

ASSIGNMENT - 004

2303A51502

NUKALA MOHAN SAI GUPTA

BATCH-25

Objective

To design, implement, and deploy a basic ERC-20 compliant token smart contract using Solidity on the Ethereum blockchain.

Requirements

- Install VS Code
- Install Solidity extension in VS Code
- Use Remix Ethereum IDE (online) or Hardhat (optional)
- Basic understanding of blockchain and Ethereum
- MetaMask wallet (for deployment testing)

Practical Implementation

Step 1: Development Environment Setup

- Install VS Code

Step 2: Create ERC20 Smart Contract

Create a Solidity smart contract that:

- Defines token name, symbol, decimals, and total supply
- Allows token transfer between accounts
- Maintains balances using mappings
- Emits events for transparency

Code:

```
call_solc(0.8.0) to install.  
PS D:\BLOCKCHAIN> pip install py-solc-x  
>>
```

```
KeyboardInterrupt  
PS D:\BLOCKCHAIN> pip install web3 py-solc-x  
Collecting web3
```

```

1  import tkinter as tk
2  from tkinter import messagebox
3  from web3 import Web3
4  from solcx import compile_source, install_solc, set_solc_version
5  # ----- Blockchain Setup -----
6  GANACHE_URL = "http://127.0.0.1:7545"
7  web3 = Web3(Web3.HTTPProvider(GANACHE_URL))
8  connected = web3.is_connected()
9  if connected:
10     account = web3.eth.accounts[0]
11 # ----- Solidity Compiler Setup -----
12 set_solc_version("0.8.0")
13 # ----- Solidity ERC20 Contract -----
14 erc20_source = """
15 // SPDX-License-Identifier: MIT
16 pragma solidity ^0.8.0;
17 contract ERC20Token {
18     string public name;
19     string public symbol;
20     uint8 public decimals = 18;
21     uint public totalSupply;
22

```

```

25     event Transfer(address indexed from, address indexed to, uint value);
26
27     constructor(string memory _name, string memory _symbol, uint _supply) {
28         name = _name;
29         symbol = _symbol;
30         totalSupply = _supply * (10 ** uint(decimals));
31         balanceOf[msg.sender] = totalSupply;
32     }
33 }
34 """
35 compiled = compile_source(erc20_source)
36 _, contract_interface = compiled.popitem()
37 # ----- Deploy Function -----
38 def deploy_token():
39     if not connected:
40         messagebox.showerror("Error", "Ganache not connected")
41         return
42
43     name = name_entry.get()
44     symbol = symbol_entry.get()
45     supply = supply_entry.get()
46

```

```

47     if not name or not symbol or not supply:
48         messagebox.showwarning("Input Error", "All fields are required")
49         return
50
51     try:
52         supply = int(supply)
53
54         Token = web3.eth.contract(
55             abi=contract_interface['abi'],
56             bytecode=contract_interface['bin']
57         )
58
59         tx_hash = Token.constructor(name, symbol, supply).transact({
60             'from': account
61         })
62
63         receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
64
65         messagebox.showinfo(
66             "Success",
67             f"Token Deployed Successfully!\n\nContract Address:\n{receipt.contract_address}"

```

```

68     )
69
70     except Exception as e:
71         messagebox.showerror("Error", str(e))
72
73     # ----- GUI -----
74
75     root = tk.Tk()
76     root.title("ERC20 Token Generator")
77     root.geometry("400x260")
78     root.resizable(False, False)
79
80     title = tk.Label(root, text="ERC20 Token Generator", font=("Arial", 14, "bold"))
81     title.pack(pady=15)
82
83     form = tk.Frame(root)
84     form.pack(pady=5)
85
86     tk.Label(form, text="Token Name").grid(row=0, column=0, sticky="w", pady=5)
87     name_entry = tk.Entry(form, width=25)
88     name_entry.grid(row=0, column=1)

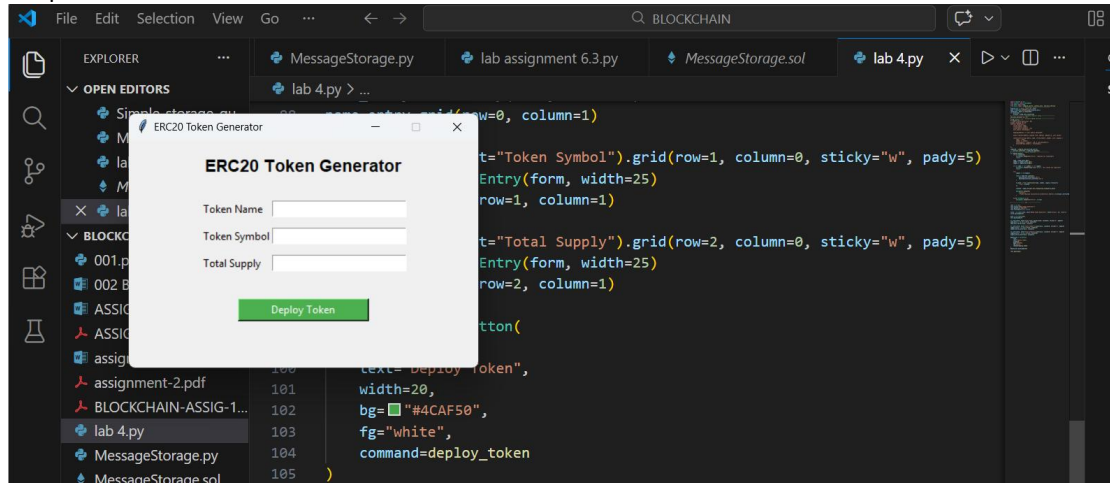
```

```

88     name_entry.grid(row=0, column=1)
89
90     tk.Label(form, text="Token Symbol").grid(row=1, column=0, sticky="w", pady=5)
91     symbol_entry = tk.Entry(form, width=25)
92     symbol_entry.grid(row=1, column=1)
93
94     tk.Label(form, text="Total Supply").grid(row=2, column=0, sticky="w", pady=5)
95     supply_entry = tk.Entry(form, width=25)
96     supply_entry.grid(row=2, column=1)
97
98     deploy_btn = tk.Button(
99         root,
100         text="Deploy Token",
101         width=20,
102         bg="#4CAF50",
103         fg="white",
104         command=deploy_token
105     )
106     deploy_btn.pack(pady=20)
107
108     root.mainloop()

```

Output:



- State variables = name, symbol, decimals, totalSupply, balanceOf
- Constructor = constructor(uint initialSupply)
- Token transfer function = transfer(address to, uint value)
- Balance storage = mapping(address => uint) balanceOf
- Event used = Transfer(address from, address to, uint value)