## Integrating Blockchain for Immutable Digital Twin Records

### To ensure trust, security, and immutability in Digital Twin data, we will

1. **Use Blockchain to store digital twin state changes securely**
2. **Create a smart contract for tamper-proof updates**
3. **Implement a decentralized ledger to track real-time congruence shifts.**

#### 1. Install Required Libraries

pip install web3 eth_account fastapi uvicorn sqlite3

### 2. Setting Up a Local Ethereum Blockchain

### Use Ganache (for testing) or Ethereum Testnet (Goerli, Sepolia, etc.) for real-world deployment.

- **Install Ganache:**

npm install -g ganache
ganache-cli

- Connect **MetaMask** to Ganache or a Testnet for deployment.

#### 3. Deploying a Smart Contract for Digital Twin Data

This **Ethereum Smart Contract ensures that digital twin records are tamper-proof.**

**Smart Contract (Solidity)**

```
// DigitalTwinStorage.sol
pragma solidity ^0.8.0;

contract DigitalTwinStorage {
    struct TwinData {
        string id;
        string timestamp;
        string dataHash; // Stores the hash of digital twin data
```

```
    }

    mapping(string => TwinData) private twins;
    event TwinUpdated(string indexed id, string timestamp, string dataHash);

    function updateTwin(string memory _id, string memory _timestamp, string memory _dataHash)
public {
        twins[_id] = TwinData(_id, _timestamp, _dataHash);
        emit TwinUpdated(_id, _timestamp, _dataHash);
    }

    function getTwin(string memory _id) public view returns (string memory, string memory, string
memory) {
        TwinData memory twin = twins[_id];
        return (twin.id, twin.timestamp, twin.dataHash);
    }
}
```

**Compile & Deploy the Contract**

```
solc --bin --abi DigitalTwinStorage.sol -o build
web3 deploy build/DigitalTwinStorage.bin --rpc http://localhost:8545
```

**4. FastAPI Integration for Blockchain Transactions**

Now, we connect **FastAPI to Ethereum blockchain using Web3.py.**

**Web3 Configuration**

```
from web3 import Web3
import json
import sqlite3
import hashlib
from fastapi import FastAPI
from pydantic import BaseModel

# Connect to Ethereum Node (Ganache or Testnet)
w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:8545"))
```

```
contract_address = "0xYourDeployedContractAddress"
contract_abi = json.load(open("DigitalTwinStorage.abi"))  # Load compiled contract ABI

# Load Smart Contract
contract = w3.eth.contract(address=contract_address, abi=contract_abi)

# Connect to SQLite for local storage
conn = sqlite3.connect("digital_twins.db", check_same_thread=False)
cursor = conn.cursor()

app = FastAPI()
```

**5. Hashing Digital Twin Data for Blockchain Integrity**

We use **SHA-256 to generate immutable hashes of digital twin data before storing it on the blockchain.**

```
def generate_data_hash(data):
    """Hashes digital twin data using SHA-256."""
    return hashlib.sha256(json.dumps(data, sort_keys=True).encode()).hexdigest()
```

**6. API to Store Digital Twin Data on Blockchain**

This endpoint **computes a hash and sends the transaction to the blockchain.**

```
class TwinUpdate(BaseModel):
    twin_id: str
    data: dict

@app.post("/update_twin_on_blockchain/")
def update_twin(data: TwinUpdate):
    """Hashes digital twin data and stores it on blockchain."""
    data_hash = generate_data_hash(data.data)
    timestamp = w3.eth.getBlock('latest')["timestamp"]  # Get current blockchain time

    tx = contract.functions.updateTwin(data.twin_id, str(timestamp), data_hash).transact({'from':
w3.eth.accounts[0]})
    w3.eth.waitForTransactionReceipt(tx)
```

```
    # Update SQLite database
    cursor.execute("UPDATE digital_twins SET expertise=?, last_updated=? WHERE id=?",
                   (json.dumps(data.data), timestamp, data.twin_id))
    conn.commit()

    return {"message": "Digital Twin updated on blockchain", "tx_hash": tx.hex()}
```

**7. API to Retrieve Digital Twin Data from Blockchain**

This endpoint **verifies stored data integrity.**

```
@app.get("/get_twin_from_blockchain/{twin_id}")
def get_twin_from_blockchain(twin_id: str):
    """Retrieves twin data from blockchain and compares with local storage."""
    twin_id, timestamp, blockchain_hash = contract.functions.getTwin(twin_id).call()

    # Fetch from local DB
    cursor.execute("SELECT expertise FROM digital_twins WHERE id=?", (twin_id,))
    local_data = cursor.fetchone()

    if not local_data:
        return {"error": "Twin not found in local storage"}

    local_hash = generate_data_hash(json.loads(local_data[0]))

    # Compare blockchain data hash with local hash
    integrity_status = "Valid" if blockchain_hash == local_hash else "Tampered"

    return {"twin_id": twin_id, "timestamp": timestamp, "integrity": integrity_status}
```

## 8. Running the Blockchain-Enabled Digital Twin System

### 1. Start Ganache (or Ethereum Testnet Node):

```
ganache-cli
```

2. **Deploy Smart Contract**:

```
solc --bin --abi DigitalTwinStorage.sol -o build
```

web3 deploy build/DigitalTwinStorage.bin --rpc http://localhost:8545

3. **Run FastAPI Server**:

uvicorn script_name:app --reload

4. **Test Blockchain Update**:

POST /update_twin_on_blockchain/
{
  "twin_id": "expert_123",
  "data": {"temperature": 25.3, "status": "operational"}
}

**9. Features & Benefits**

✅ **Immutable Data → Prevents unauthorized changes to digital twins.**
✅ **Tamper Detection → Ensures real-world & digital twin consistency.**
✅ **Decentralized Security → No single entity can alter records.**

## 10. Future Enhancements

- **Zero-Knowledge Proofs → Ensure data privacy while storing proofs on-chain.**
- **AI-Blockchain Fusion → Use smart contracts to trigger automatic twin corrections.**
- **Decentralized Oracle Networks → Fetch real-time data from IoT devices on-chain.**

**Would you like to integrate decentralized AI decision-making for Digital Twins?**