

# Assignment – 03

To understand the fundamentals of Solidity smart contract development by implementing a simple Ethereum smart contract that accepts a message from the user, stores it on the blockchain, and retrieves it securely.

## Problem Statement

Develop a basic Solidity smart contract that allows users to:

- Store a message on the blockchain
  - Update the message
  - Retrieve the stored message

This practical helps understand state variables, functions, constructors, and data types in Solidity.

```
1 import tkinter as tk
2 from tkinter import messagebox
3 import os
4 from web3 import Web3
5 from solcx import compile_standard, install_solc
6
7 # Install Solidity Compiler
8 install_solc("0.8.0")
9
10 # Connect to Ganache
11 ganache_url = "http://127.0.0.1:7545"
12 w3 = Web3(Web3.HTTPProvider(ganache_url))
13 if not w3.is_connected():
14     messagebox.showerror("Error", "Failed to connect to Ganache at " + ganache_url)
15     exit()
16 messagebox.showinfo("Success", "Connected to Blockchain")
17
18 # Read and Compile Contract (assume MessageStorage.sol exists)
19 with open("MessageStorage.sol", "") as file:
20     contract_source_code = file.read()
21
22 compiled_sol = compile_standard(
23     {
24         "language": "Solidity",
25         "sources": {"MessageStorage.sol": {"content": contract_source_code}},
26         "settings": {
27             "outputSelection": {"*": {"*": ["abi", "metadata", "evm bytecode", "evm sourceMap"]}}
28         },
29     },
30     solc_version="0.8.0",
31 )
32
33 abi = compiled_sol["contracts"]["MessageStorage.sol"]["MessageStorage"]["abi"]
34 bytecode = compiled_sol["contracts"]["MessageStorage.sol"]["MessageStorage"]["evm"]["bytecode"]["object"]
35
36 # Get Account
37 account = w3.eth.accounts[0]
38
39 # GUI Class
40 class ContractGUI:
41     def __init__(self, root):
42         self.root = root
43         self.root.title("Solidity Contract Deployer")
44         self.contract_address = None
45         self.contract_instance = None
46
47         tk.Label(root, text="Enter Message to Store:").pack(pady=10)
48         self.message_entry = tk.Entry(root, width=50)
49         self.message_entry.pack(pady=5)
50
51         tk.Button(root, text="Deploy Contract", command=self.deploy).pack(pady=10)
52         tk.Button(root, text="Get Stored Message", command=self.get_message).pack(pady=5)
53
54     def deploy(self):
55         message = self.message_entry.get()
56         if not message:
57             messagebox.showwarning("Warning", "Please enter a message to store")
58         else:
59             tx_hash = self.w3.eth.sendTransaction({
60                 "from": account,
61                 "to": self.contract_address,
62                 "data": bytecode,
63                 "gas": 1000000
64             })
65             tx_receipt = self.w3.eth.getTransactionReceipt(tx_hash)
66             self.contract_instance = self.w3.eth.contract(tx_receipt.contractAddress, abi=abi)
67             self.contract_address = tx_receipt.contractAddress
68             messagebox.showinfo("Success", "Contract deployed successfully")
69
70     def get_message(self):
71         if self.contract_instance:
72             stored_message = self.contract_instance.functions.getMessage().call()
73             messagebox.showinfo("Stored Message", f"Message stored: {stored_message}")
74         else:
75             messagebox.showwarning("Warning", "Contract has not been deployed yet")
```

Enter message to store in blockchain:

[Deploy Contract](#)

[Store Message](#)

[Retrieve Message](#)