🧨 Project Title

Blockchain-Powered Transparency and Governance for Donated Funds in NGOs

---

🧠 Step-by-Step Methodology Explanation

This end-to-end methodology transforms NGO fund management using blockchain, ensuring full transparency, global accessibility, automated governance, and donor empowerment.

Step 1: Environment Setup in Google Colab

To begin implementing a blockchain-powered donation system for NGOs, the first essential step is setting up the development environment. Google Colab provides a cloud-based Python environment, making it perfect for students and developers without high-end local hardware. The required libraries include web3.py for blockchain interactions, py-solc-x for compiling Solidity smart contracts, and data visualization libraries like matplotlib and plotly. Installing these ensures that the notebook can both manage Ethereum smart contracts and display real-time transaction analytics. This setup enables a cost-effective and accessible way to build decentralized applications while promoting transparency through blockchain data visualization.

Environment Setup in Google Colab

We begin by setting up Google Colab with all necessary libraries:

- Web3.py: For blockchain interaction.
- py-solc-x: For compiling Solidity contracts.
- matplotlib & plotly: For visualizing donations and fund flow.

```
!pip install web3 py-solc-x matplotlib plotly
```

Step 2: Connecting to the Ethereum Blockchain

Once the environment is ready, the next step is connecting to the Ethereum blockchain. This is achieved using Infura or Alchemy, which act as gateways to Ethereum nodes. By connecting via a secure HTTP provider link, the system can read and write data to the blockchain without hosting a full Ethereum node. This step ensures that all transactions are decentralized, tamper-proof, and logged immutably. It also forms the foundation for tracking donation flows, thereby directly addressing the issue of trust and transparency in NGO fund management.

Connecting to the Ethereum Blockchain

To securely log and track donations, we connect to Ethereum Mainnet via Infura:

🔎 This enables decentralized access to the blockchain

```
web3 = Web3(Web3.HTTPProvider("https://mainnet.infura.io/v3/YOUR_INFURA_PROJECT_ID"))
```

Step 3: Wallet Generation (Donor and NGO)

For decentralized fund tracking, each donor and NGO needs a digital wallet. Ethereum accounts are generated using web3.eth.account.create(). These wallets serve as unique digital identities, allowing the system to track the origin and destination of funds precisely. By simulating or creating wallets in Colab, users can securely manage private keys and public addresses, enabling end-to-end traceability of funds. This step ensures each transaction has a source (donor) and destination (NGO) that is verifiable on the Ethereum ledger, aligning with the project's goal of immutability and accountability.

3: Wallet Generation (Donor and NGO)

We simulate account creation for both:

📌 This is equivalent to assigning unique addresses to donors and NGOs for traceability

```
account = web3.eth.account.create()
```

Step 4: Smart Contract Creation for NGO Fund Management

The backbone of the solution lies in smart contracts written in Solidity. The NGOFundLedger contract includes functions to accept donations, release funds, and retrieve donation data. This contract enforces rules such as allowing only NGOs to withdraw funds and ensures that no fund movement occurs without proper authorization. By encoding business logic directly into the blockchain, this step replaces manual accounting with autonomous and transparent execution, greatly reducing the risk of mismanagement and fraud in NGO operations.

4: Smart Contract Creation for NGO Fund Management

We define a NGOFundLedger contract in Solidity that:

- Accepts donations
- Allows NGOs to release funds
- Tracks donation details

🔁 This smart contract automates donations and fund release

```
function donate() public payable

function releaseFunds() public

function getDonationDetails() public view returns (address, uint256)
```

Step 5: Compile and Deploy Smart Contract

The Solidity contract is compiled into bytecode and deployed using py-solc-x and Web3.py. Once deployed, the contract resides on Ethereum's distributed ledger, accessible from anywhere in the world. This step effectively brings the smart contract to life, allowing real-time logging and management of donation-related actions. The contract's address becomes the central hub for all interactions, ensuring that all fund inflows and outflows are visible and auditable in real-time, which is essential for building donor trust.

5: Compile and Deploy Smart Contract

Using py-solc-x, we compile the contract and deploy it with:

📍 The deployed contract becomes a public ledger

```
NGOFundLedger = web3.eth.contract(abi=abi, bytecode=bytecode)

tx_hash = NGOFundLedger.constructor(ngo_address).transact({...})
```

Step 6: Donor Sends a Donation

After deployment, the system simulates a donor sending ETH to the contract. This transaction is built, signed with the donor's private key, and broadcast to the Ethereum network. The donation is recorded immutably, enabling both the donor and the public to verify it. This mechanism ensures that the donation process is decentralized, secure, and cannot be manipulated—directly addressing the concern of opaque fund flows in traditional systems.

6: Donor Sends a Donation

A donor donates ETH via the contract:

🔐 It's recorded immutably on-chain

```
donation_tx = NGOFundLedger.functions.donate().buildTransaction({...})
```

Step 7: NGO Releases Funds

Post donation, NGOs can release funds from the contract using a restricted function. Only the predefined NGO wallet is authorized to trigger this transaction. This step upholds the integrity of the fund management process by ensuring that only the rightful recipient can withdraw funds, thereby eliminating unauthorized fund access. It also models a real-world scenario where NGOs must validate their identity before accessing donated resources.

7: NGO Releases Funds

NGO withdraws the donation:

📤 The blockchain ensures only NGOs can access these funds

```
release_tx = NGOFundLedger.functions.releaseFunds().buildTransaction({...})
```

Step 8: Verify Transactions

Every transaction on Ethereum is recorded with a unique hash. Using Web3.py, the script retrieves the transaction receipt to confirm its success and details. This capability allows donors to confirm that their contributions reached the intended NGO. It further strengthens transparency, offering indisputable proof of all monetary activities performed through the platform.

8: Verify Transactions

Every transaction gets a unique hash:

📊 This forms the basis of traceability

```
receipt = web3.eth.getTransactionReceipt(tx_hash)
```

Step 9: Retrieve Donation History

The smart contract includes a method to retrieve details of the most recent donation. This serves as a quick audit trail, confirming both the donor's address and the exact amount donated. By embedding this functionality, the platform ensures continuous access to verified donation data, supporting NGO accountability and donor engagement.

9: Retrieve Donation History

We query the contract for donor and amount:

📈 Helps track how much each donor contributed

```
donor, amount = NGOFundLedger.functions.getDonationDetails().call()
```

Step 10: Visualize Donation Timeline

To build public confidence and showcase transparency, donation data is plotted over time using Matplotlib. A line graph showing donation trends offers stakeholders a visual understanding of fund inflows. This visualization helps answer questions like: Are donations increasing? Which days see the most activity? Thus, it supports strategic decision-making and builds trust.

10: Visualize Donation Timeline

Using matplotlib, we show when and how much was donated:

⏱ Provides donors with a clear view of fund inflow

```
plt.plot(times, amounts)
```

Step 11: Visualize NGO Fund Allocation

Fund allocation among different NGOs is visualized using pie charts. This intuitive graphic provides donors and administrators with a quick snapshot of how funds are being distributed across multiple causes. Such visual summaries are essential for operational transparency and can also motivate donors by showing impact in real-time.

11: Visualize NGO Fund Allocation

Pie chart of funds distributed across NGOs:

📌 Matches Slide 8's dashboard for fund distribution.

```
plt.pie(fund_distribution.values(), labels=fund_distribution.keys())
```

Step 12: Compare Donations vs Releases

A bar graph using Plotly compares the total donated funds with actual disbursements. This comparison reveals whether NGOs are using funds efficiently and timely. It also helps in identifying bottlenecks, if any, in fund distribution. Such a comparative visualization reinforces the principles of responsible financial governance and promotes donor confidence.

12: Compare Donations vs Releases

Interactive plotly bar chart:

📋 Verifies if NGOs are utilizing the funds received

```
go.Bar(name='Donations'), go.Bar(name='Fund Releases')
```

Step 13: DAO Governance Simulation (Voting Tokens)

Incorporating a governance token model, each donor receives tokens proportional to their donation. These tokens are used in DAO-style voting mechanisms, allowing donors to vote on how funds are spent. This democratizes NGO governance, ensuring inclusivity and accountability. By empowering small donors with decision-making power, the platform redefines donor engagement in the non-profit ecosystem.

13: DAO Governance Simulation (Voting Tokens)

Donors are assigned governance tokens:

✅ Enables decision-making rights for donors

tokens = donation_amount

Step 14: Monitor Transaction Status

System health is gauged using a pie chart that shows successful vs failed transactions. This diagnostic visualization enables backend teams to monitor performance and improve reliability. For donors and auditors, it's a transparent report card on system integrity and uptime, crucial for long-term trust in the platform.

14: Monitor Transaction Status

Pie chart showing successful vs failed Ethereum transactions:

🛡 Helps maintain system reliability and transparency

```
plt.pie(transaction_counts, labels=["Success", "Failed"])
```

Step 15: Security Summary

All smart contract functions are executed on the Ethereum ledger, ensuring data integrity and security. Smart contracts are immutable once deployed and can’t be altered, reducing the risk of fraud. Moreover, decentralization prevents any single entity from controlling fund flow. This framework is censorship-resistant, especially useful in politically unstable regions, and ensures that donor funds reach NGOs without interference or loss.

15: Security Summary

- Smart contracts enforce immutability and automation
- All actions are logged on-chain
- No central authority = no censorship or fund freeze risk

✓ Fully decentralized and donor-friendly