Contents

[1. GANACHE 2](#_Toc208227437)

[2. REMIX 4](#_Toc208227438)

[3. Truffle console bị lỗi 7](#_Toc208227439)

[4. Thư viện OpenZeppelin 9](#_Toc208227440)

[5. Global installation nghĩa là gì? 9](#_Toc208227441)

[6. Lỗi cài Windows 11 Enterprise trên máy ảo VB 10](#_Toc208227442)

[7. METAMASK 12](#_Toc208227443)

[1. Ví và tài khoản trong MetaMask 12](#_Toc208227444)

[2. Cách tạo thêm tài khoản trong cùng ví MetaMask 13](#_Toc208227445)

[3. Cách 1: tạo ví MetaMask hoàn toàn mới (với seed phrase riêng) 13](#_Toc208227446)

[4. Cách 2: Tạo ví MetaMask mới bằng profile Chrome khác 13](#_Toc208227447)

[5. Mẹo quản lý nhiều ví MetaMask 16](#_Toc208227448)

[8. Tạo ERC20 Token contract 16](#_Toc208227449)

[Code tạo mới (Solidity ^0.8.20, OpenZeppelin mới): 17](#_Toc208227450)

[Khắc phục lỗi Opcode trong MoolahCoin.sol: 18](#_Toc208227451)

[9. KẾT QUẢ MIGRATE TRÊN GANACHE 19](#_Toc208227452)

[10. ABI LÀ GÌ? 22](#_Toc208227453)

[11. MỘT SỐ LỆNH DEBUG TRONG TRUFFLE 23](#_Toc208227454)

[12. LỖI KHÔNG HIỂN THỊ CONTRACT DEPLOYED IN TAB CONTRACTS GANACHE 23](#_Toc208227455)

[13. CREATING THE TOKEN WALLET FRONTEND USING REACTJS 24](#_Toc208227456)

# 1. GANACHE

* Mnemonic của Ganache chỉ dùng **tạm thời để phát triển và kiểm thử** ứng dụng blockchain trên mạng giả lập.
* **Không dùng nó để chứa tiền thật hoặc triển khai lên mạng thật.**
* Nếu cần ví thật để quản lý tài sản, hãy tạo ví mới bằng các ứng dụng uy tín như **MetaMask**, **Ledger**, hoặc **Trust Wallet** và **bảo mật kỹ mnemonic đó.**

Từ mnemonic này, Ganache dùng chuẩn **HD Wallet (Hierarchical Deterministic Wallet)** để sinh ra nhiều địa chỉ ví.

Các địa chỉ này được sinh ra tuần tự từ cùng một "cây" ví.

- Mỗi địa chỉ trong danh sách Ganache gắn với một **chỉ số (index)**.

- Index thể hiện thứ tự ví được sinh ra từ mnemonic.

- Đường dẫn (HD Path) Ganache hiển thị: **m/44'/60'/0'/0/account\_index**

Trong đó: account\_index chính là **Index** ta thấy (0, 1, 2, …) ở Ganache

**THANH MENU TRÊN CÙNG**

|  |  |
| --- | --- |
| **Biểu tượng / Tên** | **Mô tả** |
| **ACCOUNTS** | Hiển thị các ví Ethereum được sinh ra từ mnemonic. Bao gồm địa chỉ, số dư ETH, số giao dịch. |
| **BLOCKS** | Hiển thị chi tiết về các block đã được tạo (số block, gas, miner...). |
| **TRANSACTIONS** | Danh sách tất cả các giao dịch (tx) đã được thực hiện trong mạng Ganache. |
| **CONTRACTS** | Hiển thị các smart contract đã được triển khai trên mạng Ganache. |
| **EVENTS** | Ghi nhận các sự kiện (events) được phát ra từ smart contract. |
| **LOGS** | Ghi lại log hệ thống (log debug, gọi hàm, lỗi v.v.). |
| **SEARCH BAR** | Tìm kiếm theo số block, địa chỉ, hoặc mã giao dịch (tx hash). |

**THÔNG TIN MẠNG PHÍA TRÊN TRUNG TÂM**

|  |  |
| --- | --- |
| **Thông số** | **Ý nghĩa** |
| **GAS PRICE** | Giá gas mặc định khi thực hiện giao dịch (ở đây là 20000000000 wei). |
| **GAS LIMIT** | Giới hạn gas cho mỗi block. |
| **HARD FORK** | Cho biết version mạng Ethereum đang mô phỏng, ở đây là **MERGE**. |
| **NETWORK ID** | ID mạng giả lập (ở đây là 5777). Dùng để cấu hình kết nối từ MetaMask hoặc app khác. |
| **RPC SERVER** | Endpoint RPC của mạng (ở đây là http://127.0.0.1:7545) để kết nối bằng Web3, Ethers.js, Truffle,... |
| **MINING STATUS** | Trạng thái khai thác block: "AUTOMINING" nghĩa là block được tạo ngay khi có giao dịch. |

Trong Ethereum, **wei** là **đơn vị nhỏ nhất** của đồng tiền **Ether (ETH)**.

* 1 ETH = 1018 wei.
* Nói cách khác: **wei giống như “đồng xu lẻ” của ETH**.

1 ETH = 1,000,000,000,000,000,000 wei (một triệu tỷ tỷ wei).

 1 gwei (giga-wei) = 109 wei.

 Gas price thường hiển thị bằng **gwei** để dễ đọc hơn.

**GAS PRICE 20000000000 wei =** 209 wei = 20 gwei

**PHẦN ACCOUNTS (DANH SÁCH VÍ)**

**Thông tin mỗi ví:**

|  |  |
| --- | --- |
| **Trường** | **Giải thích** |
| **ADDRESS** | Địa chỉ ví Ethereum (bắt đầu bằng 0x...). |
| **BALANCE** | Số dư ETH mặc định (100 ETH cho mỗi ví từ Ganache). |
| **TX COUNT** | Số lượng giao dịch đã thực hiện từ ví đó. |
| **INDEX** | Thứ tự ví được sinh ra từ mnemonic. |
| 🔑 **(biểu tượng chìa khóa)** | Nhấn vào sẽ hiển thị **private key** của địa chỉ (để dùng ngoài như MetaMask, Remix...). |

**MNEMONIC & HD PATH**

* **MNEMONIC**: Cụm từ 12 từ để sinh ra tất cả ví.
* **HD PATH**: Đường dẫn chuẩn BIP-44 dùng để sinh ví con. Mặc định của Ethereum: m/44'/60'/0'/0/account\_index

**NÚT SAVE / SWITCH**

|  |  |
| --- | --- |
| **Nút** | **Chức năng** |
| **SAVE** | Lưu workspace hiện tại (bao gồm trạng thái ví, contract, block). |
| **SWITCH** | Chuyển sang workspace khác (nếu bạn có nhiều dự án Ganache). |

# 2. ****REMIX****

**Remix** là một **trình IDE (Integrated Development Environment) trực tuyến dành cho phát triển smart contract bằng Solidity** – ngôn ngữ lập trình chính trên Ethereum và các mạng tương thích EVM.

**Remix là gì?**

|  |  |
| --- | --- |
| **Thuộc tính** | **Mô tả** |
| T**ên đầy đủ** | Remix IDE |
| **Truy cập** | [https://remix.ethereum.org](https://remix.ethereum.org/) (không cần cài đặt) |
| **Ngôn ngữ** | Solidity |
| **Dùng để** | Viết, biên dịch (compile), triển khai (deploy), kiểm thử (test), và gỡ lỗi (debug) smart contract |
| **Tích hợp** | Web3, MetaMask, IPFS, GitHub, Truffle (Remixd) |

**Tính năng chính của Remix**

|  |  |
| --- | --- |
| **Tính năng** | **Mô tả** |
| **Code Editor** | Soạn thảo mã Solidity trực tiếp trên trình duyệt |
| **Solidity Compiler** | Biên dịch hợp đồng thông minh với nhiều version Solidity |
| **Deploy & Run** | Triển khai và chạy contract trên mạng **ảo (JavaScript VM)** hoặc **thật (MetaMask, Ganache, testnet)** |
| **Debugger** | Gỡ lỗi giao dịch để kiểm tra logic, gas, lỗi... |
| **File Explorer** | Tạo, lưu trữ và quản lý nhiều file Solidity |
| **Console + Logs** | Kiểm tra log, kết quả và lỗi khi chạy contract |
| **Plugins** | Hỗ trợ thêm các plugin như Solidity Unit Testing, Gas Profiler, Slither,... |

**Remix dùng để làm gì?**

* Học và thử nghiệm Solidity
* Phát triển smart contract nhanh chóng (không cần cấu hình máy)
* Triển khai contract lên testnet/mainnet qua MetaMask
* Kiểm thử logic contract (unit test)
* Phân tích gas và hiệu suất mã

**Ưu điểm**

* Không cần cài phần mềm
* Dễ dùng cho người mới học Solidity
* Hỗ trợ đầy đủ vòng đời smart contract
* Có giao diện đồ họa trực quan (GUI)

**Nhược điểm**

* Không phù hợp cho dự án lớn, nhiều file
* Không mạnh bằng Hardhat/Truffle khi cần test phức tạp hoặc tích hợp CI/CD

**Remix phù hợp cho ai?**

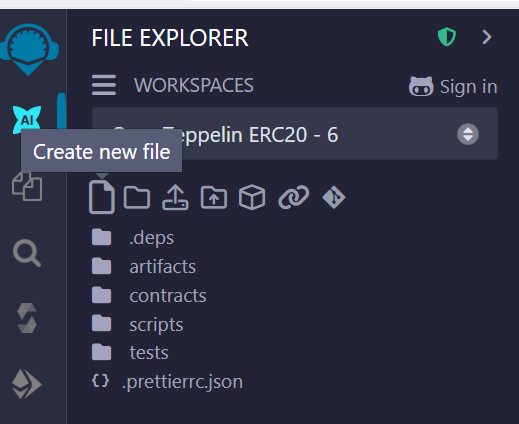
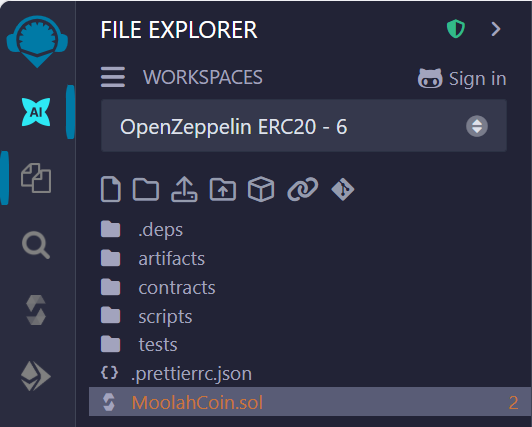
* Sinh viên, người học blockchain
* Lập trình viên thử nghiệm hợp đồng nhỏ
* Startup muốn demo hợp đồng nhanh
* Ai muốn **viết → compile → deploy** smart contract trong vòng 5 phút

**Creating the ERC20 Token contract trên remix và compile**

**Bước 1: Mở Remix:** Truy cập: <https://remix.ethereum.org>

**Bước 2: Tạo file mới**

1. Click vào biểu tượng **File Explorer** 📁 bên trái.
2. Click nút **"New File"** ➕
3. Đặt tên file, ví dụ: MoolahCoin.sol

**Bước 3: Viết mã ERC20 Token**

**Bước 4: Biên dịch hợp đồng**

1. Click vào tab **Solidity Compiler** (thanh công cụ bên trái).
2. Đảm bảo chọn **Version** là 0.8.20 (hoặc tương thích).
3. Click **Compile** MoolahCoin.sol

**Code** MoolahCoin.sol(7 chữ số thập phân)

1. // SPDX-License-Identifier: MIT
2. pragma solidity ^0.8.20;
3. // Importing required OpenZeppelin contracts
4. import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.9.3/contracts/token/ERC20/extensions/ERC20Capped.sol";
5. import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.9.3/contracts/access/Ownable.sol";
6. contract MoolahCoin is ERC20Capped, Ownable {
7. // Constructor initializing token name, symbol, decimals, and cap
8. constructor()
9. ERC20("Moolah Coin", "MC")
10. ERC20Capped(10\_000\_000\_000 \* 10 \*\* 7) // Total cap with 7 decimals
11. {
12. \_mint(msg.sender, 1\_000\_000 \* 10 \*\* 7); // Mint initial supply to owner
13. }
14. }

**Code** MoolahCoin.sol (4 chữ số thập phân)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

// Importing required OpenZeppelin contracts

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.9.3/contracts/token/ERC20/extensions/ERC20Capped.sol";

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.9.3/contracts/access/Ownable.sol";

contract MoolahCoin is ERC20Capped, Ownable {

    uint8 private constant \_customDecimals = 4;

    constructor()

        ERC20("Moolah Coin", "MC")

        ERC20Capped(10\_000\_000\_000 \* 10 \*\* \_customDecimals)

    {

        // Mint initial supply to contract owner (1,000,000 tokens with 4 decimals)

        \_mint(msg.sender, 1\_000\_000 \* 10 \*\* \_customDecimals);

    }

    // Override decimals to return 4

    function decimals() public pure override returns (uint8) {

        return \_customDecimals;

    }

}

**Code** Condos.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

contract Condos is ERC721URIStorage {

    uint32 public Id;

    mapping(address => uint32) public MDTrack;

    constructor() ERC721("Condos Token", "CONDO") payable {

        Id = 0;

    }

    function createNFT(address receiver, string calldata metadata) external returns (uint32) {

        Id++;

        \_mint(receiver, Id);

        \_setTokenURI(Id, metadata);

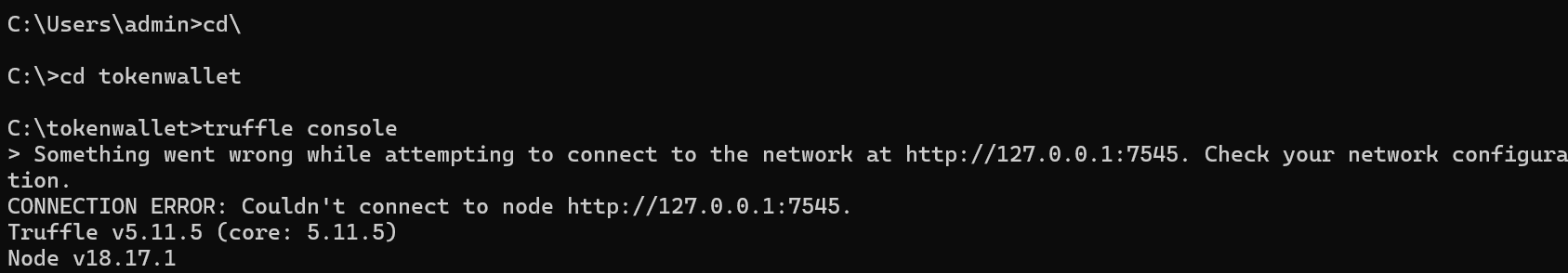
        MDTrack[receiver] = Id;

        return Id;

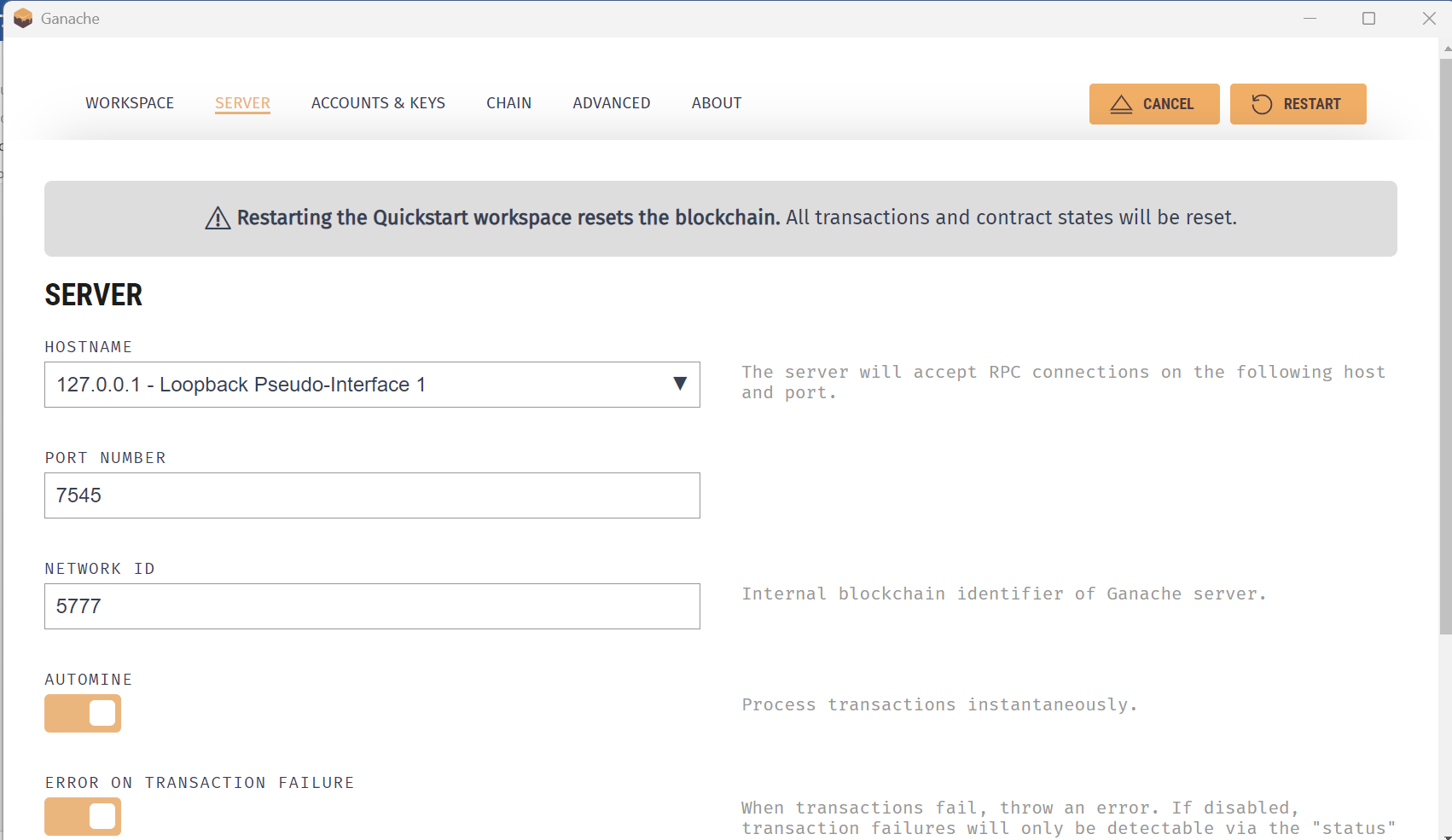
    }

}

# 3. Truffle console bị lỗi

****

Như lỗi trên là do thiết lập port kết nối ở Ganache sai. Vào Ganache -> click vào biểu tượng settings ở góc trên cùng bên phải -> chọn Server và chỉnh lại port number là 7545.



**Để thoát khỏi môi trường truffle console (hoặc truffle develop), có thể dùng một trong các cách sau:**

 truffle develop → thoát bằng Ctrl + D hoặc Ctrl + C 2 lần

 truffle console → cũng như trên

**Một số lệnh**

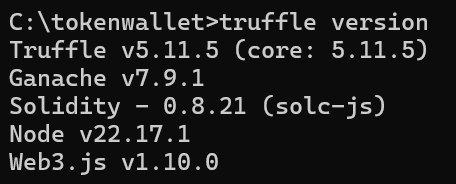
* **Dọn sạch và compile lại** để chắc chắn dùng đúng phiên bản Truffle:

truffle compile --all --reset

* **Xem Truffle version:**

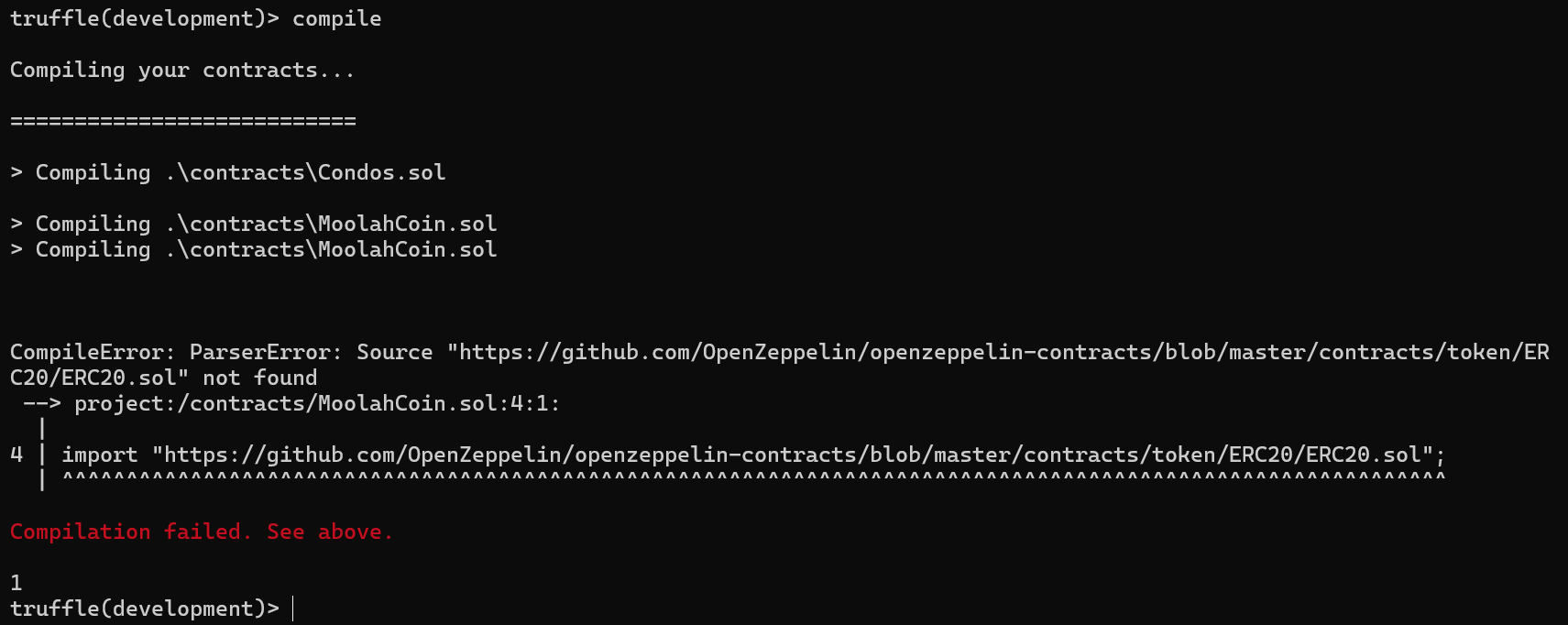
truffle version

Lưu ý thực hiện lệnh trên ở thư mục Project:



- Lỗi **cố import trực tiếp hợp đồng .sol từ GitHub bằng URL** — điều này **không được hỗ trợ bởi Truffle compiler**

Truffle không hiểu cách load file Solidity từ URL. Nó **chỉ hoạt động với file cục bộ** hoặc đã được cài bằng npm.



=> Cách khắc phục:

#### Bước 1: Cài đặt OpenZeppelin Contracts

Nếu chưa cài:

npm install @openzeppelin/contracts

#### Bước 2: Import lại đúng cách trong Solidity

Thay vì import từ GitHub, sửa lại như sau:

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

# 4. Thư viện OpenZeppelin

- Cài thư viện OpenZeppelin chuẩn qua npm: npm install @openzeppelin/contracts

# 5. Global installation nghĩa là gì?

* Khi cài **global (-g)**:
  + Package sẽ được cài vào **thư mục chung của hệ thống** (thường là C:\Users\<User>\AppData\Roaming\npm trên Windows).
  + Có thể chạy lệnh truffle ở **mọi thư mục, mọi dự án**, mà không cần cài lại cho từng project.
  + Tạo ra một “command line tool” dùng được ở bất kỳ đâu.

Ví dụ: Sau khi chạy npm install -g truffle, bạn có thể gõ:

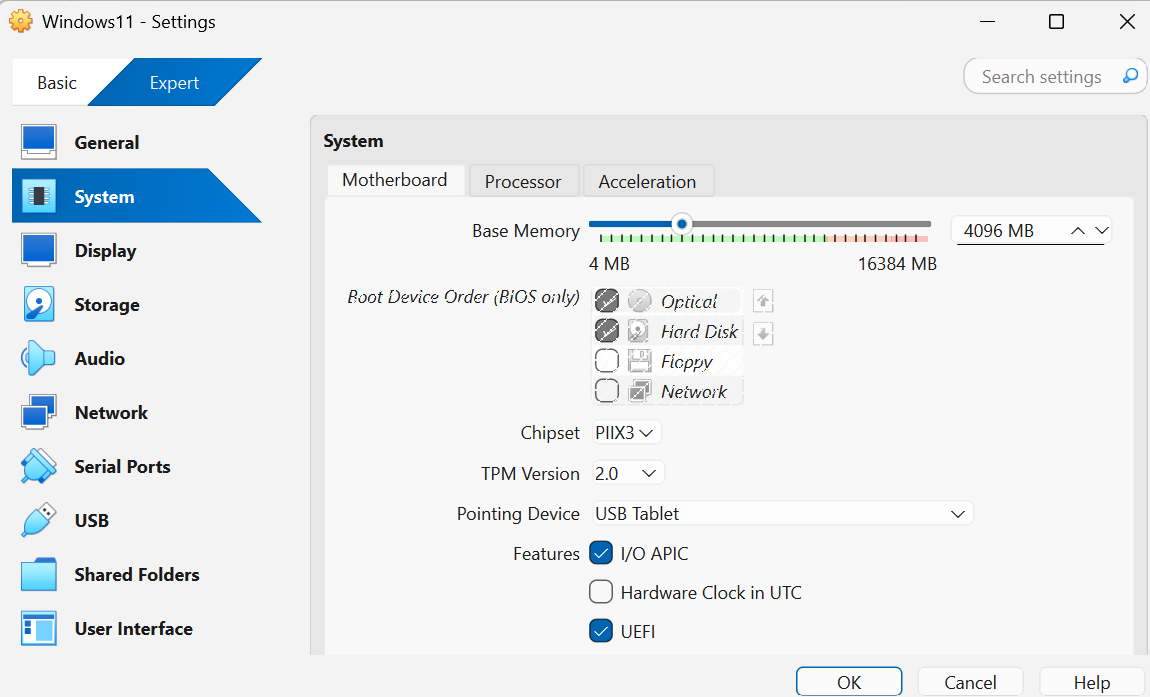
truffle version

ở bất kỳ thư mục nào và nó sẽ chạy.

# 6. Lỗi cài Windows 11 Enterprise trên máy ảo VB

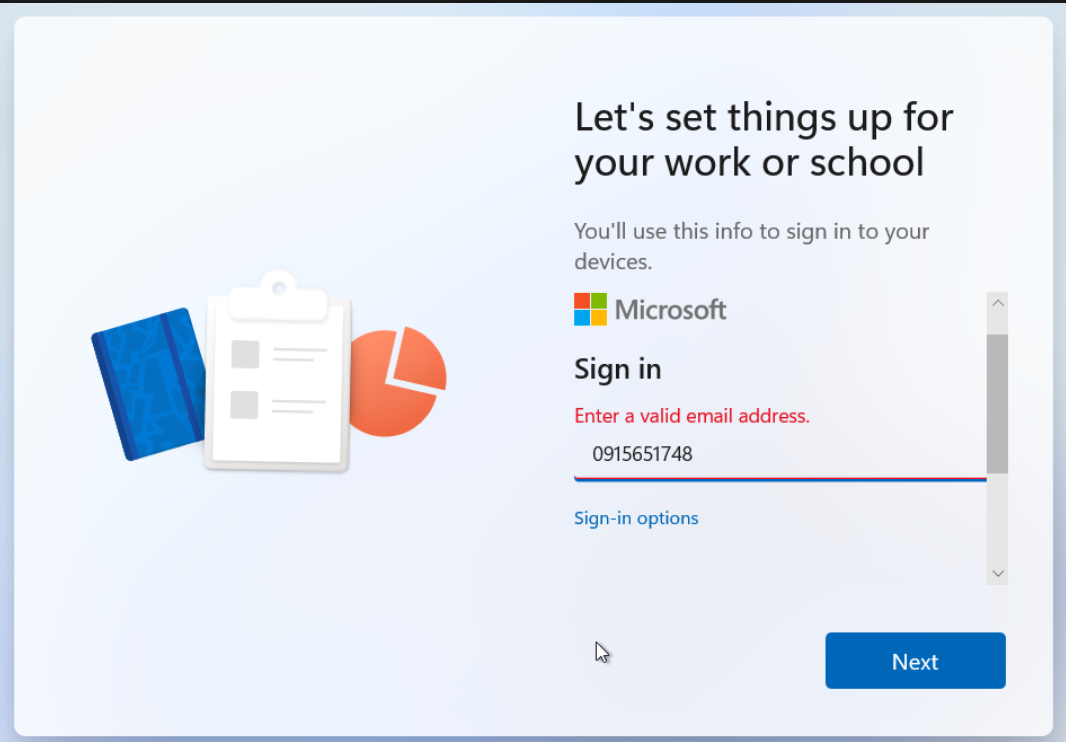
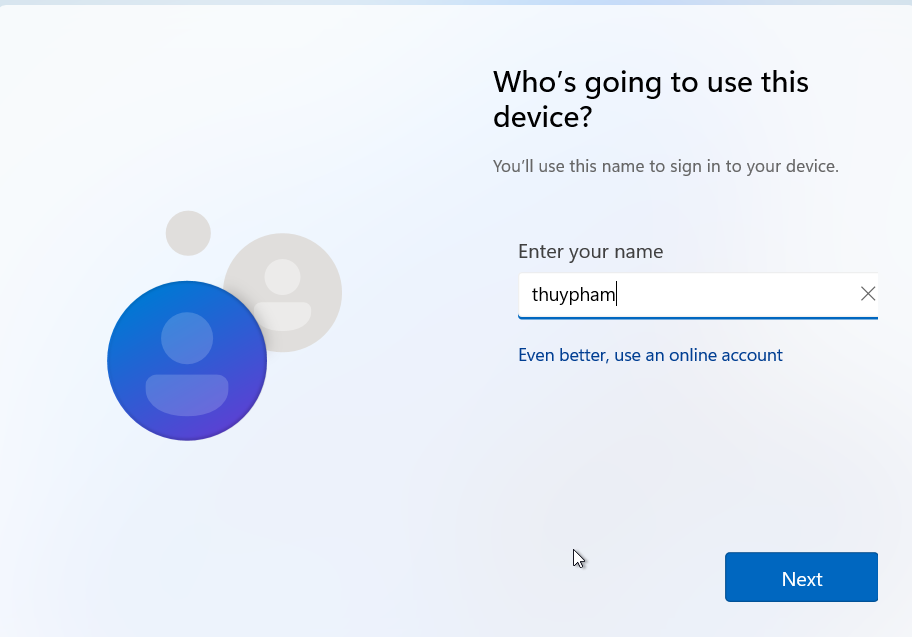
- Lưu ý phải chọn UEFI Khi cài win11

Nếu bỏ chọn thì vào set được thứ tự boot trong System -> motherboard. Nhưng sau đó lại phải chọn lại để có thể cài được win 11

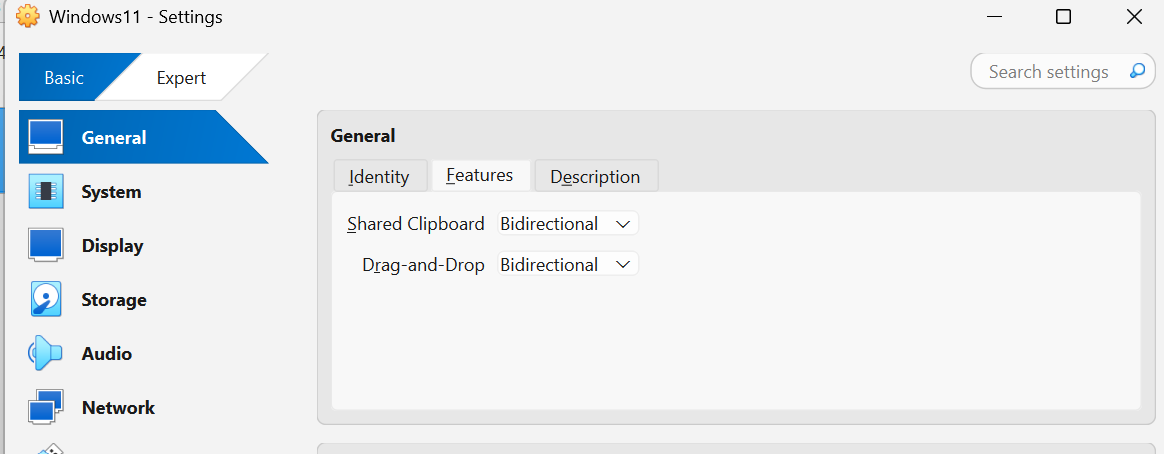


**Lưu ý:** Lúc đầu vào màn hình ấn enter để load từ CD sau đó vào màn hình cấu hình Boot để set boot device

- Khi cài windows 11 nếu yêu cầu login bằng tài khoản Microsoft thì có thể dùng tài khoản vnu. Nếu không muốn login bằng tài khoản này thì chọn **sign-in options => chọn Domain. Lúc này windows sẽ cho tạo local account để đăng nhập:**

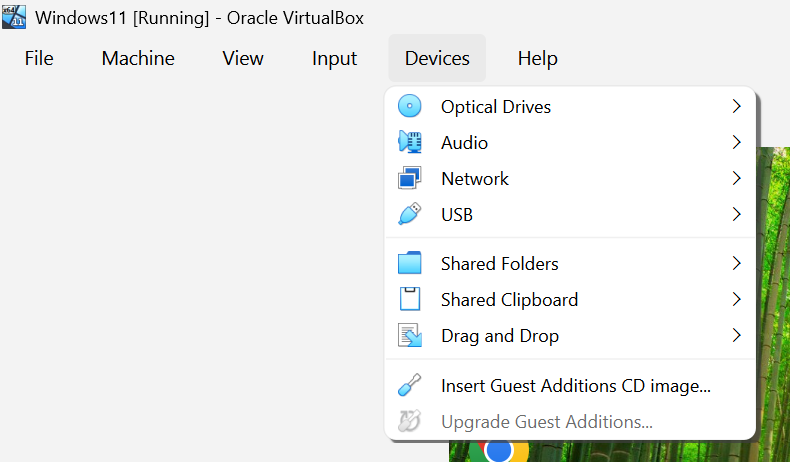
****  ****

- Để copy text từ máy ảo sang máy thật ta tắt máy ảo, vào settings và bật Bidirectional như hình sau:

****

Sau đó start máy ảo lên.

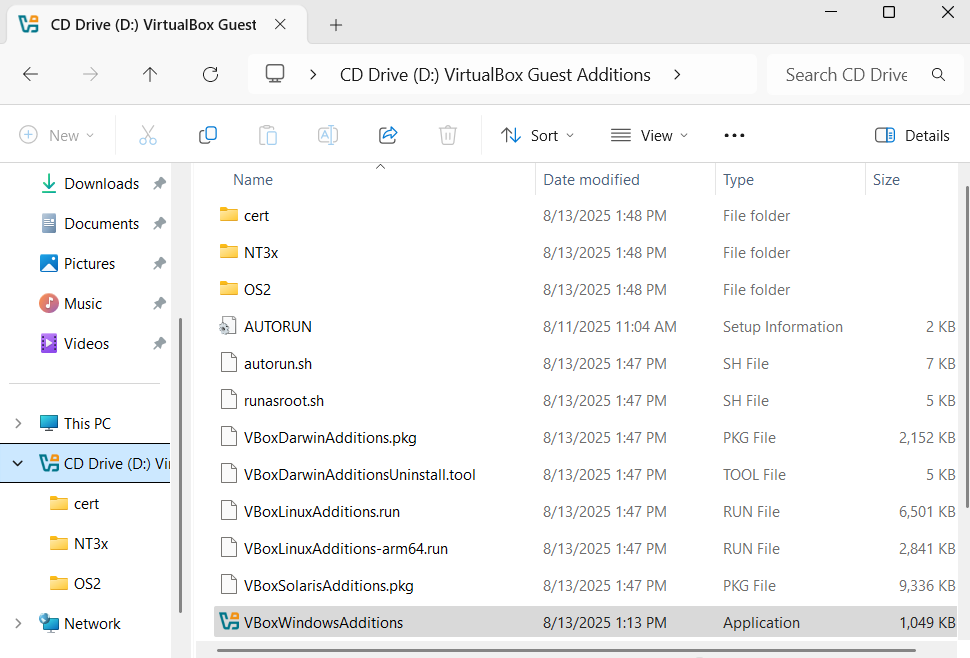
Trên menu VirtualBox → Devices → Insert Guest Additions CD Image.

****

Windows trong máy ảo sẽ nhận như một ổ CD.

Mở ổ CD trong File Explorer → chạy VBoxWindowsAdditions.exe.

Cài xong thì restart lại máy ảo.



# 7. METAMASK

## 1. Ví và tài khoản trong MetaMask

* **Ví (Wallet)**: Chính là “ngôi nhà” của bạn, được bảo vệ bằng **seed phrase** (12/24 từ khoá).
* **Tài khoản (Account)**: Là những “phòng” trong ngôi nhà đó. Mỗi tài khoản có một **địa chỉ ví riêng biệt** để nhận và gửi coin/token.

Nói cách khác:

* Bạn chỉ cần **một seed phrase** để quản lý cả “ngôi nhà”.
* Trong ngôi nhà đó, bạn có thể mở nhiều “phòng” (account), mỗi phòng là một địa chỉ ví khác nhau.

Ví dụ

* Ví MetaMask của bạn có seed phrase: apple banana ...
* Bạn tạo **Account 1**: có địa chỉ ví 0x123...
* Tạo thêm **Account 2**: có địa chỉ ví 0x456...
* Cả hai account này đều thuộc **cùng một ví**, nên chỉ cần seed phrase kia là khôi phục được hết.

Ưu điểm

* Quản lý gọn: chỉ cần nhớ 1 seed phrase.
* Có thể phân chia mục đích: ví dụ Account 1 để đầu tư, Account 2 để chơi game NFT, Account 3 để test Dapp…

Lưu ý: Vì cùng seed phrase, nếu ai đó có seed phrase này thì họ sẽ truy cập được **tất cả** các account bên trong.

## 2. Cách tạo thêm tài khoản trong cùng ví MetaMask

* Mở **extension MetaMask** trên Chrome.
* Đăng nhập bằng mật khẩu.
* Ở góc trên bên phải, nhấn vào **hình tròn avatar**.
* Chọn **Tạo tài khoản (Create Account)**.
* Đặt tên tài khoản → bấm **Tạo (Create)**.  
  Vậy là bạn đã có thêm một địa chỉ ví trong cùng seed phrase.

## 3. Cách 1: tạo ví MetaMask hoàn toàn mới (với seed phrase riêng)

* Cũng trong phần **avatar góc phải**, chọn **Cài đặt (Settings)** → **Cài đặt nâng cao (Advanced)**.
* Chọn **Tạo hồ sơ mới (Create new wallet)** hoặc đăng xuất, rồi chọn **Tạo ví mới (Create a new wallet)** khi mở lại extension.
* Lưu **seed phrase mới** thật kỹ (không chia sẻ cho ai).
* Sau khi tạo, bạn có thể chuyển đổi qua lại giữa các ví ngay trên Chrome.

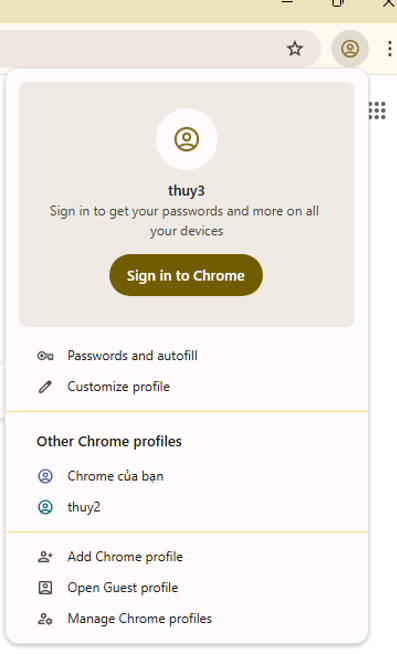
Nếu muốn tách biệt hoàn toàn, bạn còn có thể tạo thêm **profile Chrome khác**, mỗi profile cài riêng một MetaMask → hoàn toàn độc lập.

## 4. Cách 2: Tạo ví MetaMask mới bằng ****profile Chrome khác****

Các bước thực hiện

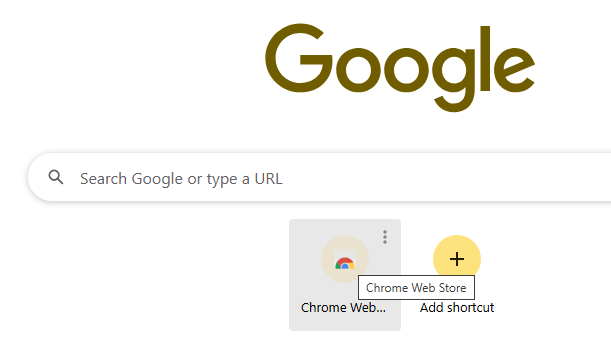
1. Tạo profile Chrome mới

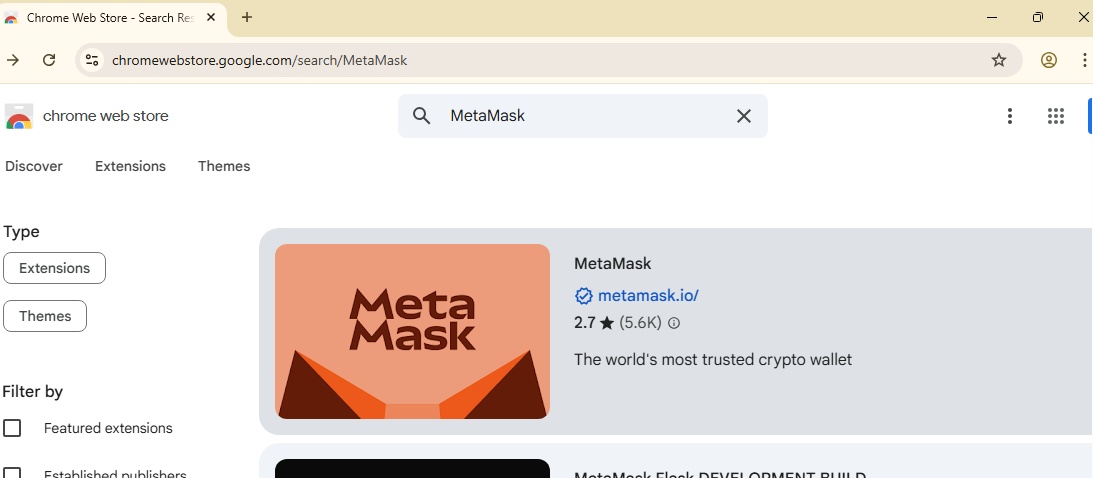
* Trên Chrome, ở góc trên cùng bên phải → nhấn vào **hình avatar (người dùng)**.
* Chọn **Thêm (Add)** hoặc **Quản lý người dùng (Manage profiles)** → **Thêm**.
* Đặt tên cho profile mới, chọn hình đại diện, rồi bấm **Hoàn tất (Done)**.  
  => Chrome sẽ mở ra một cửa sổ hoàn toàn mới, tách biệt với profile cũ.



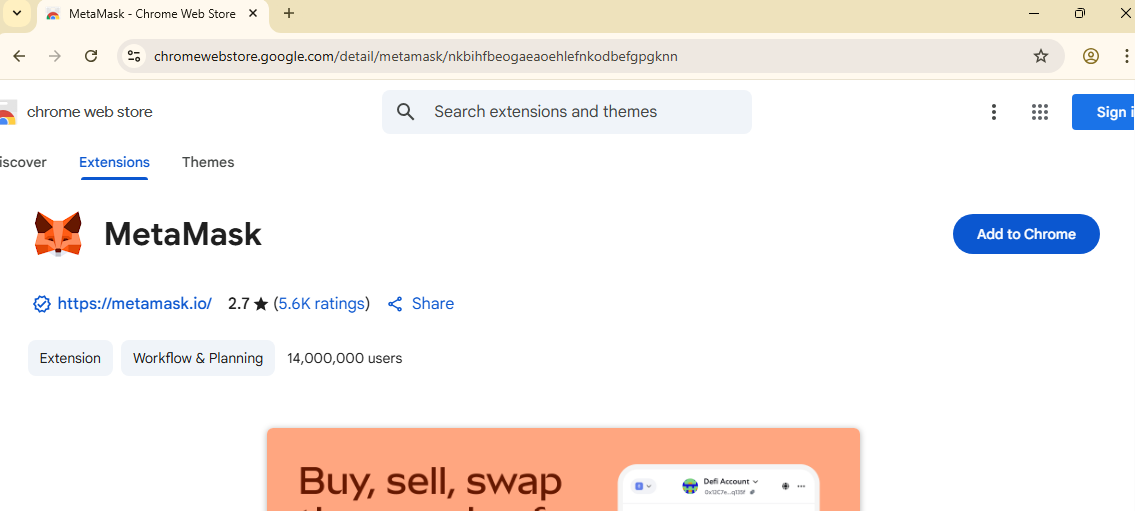
2. Cài MetaMask trong profile mới

* Vào **Chrome Web Store (Web Store)** → tìm **MetaMask**.





* Nhấn **Thêm vào Chrome (Add to Chrome)** → **Thêm tiện ích (Add extension)**.

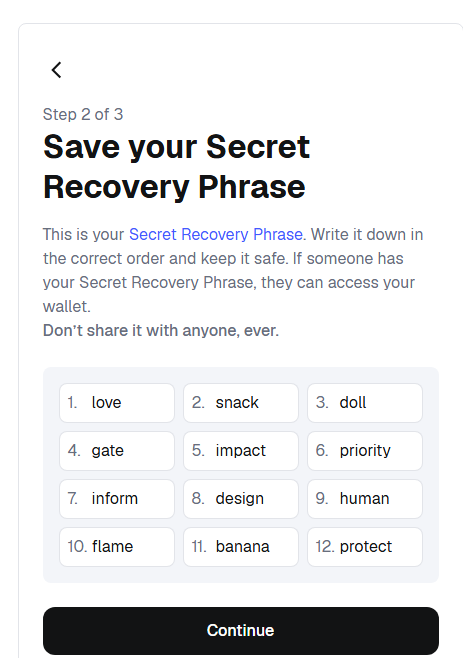
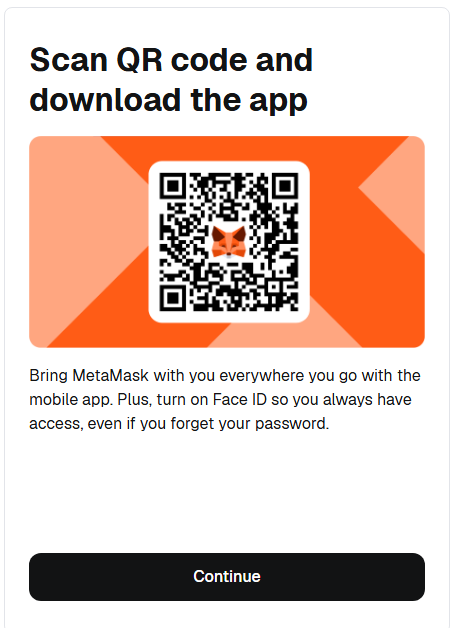


* Sau khi cài xong, biểu tượng MetaMask sẽ xuất hiện ở góc trên bên phải.

3. Tạo ví mới

* Nhấn vào biểu tượng MetaMask.
* Chọn **Tạo ví mới (Create a new wallet)**.
* Đặt mật khẩu mới → xác nhận.
* MetaMask sẽ hiển thị **Seed phrase (12/24 từ)** → bạn **phải lưu giữ an toàn** (ghi ra giấy, không chụp màn hình lưu trên máy).
* Xác nhận lại seed phrase để hoàn tất.

**Tài khoản thuy3:**

Lưu ý quan trọng

* **Mỗi profile Chrome** có MetaMask riêng, không liên quan nhau.
* Khi đăng nhập bằng profile cũ, bạn sẽ dùng ví cũ; khi mở profile mới, bạn có ví mới.
* Hãy lưu seed phrase của từng ví ở chỗ an toàn, không để lẫn lộn.

Tuyệt vời 😃 Mình chia sẻ mẹo quản lý nhiều ví MetaMask bằng cách đặt tên **profile Chrome** và **tài khoản** sao cho dễ nhớ, tránh nhầm lẫn:

## 5. Mẹo quản lý nhiều ví MetaMask

1. Đặt tên profile Chrome theo mục đích

* Khi tạo profile mới, bạn có thể đặt tên ngay lúc khởi tạo.
* Ví dụ:
  + **MetaMask – Đầu tư**
  + **MetaMask – Game NFT**
  + **MetaMask – Test DApp**
* Chrome sẽ hiển thị tên này khi bạn chọn profile, giúp bạn mở đúng ví cần dùng.

2. Đặt tên tài khoản trong MetaMask

* Trong MetaMask, mỗi account có thể đổi tên.
* Nhấn vào **hình avatar → Chỉnh sửa (Edit)** để đổi tên.
* Ví dụ:
  + Account 1 → “Ví chính”
  + Account 2 → “Ví staking”
  + Account 3 → “Ví airdrop”

Như vậy, trong **một ví** bạn cũng phân loại rõ ràng.

3. Lưu seed phrase & private key

* Mỗi ví (profile Chrome) có seed phrase riêng → hãy **ghi ra giấy và cất giữ ở nơi an toàn**.
* Nếu có nhiều ví, bạn có thể để trong phong bì riêng, ghi chú:
  + “Seed phrase – Ví Đầu tư”
  + “Seed phrase – Ví Game NFT”
* Tránh lưu online (Google Drive, email…), vì dễ bị hack.

4. Đồng bộ quản lý

* Nếu có nhiều máy tính, bạn có thể đăng nhập cùng một profile Google Chrome để đồng bộ extension MetaMask.
* Nhưng hãy cẩn thận: chỉ nên làm trên máy an toàn, không dùng chung với người khác.

# ****8. Tạo ERC20 Token contract****

Trong sách (Blockchain Development for Finance Projects)

Ở phần **"Creating the ERC20 Token contract"**, code mẫu cho MoolahCoin.sol được viết theo **Solidity 0.5.2** và thư viện **OpenZeppelin 2.2.0**. Cấu trúc nó như sau:

pragma solidity ^0.5.2;

import "openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol";

import "openzeppelin-solidity/contracts/token/ERC20/ERC20Capped.sol";

import "openzeppelin-solidity/contracts/ownership/Ownable.sol";

contract MoolahCoin is ERC20Detailed, ERC20Capped, Ownable {

constructor()

ERC20Detailed("Moolah Coin", "MC", 4)

ERC20Capped(10000000000)

payable public {}

}

* Dùng **ERC20Detailed** để set tên, ký hiệu, số chữ số thập phân.
* Dùng **ERC20Capped** để giới hạn tổng cung.
* Dùng **Ownable** để quản lý quyền sở hữu.
* Không có \_mint trong constructor, nghĩa là chưa phát hành token ngay khi deploy.

## Code tạo mới (Solidity ^0.8.20, OpenZeppelin mới):

|  |
| --- |
| // SPDX-License-Identifier: MIT  pragma solidity ^0.8.20;  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";  import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol";  import "@openzeppelin/contracts/access/Ownable.sol";  contract MoolahCoin is ERC20, ERC20Capped, Ownable {      constructor(address initialOwner)          ERC20("Moolah Coin", "MC")          ERC20Capped(10\_000\_000\_000 \* 10 \*\* 4)          Ownable(initialOwner)      {          \_mint(initialOwner, 1\_000\_000 \* 10 \*\* 4);      }      function decimals() public pure override returns (uint8) { return 4; }      // bắt buộc khi kết hợp ERC20 + ERC20Capped trong OZ v5      function \_update(address from, address to, uint256 value)          internal          override(ERC20, ERC20Capped)      { super.\_update(from, to, value); }  } |

## Khắc phục lỗi Opcode trong MoolahCoin.sol:

**“Merge” trong Ganache ≈ hardfork “Paris”** (execution layer). Nghĩa là Ganache của bạn đang chạy EVM trước Shanghai, nên nếu bytecode bạn compile ra dùng opcode mới (ví dụ **PUSH0** của Shanghai) thì khi deploy sẽ bị **invalid opcode**. Cách xử lý:

**1) Pin compiler về Paris (rất quan trọng)**

Sửa truffle-config.js để **ép** trình biên dịch nhắm đúng EVM Paris:

|  |
| --- |
| module.exports = {  networks: {  ganache: { host: "127.0.0.1", port: 7545, network\_id: "\*" },  },  compilers: {  solc: {  version: "0.8.20",  settings: {  evmVersion: "paris", // ép về Paris (Merge)  optimizer: { enabled: true, runs: 200 }  }  }  }  }; |

Nếu không đặt evmVersion, một số thiết lập có thể build theo **Shanghai** → Ganache (Merge/Paris) chạy không nổi và ném “invalid opcode”.

**2) Dùng đúng code cho OpenZeppelin v5 (giống như code trên MoolahCoin.sol)**

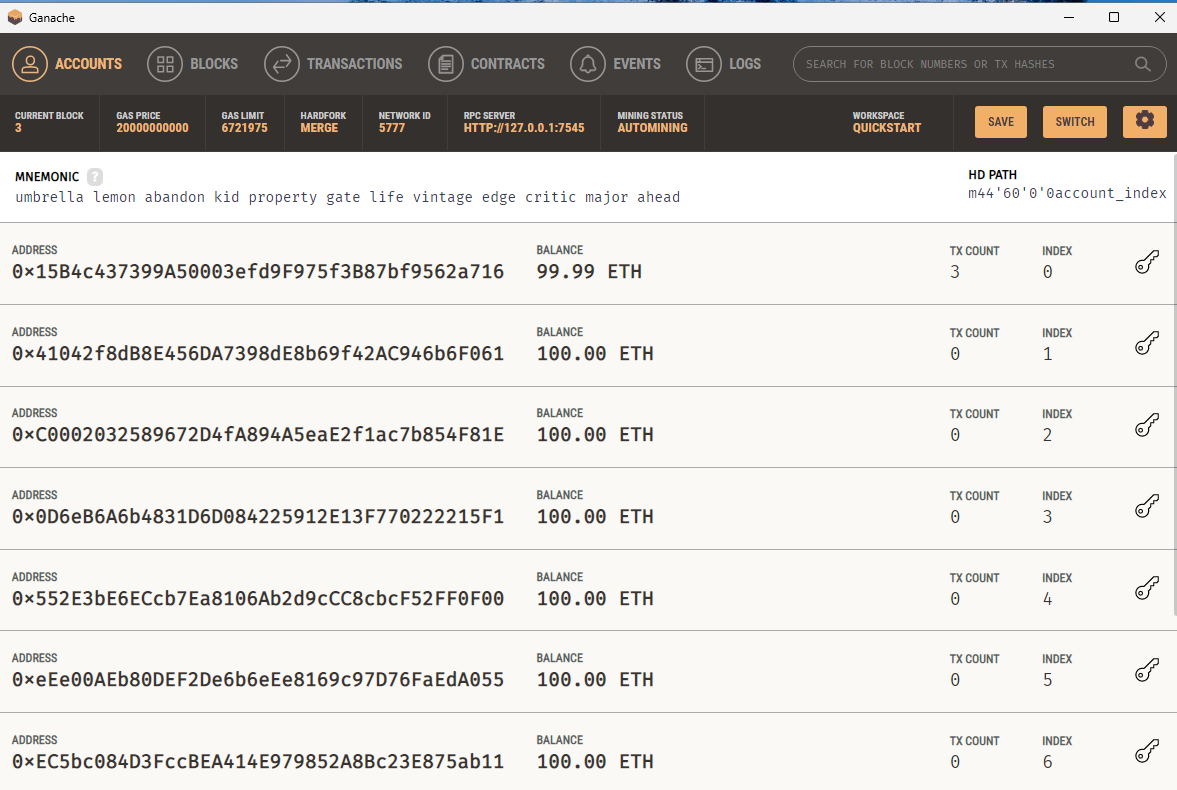
Và Migration (chỉnh file \*{timestamp}\_moolahcoin\_migration.js):

|  |
| --- |
| // migrations/2\_deploy\_moolahcoin.js  const MoolahCoin = artifacts.require("MoolahCoin");  module.exports = async (deployer, network, accounts) => {  await deployer.deploy(MoolahCoin, accounts[0]);  }; |

**3) Làm sạch & restart chain**

|  |
| --- |
| # xóa build cũ  rmdir /s /q build # Windows CMD (hoặc: rm -rf build)  truffle compile --all  # reset workspace Ganache (Save & Restart), rồi:  truffle migrate --network ganache --reset |

# 9. KẾT QUẢ MIGRATE TRÊN GANACHE

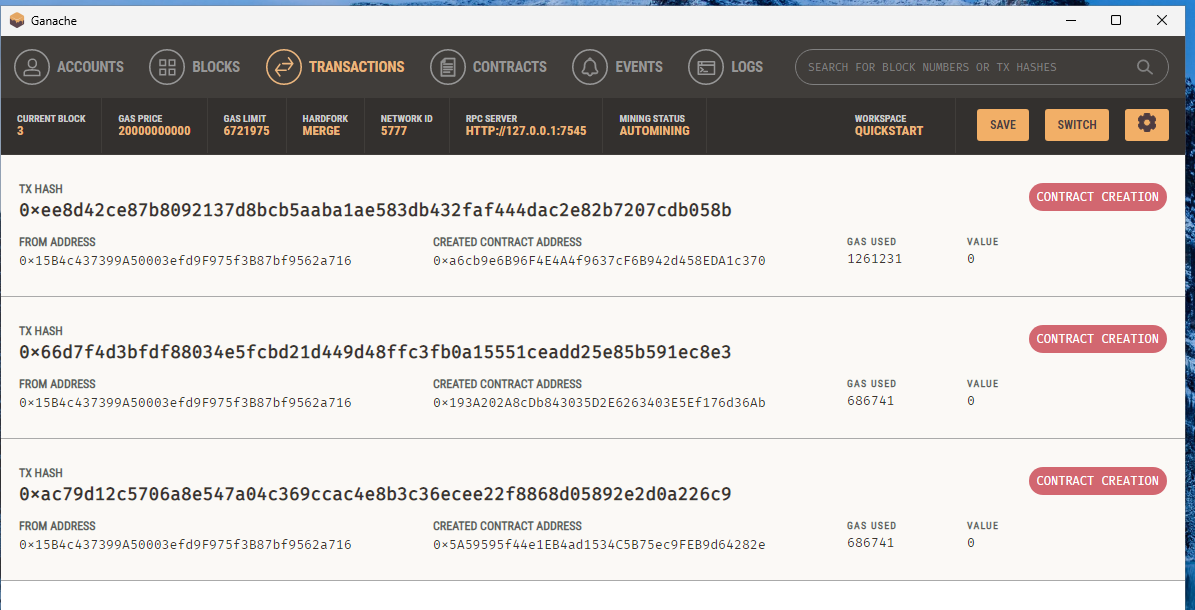


Nhìn KẾT QUẢ Ganache thì migrate đã **chạy thành công** và đây là ý nghĩa các phần:

* **ACCOUNT 0 (index 0) còn 99.99 ETH & TX COUNT = 3**  
  → Đây là ví mà Truffle dùng để deploy (accounts[0]). Nó đã ký **3 giao dịch**:
  1. Deploy Migrations (mặc định của Truffle)
  2. Deploy MoolahCoin
  3. Deploy Condos  
     ETH giảm từ 100 → 99.99 là **phí gas** cho 3 giao dịch (gas price hiển thị 2 gwei).
* **Các account còn lại vẫn 100 ETH, TX COUNT = 0**  
  → Chưa dùng đến, chưa gửi/nhận giao dịch nào.
* Thanh thông tin trên cùng:
  1. **GAS PRICE** = 2000000000 wei (2 gwei)
  2. **GAS LIMIT** ≈ 6,721,975 (mặc định Ganache)
  3. **HARDFORK** = **MERGE** (tương đương Paris)
  4. **RPC SERVER** = http://127.0.0.1:7545 (truffle-config của bạn nên trỏ đúng cổng này)
  5. **MINING STATUS: AUTOMINING** → mỗi tx sẽ được mine ngay.
* **MNEMONIC** & **HD PATH m/44'/60'/0'/account\_index**  
  → 10 địa chỉ ở dưới được sinh từ cụm 12 từ này theo chuẩn HD wallet. Bạn có thể **Import Private Key** từng account vào MetaMask để test.

**Xem địa chỉ contract & chi tiết giao dịch ở đâu?**

* Tab **CONTRACTS**: thấy danh sách contract đã deploy (địa chỉ + ABI).
* Tab **TRANSACTIONS**: xem từng tx “Contract Creation”, gas dùng, từ account nào.



Đây là tab **TRANSACTIONS** của Ganache — nó liệt kê các giao dịch mà Truffle vừa chạy khi migrate.

Các ô ở đầu trang

* **GAS PRICE 2000000000** → 2,000,000,000 wei = **2 gwei**.
* **GAS LIMIT ~ 6,721,975** → giới hạn gas mặc định mỗi block.
* **HARDFORK MERGE** (Paris) → phiên bản EVM đang dùng.
* **RPC SERVER** [**http://127.0.0.1:7545**](http://127.0.0.1:7545) → endpoint để DApp/Truffle kết nối.
* **AUTOMINING** → giao dịch được mine ngay khi gửi.

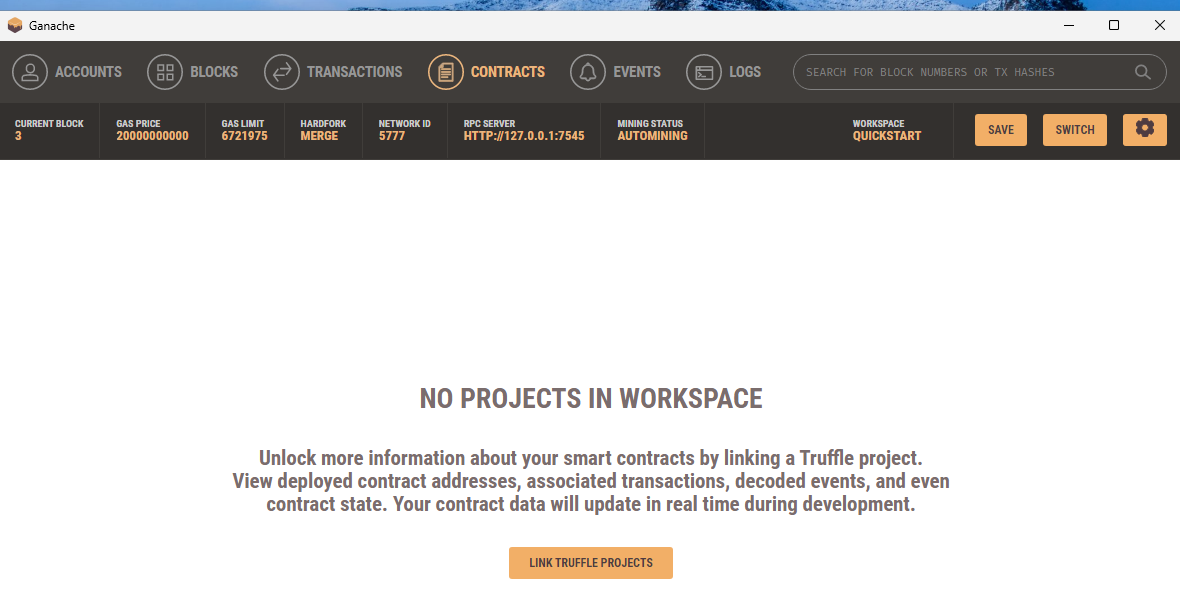
Mỗi thẻ giao dịch (từ trên xuống là mới → cũ)

* **TX HASH**: mã băm giao dịch (click để xem chi tiết).
* **FROM ADDRESS**: địa chỉ gửi (ví deployer của bạn – accounts[0]).
* **CREATED CONTRACT ADDRESS**: địa chỉ **contract mới được tạo** → đây là tx “**CONTRACT CREATION**”.
* **GAS USED**: lượng gas tiêu thụ thực tế của giao dịch.
* **VALUE**: ETH đính kèm (deploy contract thường là 0).

Trong ảnh có **3 giao dịch “CONTRACT CREATION”**:

1. Deploy **Migrations** (mặc định của Truffle) – gas used **1,261,231**
   * Chi phí ≈ 1,261,231 × 2e9 wei = **0.002522462 ETH**
2. Deploy **MoolahCoin** – gas used **686,741**
   * Chi phí ≈ **0.001373482 ETH**
3. Deploy **Condos** – gas used **686,741**
   * Chi phí ≈ **0.001373482 ETH**

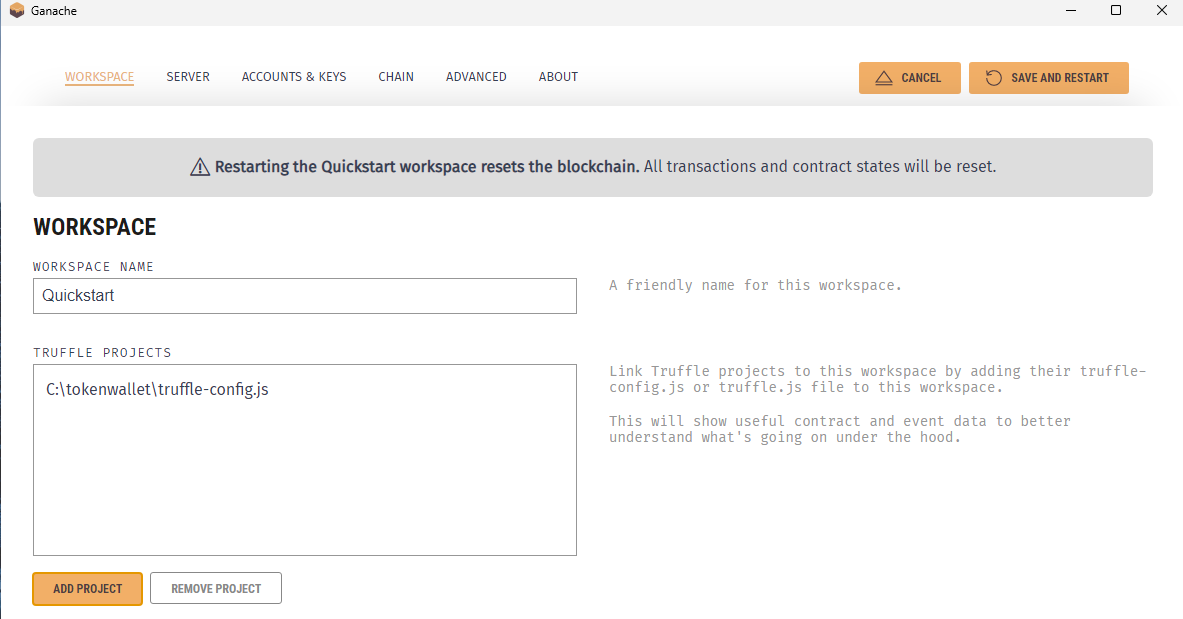
**Tab Contracts:**



Tab **CONTRACTS** của Ganache chỉ hiển thị tên contract + ABI khi **Ganache được link với dự án Truffle**. Hiện bạn đang ở workspace “Quickstart” chưa link nên nó báo **NO PROJECTS IN WORKSPACE**.

**Cách hiện danh sách contract (tên + địa chỉ + ABI)**

1. Vào tab **CONTRACTS** → bấm **LINK TRUFFLE PROJECTS**.
2. Chọn **truffle-config.js** trong thư mục dự án của bạn (ví dụ C:\tokenwallet\truffle-config.js).



1. Bấm **Save & Restart** (góc phải) để Ganache nạp ABI và map với các giao dịch đã tạo.
2. Quay lại tab **CONTRACTS** → sẽ thấy **Migrations / MoolahCoin / Condos** kèm **địa chỉ** và có thể xem **ABI, events, storage**.

**Lưu ý hay gặp**

* Port & network phải khớp (Ganache GUI mặc định http://127.0.0.1:7545, Network ID 5777).
* Nếu bạn xóa build/, nhớ truffle compile --all trước khi link.
* Sau khi **reset chain** trong Ganache, cần **migrate lại** thì danh sách mới xuất hiện.

# 10. ABI LÀ GÌ?

**ABI** (Application Binary Interface) = **“bảng hướng dẫn nói chuyện với smart contract”** ở dạng **JSON**.

* Chứa: tên hàm, tham số, kiểu trả về, sự kiện (events).
* Dùng để: **gọi hàm** (đọc/ghi) và **giải mã event**.
* Tương tác = **địa chỉ contract + ABI** (web3/ethers tạo instance từ đó).
* Lấy ở: build/contracts/YourContract.json (Truffle, mục "abi"), hoặc Remix → **ABI**.
* Khác **bytecode**: bytecode là **mã cài** trên chain; ABI là **giao diện** để gọi. Deploy cần bytecode; gọi cần ABI.

ABI (Application Binary Interface) trong Ethereum là **bản mô tả giao diện** của smart contract ở dạng **JSON**. Nó liệt kê các **hàm**, **sự kiện**, **error**, kiểu dữ liệu vào/ra, và thuộc tính (view/payable…). Nhờ ABI, thư viện như **web3.js/ethers.js** biết cách **mã hoá lời gọi hàm** và **giải mã kết quả/log** để bạn tương tác với contract.

Dùng để làm gì?

* **Gọi hàm** trên contract (read/write).
* **Giải mã event** (Transfer, Approval…).
* **Tạo instance** contract trong code frontend/backend.

Lấy ở đâu?

* Truffle: build/contracts/<Contract>.json → trường "abi".
* Remix: tab **Compilation Details** → **ABI**.
* Trong Truffle console:

const MC = artifacts.require("MoolahCoin"); MC.abi

Khác gì với bytecode?

* **Bytecode**: mã máy được deploy lên chain.
* **ABI**: chỉ là “giao diện” để gọi vào contract đã deploy.
* Triển khai: cần **bytecode + tham số constructor**.
* Tương tác: cần **địa chỉ contract + ABI** (và ký giao dịch nếu là hàm ghi).

**ABI = “hợp đồng” giữa ứng dụng và smart contract**. Không có ABI, ứng dụng không biết phải gọi hàm gì, tham số kiểu gì, và giải mã kết quả ra sao.

# 11. MỘT SỐ LỆNH DEBUG TRONG TRUFFLE

# xóa build cũ

rmdir /s /q build # Windows CMD (hoặc: rm -rf build)

truffle compile --all

# reset workspace Ganache (Save & Restart), rồi:

truffle migrate --network ganache –reset

# 12. LỖI KHÔNG HIỂN THỊ CONTRACT DEPLOYED IN TAB CONTRACTS GANACHE

Nếu vẫn chưa thấy MoolahCoin được deployed, force chỉ migration #timestamp:

|  |
| --- |
| truffle migrate --f #timestamp --to #timestamp --network ganache --reset |

# 13. LỖI TRONG CREATING THE TOKEN WALLET FRONTEND USING REACTJS

- Thực hiện lệnh npm audit fix –force trong thư mục dự án nếu không sẽ báo lỗi

* **Vào đúng thư mục dự án** (chỗ có package.json): cd C:\tokenwallet
* Nếu **chưa có lockfile**, tạo trước: npm i --package-lock-only
* Sau đó mới chạy audit:

|  |
| --- |
| npm audit  npm audit fix // chỉ sửa bản vá an toàn  npm audit fix --force // chỉ dùng khi chấp nhận risk/breaking changes |

Lưu ý: npm audit không áp dụng cho gói cài **global**; phải chạy trong **thư mục dự án** có package.json.

# 14. Kiểm tra NetID và ChainID trên Truffle console

 **Network ID**: mã nhận dạng mạng cấp bởi P2P/RPC (trả về qua net\_version / web3.eth.net.getId()).

 **Chain ID**: mã dùng để **ký giao dịch** và **chống replay** (EIP-155), trả về qua eth\_chainId / web3.eth.getChainId() (dạng hex hoặc số).

**Cách kiểm tra nhanh:**

Trong Truffle console (đúng RPC Ganache):

|  |
| --- |
| await web3.eth.getChainId() // → chainId (số)  await web3.eth.net.getId() // → networkId |

# 15. Lỗi khi run app (React App)

1. “'react-scripts' is not recognized…” xảy ra khi **chưa cài node\_modules** (hoặc cài lỗi).

* Vào đúng thư mục dự án: cd C:\tokenwallet1
* Cài lại deps sạch:

|  |
| --- |
| rmdir /s /q node\_modules  del package-lock.json  npm install |

* Nếu vẫn báo không tìm thấy, cài trực tiếp:

|  |
| --- |
| npm i -E react-scripts@5.0.1 |

* Kiểm tra có lệnh:

|  |
| --- |
| npx react-scripts --version  npm start |

* Khuyến nghị tương thích

CRA v5 hợp với **React 17/18**. Bạn đang dùng React 16.4.2 → nên nâng:

|  |
| --- |
| npm i -E react@18.2.0 react-dom@18.2.0 |

# 16. Một số lệnh npm

## 1. Để biết dự án của bạn đang dùng **Create React App version nào:**

|  |
| --- |
| npm ls react-scripts  # hoặc  npx react-scripts --version |

Nếu ra 5.0.1 (hoặc 5.x) → dự án CRA v5.

**CRA v5** chạy tốt nhất với Node 16/18/20; Node 22 có thể phát sinh lỗi toolchain cũ.

## 2. Lệnh xem **version React** đang cài

|  |
| --- |
| npm ls react |

# 17. Web3.js và Ethers.js

* **Web3.js** và **Ethers.js** đều là thư viện JavaScript để **kết nối ví (MetaMask)**, **gọi node RPC (HTTP/WS)** và **tương tác smart contract** (gọi hàm, gửi tx, đọc event).
* Chúng làm việc với cùng một provider (window.ethereum), khác nhau chủ yếu ở **API** và **trải nghiệm dev**.

Web3.js

* “Đời đầu”, lâu năm, phổ biến trong nhiều tutorial & sách.
* API kiểu new web3.eth.Contract(abi, address).methods.fn(...).call()/send({from}).
* Phiên bản:
  + **v1.10.x**: ổn định, hợp với CRA v5 (ít cấu hình).
  + **v4.x**: ESM + phụ thuộc Node core → **CRA v5 cần polyfill** (rườm rà).
* Ưu: quen thuộc, nhiều ví dụ; Nhược: API dài dòng hơn, typing kém hơn Ethers.

Ethers.js

* Nhẹ, hiện đại, **type-safe** hơn, dễ tree-shake, docs rõ.
* API gọn: new ethers.Contract(address, abi, signer); parse/format đơn vị tiện (parseUnits/formatUnits).
* **v6**: ESM-first, rất phổ biến hiện nay; chạy tốt với CRA v5 mà **không cần polyfill**.
* Ưu: DX tốt, code sạch; Nhược: khác cú pháp so với nhiều tutorial cũ.

# 18. Lỗi khi chạy npm start

* Do dùng Web3 nhiều lỗi nên chuyển sang ethers => Cài ethers:

|  |
| --- |
| npm i -E ethers@6 |

* Lỗi điển hình khi dependency react-scripts chưa được cài (hoặc bị thiếu trong node\_modules)

**sửa package.json** dòng "react-scripts": "0.0.0" thành "react-scripts": "5.0.1"

Xóa sạch & cài lại:

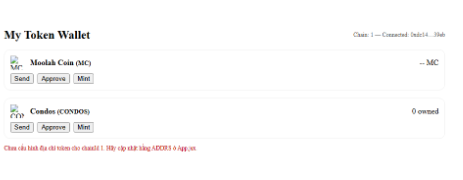
|  |
| --- |
| rmdir /s /q node\_modules  del package-lock.json  npm cache clean --force  npm install |

Kiểm tra & chạy:

|  |
| --- |
| npm ls react-scripts  dir node\_modules\.bin\react-scripts\*  npm start |

Khi đánh lệnh kiểm tra trên thì phải hiển thị ra "5.0.1".

* Lỗi Web app react kết nối tới Mainet không phải Ganache:



Dòng trên cùng bên phải Chain: 1 => có nghĩa là web đang nối vào Mainet không phải Ganache

Hoặc kiểm tra web đang nối vào đâu: trong trình duyệt gõ Ctrl+Shift+I -> Console -> paste dòng sau:

|  |
| --- |
| // ERC-20  window.ethereum.request({  method: 'eth\_getCode',  params: ['0xddF9c70eD6EeBDEEEB0FA1322D83D40fa165153E', 'latest']  }).then(console.log);  // ERC-721  window.ethereum.request({  method: 'eth\_getCode',  params: ['0x82Bd52Def9E6955128fEe68c081d467D6B79c4f8', 'latest']  }).then(console.log); |

* + Nếu eth\_chainId ra 0x539 (1337) hoặc 0x1691 (5777) ⇒ đang ở Ganache.
  + Nếu trả về "0x" trong eth\_getCode ⇒ sai mạng/sai địa chỉ.

Khắc phục:

* + Trong tab localhost:3000, mở **Ctrl+Shif+I → Console**, gõ (hoặc dán sau khi gõ allow pasting để cho phép paste):

|  |
| --- |
| // Nếu có nhiều ví cùng inject, chọn MetaMask làm provider mặc định  if (window.ethereum?.providers?.length) {  const mm = window.ethereum.providers.find(p => p.isMetaMask);  if (mm) window.ethereum = mm;  } |

* + Ép **switch network** sang Ganache:

|  |
| --- |
| // thử 1337 (0x539)  ethereum.request({  method: 'wallet\_switchEthereumChain',  params: [{ chainId: '0x539' }]  }).catch(async (e) => {  if (e.code === 4902) {  await ethereum.request({  method: 'wallet\_addEthereumChain',  params: [{  chainId: '0x539',  chainName: 'Ganache',  rpcUrls: ['http://127.0.0.1:7545'],  nativeCurrency: { name: 'ETH', symbol: 'ETH', decimals: 18 }  }]  });  } else { console.error(e); }  }); |

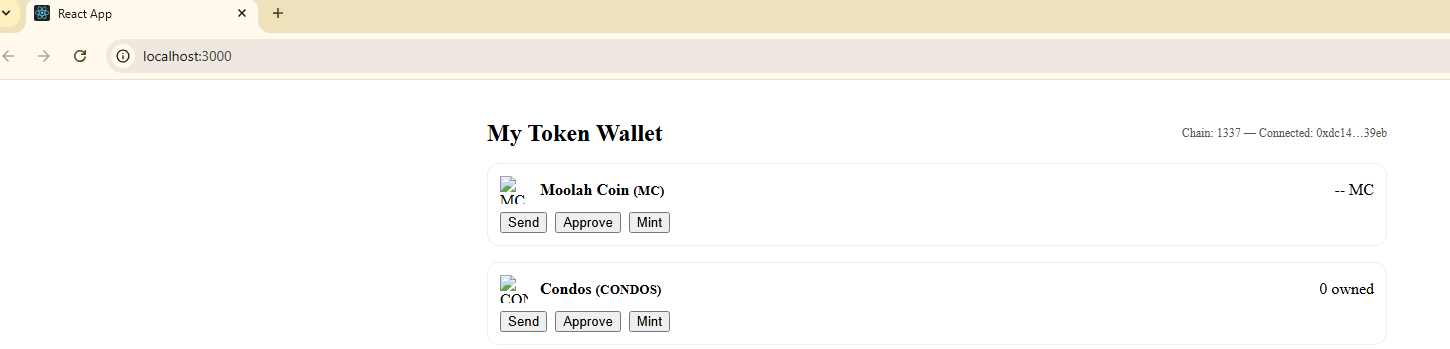
Nếu Ganache của bạn dùng **5777**, chạy thêm lệnh này:

|  |
| --- |
| ethereum.request({ method: 'wallet\_switchEthereumChain', params: [{ chainId: '0x1691' }] }); |

Xác nhận đã ở Ganache:

|  |
| --- |
| window.ethereum.request({ method: 'eth\_chainId' })  .then(id => console.log('chainId =', id, parseInt(id, 16))); |

Kỳ vọng thấy **0x539 (1337)** hoặc **0x1691 (5777)**.



Nếu vẫn là 0x1: trong MetaMask → **⋯ → Connected sites** → **Disconnect** localhost:3000, reload trang, bấm **Connect Wallet** lại rồi chạy kiểm tra trên lần nữa.

# 19. Lỗi import token trong Metamask

Nút **Import** bị xám vì ô **Token ID** đang nhập Condos1 (chuỗi chữ).  
Với **ERC-721**, **Token ID phải là một số nguyên** (0, 1, 2, …). Hãy nhập **1** (hoặc 0/2 nếu 1 không đúng) là nút sẽ sáng.

Nếu bạn **không biết tokenId** của NFT mình đã mint, lấy theo một trong các cách sau:

**Cách 1: Từ dApp của bạn**

Bấm **Mint** (ở khối Condos) → xác nhận giao dịch trong MetaMask. Hầu hết contract đặt tokenId bắt đầu từ **1** (hoặc 0). Sau khi mint xong, thử nhập **1** vào ô Token ID để import.

**Cách 2: Xem trong Ganache**

Ganache → **TRANSACTIONS** → mở giao dịch **mint** của contract Condos → xem **event Transfer** (from 0x000...000 to ví của bạn) → lấy trường **tokenId**.

**Cách 3: Hỏi contract bằng Console (nếu contract có ERC721Enumerable)**

Mở localhost:3000 → F12 → **Console**, dán:

(async () => {

const { ethers } = window;

const p = new ethers.BrowserProvider(window.ethereum);

const me = (await p.send('eth\_accounts', []))[0];

const nft = new ethers.Contract(

"0x82Bd52Def9E6955128fEe68c081d467D6B79c4f8",

["function balanceOf(address) view returns(uint256)",

"function tokenOfOwnerByIndex(address,uint256) view returns(uint256)"],

p

);

const n = await nft.balanceOf(me);

for (let i = 0; i < n; i++) console.log((await nft.tokenOfOwnerByIndex(me, i)).toString());

})();

Console sẽ in ra các **tokenId** bạn sở hữu.

Nhớ: MetaMask đang ở **đúng mạng Ganache** (chainId 1337/5777). Sau khi biết tokenId (ví dụ **1**), quay lại popup **Import NFT**, nhập:

* **Address:** 0x82Bd52Def9E6955128fEe68c081d467D6B79c4f8
* **Token ID:** 1  
  rồi **Import**.

