

How to Create a Cryptocurrency

Pre-requisites:

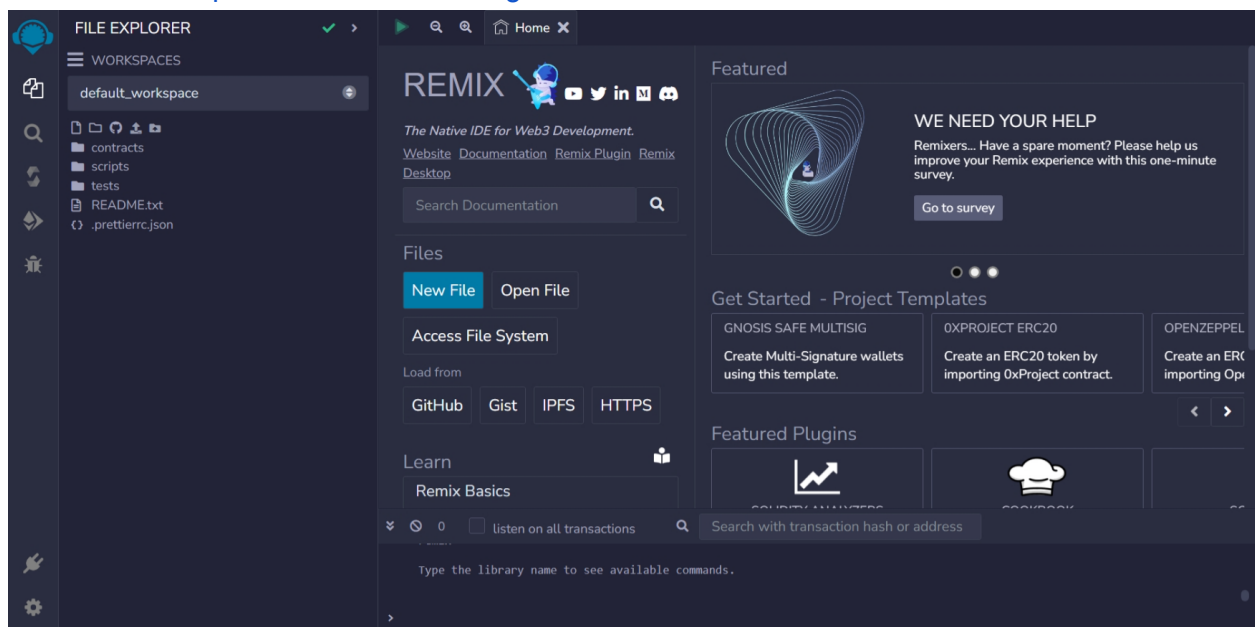
- Basic coding knowledge with Solidity
- Must have your own MetaMask Wallet. (MetaMask Wallet should be connected to Mumbai TestNet).

If you are unable to meet this prerequisite please see:

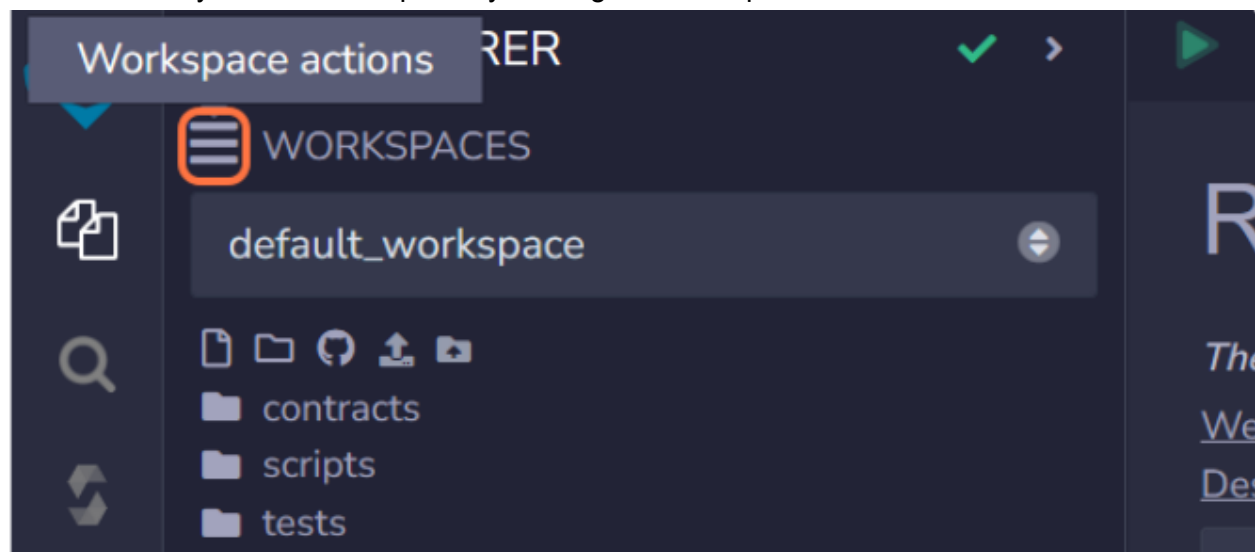
[Create a MetaMask Account - Mumbai](#)

Step 1: Coding your first token

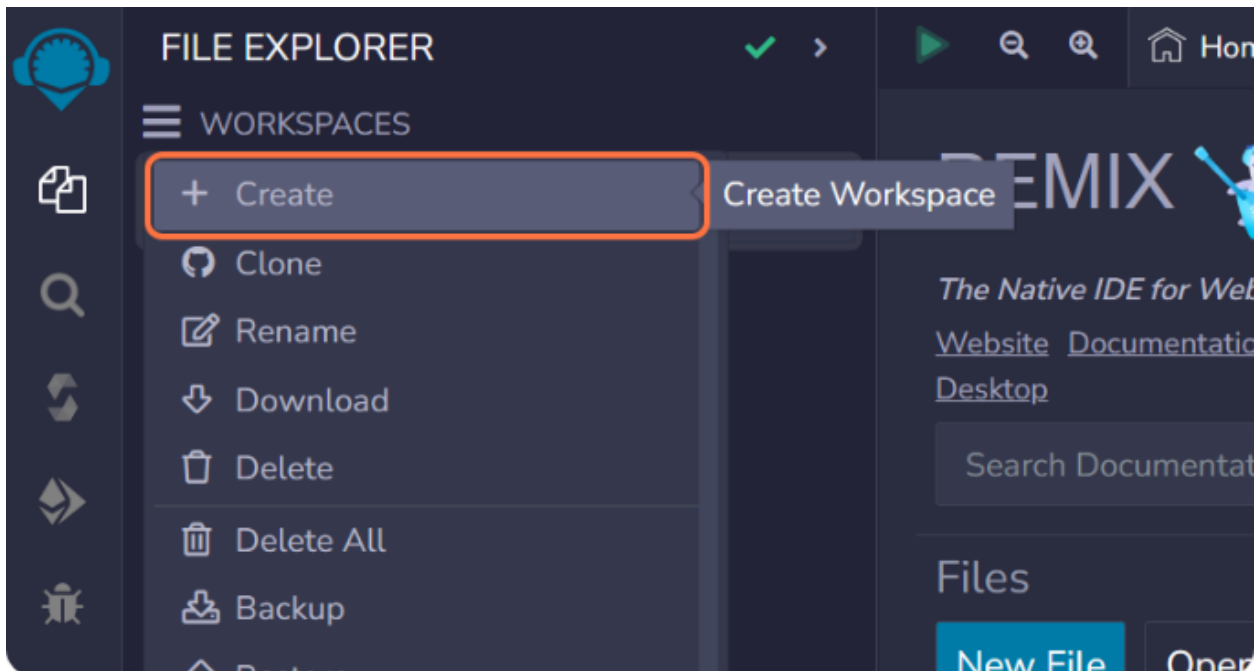
1. Go to <https://remix.ethereum.org/>



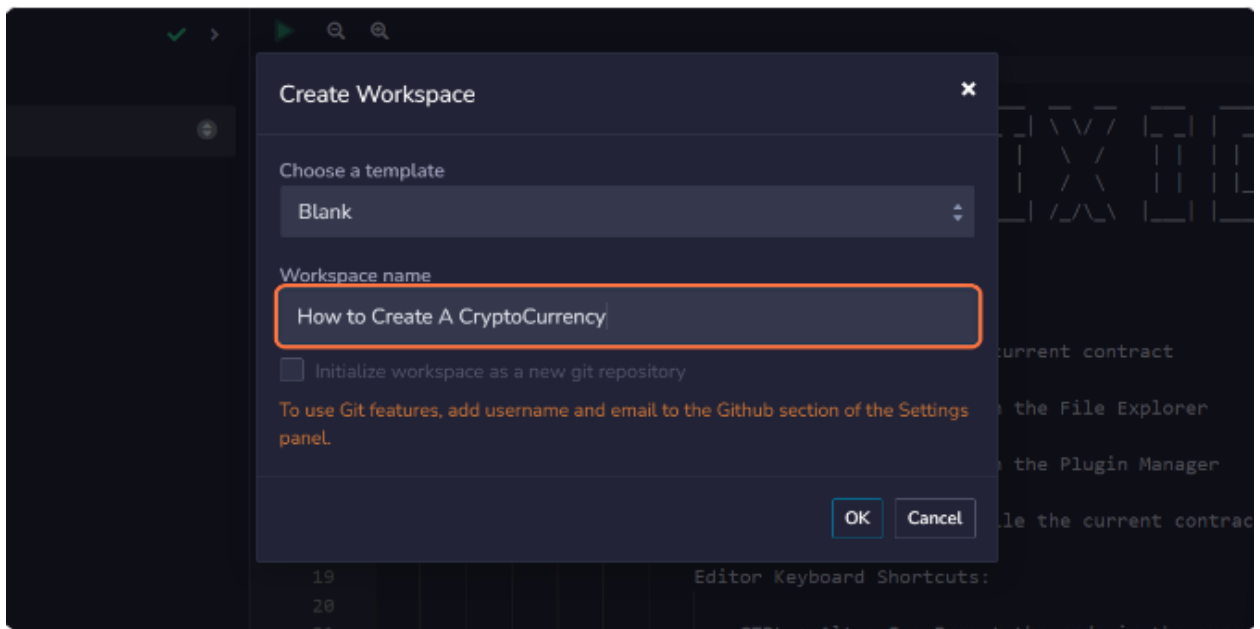
2. Create your own workspace by clicking the Workspace Actions button



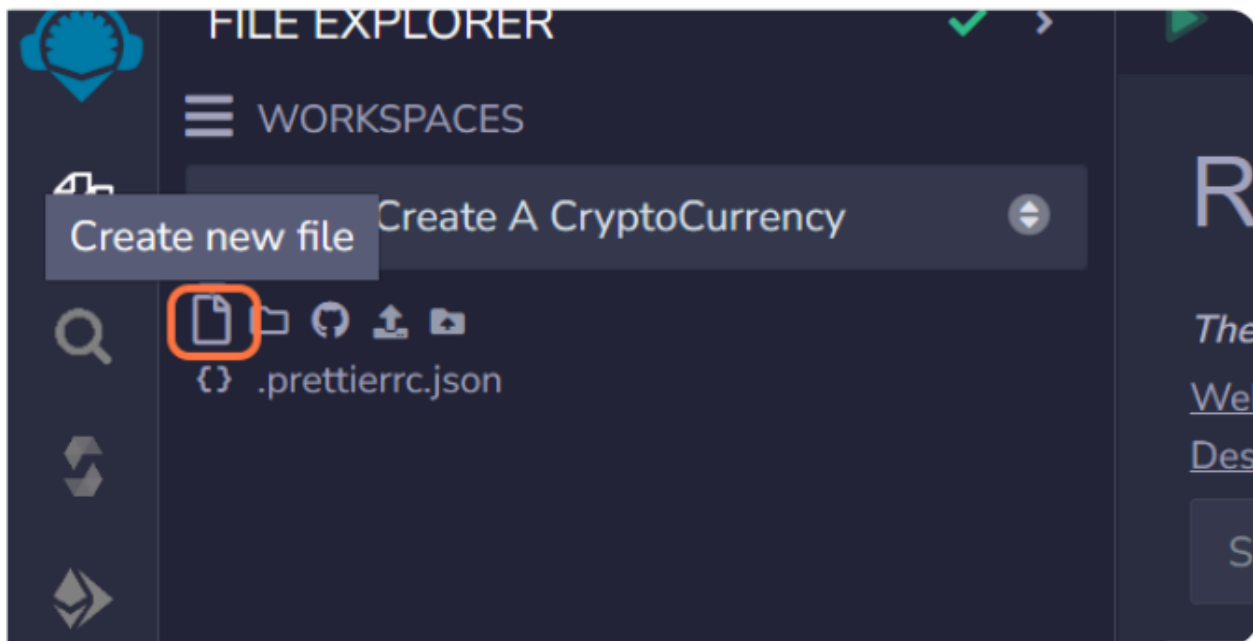
3. Choose the **Create** button.



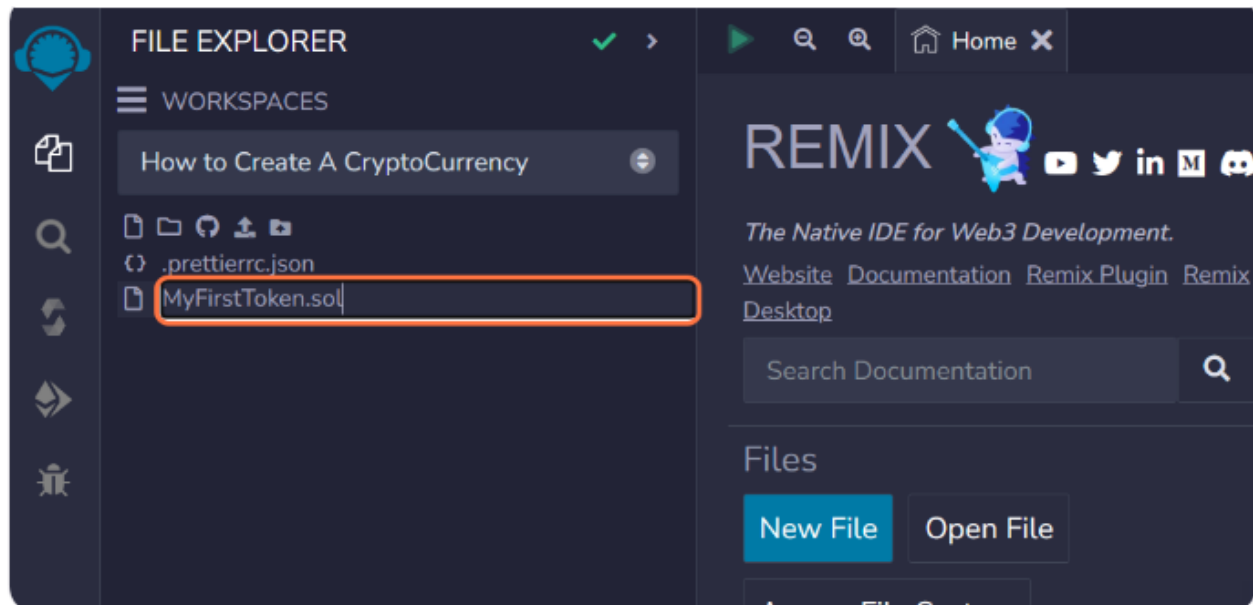
4. Select the **Blank** template and enter the name of your workspace. Then hit the **OK** button.



5. Create a new file by clicking the file icon



6. Enter your file name. Make sure it sends in .sol extension.

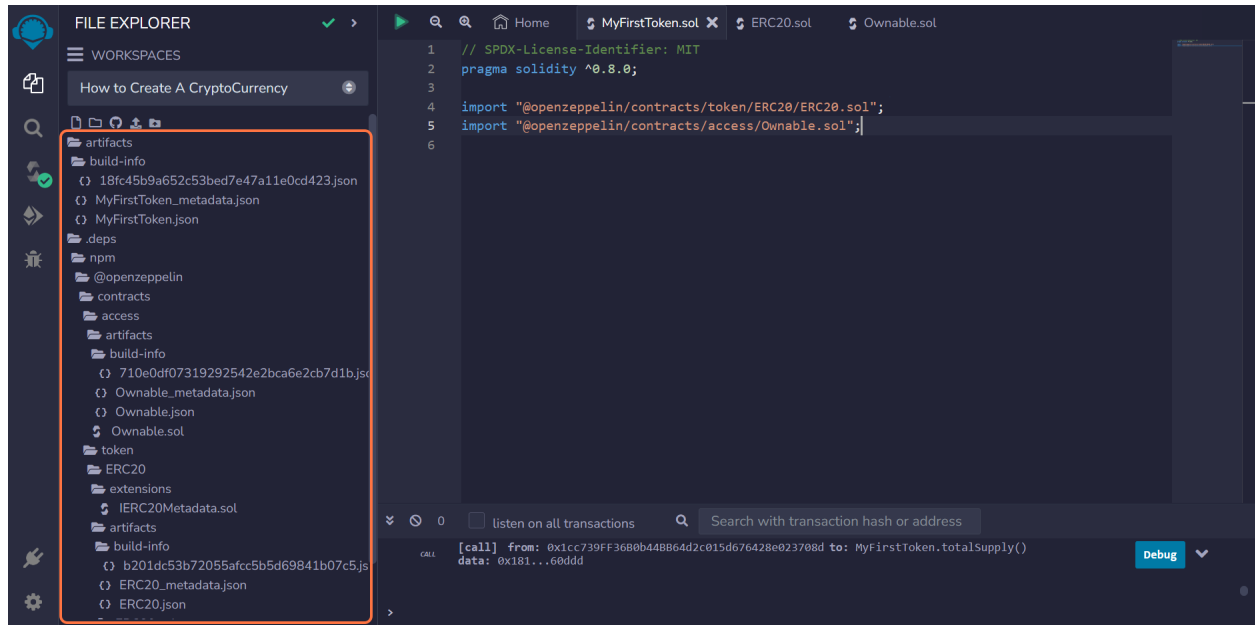


7. Write the license identifier and the version. The license identifier here would be `SPDX-License-Identifier: MIT` and the version is `pragma solidity ^0.8.0;`

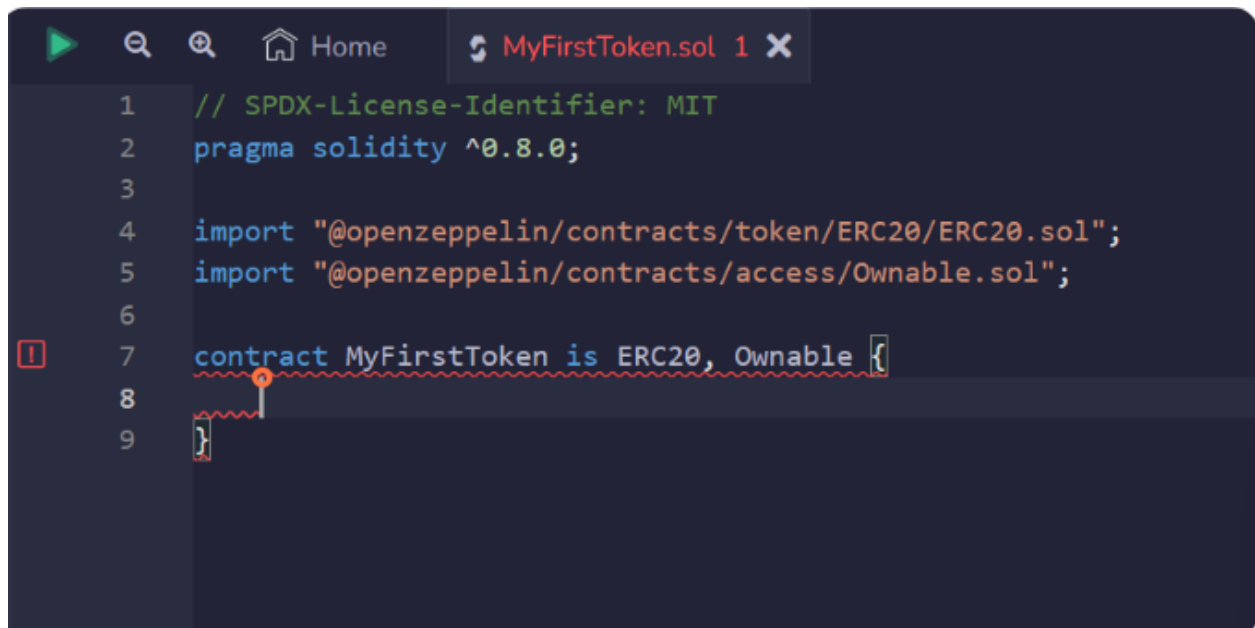
A screenshot of a code editor with a dark theme. The editor shows a file named `MyFirstToken.sol`. The first two lines of code are: `// SPDX-License-Identifier: MIT` and `pragma solidity ^0.8.0;`. The cursor is positioned at the end of the second line. On the left side, there is a sidebar with a search bar and a list of files, including `PhotoCurrency` and `7c425371117c18047.json`. The editor has a toolbar at the top with icons for running, searching, and navigating.

8. Import **ERC20** and **Ownable**. Notice that there are some files and folders that are added inside our files section.

A screenshot of the same code editor, now showing the addition of two import statements. The code is: `import "@openzeppelin/contracts/token/ERC20/ERC20.sol";` and `import "@openzeppelin/contracts/access/Ownable.sol";`. The cursor is at the end of the second import statement. On the left sidebar, the file explorer now shows additional files and folders, including `@openzeppelin` and `contracts`. The editor has a toolbar at the top with icons for running, searching, and navigating.



9. Create a contract and inherit ERC20 and Ownable



10. Create a constructor that makes a new ERC20 token named "OCC Token" with the symbol "OCC" and mints a certain amount of tokens to the address that deploys the account..

```

3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5 import "@openzeppelin/contracts/access/Ownable.sol";
6
7 contract MyFirstToken is ERC20, Ownable {
8     constructor() ERC20("OCC Token", "OCC"){
9         _mint(msg.sender, 10 * 10 ** 18);
10    }
11
12
13 }

```

This is the code for the mint function. To see it, CTRL + Click `_mint()`

The screenshot shows the VS Code interface with the following components:

- File Explorer:** Displays the project structure, including folders like `artifacts`, `build-info`, `contracts`, `access`, `token`, and `ERC20`. The `ERC20` folder is expanded, showing `ERC20Metadata.sol` and `ERC20.json`.
- Source Explorer:** Shows the `MyFirstToken.sol` file, which is the active file in the editor.
- Code Editor:** Displays the implementation of the `_mint` function. The function is marked as `internal virtual` and takes `address account` and `uint256 amount` as parameters. It includes a `require` statement to ensure the account is not the zero address, followed by a call to `_beforeTokenTransfer`, an update to `_totalSupply`, and an `emit Transfer` event. The function is decorated with `@dev Destroys 'amount' tokens from 'account', reducing the total supply.`
- Call Stack:** Shows a call to `_mint` from the constructor of `MyFirstToken`, with the transaction hash `0x1cc739FF3680b448B64d2c015d676428e023708d` and the data `0x181...60ddd`.
- Debug Console:** A `Debug` button is visible at the bottom right of the call stack.

11. Create a **mint()** function that allows the owner of the contract to mint a specified amount of tokens.
- This takes an unsigned integer which represents the amount of tokens as a parameter and the function is public.
 - This function will be modified by the **onlyOwner** modifier.
 - This mint function will call the **_mint()** function of the ERC20 library. It will take the sender's address, and the amount from the **mint()** function.

```
6
7 contract MyFirstToken is ERC20, Ownable {
8     constructor() ERC20("OCC Token", "OCC"){
9         _mint(msg.sender, 10 * 10 ** 18);
10    }
11
12    function mint(uint256 amount) public onlyOwner{
13        _mint(msg.sender, amount);
14    }
15
16
17 }
```

12. Create a **burn()** function that allows the owner of the contract to mint a specified amount of tokens.
- This takes an unsigned integer which represents the amount of tokens as a parameter and the function is public.
 - This function will be modified by the **onlyOwner** modifier.
 - This mint function will call the **_burn()** function of the ERC20 library. It will take the sender's address, and the amount from the **burn()** function.

```
9      _mint(msg.sender, 10 * 10 ** 18);
10  }
11
12  function mint(uint256 amount) public onlyOwner{
13      _mint(msg.sender, amount);
14  }
15
16  function burn(uint256 amount) public onlyOwner{
17      _burn(msg.sender, amount);
18  }
19
20 }
```

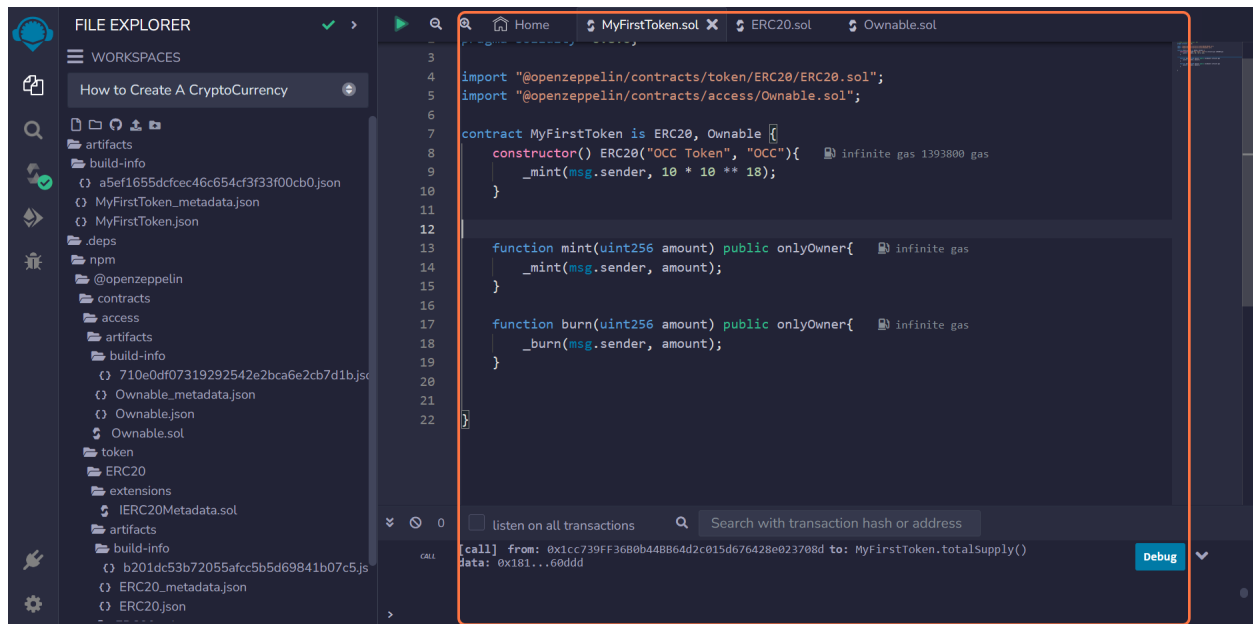
*This is the code for **_burn()** function. To see it, CTRL + **_burn()***

The screenshot shows the Visual Studio Code interface. On the left is the File Explorer showing a workspace named 'How to Create A Cryptocurrency'. The main editor area displays the code for the `_burn` function from the ERC20 library. The code includes comments and logic for burning tokens, such as checking the account address, ensuring the balance is sufficient, and updating the account balance and total supply. A call stack at the bottom shows a call from `MyFirstToken.sol` to `MyFirstToken.totalSupply()` with a data parameter.

```
273  * - 'account' cannot be the zero address.
274  * - 'account' must have at least 'amount' tokens.
275
276  function _burn(address account, uint256 amount) internal virtual {
277      require(account != address(0), "ERC20: burn from the zero address");
278
279      _beforeTokenTransfer(account, address(0), amount);
280
281      uint256 accountBalance = _balances[account];
282      require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
283      unchecked {
284          _balances[account] = accountBalance - amount;
285          // Overflow not possible: amount <= accountBalance <= totalSupply.
286          _totalSupply -= amount;
287      }
288
289      emit Transfer(account, address(0), amount);
290
291      _afterTokenTransfer(account, address(0), amount);
292  }
293
294  /**
295   * @dev Sets 'amount' as the allowance of 'spender' over the 'owner' s tokens.
296   */
```

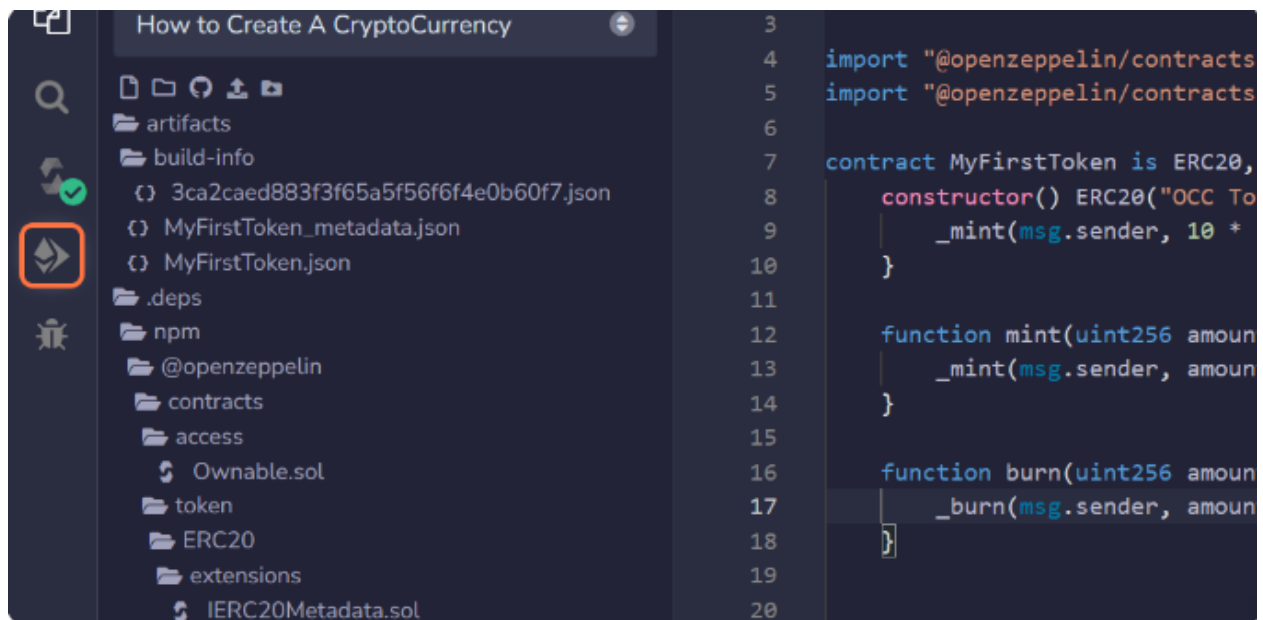
CALL [call] from: 0x1cc739FF3680b44BB64d2c015d676428e023708d to: MyFirstToken.totalSupply()
Data: 0x181...60ddd

13. This is how the final code should look like

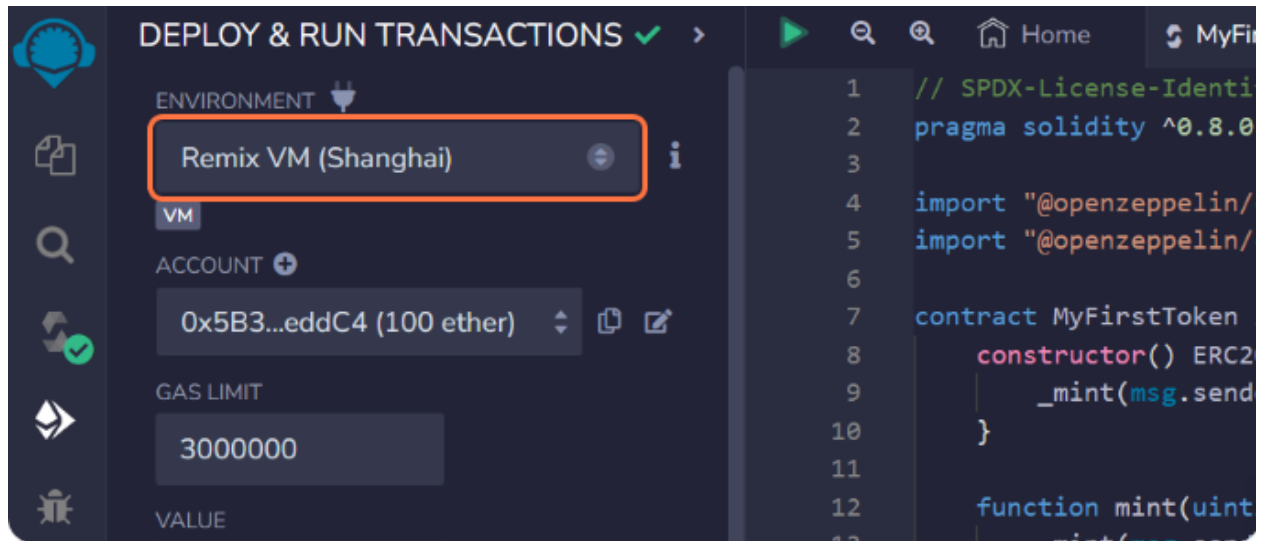


Step 2: Deploying and Testing your first token.

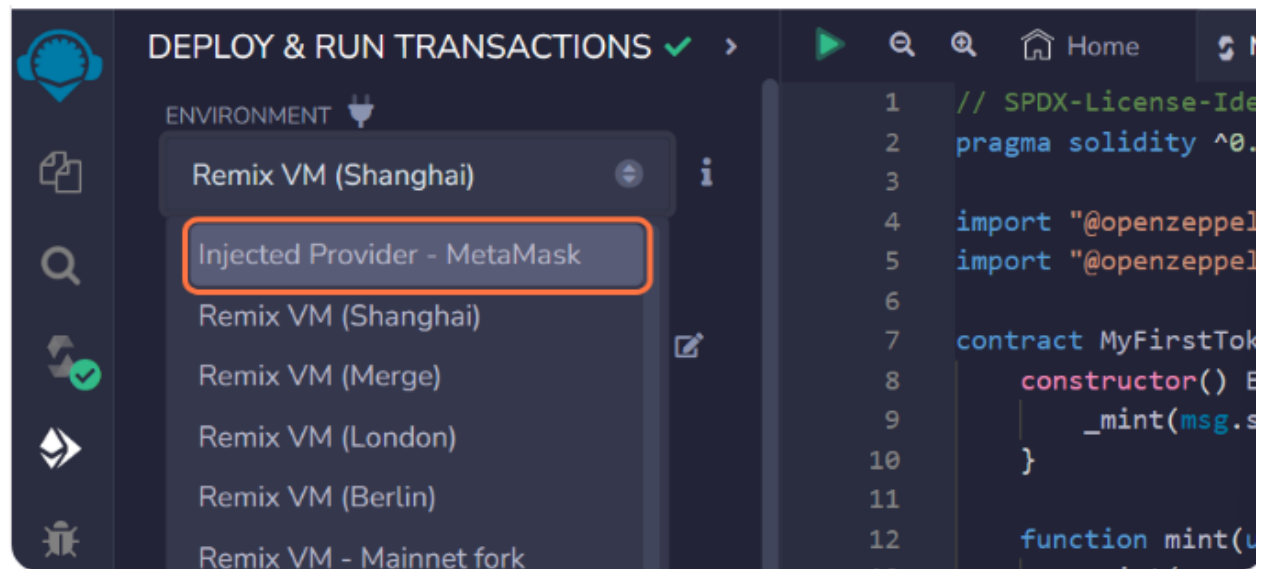
1. Go to the right side navigation bar And click on the **Deploy and Run Transactions** Button.



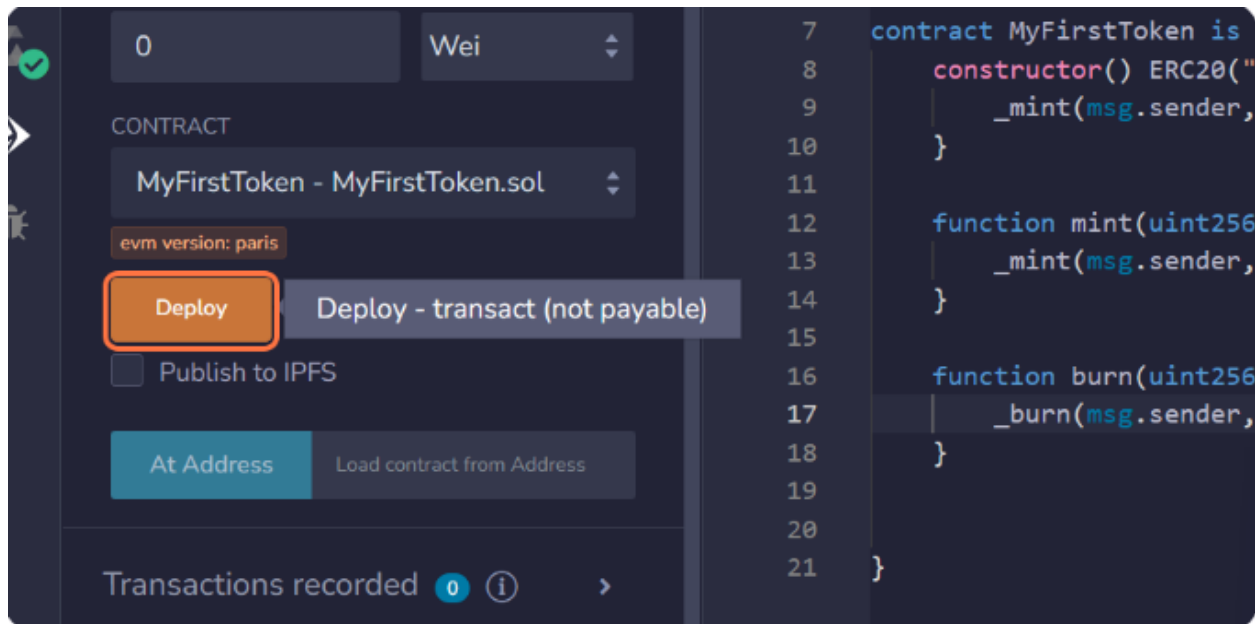
2. Click the box below the word ENVIRONMENT.



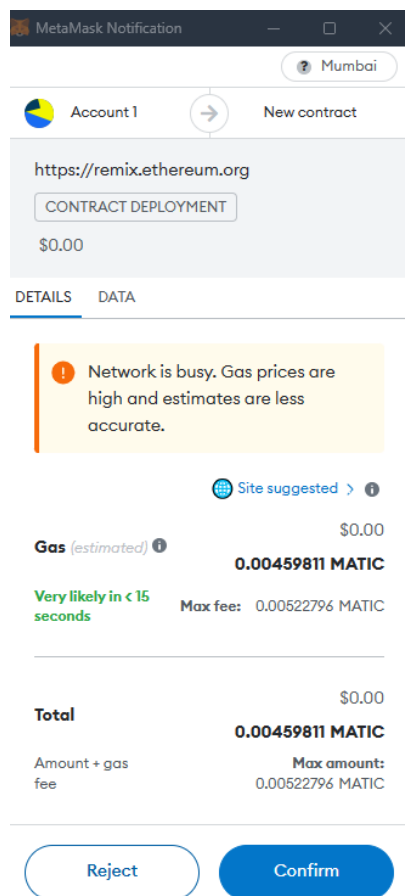
3. Choose **Injected Provider - MetaMask**.



4. Click the “Deploy” button.



5. A MetaMask prompt will open. This tells about the Contract Deployment. Click “Confirm”.



- Once you confirm, you will see inside your MetaMask Wallet a Contract Deployment Activity.

The screenshot shows the MetaMask wallet interface for 'Account 1'. The account balance is 0.1693 MATIC, equivalent to \$0.10 USD. Below the balance are buttons for Buy, Send, Swap, Bridge, and Portfolio. The 'Activity' tab is selected, showing a 'Contract deployment' transaction from Sep 7 on remix.ethereum.org, with a value of -0 MATIC (-\$0.00 USD). A detailed 'Contract deployment' modal is open on the right, showing the transaction status as 'Confirmed' and providing various transaction details.

Contract deployment	
Status	Confirmed
From	0x1cc...708d
To	New contract
Transaction	
Nonce	8
Amount	-0 MATIC
Gas Limit (Units)	1693604
Gas Used (Units)	1693604
Base fee (GWEI)	0.000380238
Priority fee (GWEI)	2.5
Total gas fee	0.004235 MATIC \$0.00 USD
Max fee per gas	0.000000003 MATIC \$0.00 USD

- Inside **Deployed Contracts**, you will see your contract name and its contract ID. Click the dropdown arrow icon beside it. Inside it you will see all the functions and variables of your contract.

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is open, showing a list of functions for the 'MyFirstToken' contract. The 'totalSupply' function is selected, and its call data is displayed. The main editor shows the Solidity code for the 'MyFirstToken' contract, which is an ERC20 token. The bottom status bar shows the transaction details for the deployment.

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

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

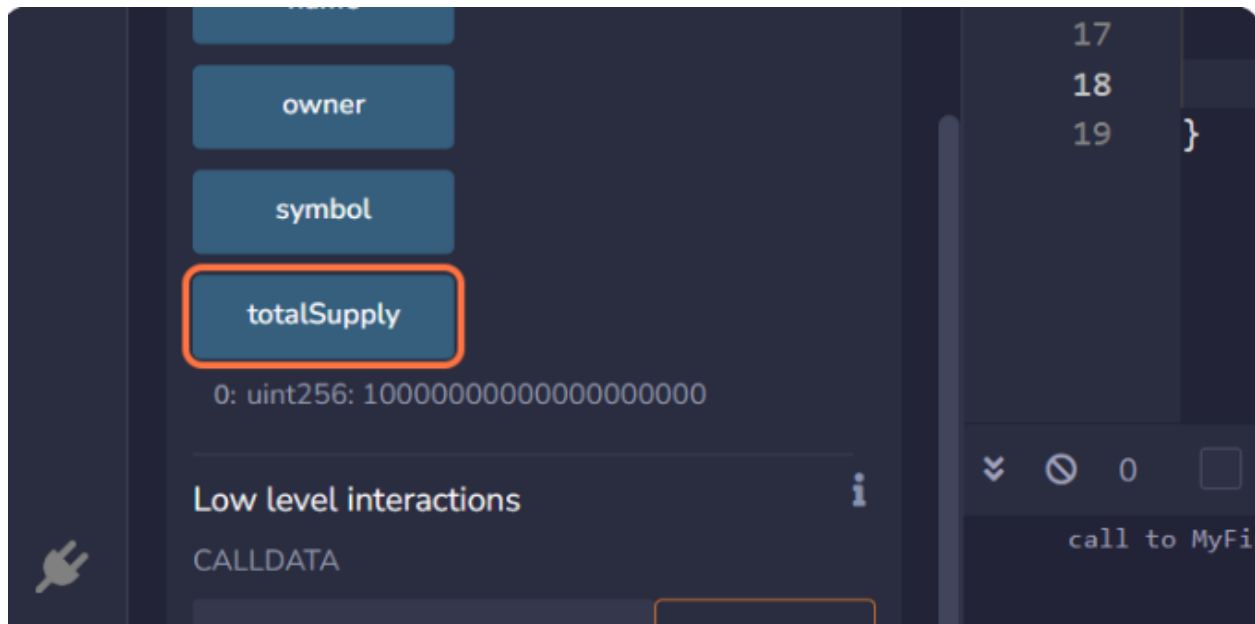
contract MyFirstToken is ERC20, Ownable {
    constructor() ERC20("OCC Token", "OCC"){
        _mint(msg.sender, 10 * 10 ** 18);
    }

    function mint(uint256 amount) public onlyOwner {
        _mint(msg.sender, amount);
    }

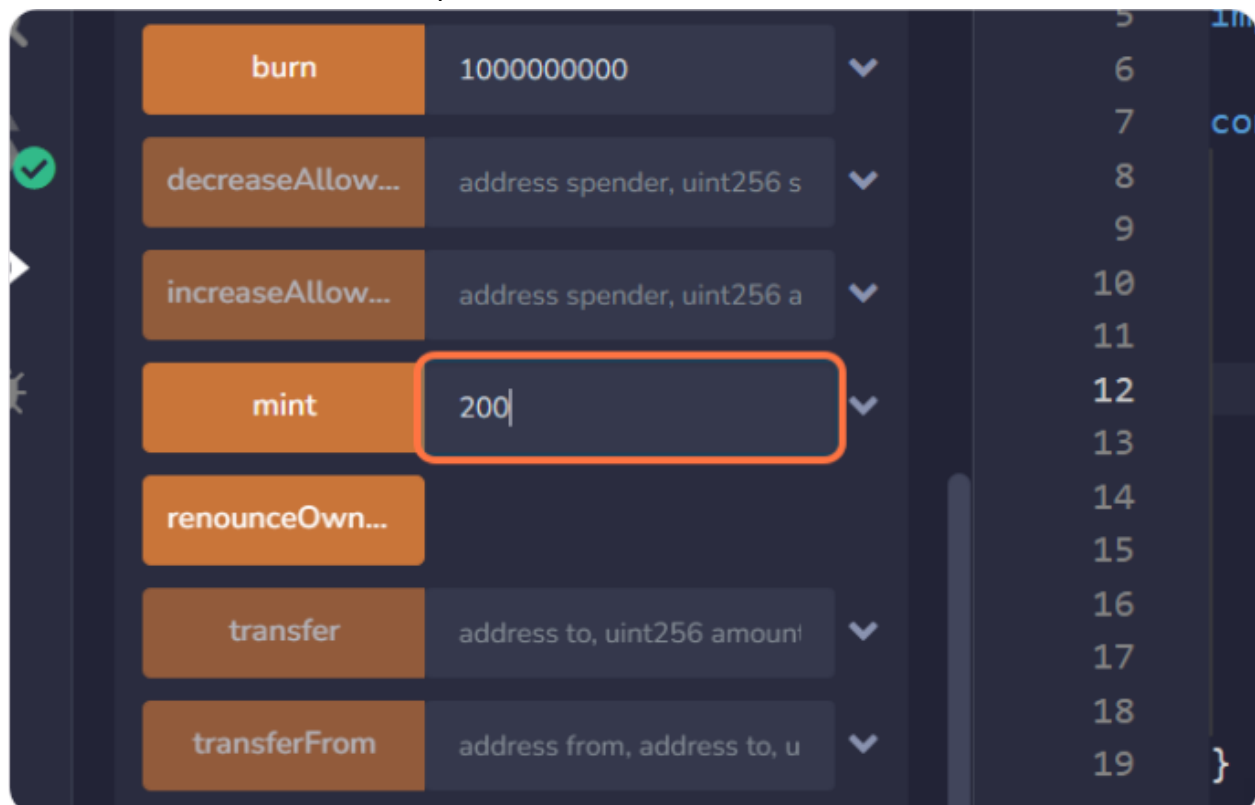
    function burn(uint256 amount) public onlyOwner {
        _burn(msg.sender, amount);
    }
}
```

Transaction details: [block:39841615 txIndex:32] from: 0x1cc...3708d to: MyFirstToken.(constructor) value: 0 wei data: 0x608...20033 logs: 2 hash: 0xf37...6755b

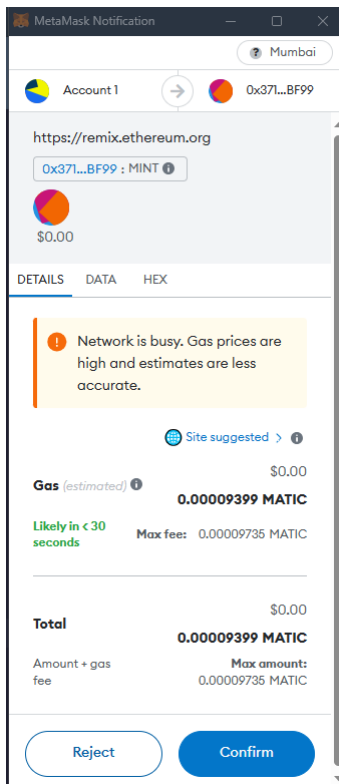
8. Click on the **“totalSupply”** button. This will tell us the amount we have.



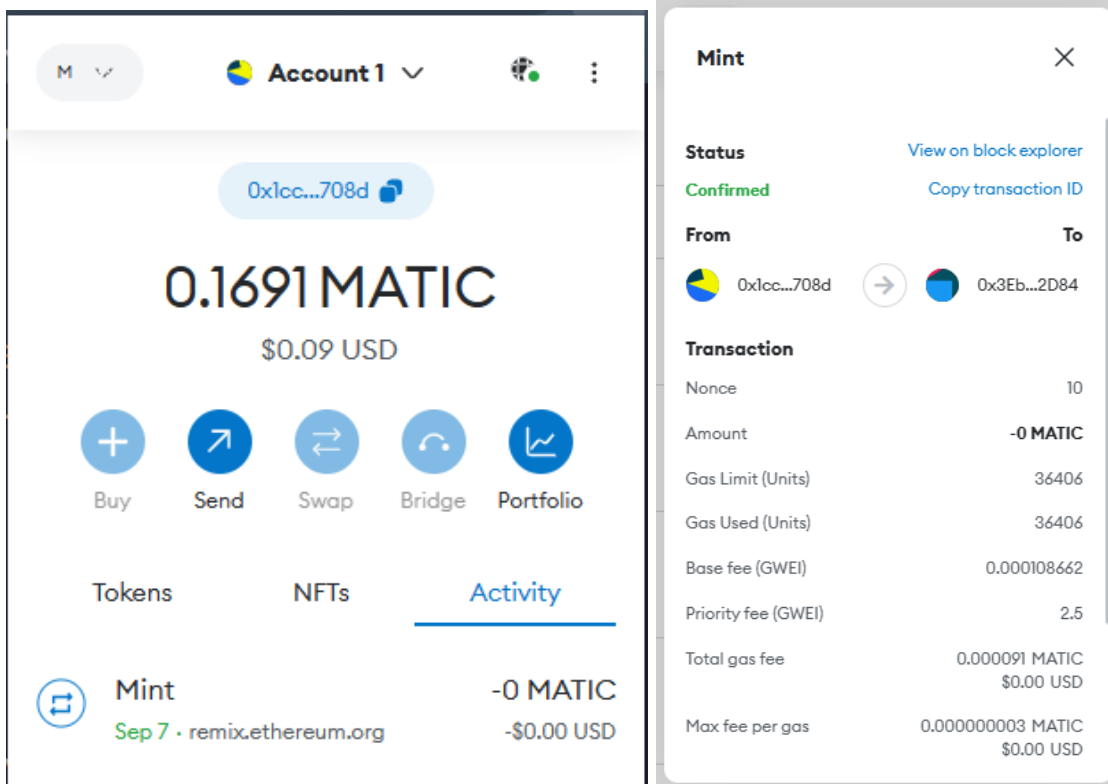
9. Enter an amount on the input field next to the **“mint”** button. Then click on **“mint”**



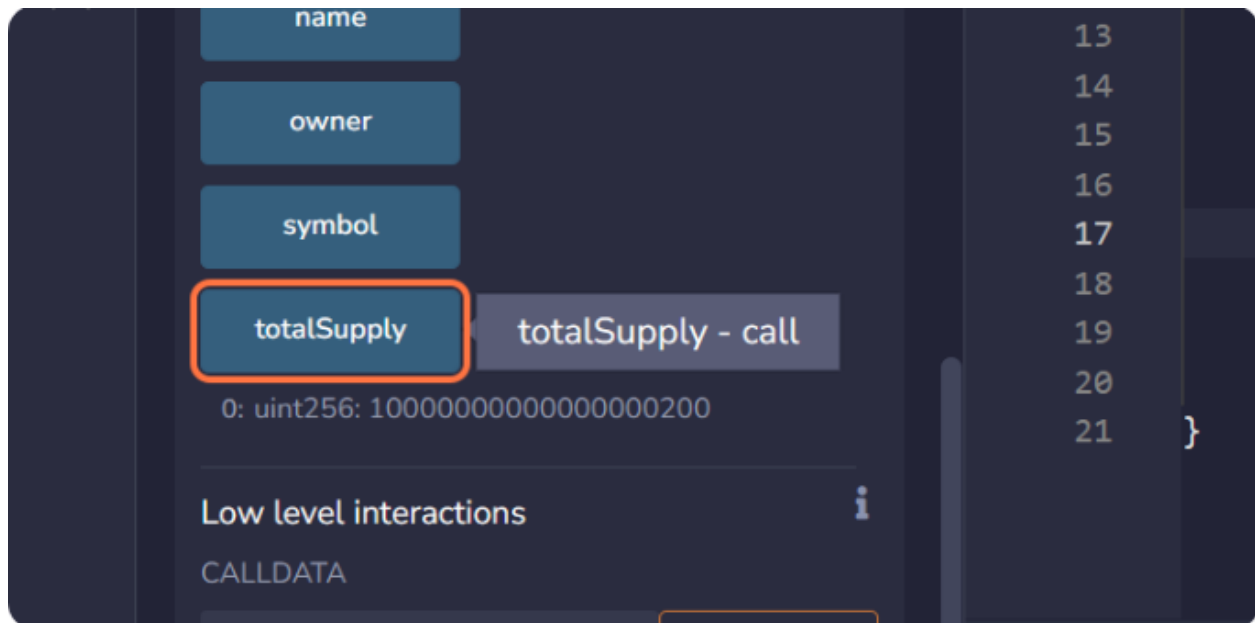
10. MetaMask will show a prompt. Click on the “**Confirm**” button.



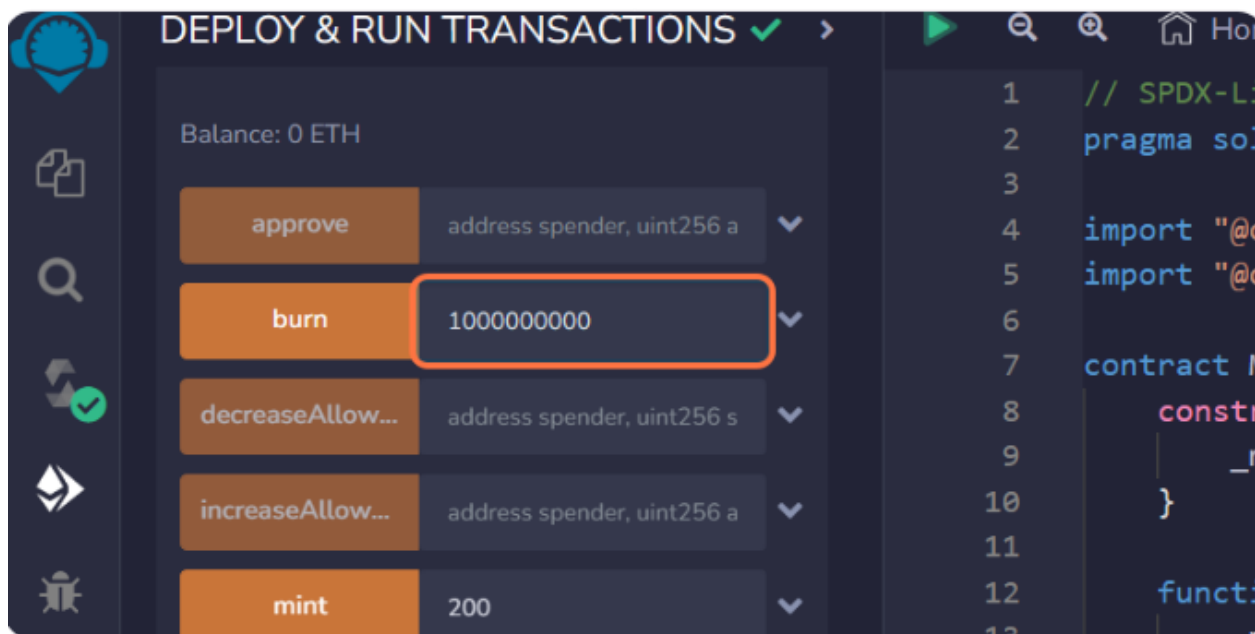
11. Once you have confirmed, you will see inside your MetaMask Wallet a Mint Activity.



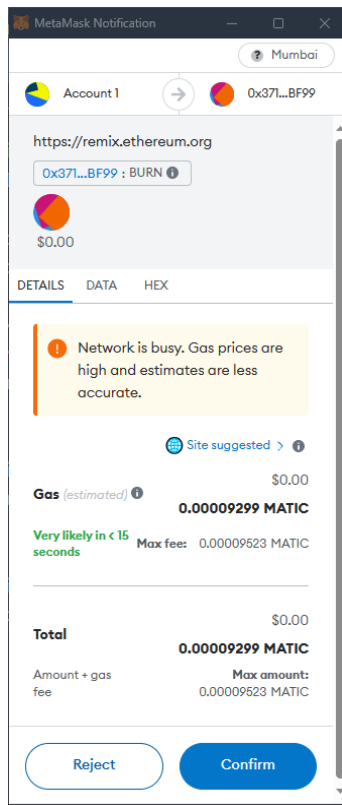
12. Click the **“totalSupply”** button. Notice that we have added the amount we entered for mint.



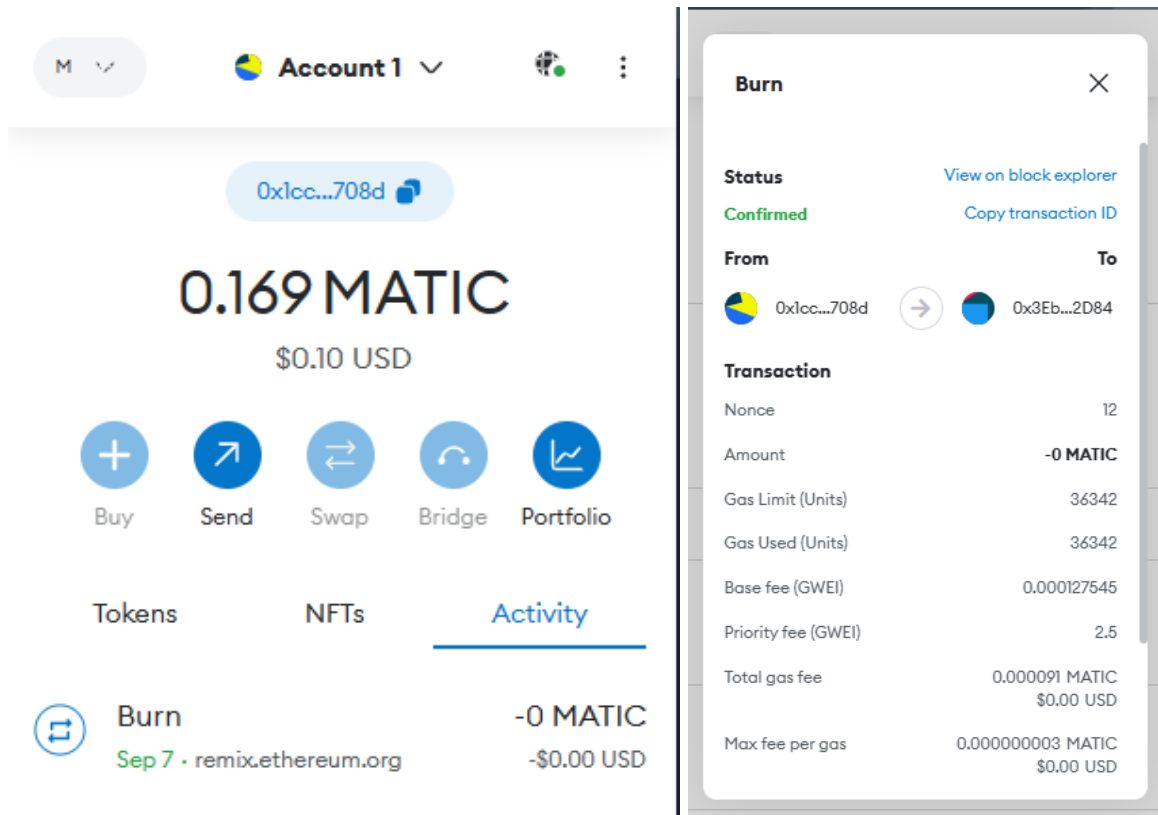
13. Enter an amount next to the **“burn”** button. Click **“burn”**.



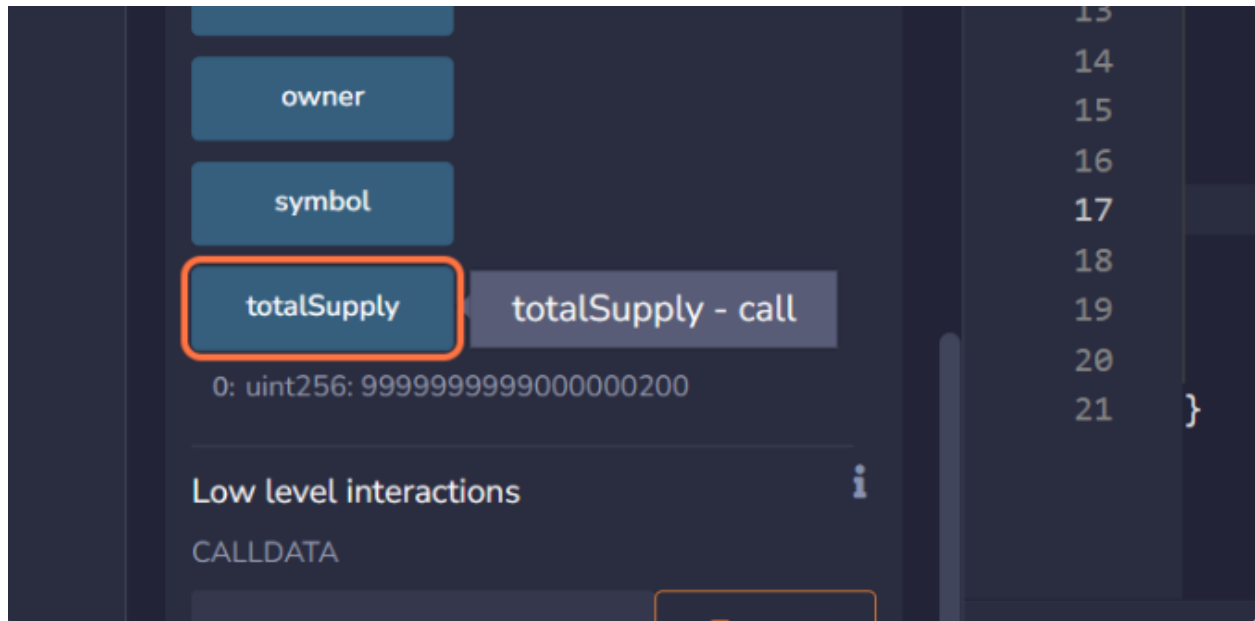
14. MetaMask will show a prompt. Click on the “**Confirm**” button.



15. Once you have confirmed, you will see inside your MetaMask Wallet a Burn Activity.

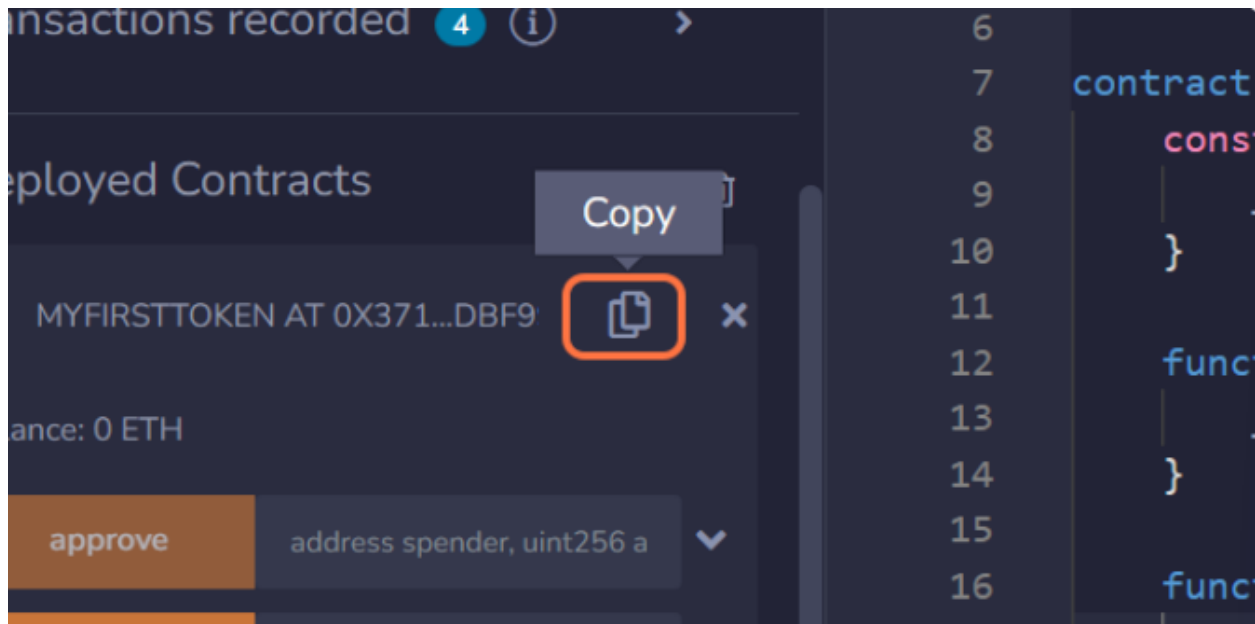


16. Click the **“totalSupply”** button. Notice that we have deducted the amount we entered for burn.



Step 3: Using Polygon Scan to Check

1. Click the **copy** icon on our deployed contract. This will copy the contract ID.



2. Got to this website <https://mumbai.polygonscan.com/>

The screenshot shows the Polygon PoS Chain Testnet Explorer homepage. The search bar is empty. The page displays 'Latest Blocks' and 'Latest Transactions' sections. The 'Latest Blocks' section shows a list of blocks with their IDs, timestamps, and validation details. The 'Latest Transactions' section shows a list of transactions with their IDs, timestamps, and details about the sender and receiver.

Latest Blocks	
Bk	39841283 11 secs ago Validated By 0x5082f249c0b2f2c1ee0... 28 txns in 2 secs 0.01248 MATIC
Bk	39841282 13 secs ago Validated By 0x5082f249c0b2f2c1ee0... 30 txns in 2 secs 0.01615 MATIC
Bk	39841281 15 secs ago Validated By 0x5082f249c0b2f2c1ee0... 54 txns in 2 secs 0.03397 MATIC
Bk	39841280 17 secs ago Validated By 0x5082f249c0b2f2c1ee0... 29 txns in 4 secs 0.03734 MATIC
Bk	39841279 21 secs ago Validated By 0xbe188d6641eb66074... 32 txns in 2 secs 0.02105 MATIC
Bk	39841278 23 secs ago Validated By 0xbe188d6641eb66074... 70 txns in 7 secs 0.01942 MATIC

Latest Transactions	
Tx	0x687aab2748f1f... 11 secs ago From 0x3d0ec56e1234e6b6c... To 0x678d34aa4fc546ba80... 0 MATIC
Tx	0xc29bdfc0b2ac0... 11 secs ago From 0xc3606a1d7d173b6a22... To 0x678d34aa4fc546ba80... 0 MATIC
Tx	0x52027529423a... 11 secs ago From 0xd3b374ce4954f3aa0e... To 0x08e573b13439d0b660... 0 MATIC
Tx	0xf031fb943849... 11 secs ago From 0xd18f4257f43bd8186d5... To 0x08e573b13439d0b660... 0 MATIC
Tx	0xe5477b661e6... 11 secs ago From 0xd5353cb83570551627... To 0x08e573b13439d0b660... 0 MATIC
Tx	0x603c50b623ad... 11 secs ago From 0xcd37314bdc0d96ea23... To 0x08e573b13439d0b660... 0 MATIC

3. Paste your contract ID on the search field. Then click the **search icon button**

The screenshot shows the Polygon PoS Chain Testnet Explorer homepage with a contract ID pasted into the search bar. The search results show the contract address and its details. The 'Addresses' section displays the contract address and its details. The 'Latest Blocks' and 'Latest Transactions' sections are also visible.

Addresses	
	0x37113d3ed443e0a994375c589dcab27b08adb99

Latest Blocks	
Bk	39841283 11 secs ago Validated By 0x5082f249c0b2f2c1ee0... 28 txns in 2 secs 0.01248 MATIC
Bk	39841282 13 secs ago Validated By 0x5082f249c0b2f2c1ee0... 30 txns in 2 secs 0.01615 MATIC
Bk	39841281 15 secs ago Validated By 0x5082f249c0b2f2c1ee0... 54 txns in 2 secs 0.03397 MATIC

Latest Transactions	
Tx	0x687aab2748f1f... 11 secs ago From 0x3d0ec56e1234e6b6c... To 0x678d34aa4fc546ba80... 0 MATIC
Tx	0xc29bdfc0b2ac0... 11 secs ago From 0xc3606a1d7d173b6a22... To 0x678d34aa4fc546ba80... 0 MATIC
Tx	0x52027529423a... 11 secs ago From 0xd3b374ce4954f3aa0e... To 0x08e573b13439d0b660... 0 MATIC

4. You will now be able to see the transactions and information of your contract. Click on the **“OCC Token(OCC)”** link in the Token Tracker Row inside the More Info table.

Contract Overview

Balance: 0 MATIC

More Info

My Name Tag: Not Available

Contract Creator: View 0x37113d3eD443E0a994375C589dcAB27B08aDBF99 Token Tracker Page

Token Tracker: [OCC Token \(OCC\)](#)

Transactions | ERC-20 Token Txns | Contract | Events

Latest 3 from a total of 3 transactions

Txn Hash	Method	Block	Age	From	To	Value	[Txn Fee]
0xc10770ca344807a586...	Burn	39841270	55 secs ago	0x1cc739f36b0b44bb64...	0x37113d3ed443e0a994...	0 MATIC	0.000093033906
0x32efe4885b03d0cecc...	Mint	39841252	1 min ago	0x1cc739f36b0b44bb64...	0x37113d3ed443e0a994...	0 MATIC	0.000093094348
0xd3371d9d337c19a0c...	0x60806040	39841179	4 mins ago	0x1cc739f36b0b44bb64...	Contract Creation	0 MATIC	0.004730985044

[Download CSV Export]

Powered by Polygon Chain

Add Mumbai Network | Preferences | Theme

5. This will open the token tracker page. You have now finally created your own Cryptocurrency!

Token OCC Token

Overview | ERC-20

Total Supply: 9.999999 **OCC**

Holders: 1 addresses

Transfers: 3

Profile Summary

Contract: 0x37113d3eD443E0a994375C589dcAB27B08aDBF99

Decimals: 18

Transfers | Holders | Contract

A total of 3 transactions found

Txn Hash	Method	Age	From	To	Quantity
0xc10770ca344807a586...	Burn	1 min ago	0x1cc739f36b0b44bb64...	0x000000000000000000...	0.000000001
0x32efe4885b03d0cecc...	Mint	2 mins ago	0x000000000000000000...	0x1cc739f36b0b44bb64...	0.0000000000000002
0xd3371d9d337c19a0c...	0x60806040	4 mins ago	0x000000000000000000...	0x1cc739f36b0b44bb64...	10

[Download CSV Export]

Powered by Polygon Chain

Add Mumbai Network | Preferences | Theme