

Tutorial 6

Building your own cryptocurrency (token) and ICO

Prepared by: Reza Parizi

The Ethereum blockchain allows you to create your own cryptocurrency, or **token**, without creating a new blockchain from scratch. Tokens in the Ethereum ecosystem can represent any fungible tradable good: coins, loyalty points, gold certificates, in-game items, etc. Since all tokens implement some basic features in a standard way (e.g. ERC20), this also means that your token will be instantly compatible with other platforms like cryptocurrency exchanges, Ethereum wallet and any other client or contract that uses the same standards.

Ways to create an Ethereum token (i.e. token contract)

-The simplest way for you to create a token is simply using [Token Factory](#) that provides a simple user interface that allows you to build a token contract with custom parameters.

Disadvantages: lack of testability, less control over code for modification [not recommended ]

-The best way for you to [create a token](#) is to code it in form of a smart contract using Solidity.

Let's get started

In this tutorial, you will learn how to code your own cryptocurrency on the Ethereum blockchain and build an ICO to sell it. To build a cryptocurrency, you simply need to create a *smart contract token* that implements [ERC-20](#) token standard.

Here is what you'll build:

You'll build an ICO website that will be governed by a crowd sale smart contract on the blockchain, which uses a token contract as well. The front-end will have a form where users can purchase tokens (symbol 'KCoin') in the crowd sale. It will show the progress for the crowd sale, like how many tokens the user has purchased, how many tokens have been purchased by all users, and the total number of tokens available in the crowd sale. It will also show the account you are connected to the blockchain with under "your account".

[a screenshot of final dapp].

KSU TOKEN

"KSU Token" (KCoin) ICO SALE!!! Token price is 0.001 Ether.

Buy Token

0 / 750000 tokens sold

Your Account: 0x5b7f69b1452068cd60810373d5f0875aed967ace

You currently have 0 KCoin.

Step 1: Build your ERC-20 Token Smart Contract (actually is the exercise you did last week)

Here is the complete ERC-20 token smart contract Solidity code for your review: [download](#)

Let's take a look at what this smart contract does, and how it implements the ERC-20 standard:

- It stores the token name `string public name = "KSU Token"`.
- It stores the token symbol for cryptocurrency exchanges `string public symbol = "KCoin"`.
- It stores the total supply of tokens in existence `uint256 public totalSupply`.
- It uses a Solidity mapping to store the balance of each account that owns tokens `mapping(address => uint256) public balanceOf`.
- It implements a `transfer` function to allow users to send tokens to another account.
- It implements an `approve` function that allows another account to spend tokens, like on a cryptocurrency exchange. This updates the `allowance` mapping to see how much the account is allowed to spend.
- It implements a `transferFrom` that allows another account to transfer tokens.

Step 2: Build your ICO (crowd sale) Smart Contract

Now you've created a token, you will have to build an ICO to allow investors to purchase tokens in an initial coin offering (ICO) for automatic selling and buying.

Here is the complete ICO/crowd sale smart contract Solidity code for your review: [download](#).

Let's take a look at what this smart contract does, and how it functions as a crowd sale:

- It stores an admin account for the crowd sale `address admin`.
- It references the ERC-20 token smart contract `KSUToken public tokenContract`.
- It stores the token price `uint256 public tokenPrice`.
- It stores the number of tokens sold `uint256 public tokensSold`.

- It implements a **sell** event so that consumers can get notifications whenever a token has been sold.
- It implements a **buyTokens** function that allows users to purchase tokens in the crowd sale.
- It implements an **endSale** function that allows an admin to end the crowd sale and collect the Ether funds that was raised during the sale.

Up to this point, you have successfully built an ERC-20 token and crowd sale/ICO smart contract on Ethereum!

Step 3: Build a Dapp

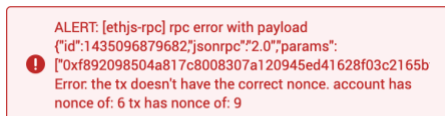
It is time to deploy smart contracts and connect them to your front-end (creating a Dapp). Use Truffle framework like you did in the previous tutorials.

You can download the full source code to this tutorial from [here](#) (review the code and try to understand how pieces are connected)

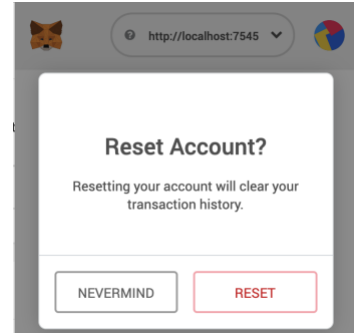
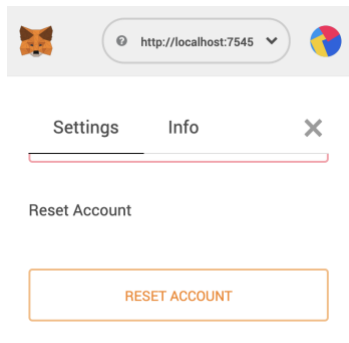
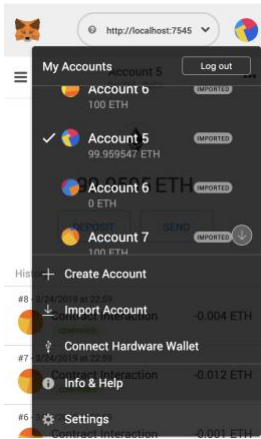
To run the code:

- Get the Ganache running first
- Open your terminal
- Go to your directory where the project is located
- Open migrations/2_deploy_contracts.js file: You must replace `from: address` by the address of the user deploying the contract—contract owner (1st account in Ganache, ex: { `from: "0xE96362E717EAF828B7Ee752a12bAa03C967D410c" }`).
- Type `truffle migrate --reset` in the terminal
- Type `npm run dev` in the terminal
- Start using the dapp with Metamask

Note, If you get this error during a ‘buy token’ transaction in Metamask:



You will simply need to reset the account’s history as follows and try again:
Settings>Reset Account



Exercise: configure the network (truffle-config.js) and connect your dapp to Ropsten now (if you've forgotten how to do so, refer to Tutorial 5).