



Developing An Automated And Transparent Wealth Framework With AI And Blockchain Interoperability

Anup Sharma^{1*}, Dr. Nitin Gupta²

^{1*}Assistant Professor & Head (Operations and IT), Research Scholar, Mittal School of Business, Lovely Professional University

²Professor and Head (Finance), Mittal School of Business, Lovely Professional University

Citation: Anup Sharma, Dr. Nitin Gupta, (2024) Developing An Automated And Transparent Wealth Framework With AI And Blockchain Interoperability, *Educational Administration: Theory and Practice*, 30(5), 10030-10041

Doi: 10.53555/kuey.v30i5.4699

ARTICLE INFO

ABSTRACT

This paper proposes a model based on the Integrated AI and Blockchain model. The objective of this paper is to propose a tested system which is transparent to customers, and other stakeholders, involved in the system. Processes are standardized to enable AI integration. Models become outdated with time when the base system is not re-trained. This model can acquire data and re-train itself, thus making the system more efficient and accurate over time. Smart contracts can be designed using AI and machine learning algorithms. These contracts can create blocks, which on validation, add up on a blockchain. The paper proposes a dummy model which can be scaled up by private or public bodies. This system intends to benefit customers through higher returns and appropriate risk profiling. A dummy ecosystem is used to generate entries and develop the model. This dummy ecosystem can be further upgraded as per the needs of the organization.

Keywords: Blockchain; Smart Contracts; Artificial Intelligence; Machine learning; Wealth Management

INTRODUCTION:

Blockchains are digital ledgers which are resistant to tempering. These ledgers are generally not kept with any central repository but rather distributed. Blockchain system allows users to record any transaction but restricts them from making any change after the event. Due to this transparency and security blockchain concept is used in various fields (Arshadi, 2019). Blockchain technology has a scope of automated processes true smart contracts. Integrating artificial intelligence in the process reduces human intervention to a very lesser extent (Antonopoulos, 2014).

Blockchain finds its major contribution to financial transactions and cryptocurrencies (Lamberti et al., 2017; Oberoi & Kansra, 2021). A common issue faced by financial institutions is the authenticity of records. Tampering and manipulation of records can be done to show good reports of any individual or organization. If every transaction of an individual and associated financial bodies are recorded on a blockchain, the addition or deletion of such data can be easily confirmed as true validation of blocks in the blockchain.

Organizations involved in the process may choose a “permissionless” network if they want to keep it open to the public. In this case, reading or writing to the blockchain requires no authorization. If organizations keep “permissioned” networks only a few individuals are associated organizations may get permission to read or write to the blockchain. Both models have their advantages, and the choice depends on the level of transparency and Flexibility the system wants to give to participating individuals and associated organizations.

Blockchain and its Components:

Blockchain technology stands at the forefront of a paradigm shift towards secure and transparent data management. It embodies a distributed ledger, a shared and synchronized database accessible to multiple participants. Unlike traditional ledgers maintained by a central authority, blockchain distributes the ledger across a peer-to-peer (P2P) network of nodes. Each node stores a complete copy of the ledger, fostering an environment of immutability and enhanced transparency (Nakamoto, 2008).

The components of a Blockchain are:

1. **Blocks:** Data generated from different sources can be stored in chronological order using blocks. Each of these blocks is secured through cryptography. Every block includes the hash of the previous block, thus ensuring, a tamper-proof resultant blockchain. As chronology is taken care of, any alteration in blocks would affect all subsequent blocks as a cryptographic hashing function is employed (Bonneau et al., 2018). Traditional blockchains have scalability-related limitations, which can be addressed by sharding. In this approach, ledgers are divided into smaller segments, maintained by a subset of validators. Sharding retains security levels but inter-shard communication and data consistency across shards emerges as a new challenge (Lu et al., 2018; Wang et al., 2019; Vou ligny et al., 2020).
2. **Nodes:** Nodes are the computers or devices which participate in the blockchain network. All nodes work in collaboration to validate all transactions recorded on the network. Every validation method depends on the consensus method used by the blockchain. At every node, a complete copy of the blockchain is maintained. This ensures network resilience and eliminates the risk of failure which may happen in traditional centralized systems (Yildiz, 2017).
3. **Consensus Mechanisms:** In a decentralized system, it is crucial to validate a transaction. It is also important to validate the current state of the ledger. Consensus mechanisms work on different sets of rules which govern how nodes collaborate and achieve consensus. Proof of Work (PoW) is a mechanism which incentivizes nodes that compete to validate transactions. Nodes are rewarded as they use their computational power to solve complex mathematical puzzles. In this mechanism, the first node which solves the puzzle gets an opportunity to add the next block. This is accompanied by a small reward for the node. This mechanism is good in terms of security but consumes high energy for computation (Krypton Foundation, 2017). Proof of Stake (PoS) is a mechanism where nodes are selected based on their total stake in the cryptocurrency. This selection process ensures higher energy efficiency, as compared to the proof-of-work mechanism (King & Nadal, 2012). A few more alternatives, like Byzantine Fault Tolerance (BFT) protocols, are suggested to prioritize scalability and efficiency (Battista et al., 2018; Cachin et al., 2018; Gervais et al., 2017).

Challenges in wealth management

There are many challenges faced by individuals and associated organizations, which makes wealth management a highly debatable and complex process.

The authenticity of an applicant is very important as all the transactions done before retirement and returns gained post-retirement are linked with an individual. Financial institutions associate risk with every individual based on history and other associated demographic factors (Gilmore and Pine, 2007). These parameters may vary slightly from institution to institution. An individual can change details as per the requirement and represent self as an eligible applicant for benefits or schemes. If every transaction of an individual is recorded by financial institutions, tampering with facts can be easily tracked by the next financial institution. It becomes almost impossible for an individual to change identity without a proper official process. This ensures, that the beneficiary of schemes would be the same person, for which investments were made in the past (Lombard and GibsonBrandon, 2024)

Financial organizations check credit history and past repayments to assess the credibility of any individual (Kanaparthi, 2024). It is quite possible to hide certain transactions. Institutions engage resources to do assessments for any applicant. If past applications and assessment results can be brought on a single platform, the resources of organizations may be spared. Any transaction of an individual can be recorded on blocks and can be validated (Lee D., 2016).

Multiple tasks are involved like data entry, report generation and identification of suitable products. The traditional system relies on a manual system which can result in errors and high labor costs. AI can automate these tasks thus reducing chances of error (Toh, 2023). AI algorithms can work with past data and can generate the most appropriate portfolio for customers. AI can also study large amounts of data and can use machine learning models to automate product offerings to customers (Agrawal et.al., 2024). Reconciliation is not required if blockchain technology is used for sharing ledgers.

Wealth management service providers need to understand the spending patterns of individuals as well as their saving tendencies (Panda et. al., 2023). Based on these patterns, organizations can understand the needs of a customer and can offer suitable products (Mehndiratta et. al., 2023). AI can help these institutions to understand the financial situation of an investor, risk tolerance and financial goals of customers. A perfect mix can be designed based on this information. Blockchain can further enhance the system by securing the information of investors (Dewasiri et. al., 2023).

Investments like private equity or venture capital can give higher returns but at higher risk. Retail investors find these options inaccessible due to the requirements of higher minimum investment. This complex fund structure can be broken down into smaller investments through blockchain technology. By tokenizing assets, smaller investors can contribute to the ownership. AI can assist managers in better understanding these systems. AI can even advise clients on appropriate purchases (Manda et. al., 2023).

Cybersecurity is another major concern. Through blockchain technology, decentralized and tamper-proof data can make data breach and hacking attempts futile (Odeyemi et. al., 2024). AI can further enhance this system by detecting fraudulent activity within wealth management systems.

Blockchain and AI Interoperability

Blockchain technology gives many ways to address challenges in businesses, specifically in wealth management. AI helps in the automation of sub-activities of a process. Integration of both concepts can add security, efficiency, and independent functionality to a process. This interoperability can be explained by a model which is a subset of a bigger system.

A blockchain and AI-based model is proposed which can help in addressing challenges faced by financial institutions. We have a pilot model where limited factors and information are used for demonstration. This model can be further scaled up to develop more complex models.

We are considering financial institutions' interactions with customers in this model. Financial institutions seek multiple types of information about customers before extending personalized and suitable wealth management options. The most common set of information includes personal information like name, address, date of birth or contact information. Identity proofs have unique serial numbers which may be required as an identity key for databases. These proofs are collected as a part of the KYC (know your customer) process, followed by financial institutions.

As a wealth management financial institution, information related to the credibility and repayment capacity of any individual is fetched, before offering any product or service. Asset and liability details, net worth, and investment history are some of the indicators for institutions.

Institutions also need to understand customers' expectations, and thus, financial goals and risk tolerance are assessed. Additional information, like family situation, inheritance, and legacy planning is also available.

Based on these requirements, we propose the following components in our model:

1. Financial Institutions (F_1, F_2, \dots, F_n): Financial institutions may be public or private sector banks offering wealth management solutions.
2. Individuals (I_1, I_2, \dots, I_n): Individuals who are salaried, still employed, and not yet retired.
3. Authorities (A_1, A_2, \dots, A_n): Authorities may be public offices authenticating the identity of an individual, validating different official documents, or providing different types of information to applicants

In our model we have considered the interaction where customers registering self on a blockchain. A customer requires identity proof for applying loan. Application given by an individual to the respective authority.

$I_1 \Rightarrow A_1$

This may be the first customer or n^{th} customer filling the request.

Two blockchain networks, N_1 and N_2 may be involved in this interaction. N_1 is a network which stores information of all individuals associated and registered in the system comprising institutions like F_1, F_2 etc.. To get self-registered, individual need to provide certain demographic details and information unique to his/her identity. This is generally done by public authorities but can be privatized for a private ecosystem.

Part 1: Developing Citizen Details based Blockchain, N_1

Steps involved in developing this system includes developing a blockchain which comprises of all relevant information of any citizen of a country. Information should be sufficiently enough to uniquely identify a citizen and his/her demographic details. We would call it N_1 blockchain. For demonstration purpose, we added blocks for three customers but same can be extended for n -number of customers. Information collected is also indicative, and in real application, can vary from the set of information collected in the following codes.

Python Code to develop a blockchain includes following steps:

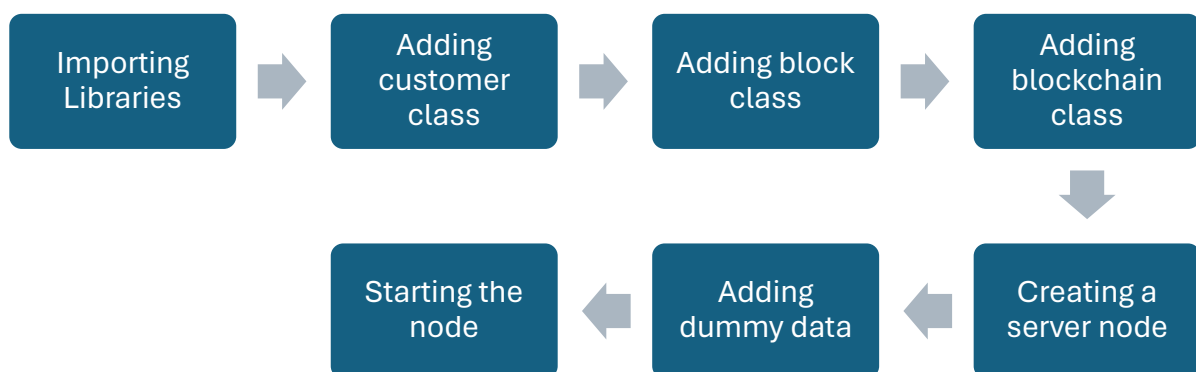


Fig.1: Citizen based blockchain – N_1

Detail code and it's output is provided in the annexure 1.

Part 2: Developing financial institutions based Blockchain, N2.

Let's assume a system where customers need to provide self details to institutions. Based on demographic and personal details, certain products are offered to individuals. Payments are made by customers, confirmation of which leads to the allotment of product.

Python Code for N2 Blockchain and it's output is provided in Annexure 2. Steps involved in these are:

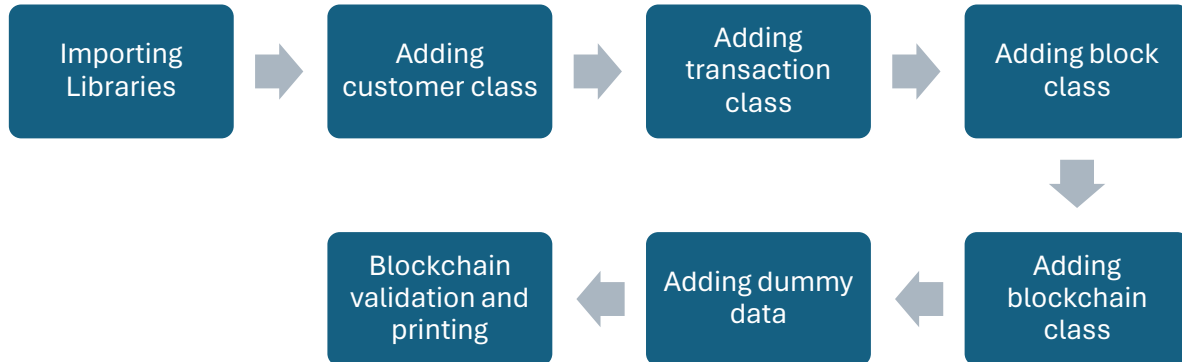


Fig.2: Financial Institution based blockchain – N2

Part 3: Interaction between two blockchains is demonstrated in this section. It is important to notice that details submitted to financial institutions need to be verified from Customer Database. As this database and further editing is recorded in blockchain N1, AI based algorithm need to check fetched details from N1 and validate it with entries submitted by customer. In case entries are validated, algorithm will return TRUE as an output. In case of mismatch and non-validation, algorithm will return FALSE. Additionally, this attempt of customer will be recorded in the blockchain so that other institutions will know customer history.

Step-1: First step in this part would be fetching data from relevant blockchains. In our dummy scenario, we are fetching customer information from N1 blockchain. Python based code, for fetching information from a blockchain, is provided in annexure 3.

Step-2: Fetching data submitted for integrating in parent blockchain. In our case, N2 blockchain represents parent blockchain.

Step-3: Validating details submitted for parent blockchain integration with details fetched from other blockchains. In our case, we will validate customer data submitted with the data fetched into database.

Step-4: In case validation is true, process will be initiated to study submitted request through machine learning model. In case of non-validation, mismatch will be reported for integration into N2 blockchain.

Step-5: Machine learning model to suggest appropriate product to customer

Step-6: If no product is opted, same is recorded otherwise transaction details are recorded

Step-7: Information generated in step 4-5 is recorded on blockchain N2.

These steps can be summarized as:

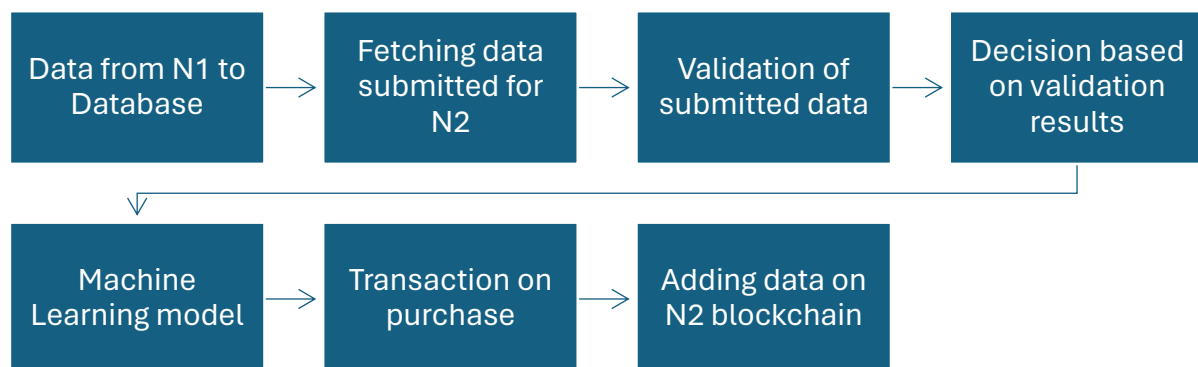


Fig.3: AI – ML based integration

Conclusion

Bringing an information on blockchain is complex but rewarding task. Project demonstrate development of two basic blockchains. These blockchains are connected through AI-ML based model. This integration of models ensures transparency through recorded history of financial institutions as well as of customers, on blockchain. Financial institutions can deploy AI based algorithms to automate services and to offer customized products to the customers. Suggested model can be further upgraded for enhanced cyber security. Further, cost cutting is possible through automation and removal of redundant information.

Annexure # 1: Python Code for Customer Data

Adding block class.

```
class Block:
```

```
def __init__(self, index, citizen_data, timestamp, previous_hash):
```

```
    self.index = index
```

```
    self.citizen_data = citizen_data
```

```
    self.timestamp = timestamp
```

```
    self.previous_hash = previous_hash
```

```
    self.nonce = self.generate_nonce()
```

```
    self.hash = self.calculate_hash()
```

#generating nonce

```
def generate_nonce(self):
```

```
    return "".join(random.choices(string.ascii_uppercase + string.digits, k=10))
```

#hashing

```
def calculate_hash(self):
```

```
    # Convert citizen data to a string
```

```
    data_string = str(vars(self.citizen_data))
```

```
    # Hash the data using SHA-256
```

```
    sha = hashlib.sha256(data_string.encode()).hexdigest()
```

```
    # Combine data with block information and hash again
```

```
    block_string = f'{self.index}{data_string}{self.timestamp}{self.previous_hash}{self.nonce}'
```

```
    return hashlib.sha256(block_string.encode()).hexdigest()
```

#Adding blockchain class.

```
class Blockchain:
```

```
def __init__(self):
```

```
    self.chain = []
```

```
    self.difficulty = 5 # Set the difficulty for mining
```

```
    self.create_genesis_block()
```

```
    # Creating genesis block with first customer data
```

```
    def create_genesis_block(self):
```

```
        customer_C1 = Citizen("C1 Name", "C1 Father Name", "C1 Address",
```

```
                                "C1 Phone Number", "c1@email.com", "PANooo1", "AADHARooo1")
```

```
        genesis_block = Block(0, customer_C1, time.time(), "o")
```

```
        self.mine_block(genesis_block)
```

```
        self.chain.append(genesis_block)
```

```
    # Mining block
```

```
    def mine_block(self, block):
```

```
        while block.hash[:self.difficulty] != "o" * self.difficulty:
```

```
            block.nonce = block.generate_nonce()
```

```
            block.hash = block.calculate_hash()
```

```
            print(f"Block mined: {block.hash}")
```

```
    # Adding block
```

```
    def add_block(self, new_block):
```

```
        new_block.previous_hash = self.chain[-1].hash
```

```
        self.mine_block(new_block)
```

```
        self.chain.append(new_block)
```

```
    # Validity Check
```

```
    def is_valid(self):
```

```
        for i in range(1, len(self.chain)):
```

```
            current_block = self.chain[i]
```

```
            previous_block = self.chain[i - 1]
```

```
            if current_block.hash != current_block.calculate_hash():
```

```
                print("Hash mismatch!")
```



```

return False
if current_block.previous_hash != previous_block.hash:
    print("Previous hash mismatch!")
    return False
return True
#Creating a server node
def get_blockchain_data(self):
    return self.chain
def handle_client(conn, blockchain):
    blockchain_data = blockchain.get_blockchain_data()
    conn.send(pickle.dumps(blockchain_data))
    conn.close()
# Creating a server node at port 8080, to which client node can connect.
def start_node(blockchain):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', 8080))
    server_socket.listen(5)
# Generating message on successful connection of client node
    print("Node is listening for incoming connections...")
    while True:
        conn, addr = server_socket.accept()
        print(f"Connection from {addr} has been established.")
        threading.Thread(target=handle_client, args=(conn, blockchain)).start()
    if __name__ == "__main__":
        blockchain_n1 = Blockchain()
#Adding dummy citizen objects
# Create Citizen objects with C2 information
        customer_C2 = Citizen("C2 Name", "C2 Father Name", "C2 Address",
                               "C2 Phone Number", "c2@email.com", "PANooo2", "AADHARooo2")
        block_1 = Block(1, customer_C2, time.time(), "")
        blockchain_n1.add_block(block_1)
# Create Citizen objects with C3 information
        customer_C3 = Citizen("C3 Name", "C3 Father Name", "C3 Address",
                               "C3 Phone Number", "c3@email.com", "PANooo3", "AADHARooo3")
        block_2 = Block(2, customer_C3, time.time(), "")
        blockchain_n1.add_block(block_2)
#Starting the node
        start_node(blockchain_n1)

```

Results:

```

Block mined: 000e4207f1b3adb8550b327e9bf1129bb5160acd7db4d33b11bdo877c59d5cbd
Block mined: 00045251a274a08236c7923bdab94e6744b2955f4b16e5c985006d7995f971
Block mined: 000248989601ddbcb7e422a0a06eb273a96bbod15d2ef9e2de83ce4530bcad28
Node is listening for incoming connections...
Connection from ('127.0.0.1', 57238) has been established.

```

Annexure#2: Python Code for Financial Institution

Nonce added

```

def generate_nonce(self):
    return ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))
# Hashing
def calculate_hash(self):
    # Convert transaction data to a string
    data_string = str([(trans.institution, trans.customer.name, trans.investment_amount, trans.product,
                        trans.result, trans.timestamp) for trans in self.transactions])
    # Hash the data using SHA-256
    sha = hashlib.sha256(data_string.encode()).hexdigest()
    # Combine data with block information and hash again
    block_string = f'{self.index}{data_string}{self.timestamp}{self.previous_hash}{self.nonce}'
    return hashlib.sha256(block_string.encode()).hexdigest()
#Adding Blockchain Class. It has an option to set difficulty level
class Blockchain:
    def __init__(self):
        self.chain = []

```

```

self.difficulty = 4 # Set the difficulty for mining
self.create_genesis_block()
# Creating genesis block
def create_genesis_block(self):
    genesis_block = Block(0, ["F1", "F2", "F3"], [], time.time(), "o")
    self.mine_block(genesis_block)
    self.chain.append(genesis_block)
# Defining mining block
def mine_block(self, block):
    while block.hash[:self.difficulty] != "o" * self.difficulty:
        block.nonce = block.generate_nonce()
        block.hash = block.calculate_hash()
    print(f"Block mined: {block.hash}")
# Setting definition of transaction where involved institution and customer, amount invested, product offered and result of final deal is captured.
def add_transaction(self, institution, customer, investment_amount, product, result):
    new_transaction = Transaction(institution, customer, investment_amount, product, result, time.time())
    if self.validate_transaction(new_transaction):
        new_block = Block(len(self.chain), [institution], [new_transaction], time.time(), self.chain[-1].hash)
        self.mine_block(new_block)
        self.chain.append(new_block)
    else:
        print("Transaction is not valid and will not be added to the blockchain.")
# Setting definition to validate transaction. Details entered by customers to be verified in this step. AI intervention is required here which can interact with N1 and can verify details entered in N2.
def validate_transaction(self, transaction):
# Placeholder for validation logic, returns True for now. It is separately coded.
    return True
# Hash validation
def is_valid(self):
    for i in range(1, len(self.chain)):
        current_block = self.chain[i]
        previous_block = self.chain[i - 1]
        if current_block.hash != current_block.calculate_hash():
            print("Hash mismatch!")
            return False
        if current_block.previous_hash != previous_block.hash:
            print("Previous hash mismatch!")
            return False
    return True
# Printing information on blockchain. This is not required on actual blockchain. It is included in this code only for developmental purpose and can be remove before publishing.
def print_blockchain(self):
    for block in self.chain:
        print(f"\n--- Block {block.index} ---")
        print(f"Institutions: {block.institutions}")
        print("Transactions:")
        for transaction in block.transactions:
            print(f"Institution: {transaction.institution}")
            print(f"Customer Name: {transaction.customer.name}")
            print(f"Investment Amount: {transaction.investment_amount}")
            print(f"Product: {transaction.product}")
            print(f"Result: {transaction.result}")
            print(f"Timestamp: {transaction.timestamp}")
            print(f"Timestamp: {block.timestamp}")
            print(f"Previous Hash: {block.previous_hash}")
            print(f"Hash: {block.hash}")
# Main code to create blockchain using above created classes and variables.
# Main code
blockchain = Blockchain()
# Sample transactions
customer_C1 = Citizen("C1 Name", "C1 Father Name", "C1 Address",
"C1 Phone Number", "c1@email.com", "PANooo1", "AADHARooo1")

```

```

blockchain.add_transaction("F1", customer_C1, 10000, "p1", "Approved")
customer_C2 = Citizen("C2 Name", "C2 Father Name", "C2 Address",
"C2 Phone Number", "c2@email.com", "PAN0002", "AADHAR0002")
blockchain.add_transaction("F2", customer_C2, 15000, "p2", "Approved")
customer_C3 = Citizen("C3 Name", "C3 Father Name", "C3 Address",
"C3 Phone Number", "c3@email.com", "PAN0003", "AADHAR0003")
blockchain.add_transaction("F3", customer_C3, 20000, "p3", "Approved")

```

Print Blockchain

```

if blockchain.is_valid():
    print("\nBlockchain is valid!")
    blockchain.print_blockchain()
else:
    print("\nBlockchain is invalid!")

```

Results:

```

Block mined: 00006c2f8dfe2f1c28dec244b1d8oad95987c2802d67eca27cb92ff2a621c8b4
Block mined: 00002357d78ff6a4e422481d9e451e9f8e804a3f4bd8608183f5d4922643489d
Block mined: 00009ae37c7f6833c6f9bade319f394417c70dd901e72b68362b8d4dec6d4d32
Block mined: 0000479b54aa77f99158a5908cfca0c3eafa9c5b25c0ce3282e07d787d1fa944

```

Blockchain is valid!

--- Block 0 ---

Institutions: ['F1', 'F2', 'F3']

Transactions:

Timestamp: 1713199640.3593652

Previous Hash: 0

Hash: 00006c2f8dfe2f1c28dec244b1d8oad95987c2802d67eca27cb92ff2a621c8b4

--- Block 1 ---

Institutions: ['F1']

Transactions:

Institution: F1

Customer Name: C1 Name

Investment Amount: 10000

Product: p1

Result: Approved

Timestamp: 1713199640.43437

Timestamp: 1713199640.4343731

Previous Hash: 00006c2f8dfe2f1c28dec244b1d8oad95987c2802d67eca27cb92ff2a621c8b4

Hash: 00002357d78ff6a4e422481d9e451e9f8e804a3f4bd8608183f5d4922643489d

--- Block 2 ---

Institutions: ['F2']

Transactions:

Institution: F2

Customer Name: C2 Name

Investment Amount: 15000

Product: p2

Result: Approved

Timestamp: 1713199640.476576

Timestamp: 1713199640.476577

Previous Hash: 00002357d78ff6a4e422481d9e451e9f8e804a3f4bd8608183f5d4922643489d

Hash: 00009ae37c7f6833c6f9bade319f394417c70dd901e72b68362b8d4dec6d4d32

--- Block 3 ---

Institutions: ['F3']

Transactions:

Institution: F3

Customer Name: C3 Name

Investment Amount: 20000

Product: p3

Result: Approved

Timestamp: 1713199640.7982762

Timestamp: 1713199640.7982779

Previous Hash: 00009ae37c7f6833c6f9bade319f394417c70dd901e72b68362b8d4dec6d4d32

Hash: 0000479b54aa77f99158a5908cfca0c3eafa9c5b25c0ce3282e07d787d1fa944

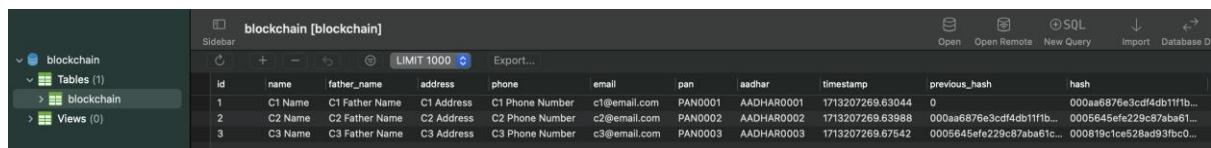
Annexure # 3: Python Code for Fetching information from Blockchain**# defining blockchain data fetching. Establishing connection to host socket: 8080**

```

def fetch_blockchain_data():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8080))
    client_socket.send(b"get_blockchain_data")
    blockchain_data = pickle.loads(client_socket.recv(4096))
    client_socket.close()
    return blockchain_data

# defining fetched data saving to a database
def save_to_database(blockchain_data):
    conn = sqlite3.connect('blockchain.db')
    c = conn.cursor()
    # Creating a blankdata base for first time and updating records to this database in subsequent
    # addition
    c.execute("""CREATE TABLE IF NOT EXISTS blockchain
    (id INTEGER PRIMARY KEY, name TEXT, father_name TEXT, address TEXT,
    phone TEXT, email TEXT, pan TEXT, aadhar TEXT, timestamp REAL,
    previous_hash TEXT, hash TEXT)""")
    # for block in blockchain_data:
    citizen_data = block.citizen_data
    data = (citizen_data.name, citizen_data.father_name,
    citizen_data.address, citizen_data.phone, citizen_data.email,
    citizen_data.pan, citizen_data.aadhar, block.timestamp,
    block.previous_hash, block.hash)
    c.execute("INSERT INTO blockchain (name, father_name, address, phone, email, pan, aadhar, timestamp,
    previous_hash, hash) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", data)
    conn.commit()
    conn.close()
    if __name__ == "__main__":
        blockchain_data = fetch_blockchain_data()
        save_to_database(blockchain_data)
        print("Blockchain data saved to database successfully.")

```

It will be saved in a sqldatabase as:


id	name	father_name	address	phone	email	pan	aadhar	timestamp	previous_hash	hash
1	C1 Name	C1 Father Name	C1 Address	C1 Phone Number	c1@email.com	PAN0001	AADHAR0001	1713207269.63044	0	000aa6876e3cdf4db11fb...
2	C2 Name	C2 Father Name	C2 Address	C2 Phone Number	c2@email.com	PAN0002	AADHAR0002	1713207269.63988	000aa6876e3cdf4db11fb...	0005645efe229c87aba61...
3	C3 Name	C3 Father Name	C3 Address	C3 Phone Number	c3@email.com	PAN0003	AADHAR0003	1713207269.67542	0005645efe229c87aba61...	000819c1ce528ad93fbc0...

Annexure # 4: Validation and Updation of blockchain

```

class FinancialBlockchain(Blockchain):
    def add_transaction(self, institution, customer, investment_amount, product, result):
        new_transaction = Transaction(institution, customer, investment_amount, product, result, time.time())
        if self.validate_transaction(new_transaction):
            new_block = Block(len(self.chain), [institution], [new_transaction], time.time(), self.chain[-1].hash)
            self.mine_block(new_block)
            self.chain.append(new_block)
        else:
            print("Transaction is not valid and will not be added to the blockchain.")
        def validate_transaction(self, transaction):
            blockchain_n1_data = fetch_blockchain_data()
            for block in blockchain_n1_data:
                citizen_data = block.citizen_data
                if (citizen_data.name == transaction.customer.name and
                citizen_data.father_name == transaction.customer.father_name and
                citizen_data.address == transaction.customer.address and
                citizen_data.phone == transaction.customer.phone and
                citizen_data.email == transaction.customer.email and
                citizen_data.pan == transaction.customer.pan and
                citizen_data.aadhar == transaction.customer.aadhar):
                    print(f"Validation success for customer: {transaction.customer.name}")

```

```

return True
print(f"Validation failure for customer: {transaction.customer.name}")
return False
class Transaction:
def __init__(self, institution, customer, investment_amount, product, result, timestamp):
self.institution = institution
self.customer = customer
self.investment_amount = investment_amount
self.product = product
self.result = result
self.timestamp = timestamp
if __name__ == "__main__":
blockchain_n2 = FinancialBlockchain()
customer_C1 = Citizen("C1 Name", "C1 Father Name", "C1 Address",
"C1 Phone Number", "c1@email.com", "PAN0001", "AADHAR0001")
blockchain_n2.add_transaction("F1", customer_C1, 10000, "p1", "Approved")
customer_C2 = Citizen("C2 Name", "C2 Father Name", "C2 Address",
"C2 Phone Number", "c2@email.com", "PAN0002", "AADHAR0002")
blockchain_n2.add_transaction("F2", customer_C2, 15000, "p2", "Approved")
customer_C3 = Citizen("C3 Name", "C3 Father Name", "C3 Address",
"C3 Phone Number", "c3@email.com", "PAN0003", "AADHAR0003")
blockchain_n2.add_transaction("F3", customer_C3, 20000, "p3", "Approved")
if blockchain_n2.is_valid():
print("\nBlockchain N2 is valid!")
else:
print("\nBlockchain N2 is invalid!")

```

Output:

Blockchain N2 is valid!

Annexure# 5: Machine Learning Model suggesting product

```

# Machine Learning Model
def train_model():
# Sample data for training
data = {
'name': ['C1', 'C2', 'C3', 'C4'],
'father_name': ['C1 Father', 'C2 Father', 'C3 Father', 'C4 Father'],
'address': ['Address1', 'Address2', 'Address3', 'Address4'],
'phone': ['Phone1', 'Phone2', 'Phone3', 'Phone4'],
'email': ['Email1', 'Email2', 'Email3', 'Email4'],
'pan': ['PAN1', 'PAN2', 'PAN3', 'PAN4'],
'aadhar': ['AADHAR1', 'AADHAR2', 'AADHAR3', 'AADHAR4'],
'investment_amount': [10000, 15000, 20000, 25000],
'product': ['p1', 'p2', 'p3', 'p4'],
'feedback': [1, 0, 1, 0], # 1 for positive feedback, 0 for negative feedback
'accepted': [1, 0, 1, 0] # 1 for accepted, 0 for rejected
}
df = pd.DataFrame(data)
X = df[['investment_amount', 'feedback']]
y = df['accepted']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Model accuracy: {accuracy_score(y_test, y_pred)}")
return model
def offer_product(model, customer, investment_amount, feedback):
customer_data = {'investment_amount': [investment_amount], 'feedback': [feedback]}
df = pd.DataFrame(customer_data)
prediction = model.predict(df)
return prediction[0] == 1
if __name__ == "__main__":
model = train_model()
blockchain_n2 = FinancialBlockchain()
customer_C1 = Citizen("C1 Name", "C1 Father Name", "C1 Address",

```

```

"C1 Phone Number", "c1@email.com", "PAN0001", "AADHAR0001")
investment_amount = 10000
feedback = 1 # Positive feedback
if offer_product(model, customer_C1, investment_amount, feedback):
    print("Product offered to customer C1")
    blockchain_n2.add_transaction("F1", customer_C1, investment_amount, "p1", "Accepted")
else:
    print("Product not offered to customer C1")
    blockchain_n2.add_transaction("F1", customer_C1, investment_amount, "p1", "Rejected")
if blockchain_n2.is_valid():
    print("\nBlockchain N2 is valid!")
else:
    print("\nBlockchain N2 is invalid!")

```

Output

Model accuracy: 1.0

Product offered to customer C1

Block mined: 00000a1b2c3d4e5f6g7h8i9jok1l2m3n4o5p6q7r8s9tou1v2w3x4y5z

Transaction added to blockchain: Institution: F1, Customer: C1 Name, Investment Amount: 10000, Product: p1, Result: Accepted

Blockchain N2 is valid!

References

1. Agrawal, S.S., Rose, N. and PrabhuSahai, K., 2024. THE FINTECH REVOLUTION: AI'S ROLE IN DISRUPTING TRADITIONAL BANKING AND FINANCIAL SERVICES. *Decision Making: Applications in Management and Engineering*, 7(1), pp.243-256.
2. Battista, M., Conti, M., & Hahn, C. (2018, July). Blockchain and Byzantine Fault Tolerance: The state of the art. <https://arxiv.org/pdf/2210.14003>
3. Bonneau, J., Bünz, B., Danezis, C., Günther, D., Kalra, V., Lerner, H., Lucks, C., Millien, P., Nascimento, D., Segfault, M., Silva, M., & Wuille, M. (2018). Bitcoin: A decentralized consensus mechanism. <https://eprint.iacr.org/>
4. Cachin, C., Vukolić, M., & Zhao, Q. (2018, April). On the long-standing security concerns of permissionless blockchains. <https://eprint.iacr.org/2018/459>
5. Dewasiri, N.J., Karunarathne, K.S.S.N., Menon, S., Jayarathne, P.G.S.A. and Rathnasiri, M.S.H., 2023. Fusion of Artificial Intelligence and Blockchain in the Banking Industry: Current Application, Adoption, and Future Challenges. In *Transformation for Sustainable Business and Management Practices: Exploring the Spectrum of Industry 5.0* (pp. 293-307). Emerald Publishing Limited.
6. Gervais, A., Ritzdorf, H., Rivier, G., & Schulte-Naumburg, D. (2017, August). How to make a Hyperledger Fabric network scalable by reducing consensus overhead
7. Gilmore, J.H. and Pine, B.J., 2007. *Authenticity: What consumers really want*. Harvard Business Press.
8. Kanaparthi, V., 2024. AI-based Personalization and Trust in Digital Finance. *arXiv preprint arXiv:2401.15700*.
9. King, S., & Nadal, S. (2012, August 19). PPcoin: Peer-to-peer cryptocurrency with proof of stake. <https://assets.press.princeton.edu/chapters/s10908.pdf>
10. Lombard, E. and GibsonBrandon, R.N., 2024. Do Wealth Managers understand Codes of Conduct and their ethical dilemmas? Lessons from an online survey. *Journal of Business Ethics*, 189(3), pp.553-572.
11. Lu, J., Wang, Y., Li, X., & Li, L. (2018, June). A survey of blockchain applications: A multi-dimensional perspective. <https://ieeexplore.ieee.org/document/8669067>
12. Manda, V.K., Manda, V.K. and Katneni, V., 2023. Blockchain for the asset management industry. *World Review of Science, Technology and Sustainable Development*, 19(1-2), pp.170-185.
13. Mehndiratta, N., Arora, G. and Bathla, R., 2023, May. The use of Artificial Intelligence in the Banking Industry. In *2023 International Conference on Recent Advances in Electrical, Electronics & Digital Healthcare Technologies (REEDCON)* (pp. 588-591). IEEE.
14. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. <https://bitcoin.org/bitcoin.pdf>
15. Odeyemi, O., Okoye, C.C., Ofodile, O.C., Adeoye, O.B., Addy, W.A. and Ajayi-Nifise, A.O., 2024. Integrating AI with blockchain for enhanced financial services security. *Finance & Accounting Research Journal*, 6(3), pp.271-287.
16. Panda, A.C., Patnaik, A., Pawar, A. and Birari, A., 2023. Adoption of Fintech Towards Asset and Wealth Management: Understanding the Recent Scenario in India. In *INDAM: Indian Academy of Management at SBM-NMIMS Mumbai* (pp. 357-367). Singapore: Springer Nature Singapore.
17. Toh, Y.L., 2023. Addressing Traditional Credit Scores as a Barrier to Accessing Affordable Credit. *Economic Review*, 108(3), pp.1-22.

18. Yildiz, B. (2017, December). The future of commerce: Blockchain.
<https://www.sciencedirect.com/science/article/abs/pii/S0007681321000355>