

Assignment _005

2303A510H5

Batch _30

To develop and perform unit testing of smart contracts using the Truffle framework

Objective

Implement a simple Solidity Smart Contract and perform UNIT Testing using Truffle to verify contract functionality.

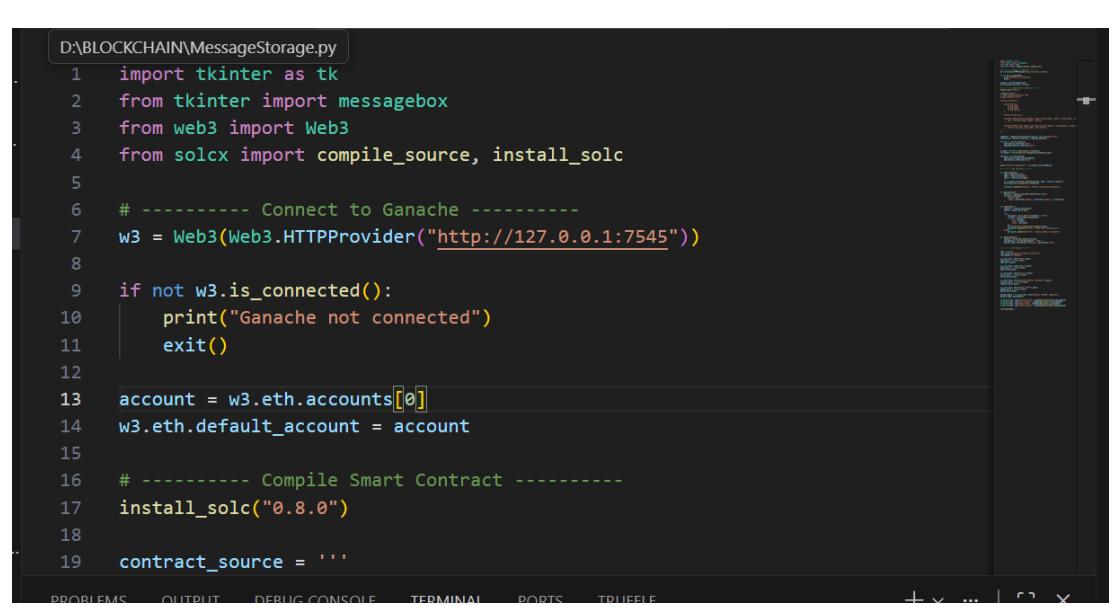
Problem Statement (Equivalent to Given Cipher Practical)

Develop a basic Personal Portfolio Smart Contract that stores and retrieves user profile data, and validate its behavior using Truffle unit tests.

Requirements

Development Environment

- Install Node.js
- Install Truffle Framework
- Install Ganache (Local Blockchain)
- Use VS Code with:
 - Solidity Extension
 - JavaScript Extension



A screenshot of the Visual Studio Code interface. The main editor window displays a Python script named 'D:\BLOCKCHAIN\MessageStorage.py'. The code uses the `web3` library to interact with a Ganache blockchain. It includes logic to connect to Ganache, set a default account, compile a Solidity contract, and interact with it. The right side of the screen shows the Truffle UI, which is a separate application window titled 'Truffle UI' showing network status and other blockchain-related information.

```
D:\BLOCKCHAIN\MessageStorage.py
1 import tkinter as tk
2 from tkinter import messagebox
3 from web3 import Web3
4 from solcx import compile_source, install_solc
5
6 # ----- Connect to Ganache -----
7 w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:7545"))
8
9 if not w3.is_connected():
10     print("Ganache not connected")
11     exit()
12
13 account = w3.eth.accounts[0]
14 w3.eth.default_account = account
15
16 # ----- Compile Smart Contract -----
17 install_solc("0.8.0")
18
19 contract_source = '''
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS TRUFFLE

```
ge.py    lab assignment 6.3.py    lab 4.py    portfolio.sol    portfolio_test.py    CHAT  
portfolio_test.py > ...  
16  # ----- Compile Smart Contract -----  
17  install_solc("0.8.0")  
18  
19  contract_source = '''  
20 // SPDX-License-Identifier: MIT  
21 pragma solidity ^0.8.0;  
22  
23 contract Portfolio {  
24  
25     struct Profile {  
26         string name;  
27         string email;  
28         string skills;  
29     }  
30  
31     Profile private user;  
32  
33     function setProfile(string memory _name, string memory _email, string memory _skills) external  
34         user = Profile(_name, _email, _skills);  
35     }  
36  
37     function getProfile() public view returns(string memory, string memory, string memory)  
38         return (user.name, user.email, user.skills);  
39     }  
40 }  
41 ...  
42  
43 compiled = compile_source(contract_source, solc_version="0.8.0")  
44 contract_id, contract_interface = compiled.popitem()  
45  
46 Portfolio = w3.eth.contract(  
47     abi=contract_interface['abi'],  
48     bytecode=contract_interface['bin']  
49 )  
50  
51 tx_hash = Portfolio.constructor().transact()  
52 tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)  
53  
54 contract = w3.eth.contract(  
55     address=tx_receipt.contractAddress,  
56     abi=contract_interface['abi']  
57 )  
58  
59 print("Contract deployed at:", tx_receipt.contractAddress)  
60  
61 # ----- GUI Functions -----  
62  
63 def store_profile():  
64     name = name_entry.get()  
65     email = email_entry.get()  
66     skills = skills_entry.get()  
67  
68     tx = contract.functions.setProfile(name, email, skills).transact()  
69     w3.eth.wait_for_transaction_receipt(tx)  
70  
71     messagebox.showinfo("Success", "Profile Stored on Blockchain")  
72  
73  
74 def get_profile():  
75     profile = contract.functions.getProfile().call()  
76     messagebox.showinfo(  
77         "Profile Data",  
78         f"Name: {profile[0]}\nEmail: {profile[1]}\nSkills: {profile[2]}"  
79     )  
80  
81
```

SESSIONS

portfolio_test.py

Describe what to build

Auto

OVR Ln 13, Col 29 Spaces: 4 UTF-8 CRLF {} Python

Search

ENG IN

```
portfolio_test.py > ...
82     def send_ether():
83         receiver = receiver_entry.get()
84         amount = amount_entry.get()
85
86     try:
87         wei_value = w3.to_wei(float(amount), 'ether')
88         tx_hash = w3.eth.send_transaction({
89             'to': receiver,
90             'from': account,
91             'value': wei_value
92         })
93         w3.eth.wait_for_transaction_receipt(tx_hash)
94         messagebox.showinfo("Success", "Ether Sent Successfully")
95     except:
96         messagebox.showerror("Error", "Invalid Address or Amount")
97
98
99    def check_balance():
100        balance = w3.eth.get_balance(account)
101        eth_balance = w3.from_wei(balance, 'ether')
102        balance_label.config(text=f"Balance: {eth_balance} ETH")
103
104    # ----- GUI Layout -----
105
106    root = tk.Tk()
107    root.title("Blockchain Portfolio Interface")
108    root.geometry("350x420")
109
```

OVR Ln 13, Col 29 Spaces: 4 UTF-8 CRLF {} Python 3.12.4

```
portfolio_test.py > ...
113    name_entry.pack()
114
115    tk.Label(root, text="Email").pack()
116    email_entry = tk.Entry(root)
117    email_entry.pack()
118
119    tk.Label(root, text="Skills").pack()
120    skills_entry = tk.Entry(root)
121    skills_entry.pack()
122
123    tk.Label(root, text="Receiver Address (optional)").pack()
124    receiver_entry = tk.Entry(root)
125    receiver_entry.pack()
126
127    tk.Label(root, text="Amount (ETH)").pack()
128    amount_entry = tk.Entry(root)
129    amount_entry.pack()
130
131    balance_label = tk.Label(root, text="Balance checked", fg="blue")
132    balance_label.pack(pady=5)
133
134    tk.Button(root, text="Store Profile", command=store_profile).pack(pady=5)
135    tk.Button(root, text="Get Profile", command=get_profile).pack(pady=5)
136    tk.Button(root, text="Send Ether", command=send_ether).pack(pady=5)
137    tk.Button(root, text="Check Balance", command=check_balance).pack(pady=5)
138
139    root.mainloop()
140
```

OVR Ln 13, Col 29 Spaces: 4 UTF-8 CRLF {} Python 3.12.4

