Enhanced Methodology for Python Implementation Using Google Colab and Ethereum Ledger

Author : SIYA SINGH

This enhanced approach is structured so that Google Colab will be used to:

```
Connect to the Ethereum Ledger (via Infura/Alchemy).
Deploy and interact with smart contracts on Ethereum.
Send and verify transactions on the Ethereum ledger.
Ensure that all transactions are authorized and handled through the Ethereum ledger, with smart contracts managing fund flows.
```

The methodology to focus on the Python implementation using Google Colab will involve breaking down each step into concrete Python code using Web3.py to interact with the Ethereum Ledger. We'll use Google Colab's Python environment for deploying smart contracts, sending transactions, and verifying blockchain states.

1. Environment Setup on Google Colab

Install Necessary Packages

The Web3.py library will allow us to interact with the Ethereum blockchain. First, we need to install it in Google Colab.

```
# Install Web3.py to interact with Ethereum blockchain
!pip install web3
```

Connect to Ethereum Ledger via Infura/Alchemy

Once Web3 is installed, we'll connect to the Ethereum ledger via a service like Infura or Alchemy.

```
from web3 import Web3

# Connect to Ethereum Mainnet through Infura (replace YOUR_INFURA_PROJECT_ID with your own)
infura_url = "https://mainnet.infura.io/v3/YOUR_INFURA_PROJECT_ID"
web3 = Web3(Web3.HTTPProvider(infura_url))

# Check if the connection is successful
if web3.isConnected():
    print("Connected to Ethereum Mainnet")
else:
    print("Failed to connect")
```

Create Wallets and Fund Transfer Accounts

To interact with Ethereum, we will need a wallet for both donors and NGOs. We can create or import Ethereum accounts directly into Google Colab.

```
# Create new Ethereum account (for demo purposes)
account = web3.eth.account.create()
private_key = account.privateKey.hex()
public_address = account.address

print(f"New Ethereum Wallet Created: {public_address}")
```

2. Smart Contract Development

Create a Smart Contract for NGO Fund Management

The smart contract will handle donations and manage the flow of funds from donors to NGOs. We can write and deploy the smart contract using Solidity.

Here's the Solidity code for a simple NGO fund management contract:

```
// Solidity contract for managing NGO donations
pragma solidity ^0.8.0;

contract NGOFundLedger {
    address public ngo;
    address public donor;
    uint256 public donationAmount;

    constructor(address _ngo) {
        ngo = _ngo;
    }

    // Donor can send donations
    function donate() public payable {
        require(msg.value > 0, "Donation must be greater than zero");
        donor = msg.sender;
        donationAmount = msg.value;
    }

    // Only the NGO can release the funds
    function releaseFunds() public {
        require(msg.sender == ngo, "Only NGO can release funds");
        payable(ngo).transfer(donationAmount);
    }

    // Function to check donation details
    function getDonationDetails() public view returns (address, uint256) {
        return (donor, donationAmount);
    }
}
```

Compile and Deploy the Smart Contract Using Google Colab

We need to use Web3.py to deploy this contract on the Ethereum Ledger. For that, we'll compile the Solidity code using a tool like Solcx or use online tools like Remix for compilation.

```
# Install Solcx for compiling Solidity code
!pip install py-solc-x

from solcx import compile_standard

# Solidity source code
solidity_code = '''
// Solidity contract for managing NGO donations
pragma solidity ^0.8.0;

contract NGOFundLedger {
    address public ngo;
    address public donor;
    uint256 public donationAmount;

    constructor(address _ngo) {
        ngo = _ngo;
    }

    function donate() public payable {
        require(msg.value > 0, "Donation must be greater than zero");
        donor = msg.sender;
        donationAmount = msg.value;
    }

    function releaseFunds() public {
        require(msg.sender == ngo, "Only NGO can release funds");
        payable(ngo).transfer(donationAmount);
    }

    function getDonationDetails() public view returns (address, uint256) {
        return (donor, donationAmount);
    }
}
'''

# Compile the contract
compiled_sol = compile_standard({
    "language": "Solidity",
    "sources": {
        "NGOFundLedger.sol": {
            "content": solidity_code
        }
    },
    "settings": {
```

```
        "outputSelection": {
            "*": {
                "*": ["metadata", "evm.bytecode", "evm.sourceMap"]
            }
        }
    }
})

# Bytecode and ABI
bytecode = compiled_sol['contracts']['NGOFundLedger.sol']['NGOFundLedger']['evm']['bytecode']['object']
abi = compiled_sol['contracts']['NGOFundLedger.sol']['NGOFundLedger']['abi']

# Create the contract in Python
NGOFundLedger = web3.eth.contract(abi=abi, bytecode=bytecode)
```

Now, we deploy the smart contract on the Ethereum ledger.

```
# NGO wallet address (from previously created account)
ngo_address = "0xYourNGOAddress"

# Deploy the smart contract
tx_hash = NGOFundLedger.constructor(ngo_address).transact({
    'from': public_address,  # This is the address of the sender
    'gas': 2000000,  # Gas limit
    'gasPrice': web3.toWei('20', 'gwei')  # Gas price
})

# Get transaction receipt after deployment
tx_receipt = web3.eth.waitForTransactionReceipt(tx_hash)
contract_address = tx_receipt.contractAddress

print(f"Smart Contract Deployed at: {contract_address}")
```

3. Transactions and Fund Management

Donation Flow

Once the smart contract is deployed, we will allow donors to send ETH to the smart contract.

```
# Donor sends ETH as a donation to the smart contract
donation_tx = NGOFundLedger.functions.donate().buildTransaction({
    'from': donor_address,
    'value': web3.toWei(1, 'ether'),  # 1 ETH donation
    'nonce': web3.eth.getTransactionCount(donor_address),
    'gas': 2000000,
    'gasPrice': web3.toWei('20', 'gwei')
})

# Sign the transaction with the donor's private key
signed_donation_tx = web3.eth.account.signTransaction(donation_tx, donor_private_key)

# Send the transaction to the Ethereum network
tx_hash = web3.eth.sendRawTransaction(signed_donation_tx.rawTransaction)
print(f"Donation transaction hash: {web3.toHex(tx_hash)}")
```

Fund Release by NGO

The NGO can now release the funds to their own account once donations are received.

```
# NGO releases funds from the smart contract
release_tx = NGOFundLedger.functions.releaseFunds().buildTransaction({
    'from': ngo_address,
    'nonce': web3.eth.getTransactionCount(ngo_address),
    'gas': 2000000,
    'gasPrice': web3.toWei('20', 'gwei')
})

# Sign and send the transaction
signed_release_tx = web3.eth.account.signTransaction(release_tx, ngo_private_key)
release_tx_hash = web3.eth.sendRawTransaction(signed_release_tx.rawTransaction)

print(f"Funds Released, Tx Hash: {web3.toHex(release_tx_hash)}")
```

4. Testing and Validation

After each transaction (donation or fund release), we can validate it by checking the blockchain.

Check Donation Details

```
# View donation details
donor, amount = NGOFundLedger.functions.getDonationDetails().call()
print(f"Donor Address: {donor}, Donation Amount: {web3.fromWei(amount, 'ether')} ETH")
```

Check Transaction on Ethereum Ledger

To validate the transactions, we can query the Ethereum ledger.

```
# Check the receipt of the donation transaction
donation_receipt = web3.eth.getTransactionReceipt(donation_tx_hash)
print(f"Donation Transaction Status: {donation_receipt.status}")
```

5. Conclusion: Exclusive Authorization of Ethereum Transactions

Using the Ethereum Ledger, the implemented solution ensures that:

```
All fund management transactions occur directly on Ethereum, providing full transparency and immutability.
Donors and NGOs can track fund flows and validate all interactions via smart contracts.
No off-ledger transactions are authorized, securing the entire process and ensuring accountability.
```

5.1. Visualization of Results using Python

We will visualize the following:

```
Transaction Flow: Show the donation amounts received over time.
Fund Allocation: Visualize how funds are allocated to different NGOs.
Ethereum Transaction Status: Track successful and failed transactions.
```

By integrating these Python visualization techniques in Google Colab, you can:

```
Track donations and fund releases over time, showing trends and performance.
Visualize fund distribution among NGOs, ensuring transparency in allocation.
Monitor Ethereum transaction status, helping to maintain trust in the blockchain system.
```

These visual tools offer real-time insights into the blockchain-powered NGO fund management system, further reinforcing transparency, accountability, and operational efficiency.

Step 5.1: Install Necessary Libraries for Visualization

```
# Install Plotly for interactive visualizations
!pip install plotly

# Install Matplotlib for simple static visualizations
!pip install matplotlib
```

Step 5.2: Set Up Data Collection from Ethereum Ledger

We can query the Ethereum blockchain to get details of transactions like donation amounts and timestamps. This data will be plotted to visualize the trends.

```
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import time

# Dummy data for the example (in real usage, pull this data from the Ethereum ledger)
donation_data = [
```

```
    {"time": "2024-10-12 10:00", "amount": 1.5},
    {"time": "2024-10-13 12:30", "amount": 2.0},
    {"time": "2024-10-14 14:00", "amount": 3.2},
    {"time": "2024-10-15 16:15", "amount": 1.0},
]

times = [entry["time"] for entry in donation_data]
amounts = [entry["amount"] for entry in donation_data]
```

Step 5.3: Visualize Donation Flow Over Time

We can use Matplotlib to create a simple line plot that shows the donation amounts received over time.

This plot will show a clear trend of donations over time, helping to visualize when and how much was donated to the NGO.

```
# Simple plot of donations over time
plt.figure(figsize=(10,6))
plt.plot(times, amounts, marker='o', color='b', linestyle='-', label='Donation Amount (ETH)')
plt.title('Donation Flow Over Time')
plt.xlabel('Time')
plt.ylabel('Donation Amount (ETH)')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Step 5.4: Visualize Fund Allocation with a Pie Chart

We can create a pie chart to visualize how funds are distributed to various NGOs. Assume the funds are being distributed among several projects run by the NGOs.

This chart will show the proportional distribution of funds to different NGOs, giving a snapshot of how donations are being utilized.

```
# Dummy data for NGO fund distribution (example)
fund_distribution = {
    "NGO 1": 4.5,
    "NGO 2": 2.5,
    "NGO 3": 3.2,
}

# Pie chart of fund allocation
plt.figure(figsize=(7,7))
plt.pie(fund_distribution.values(), labels=fund_distribution.keys(), autopct='%1.1f%%', startangle=140)
plt.title('Fund Allocation to NGOs')
plt.axis('equal')  # Equal aspect ratio ensures the pie chart is circular
plt.show()
```

Step 5.5: Interactive Visualization Using Plotly

Using Plotly, we can create an interactive bar chart to compare donations and fund releases, or visualize the time progression of donation and release events.

This interactive bar chart allows users to visualize and compare how much money is being donated versus how much is being released to the NGOs, over time.

```
# Create a bar chart comparing donations vs fund releases (dummy data)
donations = [1.5, 2.0, 3.2, 1.0]
releases = [1.0, 2.5, 2.5, 1.2]

fig = go.Figure(data=[
    go.Bar(name='Donations (ETH)', x=times, y=donations),
    go.Bar(name='Fund Releases (ETH)', x=times, y=releases)
])

# Customize the layout
fig.update_layout(
    title='Donations vs Fund Releases Over Time',
    xaxis_title='Time',
    yaxis_title='Amount (ETH)',
    barmode='group',
    xaxis_tickangle=-45
)
```

```
# Display the interactive plot
fig.show()
```

Step 5.6: Monitor Ethereum Transaction Status

We can also visualize the success and failure rate of Ethereum transactions, which could be useful for monitoring network performance.

This pie chart provides a simple but effective way to track the overall performance and reliability of the system.

```
# Data for transaction success/failure
transaction_statuses = ["Success", "Success", "Failed", "Success"]
transaction_counts = [3, 1]  # 3 successful, 1 failed

# Pie chart for transaction status
plt.figure(figsize=(6,6))
plt.pie(transaction_counts, labels=["Success", "Failed"], autopct='%1.1f%%', startangle=90, colors=['green', 'red'])
plt.title('Ethereum Transaction Status')
plt.axis('equal')
plt.show()
```
