@maryemhadjwannes

Create ERC-20 Token in 10 mins

Start Tutorial

Tools

REMIX https://remix.ethereum.org/

METAMASK https://metamask.io/download/

No local installation is needed; everything is online!

@maryemhadjwannes

How Tokens Work in Ethereum Virtual Machine

Tokens on Ethereum act as a ledger or database that keeps track of addresses and their corresponding balances. This database ensures secure and transparent transfer of tokens between addresses.

- **Visualizing the Token Ledger**

The token ledger can be imagined as a table with two columns:

Address	Balance
0x123	100
0x456	0
0x789	0

Token Basics

- **Example Transactions**

1. Transfer 25 Tokens from 0x123 to 0x456

The ledger updates as follows:

Address	Balance
0x123	75
0x456	25
0x789	0

2. Transfer 70 Tokens from 0x123 to 0x789

The ledger updates again:

Address	Balance
0x123	5
0x456	25
0x789	70

@maryemhadjwannes

Key Takeaways

- Each address represents a user or wallet.
- The balance column reflects the tokens each address holds.
- Transactions update balances by deducting tokens from the sender and adding them to the recipient.
- This simple structure is maintained and enforced by the Ethereum network, ensuring secure and tamper-proof token management.

OpenZeppelin

OpenZeppelin is a renowned company specializing in security audits for blockchain projects. They also provide a suite of free, secure, and audited smart contracts that anyone can use with confidence.

Their contracts are:

- Pre-audited: No need to worry about vulnerabilities.
- Standard-Compliant: Built to adhere to widely accepted token standards (like ERC20, ERC721, etc.).

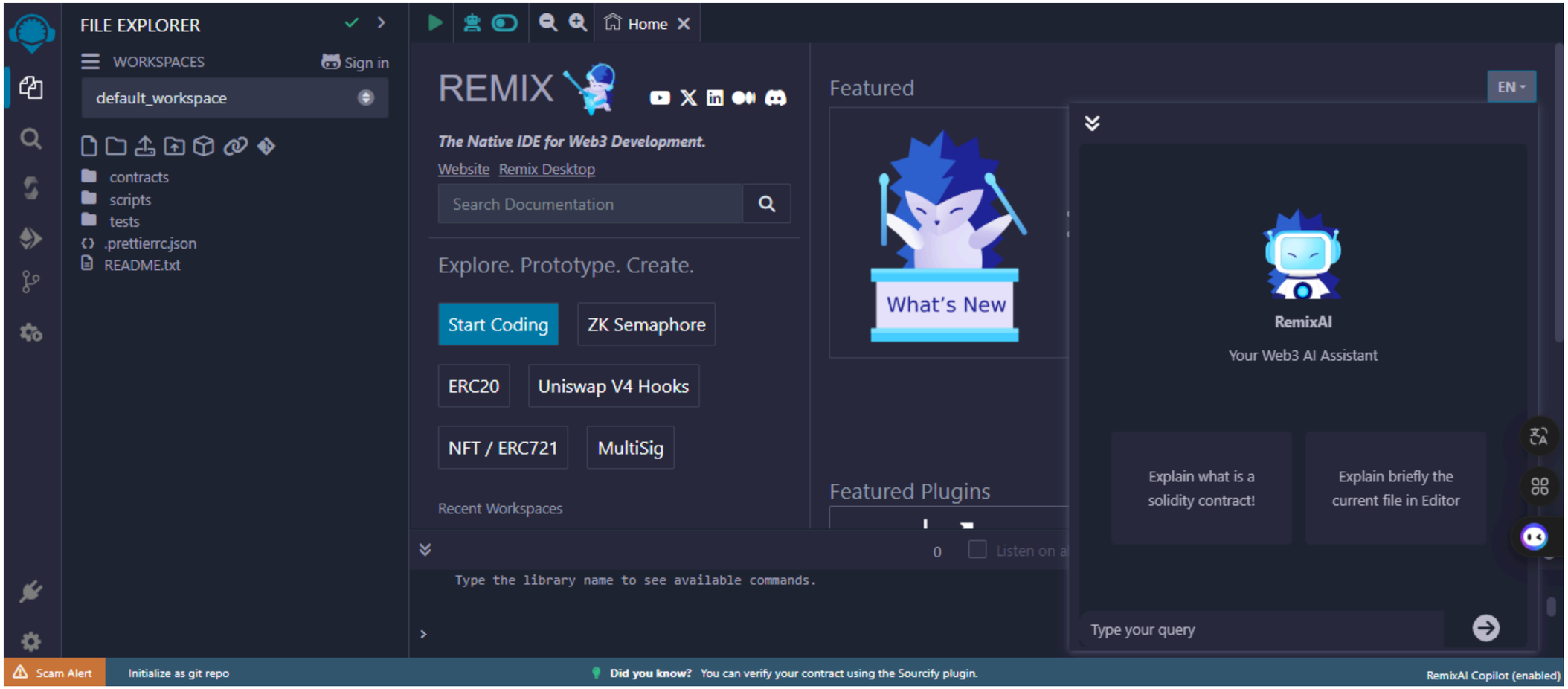
By using OpenZeppelin's contracts, you save time and ensure your token meets industry best practices. For this tutorial, we'll focus on the ERC20 contract.

Explore their free contracts here: [OpenZeppelin GitHub]

Create Token

@maryemhadjwannes

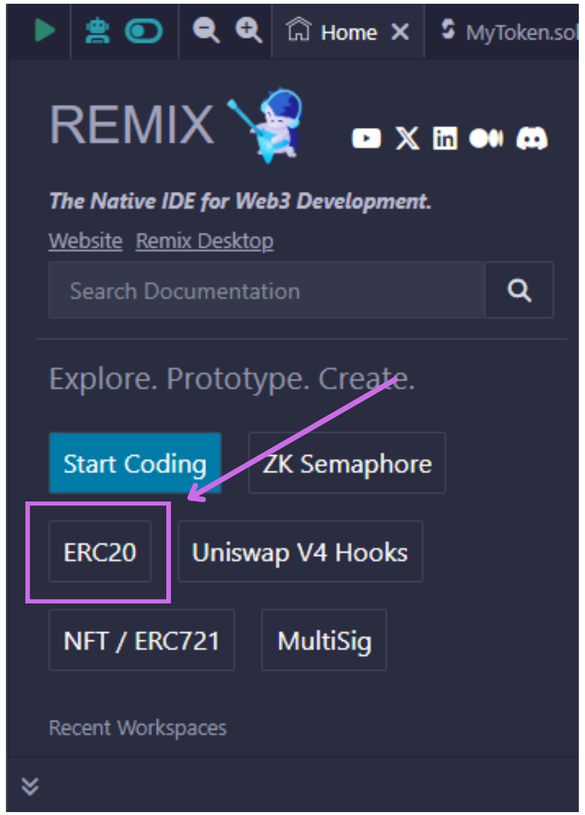
1- Open Remix IDE : This is the default page

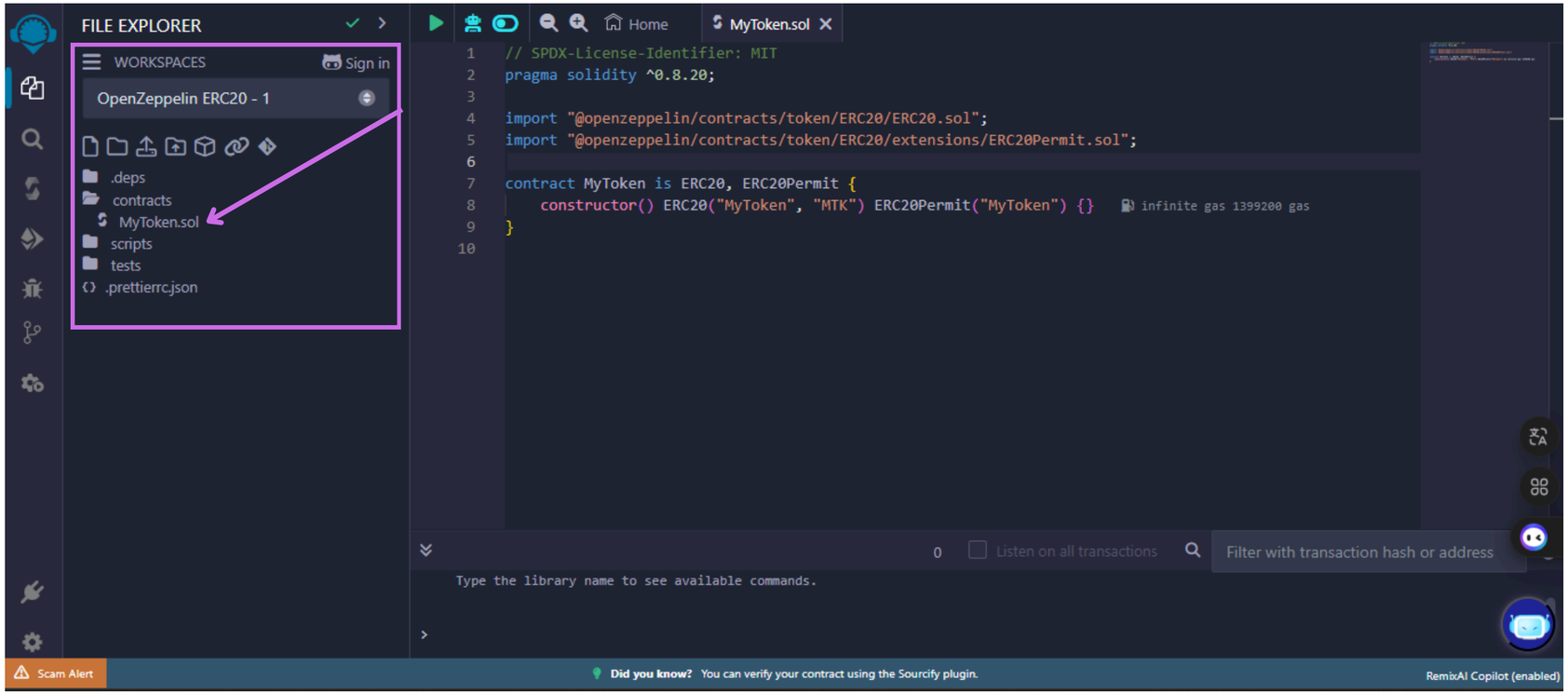


Create Token

@maryemhadjwannes

2- Choose ERC20 directly, and it will take you to the OpenZeppelin ERC20 workspace and create a MyToken.sol file with the necessary imports. You can create the file manually, but this method automates the imports and initial setup.





Code Breakdown

1- License Declaration

```
// SPDX-License-Identifier: MIT
```

- Specifies the license type for the contract, which in this case is MIT, a permissive open-source license.

2- Solidity Version

```
pragma solidity ^0.8.20;
```

- Indicates the version of Solidity required to compile the contract, ensuring compatibility and access to specific language features.

Code Breakdown

3- Import Statements

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
```

- **ERC20.sol:** Contains the base implementation of the ERC20 token standard, providing functionality like transfers, balance tracking, and allowance mechanisms.
- **ERC20Permit.sol:** Adds support for "permit" functionality, allowing approvals via digital signatures (EIP-2612) instead of requiring on-chain transactions.

4- Contract Declaration

```
contract MyToken is ERC20, ERC20Permit {
```

- Defines a new contract called **MyToken** that inherits functionality from both **ERC20** and **ERC20Permit.**

Code Breakdown

5- Constructor

```
constructor() ERC20("MyToken", "MTK") ERC20Permit("MyToken") {}
```

- The constructor initializes the **ERC20** and **ERC20Permit** parent contracts:
  - **ERC20:**
    - "**MyToken**": The name of the token.
    - "**MTK**": The symbol of the token.
  - **ERC20Permit:**
    - "**MyToken**": The name of the token used in the permit mechanism.

Create Token

@maryemhadjwannes

3- In this step, we extend our token contract with a custom minting function minHundred() . This function allows the caller to mint 100 tokens directly into their wallet

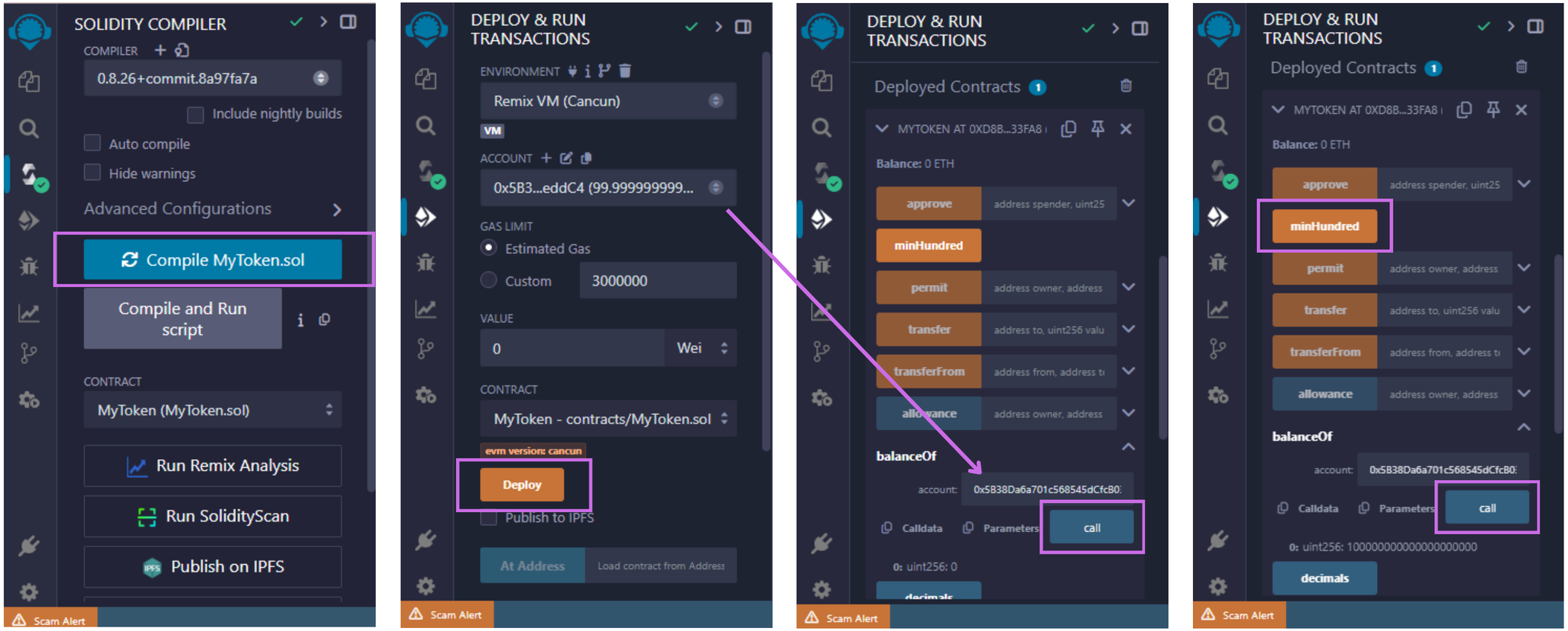
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";

contract MyToken is ERC20, ERC20Permit {
    constructor() ERC20("MyToken", "MTK") ERC20Permit("MyToken") {}   infinite gas 1436200 gas

    function minHundred() public {   infinite gas
        _mint(msg.sender, 100 * 10**18);
    }
}
```

- **Public Access:** Any user can call this function.
- **Minting:** Creates 100 tokens, considering the default 18 decimals.
- **Assignment:** Tokens are sent to the caller's address **(msg.sender).**

Create Token

@maryemhadjwannes

4- Compile and deploy the contract. In the deployed contracts section, you will find functions that you can test, such as balanceOf. Enter your address (from the Accounts section) into the balanceOf function, and it will initially return 0. Then, call the mintHundred function and check balanceOf again to see the new balance updated to 100 tokens.



Create Token

@maryemhadjwannes

5- Currently, anyone can call the mintHundred function to mint 100 tokens for themselves, which is not ideal for controlling token supply. To fix this, we can restrict the function so that only the contract owner can mint tokens.

ERC20 contracts include a modifier called onlyOwner. This modifier allows us to restrict access to specific functions, ensuring only the owner can execute them. By adding this, we secure the mintHundred function and prevent unauthorized minting.

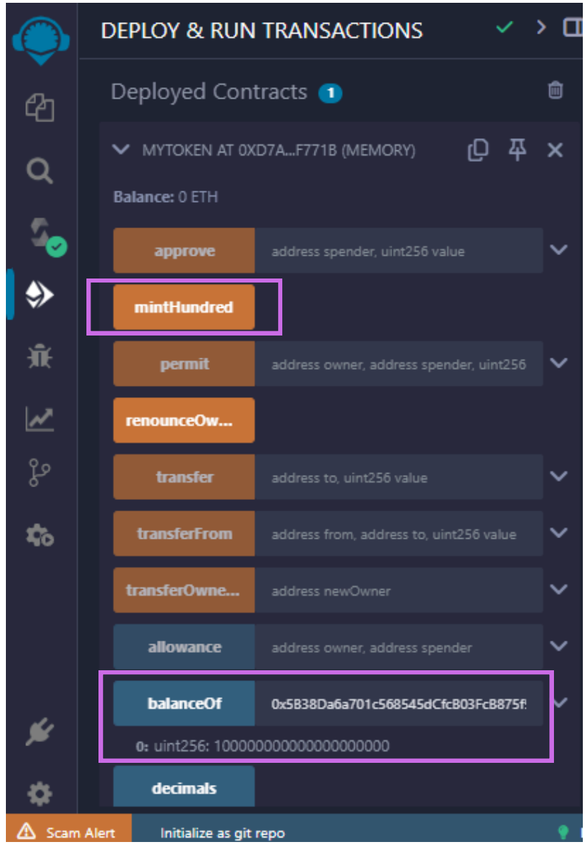
Home | MyToken.sol | Ownable.sol

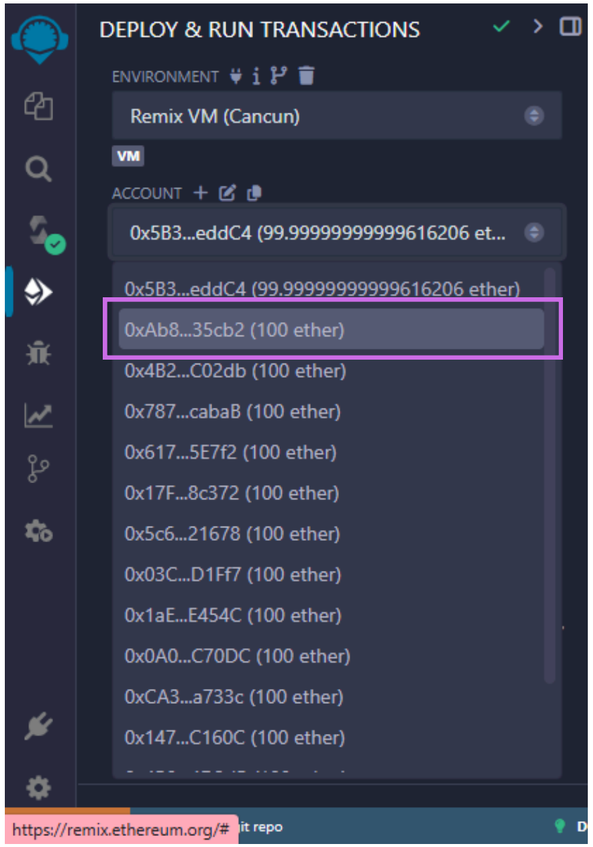
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract MyToken is ERC20, ERC20Permit, Ownable {

    constructor(address initialOwner) ERC20("MyToken", "MTK") ERC20Permit("MyToken")
    Ownable(initialOwner) {}

    function mintHundred() public onlyOwner {    infinite gas
        _mint(msg.sender, 100 * 10**18);
    }
}
```

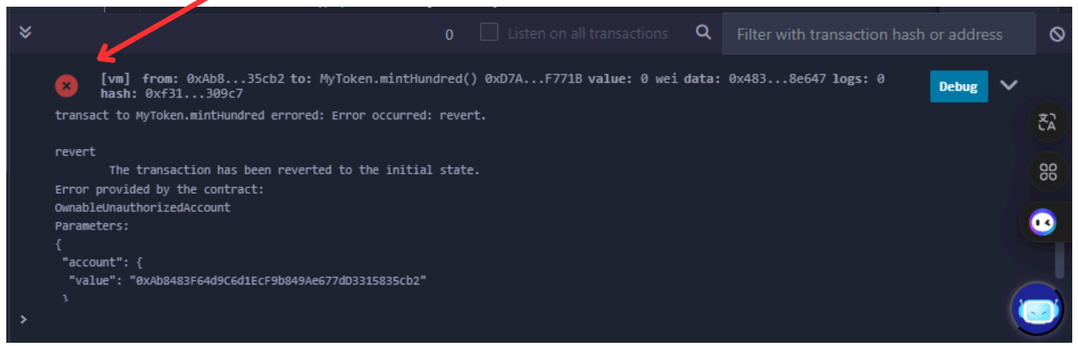
Create Token

@maryemhadjwannes

6- When we mint tokens for the owner of the contract, 100 tokens are successfully minted. However, if we try to switch to another address and attempt to mint, it will return an error due to the onlyOwner modifier, which restricts the minting function to the contract owner only.



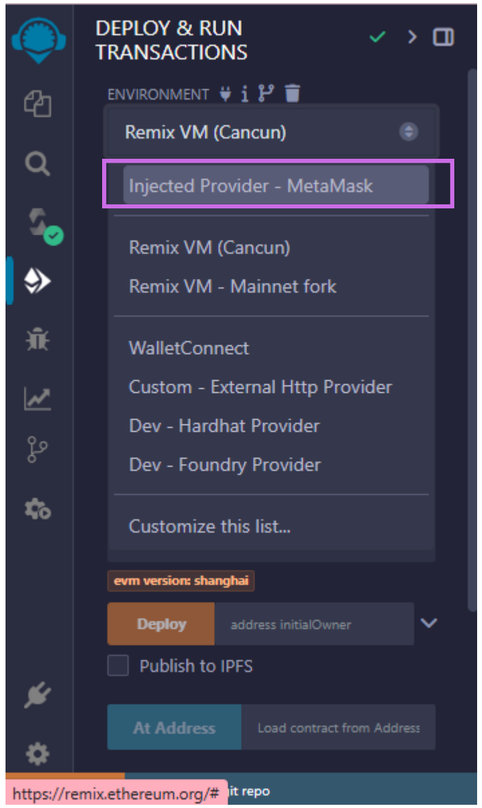


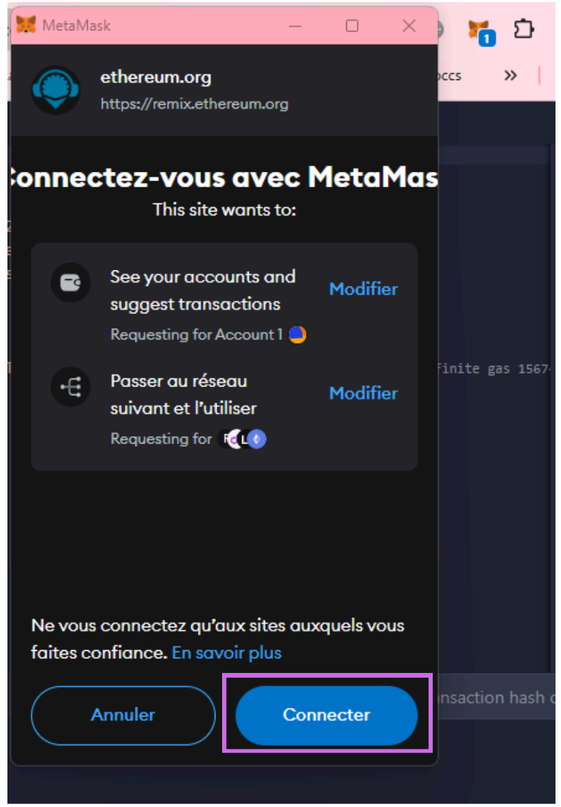


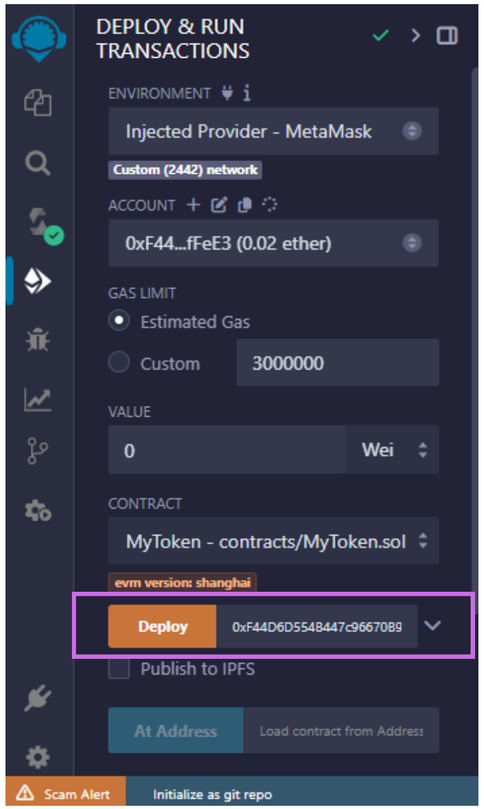
Create Token

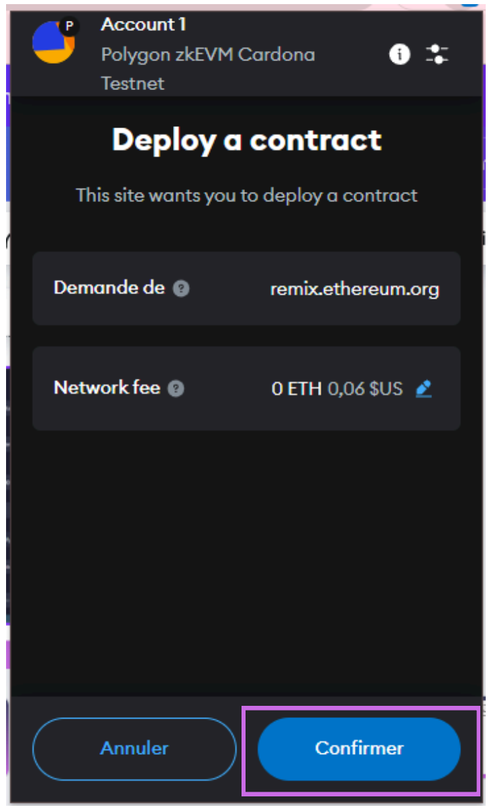
@maryemhadjwannes

7- Now, let's update the web page. Go to the Deploy section, select Injected Provider (MetaMask). MetaMask will pop up, prompting you to connect your wallet. After connecting, click Deploy. MetaMask will ask for confirmation to deploy the contract, and once approved, your token will be successfully launched!





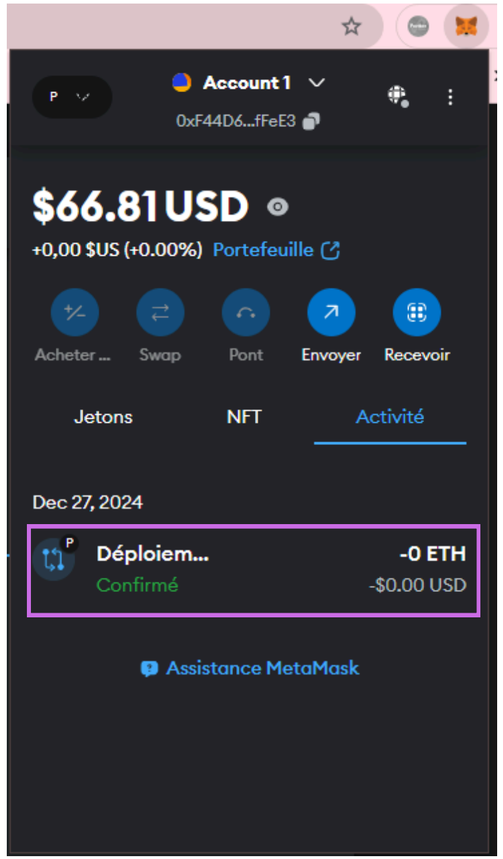


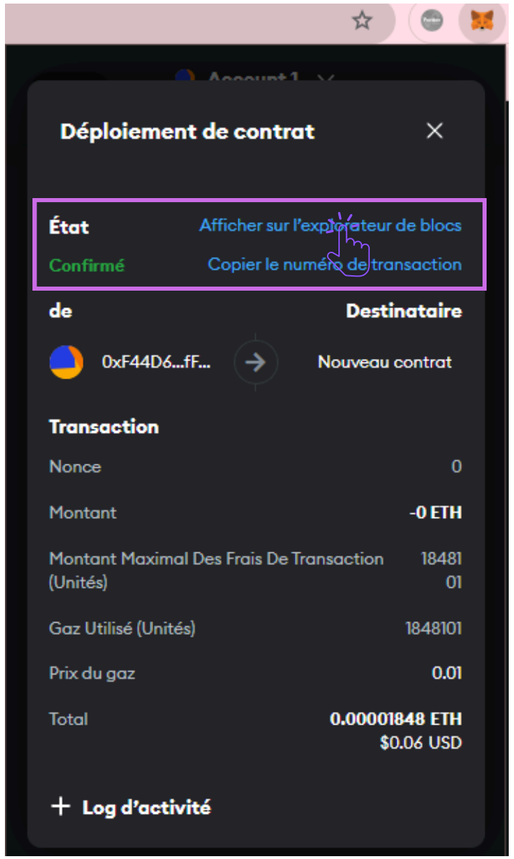


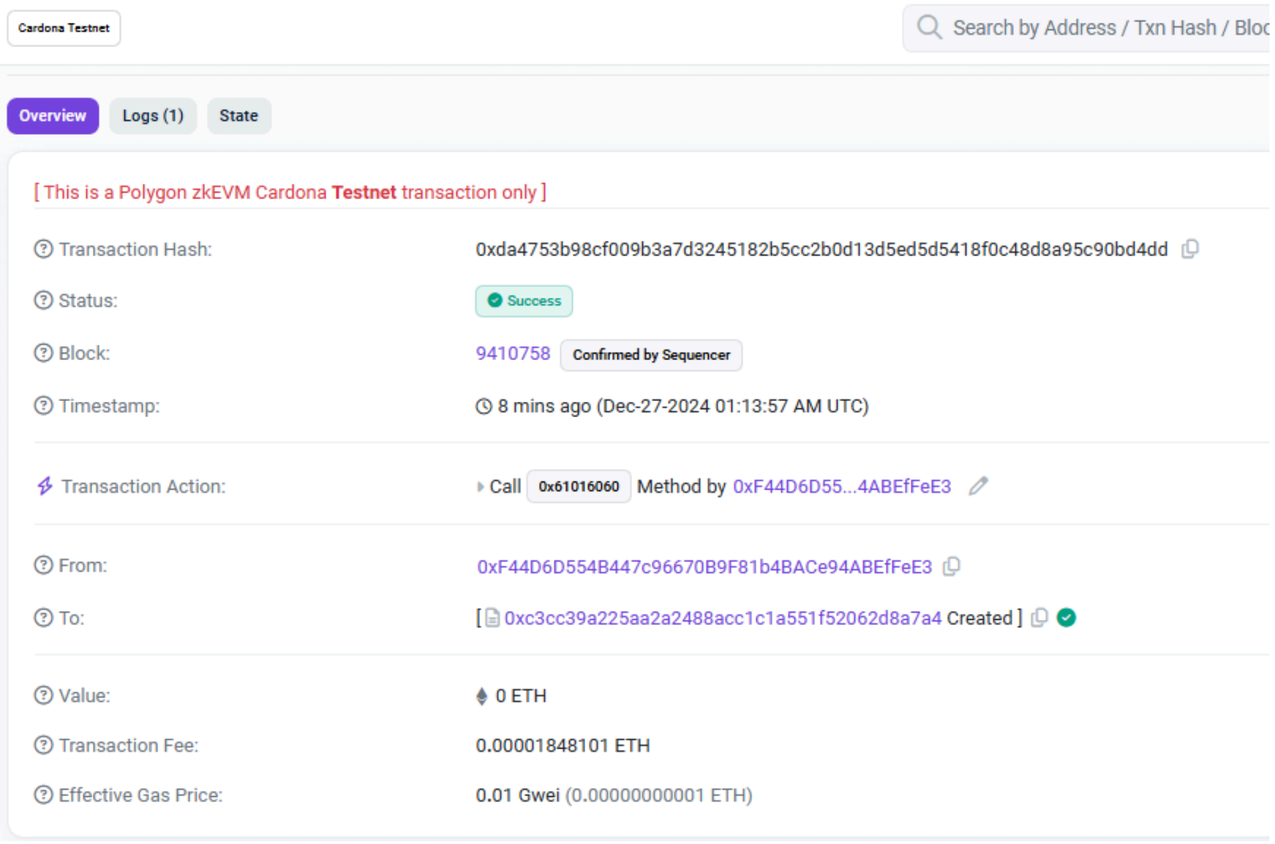
Create Token

@maryemhadjwannes

8- In MetaMask's transaction history, you can click on the deployment transaction. This will take you to PolygonScan (on the testnet), where you can view detailed information about the transaction, including the deployment transaction fee and the address that created the contract.



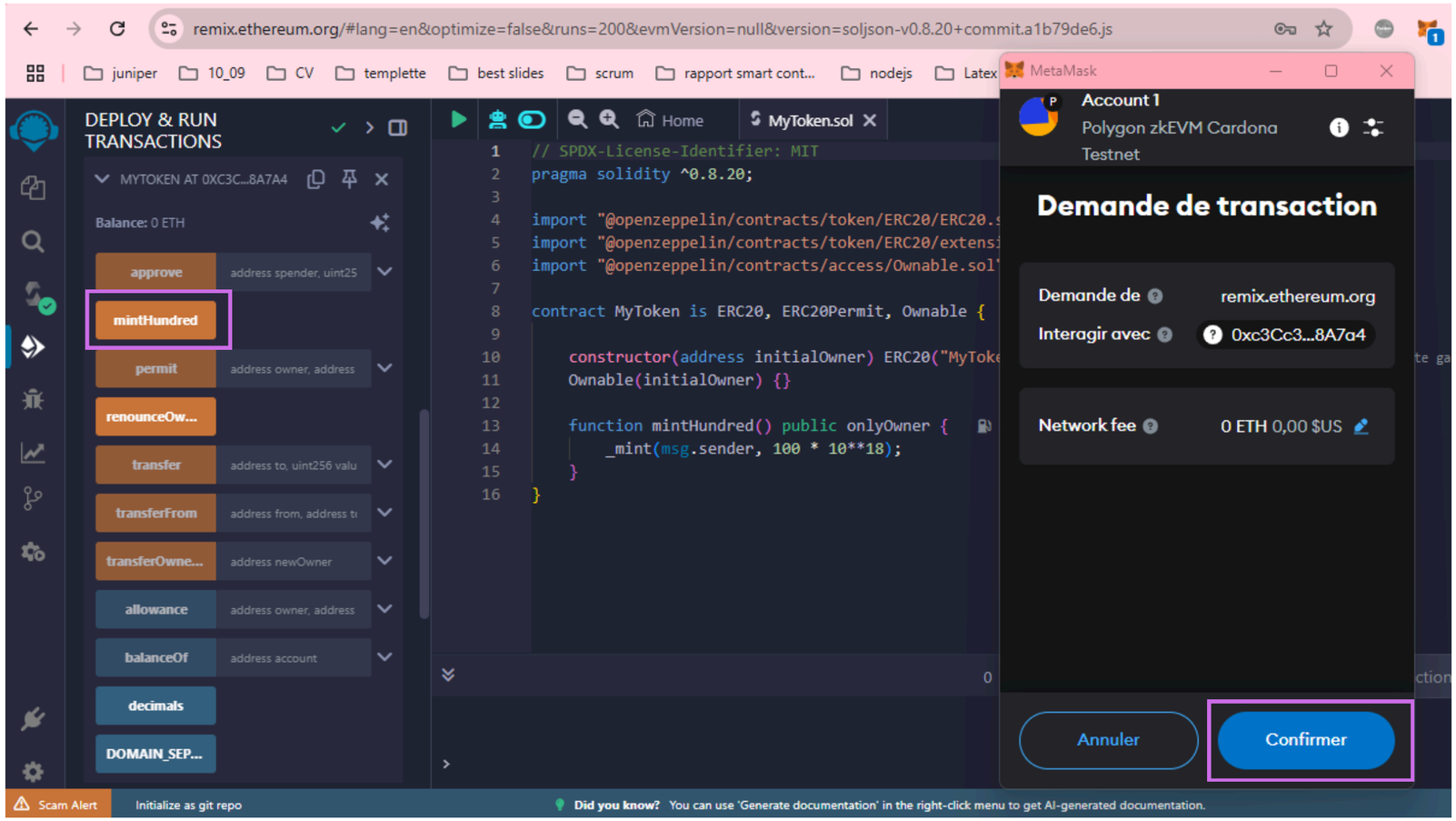


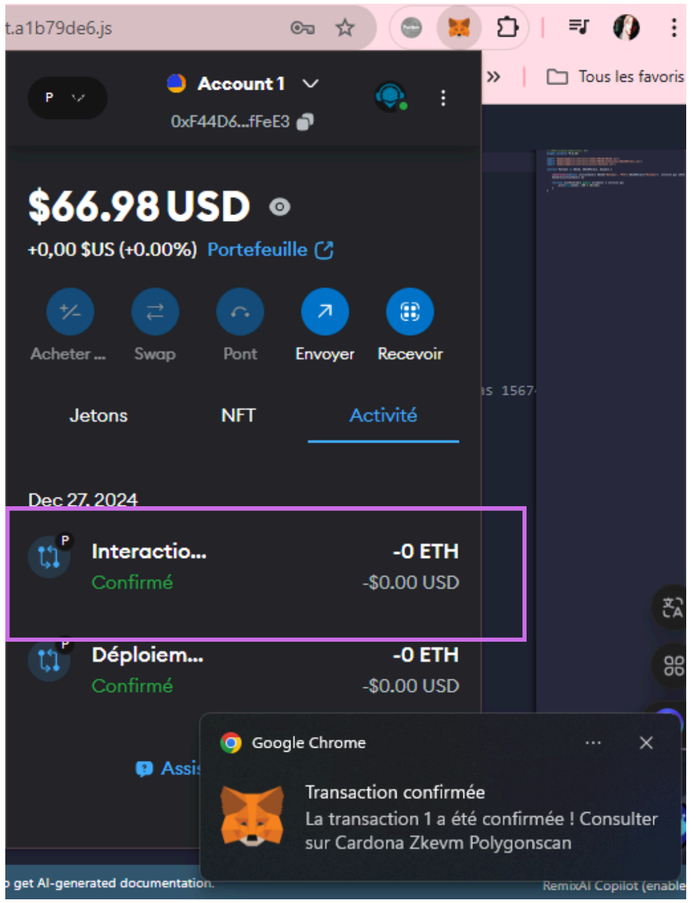


Create Token

@maryemhadjwannes

9- Since PolygonScan doesn't recognize it as a token yet, return to Remix and call the mintHundred function to mint 100 tokens.

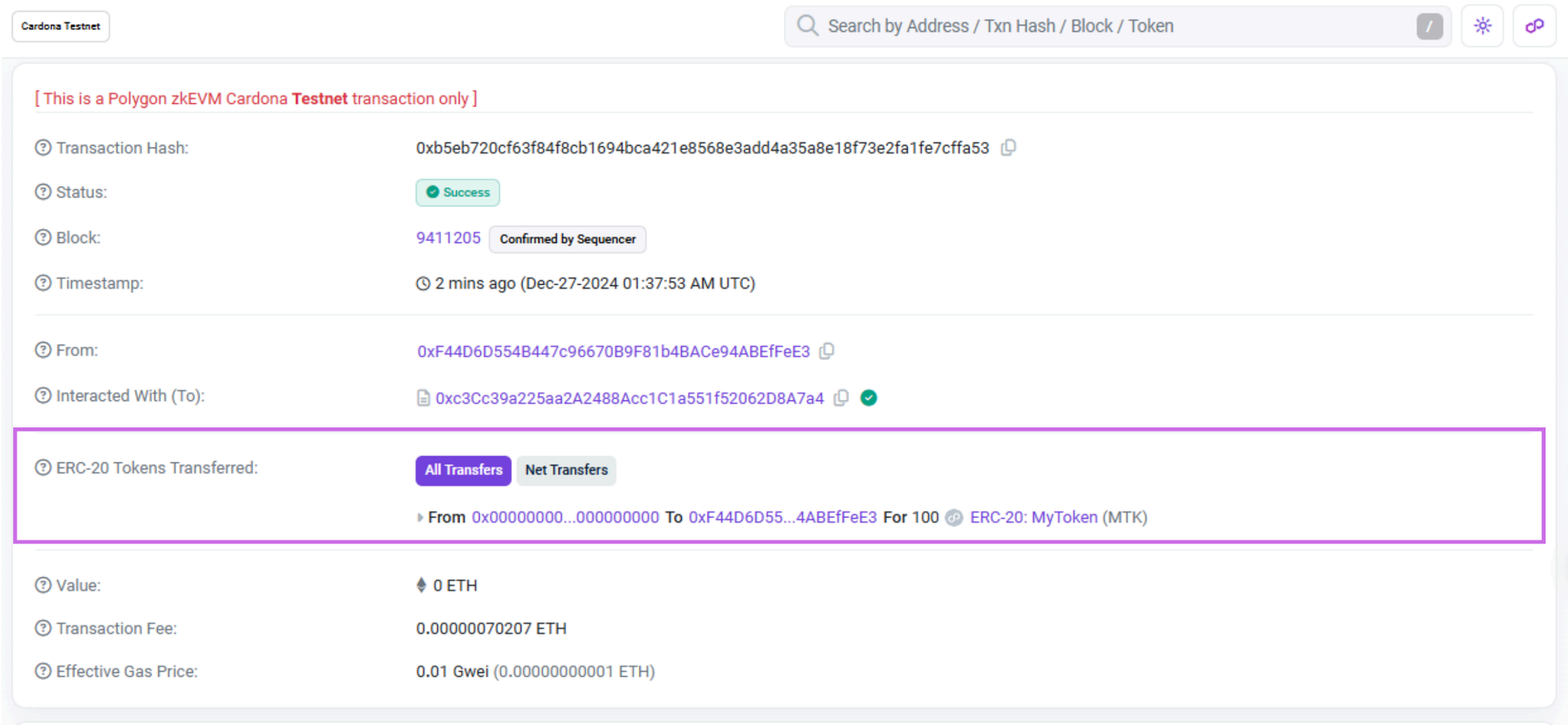




Create Token

@maryemhadjwannes

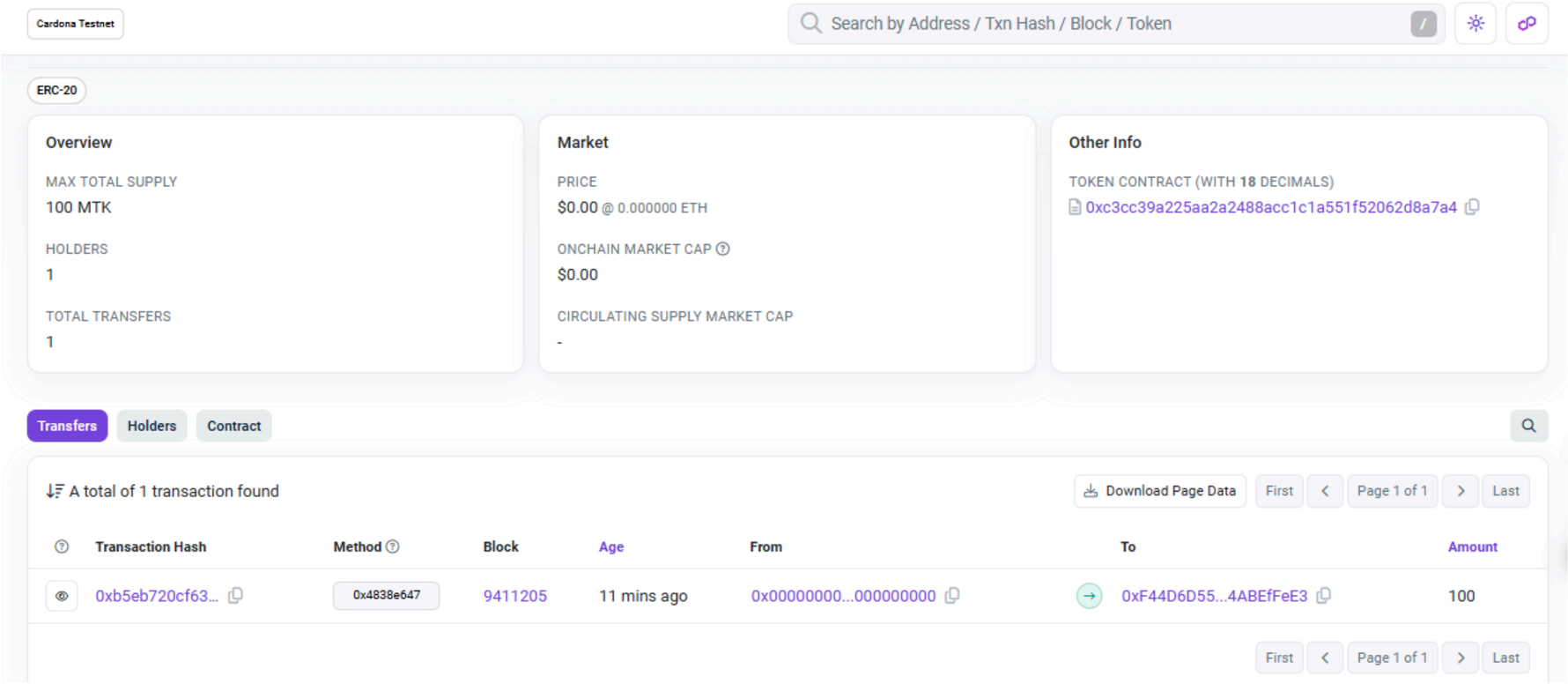
10- **After minting 100 tokens** using the mintHundred function, PolygonScan will show that 100 tokens were added to your address. This usually appears as if the tokens are coming from the null address (0x0...0), and it will display the token name as **MyToken**, the symbol as **MTK**, and the type as **ERC20**.



Create Token

@maryemhadjwannes

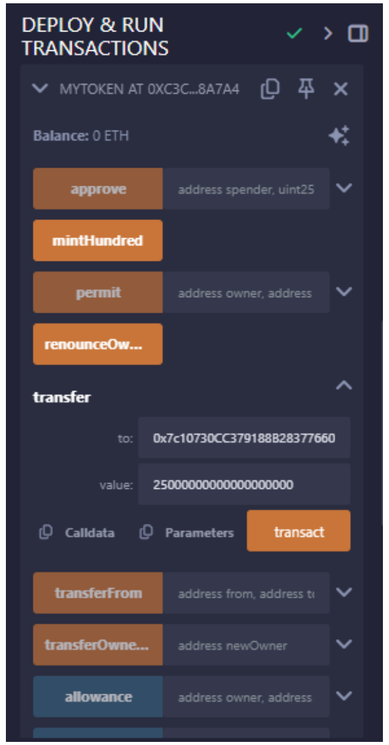
11- To display more information, the total supply of our token is 100 MTK, with 1 holder (my address) and 1 total transfer (the minting transaction). This shows that the minting process has successfully created 100 tokens and assigned them to the first holder. And of course, the market value is 0 ETH for now, Haha! But hey, it's a start! 😋

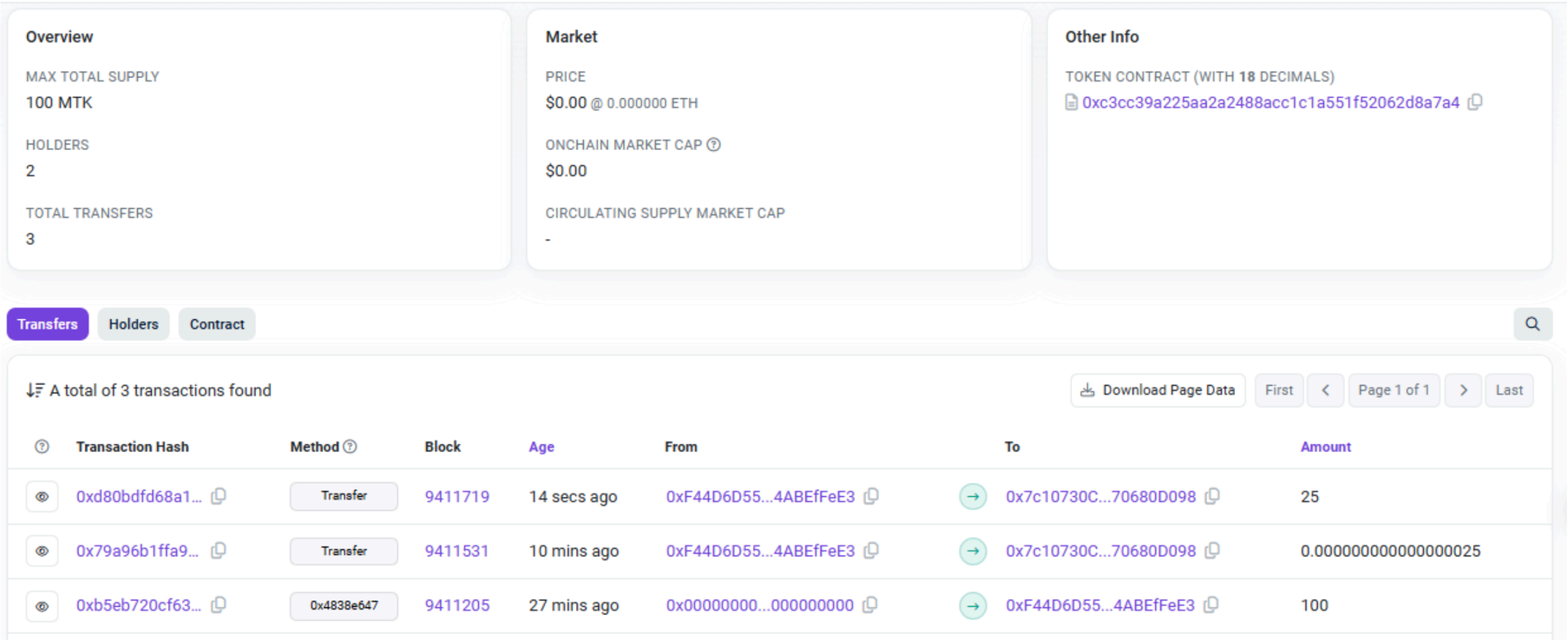


Create Token

@maryemhadjwannes

12- After transferring 25 MTK to another account, the token information updates accordingly. Now, there are 2 holders. The details show the amount held by each account: my first address holds 75 MTK, and the recipient holds 25 MTK.
Don't be surprised by 25000000000000000000 because it's in Wei, which is equal to 25 MTK.





I hope you found this tutorial helpful!