

Token Launch – Deploying a Token Locally

School: ............................................................................................................. Campus: ....................................................... Academic Year: ...................... Subject Name: ........................................................... Subject Code: ..........................

Semester: ............... Program: ........................................ Branch: ......................... Specialization: .......................... Date: .....................................

(Learning by Doing and Discovery)

**\* Coding Phase: Pseudo Code / Flow Chart / Algorithm**

ALGORITHM:

1. Start
2. Open Remix IDE in your browser.
3. Import the ERC20 contract from the OpenZeppelin library.
4. Write a constructor to set:

* Token Name
* Token Symbol
* Mint Initial Supply to the deployer account.

1. Compile the smart contract using Solidity ^0.8.20.
2. Deploy the contract on the local blockchain using Account 1.
3. Verify token details using functions:

* name() → returns token name.
* symbol() → returns token symbol.
* totalSupply() → returns total supply minted.
* balanceOf(owner) → shows deployer’s balance

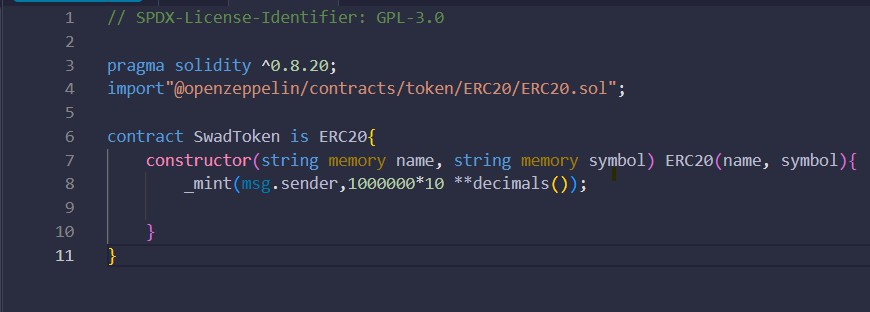
1. Add the deployed token to MetaMask by importing the contract address.
2. Use the transfer() function to send tokens to another account and verify with balanceOf().
3. End

# \* Software Used:

1. OS: Windows or others.
2. Remix IDE.
3. Wallet: MetaMask.
4. Library: OpenZeppelin ERC20

1. **Open Remix IDE:**

* Open Browser/Brave.
* Search Remix IDE.
* Write Smart Contract.
* Create a new file in inside of contract using .sol.

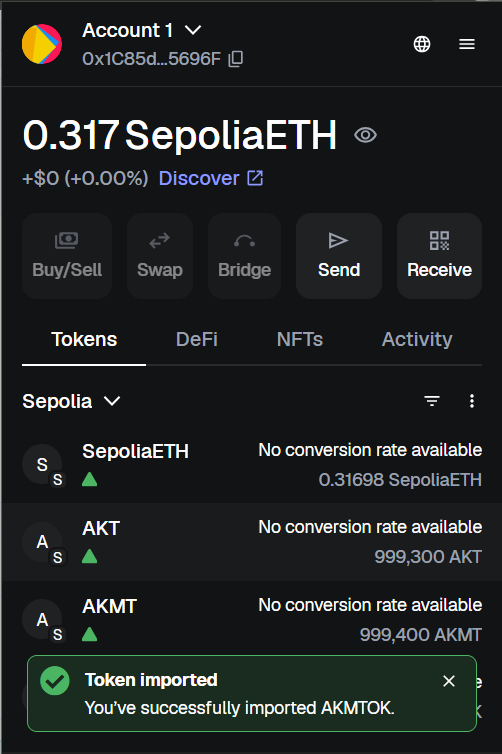


1. **Code Compile and Deploy**

* Click the Solidity compiler and compile this file.
* Click Deploy & run transaction Option.
* Deploy the write of some string name and symbol.

1. **Add Token to MetaMask:**

* In MetaMask, click **"Import Tokens"**.
* Enter your deployed contract address.
* Your token should appear in the wallet.



1. **Deploy:**

In the **Deploy** section, enter constructor values:

* name: **AnweshToken.**
* symbol: **AKMTOK**

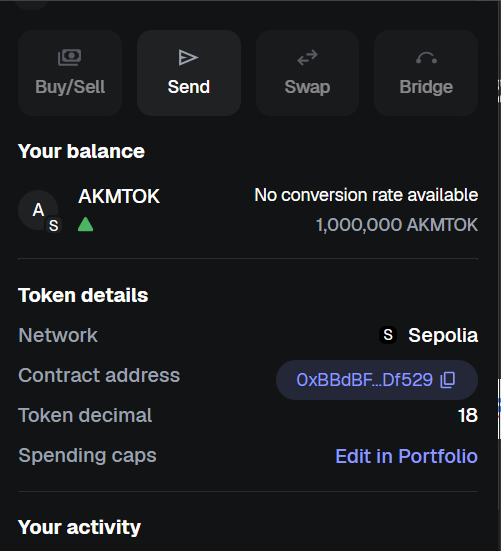
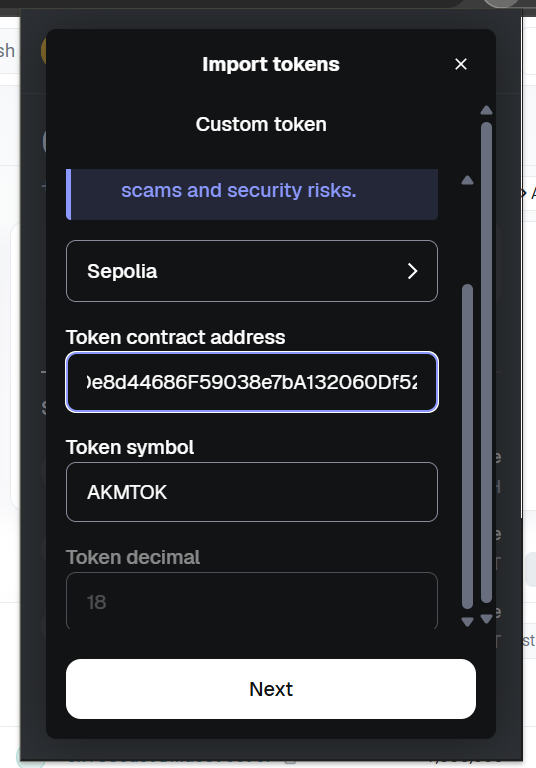
Click **Deploy**,Copy the **contract address** fromthe deployed panel.

1. **Verify core functions (from the deployed contract panel):**

* name() → should return **“AnweshToken”**
* symbol() → should return **“AKMTOK”**
* decimals() → **18**
* totalSupply() → **1,000,000 × 10^18** (because of decimals)
* balanceOf(deployer) → equals totalSupply (all tokens minted to deployer).

1. **Add token to MetaMask:**
2. Open MetaMask
3. Import Tokens
4. paste your contract address
5. symbol AKMTOK
6. decimals 18.

You’ll see the SWA balance in your wallet.

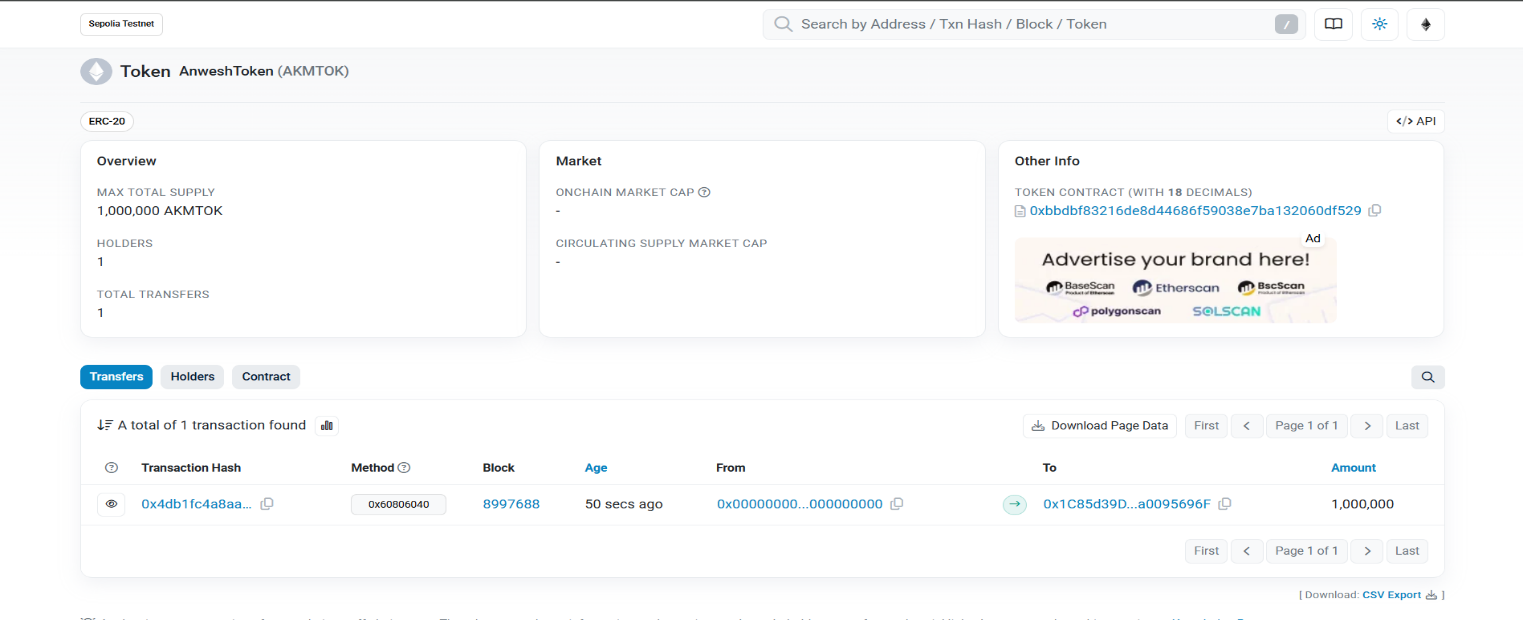


1. **Check Token Details:**

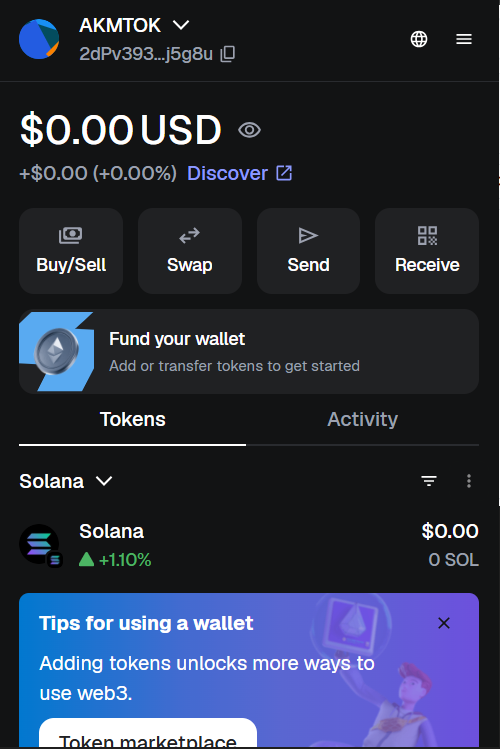
name() → should return "AnweshToken"

symbol() → should return "AKMTOK"

totalSupply() → should return 10000000.

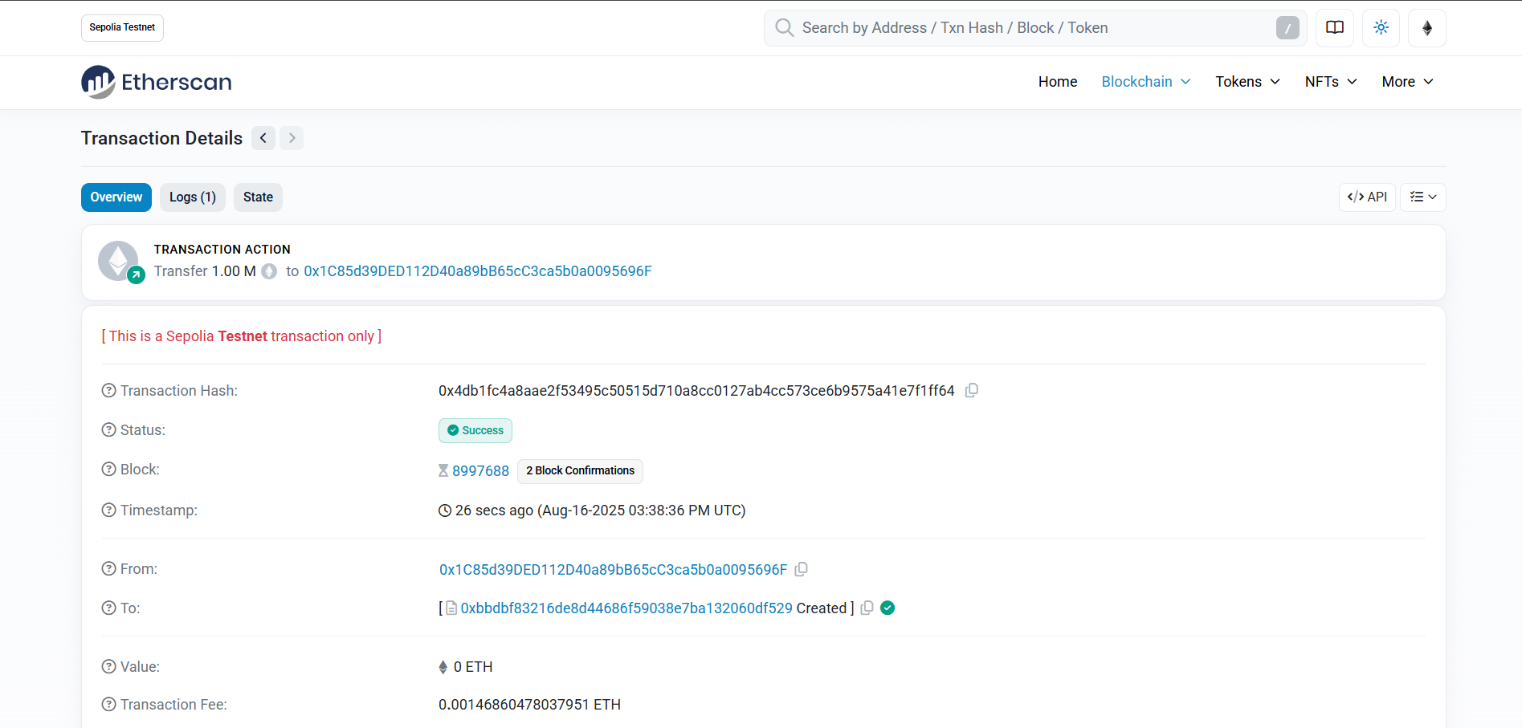


1. **Analyze the Transaction Details:**

* **TX Hash** → Unique ID
* **From / To** → Sender & Receiver
* **Block & Time** → When stored
* **Gas & Fee** → Cost paid
* **Status** → Success/Fail
* **Value / Data** → ETH sent or contract call

## \* Implementation Phase: Final Output (no error)

Applied and Action Learning

****

# Observation:



Each blockchain transaction is **public, traceable, and immutable.**

Details like **gas, sender, receiver, and status** give transparency to how the transaction was processed.

For smart contracts, **input data** reveals the function called (e.g., transfer() or mint()).

|  |  |  |  |
| --- | --- | --- | --- |
| **Rubrics** |  |  |  |
| Concept | 10 |  |  |
| Planning and Execution/  Practical Simulation/ Programming | 10 |  |  |
| Result and Interpretation | 10 |  |  |
| Record of Applied and Action Learning | 10 |  |  |
| Viva | 10 |  |  |
| **Total** | **50** |  |  |

***Signature of the Student:***



***Signature of the Faculty:***