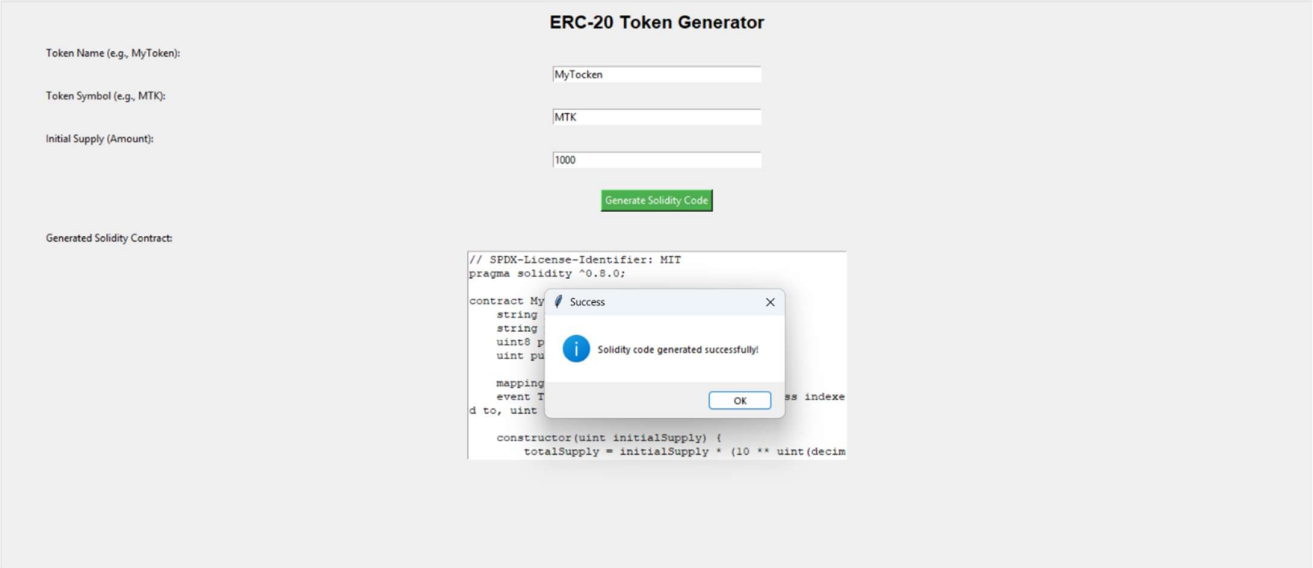
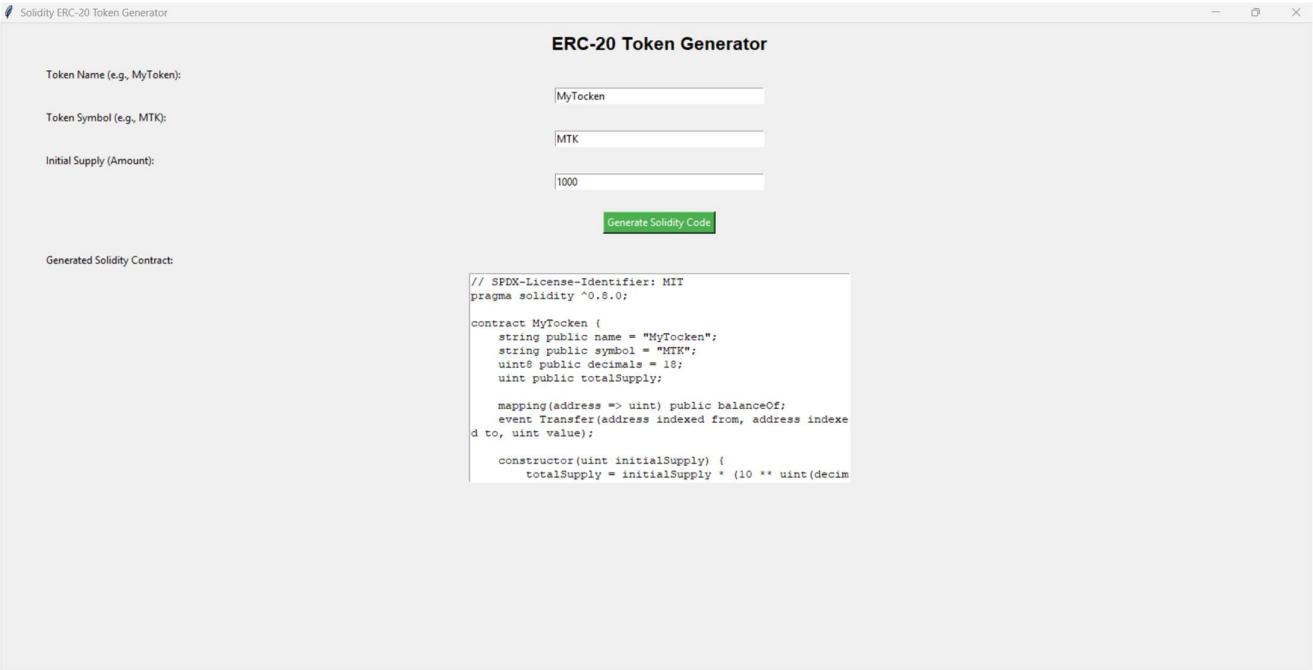
Code :

```
#_____________________________________________________________________________
#LAB-ASSIGNMENT-04: BLOCKCHAIN INTERACTION GUI

import tkinter as tk
from tkinter import messagebox, ttk
from web3 import Web3

class TokenGeneratorApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Solidity ERC-20 Token Generator")
        self.root.geometry("500x600")

        # UI Styling and Labels
        tk.Label(root, text="ERC-20 Token Generator", font=("Arial", 16, "bold")).pack(pady=10)

        # Token Name
        tk.Label(root, text="Token Name (e.g., MyToken):").pack(anchor="w", padx=50)
        self.name_entry = tk.Entry(root, width=40)
        self.name_entry.pack(pady=5)

        # Token Symbol
        tk.Label(root, text="Token Symbol (e.g., MTK):").pack(anchor="w", padx=50)
        self.symbol_entry = tk.Entry(root, width=40)
        self.symbol_entry.pack(pady=5)

        # Total Supply
        tk.Label(root, text="Initial Supply (Amount):").pack(anchor="w", padx=50)
        self.supply_entry = tk.Entry(root, width=40)
        self.supply_entry.pack(pady=5)

        # Generate Button
        self.gen_button = tk.Button(root, text="Generate Solidity Code", command=self.generate_code, bg="#4CAF50", fg="white")
        self.gen_button.pack(pady=20)

        # Code Output Area
        tk.Label(root, text="Generated Solidity Contract:").pack(anchor="w", padx=50)
        self.code_output = tk.Text(root, height=15, width=55)
        self.code_output.pack(pady=5, padx=10)

    def generate_code(self):
        name = self.name_entry.get()
        symbol = self.symbol_entry.get()
        supply = self.supply_entry.get()

        if not name or not symbol or not supply:
            messagebox.showwarning("Input Error", "Please fill in all fields")
            return

        # Template for the Smart Contract
        solidity_template = f"""// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract {name.replace(" ", "")} {{
    string public name = "{name}";
    string public symbol = "{symbol}";
    uint8 public decimals = 18;
    uint public totalSupply;

    mapping(address => uint) public balanceOf;
    event Transfer(address indexed from, address indexed to, uint value);

    constructor(uint initialSupply) {{
        totalSupply = initialSupply * (10 ** uint(decimals));
        balanceOf[msg.sender] = totalSupply;
        emit Transfer(address(0), msg.sender, totalSupply);
    }}

    function transfer(address to, uint value) public returns (bool success) {{
        require(balanceOf[msg.sender] >= value, "Insufficient balance");
        balanceOf[msg.sender] -= value;
        balanceOf[to] += value;
        emit Transfer(msg.sender, to, value);
        return true;
    }}
}}
"""
        self.code_output.delete(1.0, tk.END)
        self.code_output.insert(tk.END, solidity_template)
        messagebox.showinfo("Success", "Solidity code generated successfully!")

if __name__ == "__main__":
    root = tk.Tk()
    app = TokenGeneratorApp(root)
    root.mainloop()
```

Output:


ERC-20 Token Generator
Token Name (e.g., MyToken):
MyTocken
Token Symbol (e.g., MTK):
MTK
Initial Supply (Amount):
1000
Generate Solidity Code
Generated Solidity Contract:
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract My
string
string
uint8 p
uint pu
mapping
event T
ss indexe
d to, uint
constructor(uint initialSupply) {
totalSupply = initialSupply * (10 ** uint(decim
Success
Solidity code generated successfully!
OK


Solidity ERC-20 Token Generator
ERC-20 Token Generator
Token Name (e.g., MyToken):
MyTocken
Token Symbol (e.g., MTK):
MTK
Initial Supply (Amount):
1000
Generate Solidity Code
Generated Solidity Contract:
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract MyTocken {
string public name = "MyTocken";
string public symbol = "MTK";
uint8 public decimals = 18;
uint public totalSupply;
mapping(address => uint) public balanceOf;
event Transfer(address indexed from, address indexe
d to, uint value);
constructor(uint initialSupply) {
totalSupply = initialSupply * (10 ** uint(decim