**Nhóm 5: Hướng dẫn Tạo ra token ERC20**

***Bước 1: Lên đặc điểm của token***

Để tạo ra một token ERC20, bạn phải định nghĩa những thuộc tính sau:

* Tên của token
* Mã của token
* Số chữ số thập phân sau dấu phẩy của token
* Tổng số token muốn phát hành
* Ví dụ chúng ta sẽ tạo token như sau:
* Tên: Cryptolife
* Mã: CTL
* Số chữ số thập phân sau dấu phẩy: 0
* Tổng số token phát hành: 1000

Thuộc tính Số chữ số thập phân chỉ có ý nghĩa trong hiển thị, và nó được thiết lập khác nhau tuỳ vào ý đồ của người tạo ra nó. Theo mặc định thì sẽ có 18 chữ số thập phân sau dấu phẩy, tức là token có thể được chia nhỏ đến .0000000000000000001 token. Với ví dụ trên, chúng ta muốn token sẽ ở dạng nguyên nên sẽ không có chữ số thập phân nào.

Tổng số token phát hành được tính theo đơn vị nhỏ nhất của token đó. Ví dụ khi số chữ số thập phân là 0 thì với tổng số token phát hành sẽ được đếm trong khoảng 1 - 1000. Khi chữ số thập phân là 1, tổng số token là 1000 thì token sẽ được hiển thị trong phạm vi 0.1 – 100

***Bước 2: Viết mã cho phần Smart Contract chuẩn ERC20***

Link code: <https://github.com/hungngph/DemoMHH/tree/master/TokenERC20Solidity>

Trước tiên bạn sẽ cần copy một contract chuẩn ERC20 về để từ đó sửa những chỗ cần thiết thôi. Contract chúng ta sẽ tham khảo tại Token-Factory của ConsenSys. Copy nội dung của Token.sol, StandardToken.sol và HumanStandardToken.sol vào một file Solidity duy nhất như dưới.

pragma solidity ^0.4.4;

contract Token {

*/// @return total amount of tokens*

function totalSupply() constant returns (uint256 supply) {}

*/// @param \_owner The address from which the balance will be retrieved*

*/// @return The balance*

function balanceOf(address \_owner) constant returns (uint256 balance) {}

*/// @notice send `\_value` token to `\_to` from `msg.sender`*

*/// @param \_to The address of the recipient*

*/// @param \_value The amount of token to be transferred*

*/// @return Whether the transfer was successful or not*

function transfer(address \_to, uint256 \_value) returns (bool success) {}

*/// @notice send `\_value` token to `\_to` from `\_from` on the condition it is approved by `\_from`*

*/// @param \_from The address of the sender*

*/// @param \_to The address of the recipient*

*/// @param \_value The amount of token to be transferred*

*/// @return Whether the transfer was successful or not*

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {}

*/// @notice `msg.sender` approves `\_addr` to spend `\_value` tokens*

*/// @param \_spender The address of the account able to transfer the tokens*

*/// @param \_value The amount of wei to be approved for transfer*

*/// @return Whether the approval was successful or not*

function approve(address \_spender, uint256 \_value) returns (bool success) {}

*/// @param \_owner The address of the account owning tokens*

*/// @param \_spender The address of the account able to transfer the tokens*

*/// @return Amount of remaining tokens allowed to spent*

function allowance(address \_owner, address \_spender) constant returns (uint256 remaining) {}

event Transfer(address indexed \_from, address indexed \_to, uint256 \_value);

event Approval(address indexed \_owner, address indexed \_spender, uint256 \_value);

}

contract StandardToken is Token {

function transfer(address \_to, uint256 \_value) returns (bool success) {

*//Default assumes totalSupply can't be over max (2^256 - 1).*

*//If your token leaves out totalSupply and can issue more tokens as time goes on, you need to check if it doesn't wrap.*

*//Replace the if with this one instead.*

*//if (balances[msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {*

if (balances[msg.sender] >= \_value && \_value > 0) {

balances[msg.sender] -= \_value;

balances[\_to] += \_value;

Transfer(msg.sender, \_to, \_value);

return true;

} else { return false; }

}

function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success) {

*//same as above. Replace this line with the following if you want to protect against wrapping uints.*

*//if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && balances[\_to] + \_value > balances[\_to]) {*

if (balances[\_from] >= \_value && allowed[\_from][msg.sender] >= \_value && \_value > 0) {

balances[\_to] += \_value;

balances[\_from] -= \_value;

allowed[\_from][msg.sender] -= \_value;

Transfer(\_from, \_to, \_value);

return true;

} else { return false; }

}

function balanceOf(address \_owner) constant returns (uint256 balance) {

return balances[\_owner];

}

function approve(address \_spender, uint256 \_value) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

return true;

}

function allowance(address \_owner, address \_spender) constant returns (uint256 remaining) {

return allowed[\_owner][\_spender];

}

mapping (address => uint256) balances;

mapping (address => mapping (address => uint256)) allowed;

uint256 public totalSupply;

}

*//name this contract whatever you'd like*

contract ERC20Token is StandardToken {

function () {

*//if ether is sent to this address, send it back.*

throw;

}

*/\* Public variables of the token \*/*

*/\**

*NOTE:*

*The following variables are OPTIONAL vanities. One does not have to include them.*

*They allow one to customise the token contract & in no way influences the core functionality.*

*Some wallets/interfaces might not even bother to look at this information.*

*\*/*

string public name; *//fancy name: eg Simon Bucks*

uint8 public decimals; *//How many decimals to show. ie. There could 1000 base units with 3 decimals. Meaning 0.980 SBX = 980 base units. It's like comparing 1 wei to 1 ether.*

string public symbol; *//An identifier: eg SBX*

string public version = 'H1.0'; *//human 0.1 standard. Just an arbitrary versioning scheme.*

*//*

*// CHANGE THESE VALUES FOR YOUR TOKEN*

*//*

*//make sure this function name matches the contract name above. So if you're token is called TutorialToken, make sure the //contract name above is also TutorialToken instead of ERC20Token*

function ERC20Token(

) {

balances[msg.sender] = NUMBER\_OF\_TOKENS\_HERE; *// Give the creator all initial tokens (100000 for example)*

totalSupply = NUMBER\_OF\_TOKENS\_HERE; *// Update total supply (100000 for example)*

name = "NAME OF YOUR TOKEN HERE"; *// Set the name for display purposes*

decimals = 0; *// Amount of decimals for display purposes*

symbol = "SYM"; *// Set the symbol for display purposes*

}

*/\* Approves and then calls the receiving contract \*/*

function approveAndCall(address \_spender, uint256 \_value, bytes \_extraData) returns (bool success) {

allowed[msg.sender][\_spender] = \_value;

Approval(msg.sender, \_spender, \_value);

*//call the receiveApproval function on the contract you want to be notified. This crafts the function signature manually so one doesn't have to include a contract in here just for this.*

*//receiveApproval(address \_from, uint256 \_value, address \_tokenContract, bytes \_extraData)*

*//it is assumed that when does this that the call \*should\* succeed, otherwise one would use vanilla approve instead.*

if(!\_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))), msg.sender, \_value, this, \_extraData)) { throw; }

return true;

}

}

Chúng ta sẽ sửa lại từ đoạn comment CHANGE THESE VALUES FOR YOUR TOKEN:

* Tên token
* Mã token
* Số chữ số thập phân
* Số token mà người phát hành nắm
* Tổng cung

Bạn cần chú ý một số chỗ sau:

Bạn có thể đặt tên tuỳ ý cho contract ERC20Token, chẳng hạn như mình sẽ đặt tên là CryptolifeToken. Miễn sao hàm khởi tạo cũng phải đúng tên đó.

Tổng cung token có liên quan tới số chữ số thập phân của token. Ví dụ khi bạn muốn phát hành 1000 token có số chữ số thập phân là 2 thì bạn cần tổng cung là 10000. Nhưng khi bạn muốn phát hành 1 token có số chữ số thập phân là 5 thì bạn cần tổng cung là 100000

Thiết lập số token mà người phát hành nắm giữ balances[msg.sender] là số token sẽ được tạo ra tại ví của người phát hành smart contract.

Dưới đây là phần code mà chúng ta đã chỉnh:

...

*//Ham khoitao token*

function CryptolifeToken() {

balances[msg.sender] = 1000; *// Nguoi phathanh se namgiu toanbo token*

totalSupply = 1000; *// Tong cung token*

name = "NPH TOKEN"; *// Ten cua token*

decimals = 0; *// Token khong co phan thapphan (so nguyen thoi)*

symbol = "NPH"; *// Ma token*

}

...

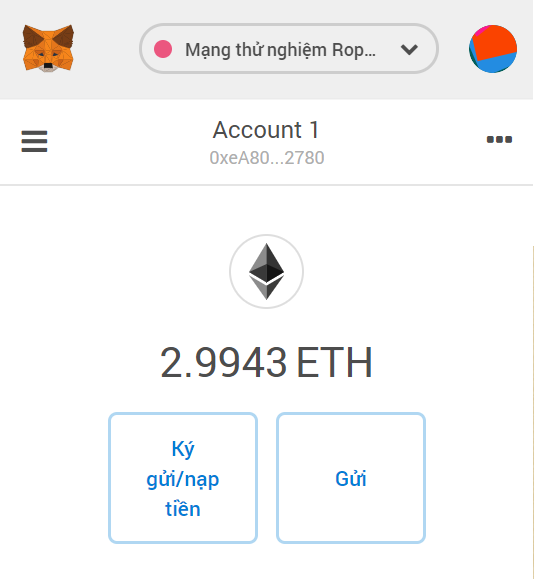
Sau khi đã thiết lập xong, giờ là lúc test token thôi!

***Bước 3: Test token trên Testnet***

Ở bước này, chúng ta sẽ phát hành token lên một mạng lưới test (Test net) để xem nó có hoạt động ok hay không. Chúng ta sẽ chọn Ropsten Testnet để test token này.

À quên, nhưng trước tiên để tương tác với Ropsten Testnet này thì bạn cần phải cài đặt MetaMask. Mình sẽ có một bài viết riêng hướng dẫn một số thiết lập và thao tác quản lý ví cơ bản sử dụng MetaMask sau.

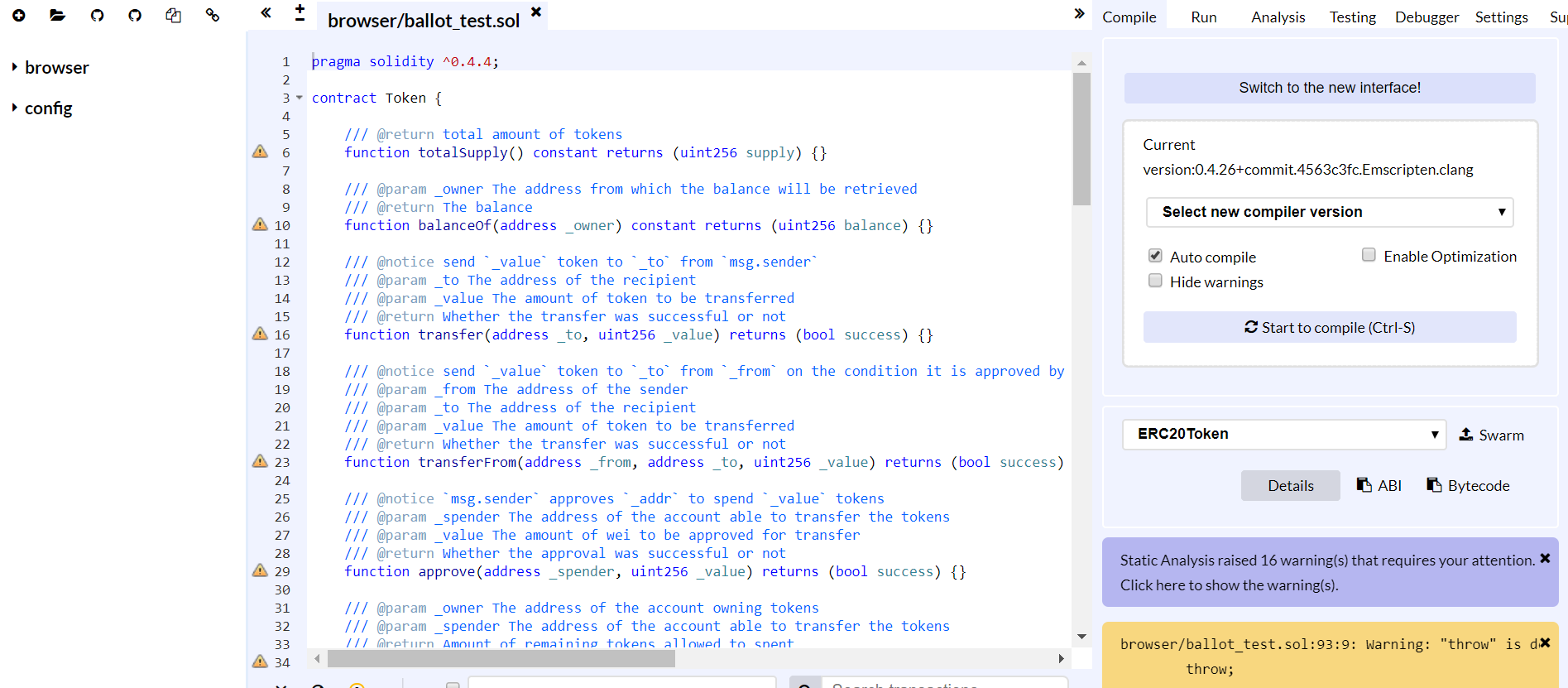
Sau khi cài xong MetaMask vào trình duyệt Chrome, bạn đăng nhập và chọn network là Ropsten Network như hình.



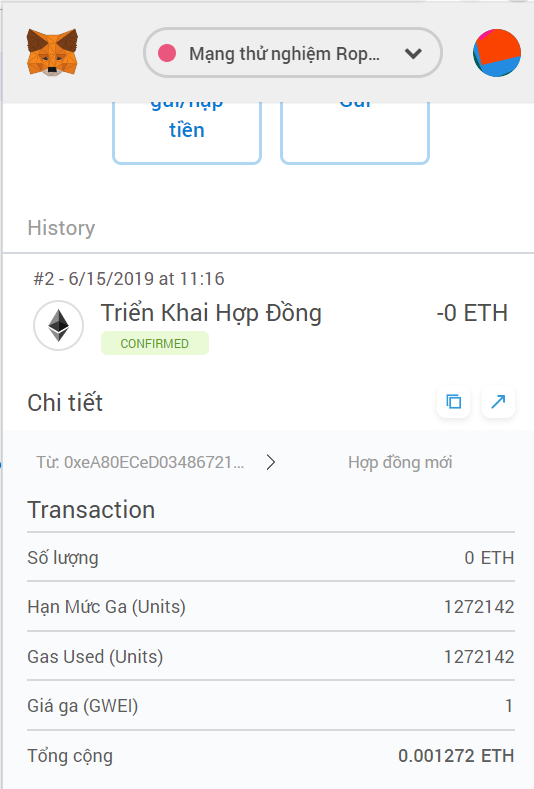
Địa chỉ ví này sẽ là chủ của contract sắp phát hành nên bạn không được để mất ví. Ngoài MetaMask ra bạn có thể dùng Mist hoặc MyEtherWallet để tạo contract cũng được. Mình sử dụng MetaMask vì nó khá đơn giản và bạn có thể export ra private key để có thể dùng với MyEtherWallet sau này.

Tiếp theo là biên dịch mã nguồn contract bằng Solidity của chúng ta. Để làm việc này, chúng ta sẽ sử dụng Solidity Remix Compiler, một trình biên dịch online cho phép chúng ta phát hành contract ngay lên blockchain sau khi biên dịch xong.

Bạn copy và dán mã nguồn đã chỉnh sửa vào Soidity Remix Compiler. Đổi tên file contract lại một chút chúng ta sẽ có kết quả như hình dưới.



Sau khi đã chuẩn bị xong thì chúng ta sẽ deploy và nhận được kết quả.



Và đây là kết quả Token NPH mình đã tạo.

