cụ của bạn.

- **Remix URL Resolvers** – Cung cấp người trợ giúp để giải quyết nội dung từ các

URL bên ngoài.

2.3 So sánh 2 công cụ hỗ trợ lập trình smart contract (Remix & Hardhat)

Remix	Hardhat
IDE chạy trên browser	Framework JS sử dụng npm để install và code ở local
Không cần install thư viện openzeppelin để implement ERC721	Install thư viện openzeppelin để implement ERC721
Được cung cấp các provider ảo để làm ví deploy SC ở memory ảo của IDE	Được cung cấp các provider ảo để làm ví deploy SC nhưng chỉ hoạt động local
Khi deploy testnet hoặc mainnet có thể sử dụng ví metamask để deploy SC và metamask sẽ tự động mở trình duyệt để mình chấp nhận	Khi deploy testnet hay mainnet phải cung cấp private key của ví để chạy SC
Vì là IDE nên các lệnh đều có được giao diện để compile, test, deploy	Tất cả các lệnh phải qua terminal sử dụng npx
Khi deploy SC có thể chạy các hàm trên giao diện của Remix tùy lúc tùy ý	

B. Thực hành

- Cách thức thực hiện phần thực hành:

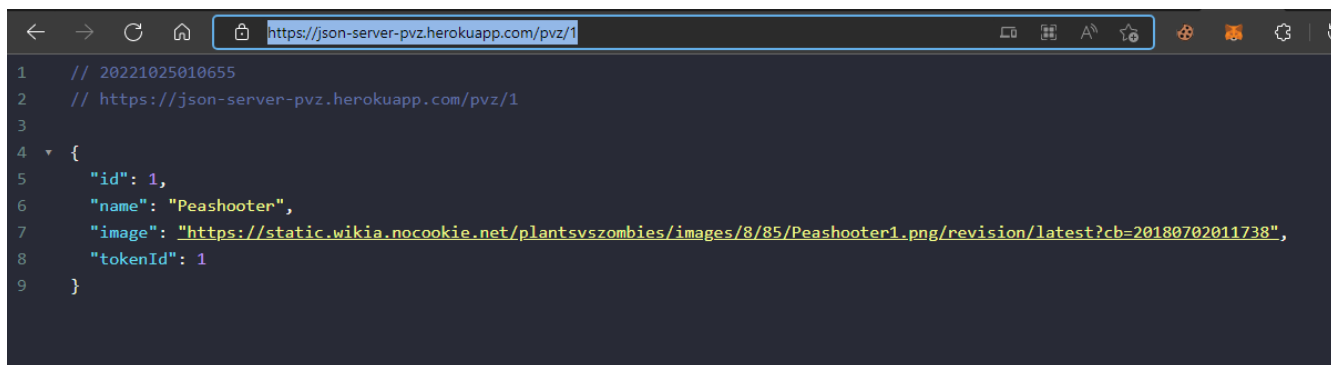
Trên các chain chạy EVM ngày nay ngoài Native coin (ETH) và Fungible Token (ERC20) thì chúng còn có Non-Fungible Token (ERC721 và ERC1155). Native coin là coin được sinh ra trong quá trình đào của các máy, còn Fungible Token và Non-Fungible Token được sinh ra và vận hành theo source code được lập trình trong smart contract.

- Non-Fungible Token: Nếu Non-Fungible Token này implement theo chuẩn của

openseppelin thì nó cũng tương tự như Native Token, một địa chỉ ví có thể dễ dàng nhận Non-Fungible Token và nhóm em tạo NFT theo ERC721 và kế thừa smart contract ERC721 qua thư viện Openzeppelin.

Với nội dung NFT sẽ là Object “Plants Vs Zombies” qua API <https://json-server-pvz.herokuapp.com/pvz/<tokenId>>. Ví dụ: với tokenId = 1 thì Server sẽ trả về đối tượng

a. Lập trình NFT trên Remix:



```
1 // 20221025010655
2 // https://json-server-pvz.herokuapp.com/pvz/1
3
4 {
5   "id": 1,
6   "name": "Peashooter",
7   "image": "https://static.wikia.nocookie.net/plantsvszombies/images/8/85/Peashooter1.png/revision/latest?cb=20180702011738",
8   "tokenId": 1
9 }
```

Và nhóm em sẽ tạo NFT qua các Object trên API này

Đoạn code smart contract tạo ra các NFT và trao đổi NFT “PVZ.sol”



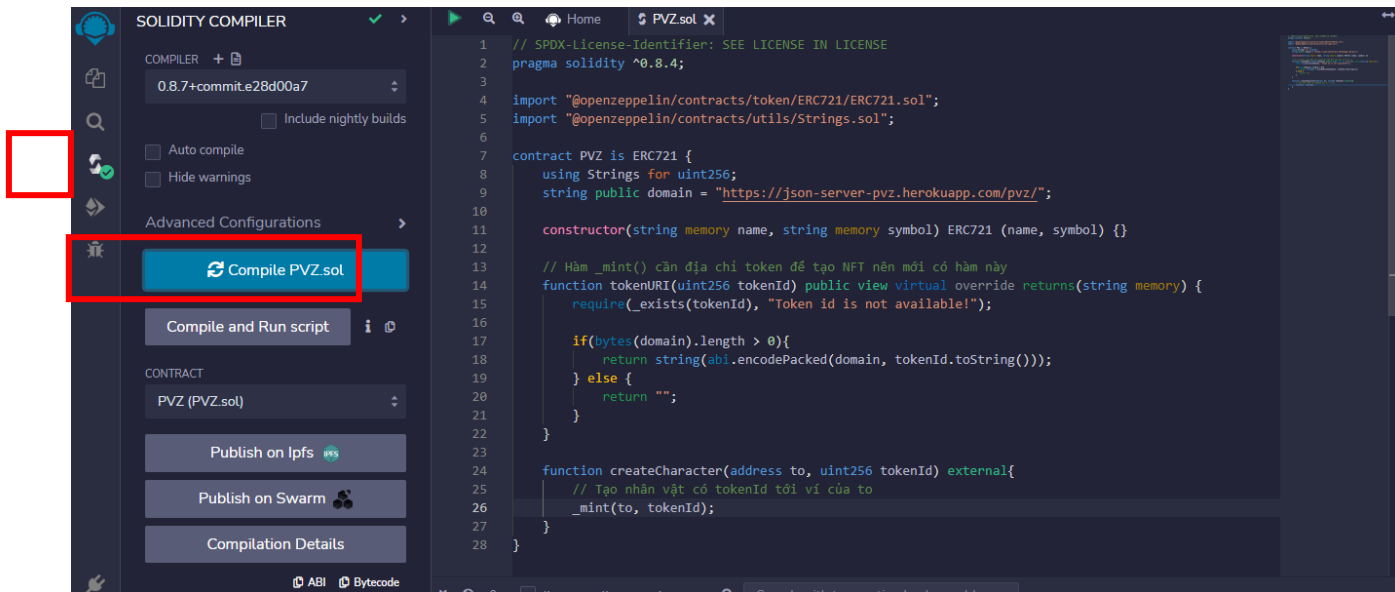
```
1 // SPDX-License-Identifier: SEE LICENSE IN LICENSE
2 pragma solidity ^0.8.4;
3
4 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5 import "@openzeppelin/contracts/utils/Strings.sol";
6
7 contract PVZ is ERC721 {
8   using Strings for uint256;
9   string public domain = "https://json-server-pvz.herokuapp.com/pvz/";
10
11   constructor(string memory name, string memory symbol) ERC721 (name, symbol) {}
12
13   // Hàm _mint() cần địa chỉ token để tạo NFT nên mới có hàm này
14   function tokenURI(uint256 tokenId) public view virtual override returns(string memory) {
15     require(_exists(tokenId), "Token id is not available!");
16
17     if(bytes(domain).length > 0){
18       return string(abi.encodePacked(domain, tokenId.toString()));
19     } else {
20       return "";
21     }
22   }
23
24   function createCharactor(address to, uint256 tokenId) external{
25     // Tạo nhân vật có tokenId tối ví của to
26     _mint(to, tokenId);
27   }
28 }
```

Contract PVZ kế thừa contract ERC721 đã lập trình và nhóm em đã tận dụng lại để lập trình NFT cho riêng mình.

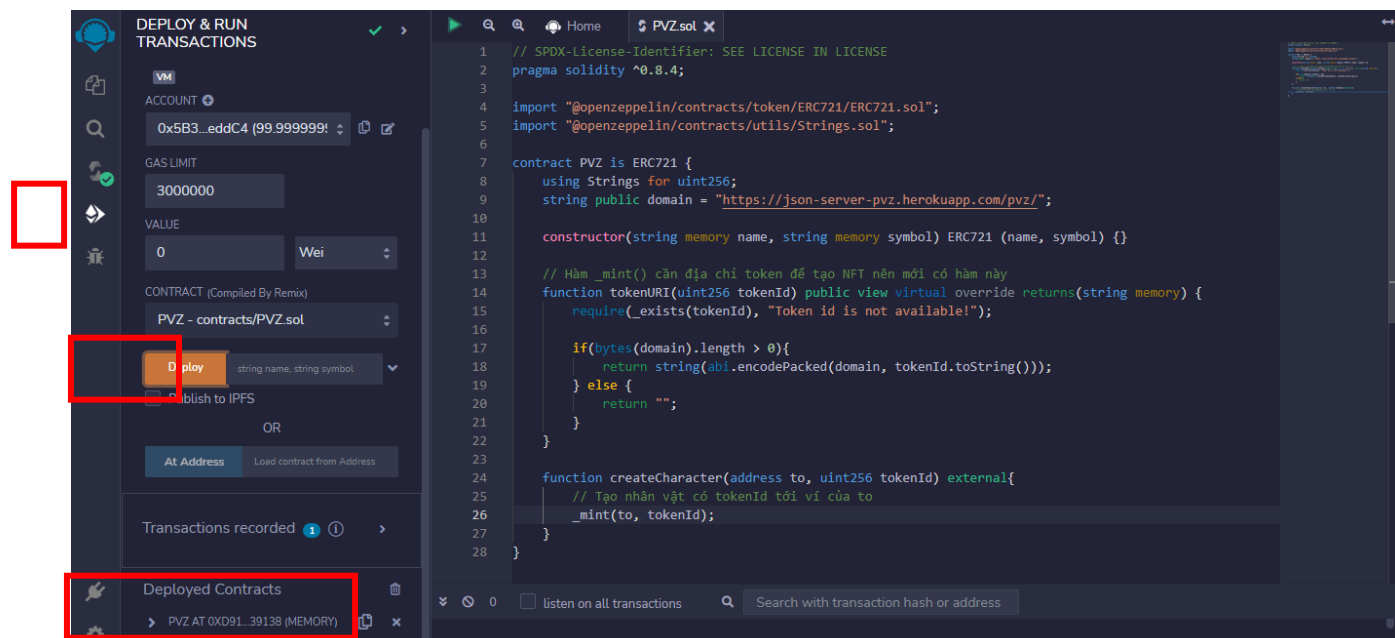
Hàm createCharacter() dùng lại hàm _mint() tạo ra NFT

Hàm tokenURI() sẽ dùng để tạo đường dẫn cho các NFT được mint ra

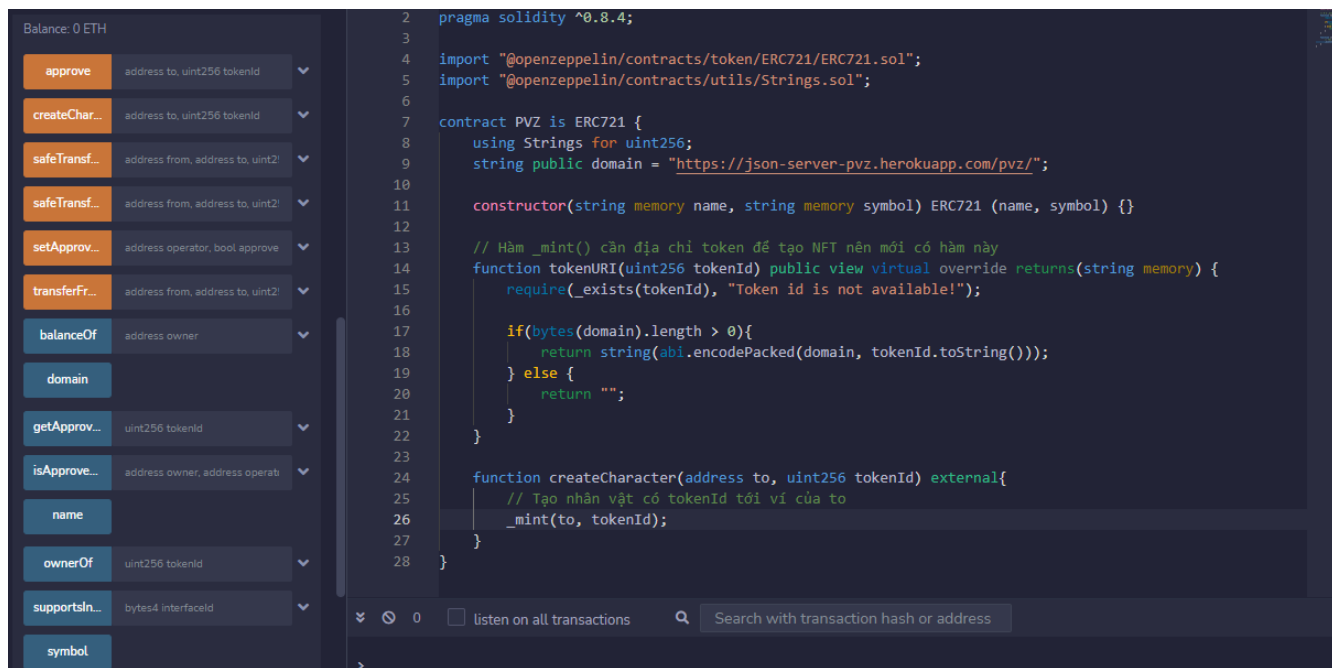
Compile trước khi Deploy contract



Deploy contract ở địa chỉ “0xd9145CCE52D386f254917e481eB44e9943F39138”

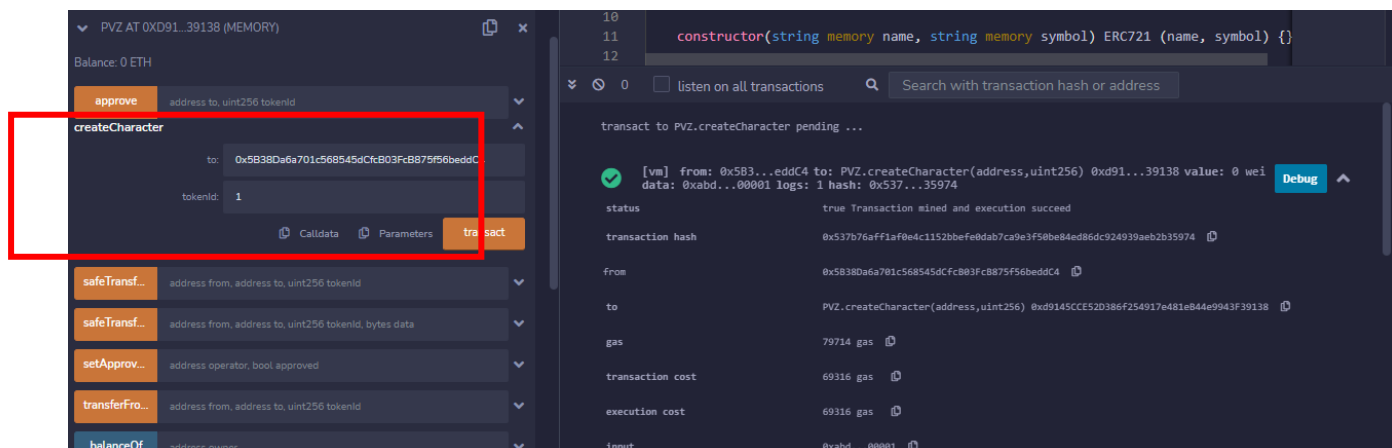


Mở rộng phần Deployed Contracts ta sẽ có các hàm

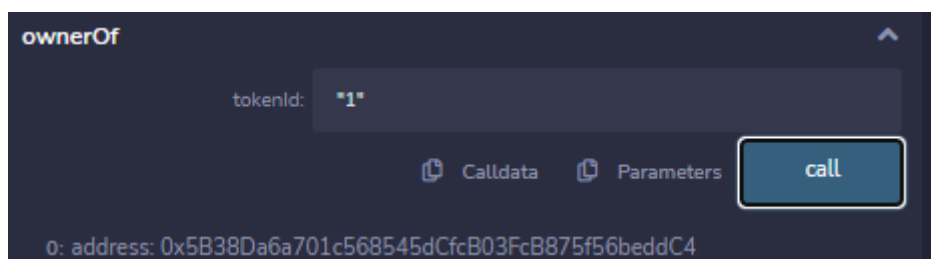


Em sẽ tạo tokenId = 1 cho address

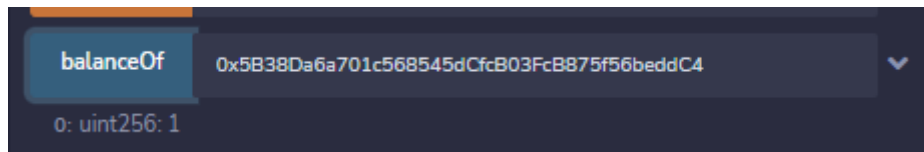
“0x5B38Da6a701c568545dCfcB03FcB875f56beddC4” qua hàm createCharacter()



Check hàm ownerOf() kiểm tra address NFT



Check hàm balanceOf() kiểm tra số lượng NFT mà address sở hữu

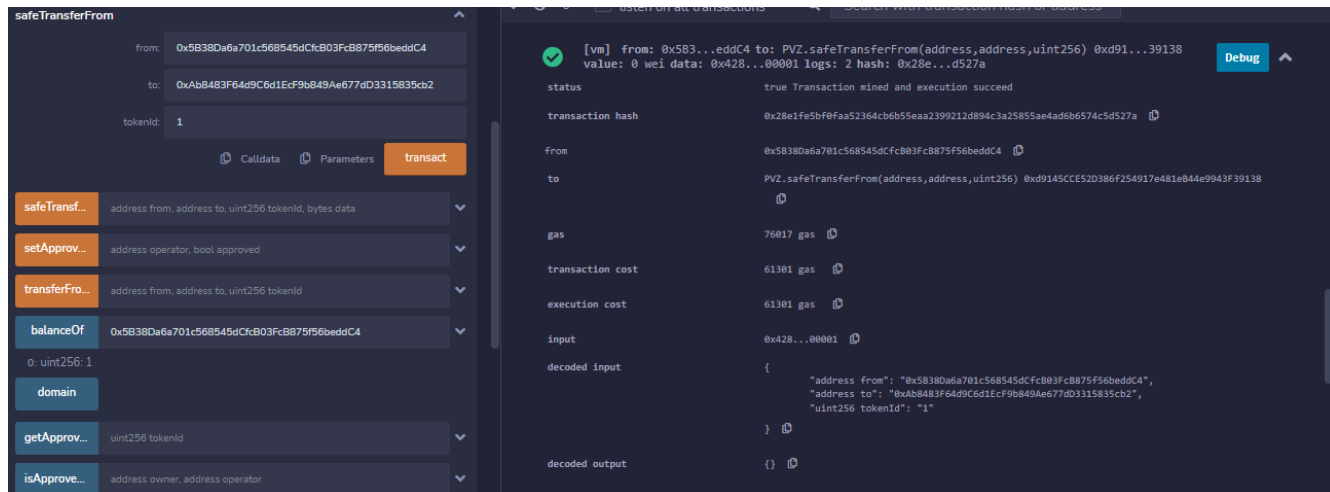


Chuyển NFT đang sở hữu sang address khác qua qua hàm `safeTransferFrom()`

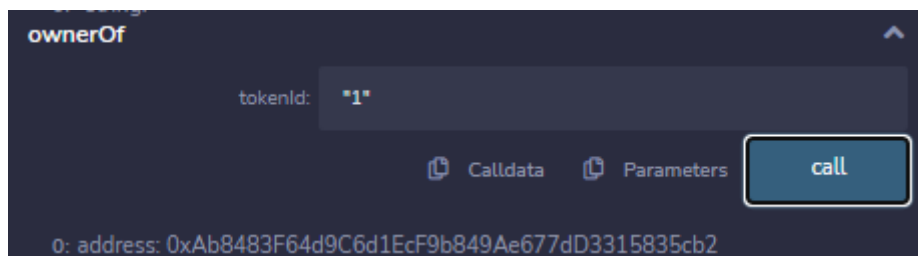
Chuyển tokenId=1 từ address1 =

“0x5B38Da6a701c568545dCfcB03FcB875f56beddC4” đến

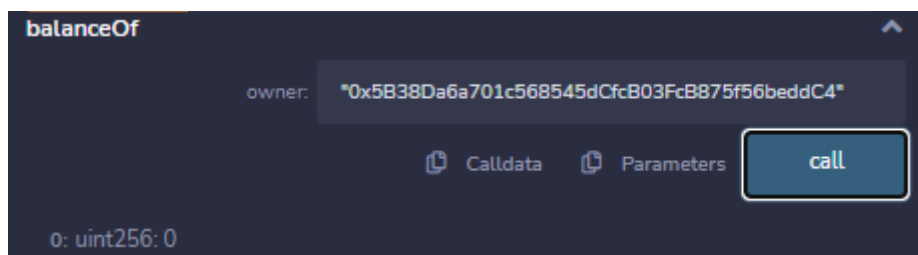
address2=“0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2”



Lúc này address2 sở hữu NFT tokenId = 1



Và số lượng NFT của address1 trở về 0



b. Lập trình NFT trên Hardhat:

Đầu tiên nhóm em khởi tạo Hardhat qua câu lệnh “`npm install --save-dev hardhat`”

Và cài đặt Hardhat “npx hardhat”

Phần code contract tương tự như ở Remix.

Để compile phần contract thì chạy câu lệnh “npx hardhat compile”

Nhóm em đã thêm các testcase để kiểm tra chạy code có như yêu cầu không.

```
21 describe("PVZ NFT", async () =>
22 {
23     let [acc01, acc02, acc03] = [];
24     let address0 = "0x0000000000000000000000000000000000000000";
25     let PVZContract;
26     let pvz
27
28     beforeEach(async () => {
29         [acc01, acc02, acc03] = await ethers.getSigners();
30         PVZContract = await ethers.getContractFactory("PVZ");
31         pvz = await PVZContract.deploy("Plants Vs Zombies", "PVZ");
32         await pvz.deployed();
33     });
34
```

Testcase 1: kiểm tra tên và symbol

```
35     it("Should return correct name and symbol", async () =>
36     {
37         expect(await pvz.name()).to.be.equal("Plants Vs Zombies");
38         expect(await pvz.symbol()).to.be.equal("PVZ");
39     });
40
```

Testcase 2: Kiểm tra tạo các NFT

```
41     it("Should mint token correctly!", async () =>
42     {
43         //console.log([acc01.address, acc02.address, acc03.address]);
44         await pvz.createCharacter(acc01.address, 1);
45         expect(await pvz.balanceOf(acc01.address)).to.be.equal(1);
46         expect(await pvz.ownerOf(1)).to.be.equal(acc01.address);
47
48         await pvz.createCharacter(acc02.address, 2);
49         expect(await pvz.balanceOf(acc02.address)).to.be.equal(1);
50         expect(await pvz.ownerOf(2)).to.be.equal(acc02.address);
51     });
52
```

Testcase 3: Chuyển token

```

53   it("Should transfer token correctly!", async () =>
54   {
55       // Tạo tokenId 1 cho acc01 và kiểm tra số lượng
56       await pvz.createCharacter(acc01.address, 1);
57       expect(await pvz.balanceOf(acc01.address)).to.be.equal(1);
58       expect(await pvz.ownerOf(1)).to.be.equal(acc01.address);
59
60       // Chuyển token từ acc01 sang acc02 và kiểm tra số lượng
61       await pvz.transfer(acc01.address, acc02.address, 1)
62       expect(await pvz.ownerOf(1)).to.be.equal(acc02.address);
63       expect(await pvz.balanceOf(acc02.address)).to.be.equal(1);
64       expect(await pvz.balanceOf(acc01.address)).to.be.equal(0);
65   });
66

```

Sau khi code xong phần testcase thì chạy lệnh “npx hardhat test” và được kết quả

```

PS D:\Learning\Nam-4\Sem-1\Blockchain\nft-demo-2> npx hardhat test

PVZ NFT
✓ Should return correct name and symbol (57ms)
✓ Should mint token correctly! (134ms)
✓ Should transfer token correctly! (140ms)

3 passing (3s)

```

Để deploy Contract thì dùng câu lệnh “npx hardhat run”.

III. Tài liệu tham khảo

- [1] <https://moralis.io/hardhat-explained-what-is-hardhat/>
- [2] <https://viblo.asia/p/quen-truffle-di-tu-nay-chung-ta-da-co-hardhat-yMnKM2BaZ7P>
- [3] <https://theblockchainguy.dev/hardhat-vs-truffle-vs-remix>
- [4] <https://moralis.io/remix-explained-what-is-remix/>
- [5] <https://hardhat.org/hardhat-runner/docs/getting-started#overview>
- [6] <https://viblo.asia/p/lam-the-nao-de-mot-smart-contract-co-the-nhan-nft-non-fungible-token-E375z6LW5GW>