Build your own cryptocurrency using ERC-20 tokens

Create and deploy an ERC-20 token on Ethereum.

Tool used:

1- Metamask
2- Remix IDE

But first what us ERC-20?

`ERC` stands for `Ethereum Request for Comment`. Essentially, they are standards that have been approved by the community and are used to convey technical requirements and specifications for certain use cases.

`ERC-20` specifically is a standard which outlines the technical specification of a fungible token.

Most tokens on Ethereum comply with the `ERC-20` specification. Following a standard like `ERC-20` allows application developers which use `ERC-20` tokens to easily support *all* `ERC-20` tokens without having to write specialized code for them individually.

For example, decentralized exchanges like Uniswap allow you to swap any token for any other token. This is only possible because pretty much all tokens follow the `ERC-20` standard, so Uniswap could write code which works with all tokens following the standard.

Prerequisites

- Make sure you have downloaded and installed Metamask.
- Select the `Goerli Testnet` network to work with
- Request some testnet ether on Goerli through any one of the following faucets:
    - Metamask Faucet
    - Chainlink Faucet
    - Paradigm Faucet

Writing the code

NB: OZ is an Ethereum security company. Among other things, OZ develops reference contracts for popular smart contract standards which are thoroughly tested and secure. Whenever implementing a smart contract which needs to comply with a standard, try to find an OZ reference implementation rather than rewriting the entire standard from scratch.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol";

contract LW3Token is ERC20 {
    constructor(string memory _name, string memory _symbol) ERC20(_name, _symbol) {
        _mint(msg.sender, 10 * 10 ** 18);
    }
}
```

Let's break it down and explain each line of it:

1- Pragma : compiler version of Solidity to be used
2- Import the ERC-20 token from OpenZeppelin (OZ).
3- Then we start creating the Contract and we name it LW3Token wich is extends or instance of the ERC20, so then all the functions and logic

that is built into `ERC20` is available for us to use, and we can add our own custom logic on top of it.

4- Then we define the constructor (like in java, each class has a constructor) that it's called when the smart contract is first deployed.

    2 argument in constructor : _name and _symbol, which are the name and symbol of the cryptocurrency, (ex : ETHERIUM and ETH)

    Therefore, we are providing `_name` and `_symbol` variables to our contract, which we immediately pass on to the `ERC20` constructor, thereby initializing the `ERC20` smart contract.

5-
```
_mint(msg.sender, 10 * 10 ** 18);
```
`_mint` is an `internal` function within the `ERC20` standard contract, which means that it can only be called by the contract itself. External users cannot call this function.

`msg.sender` is a global variable injected by the Ethereum Virtual Machine, which is the address which made this transaction. Since you will be the one deploying this contract, your address will be there in `msg.sender`. `10 * 10 ** 18` specifies that you want 10 full tokens to be minted to your address.

Finally compile it and deploy it after connecting your Metamask wallet and selecting the Goerli Test Network.

And Finally you can see it