Blockchain & Cryptocurrency Lab 01
K21-4579 K21-4871

Task 1: Client Registration, Communication, and Ledger Synchronization

Objective
The objective of this task is to implement a basic client-server architecture that registers clients, facilitates message exchanges between clients and servers, and synchronizes transactions across two servers using a file-based ledger.

Implementation Details
Key Components
- Ledger File: A JSON file that stores registered clients and transaction history.
- Client Hash: SHA-256 hash of client names used as a unique identifier.

register_clients.py
- Registers clients by hashing their names and storing them in the ledger.
- Ensures each client is uniquely added.

client1.py and client2.py
- Each client generates a message and sends it to two server endpoints.
- Threads are used for concurrent message sending to multiple servers.

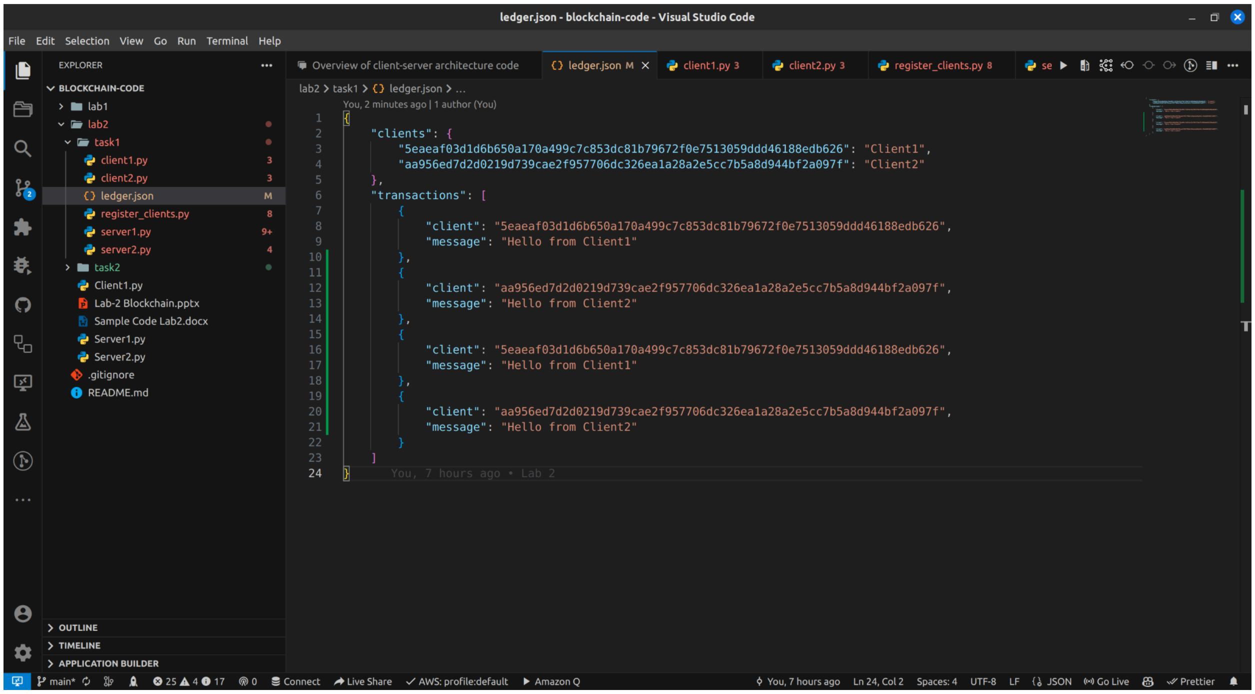
server1.py
- Validates incoming messages using the client hash and the ledger.
- Logs transactions in the ledger and synchronizes the transaction with Server 2.

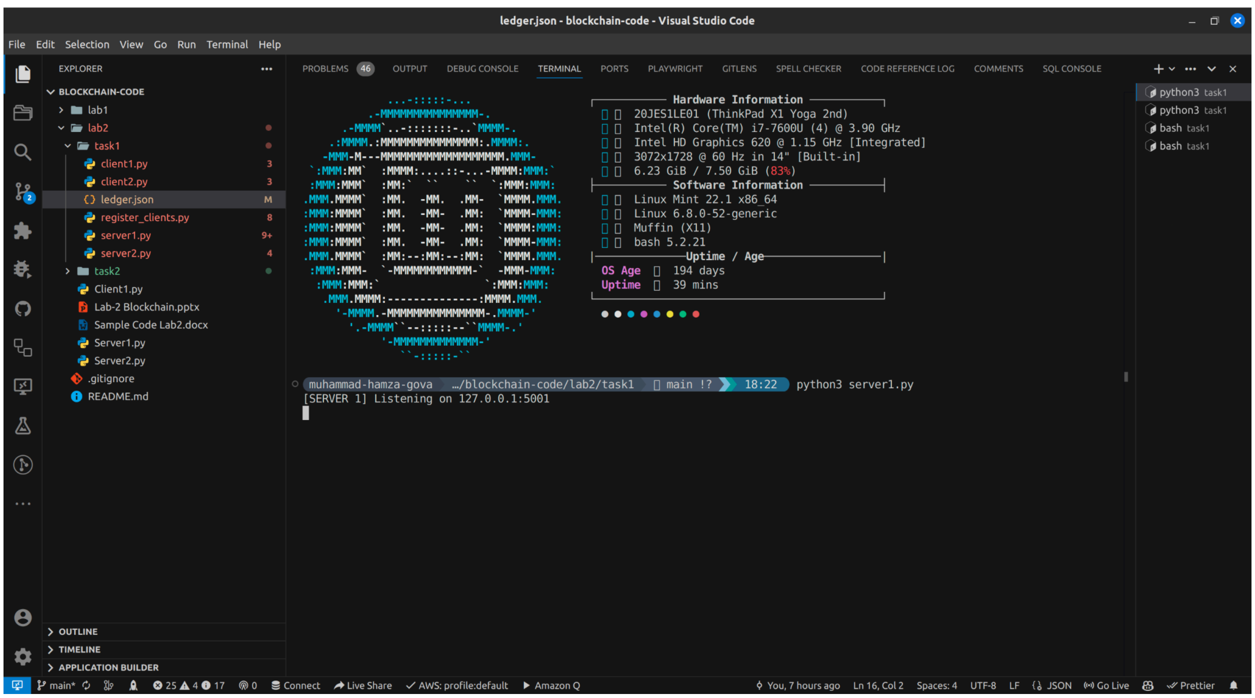
server2.py
- Listens for ledger updates from Server 1 and prints the receive transactions.

Results
- Clients are successfully registered and recognized using their hashed identifiers.
- Both Client1 and Client2 send messages to the servers; Server1 logs the transactions and synchronizes with Server2.
- Server2 receives and displays the ledger updates sent from Server1.

Conclusion
This task demonstrates an integrated approach to client registration, secure message exchange, and transaction logging with ledger synchronization between servers. The system forms a foundational step toward building more complex distributed applications with enhanced security and communication protocols.

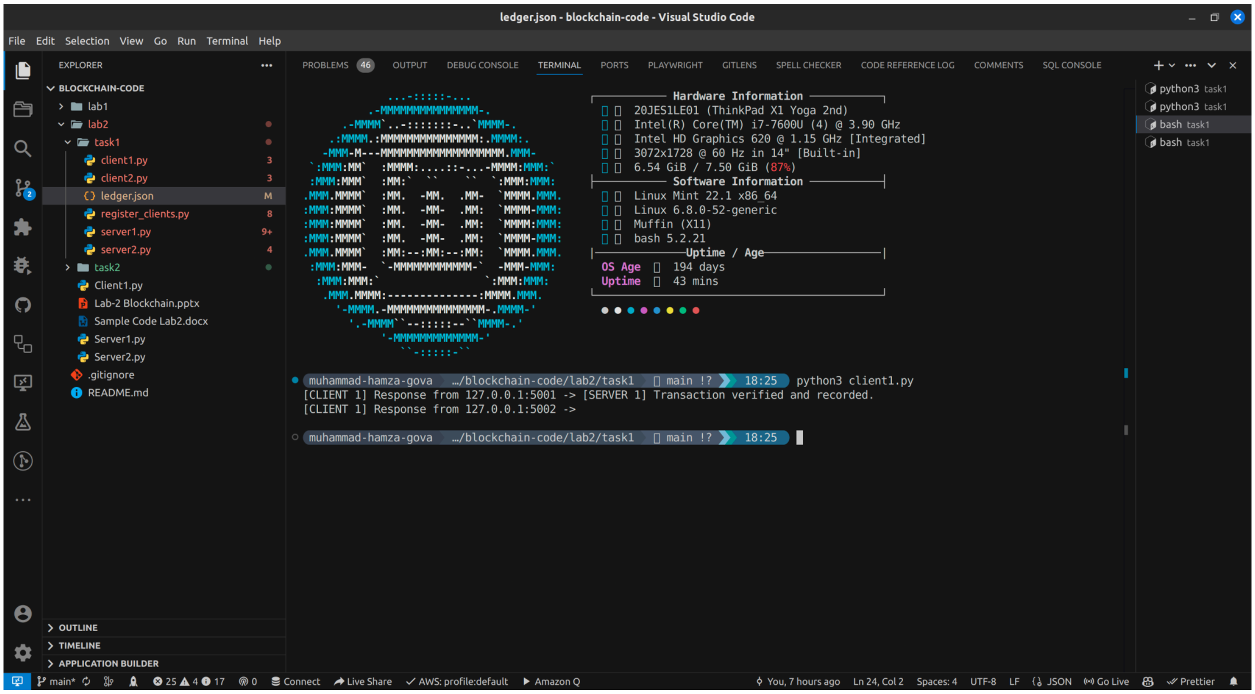
Blockchain & Cryptocurrency Lab 01
K21-4579 K21-4871

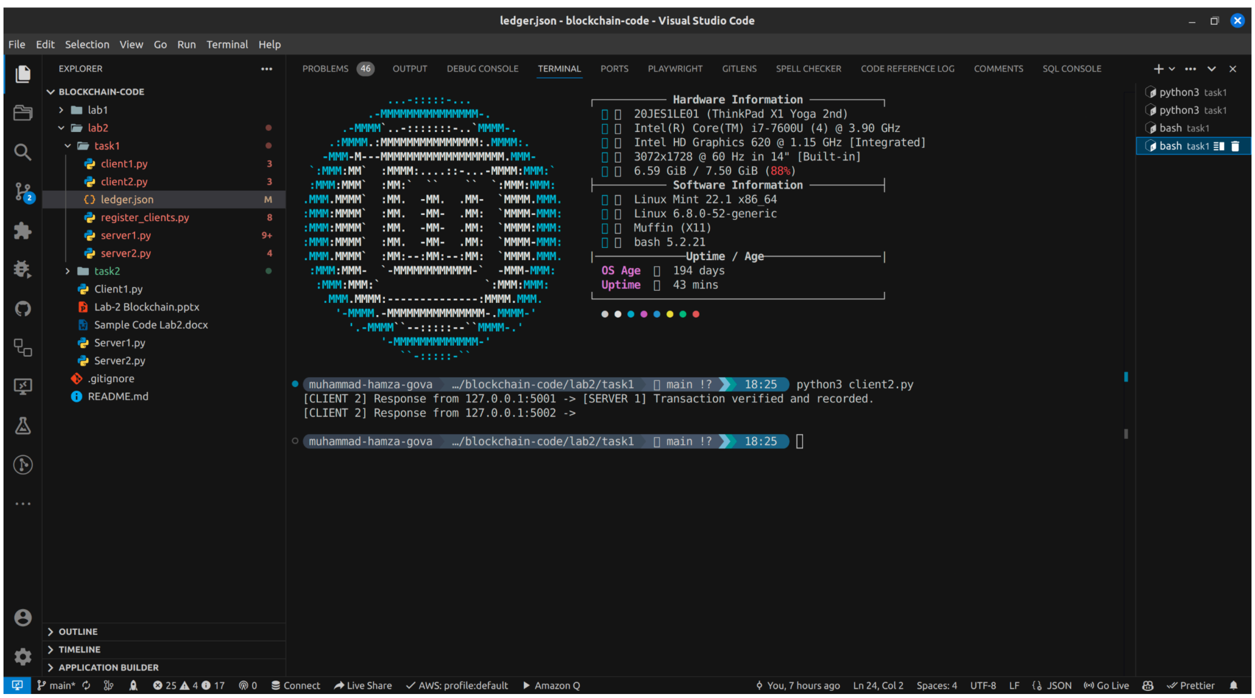




Blockchain & Cryptocurrency Lab 01

K21-4579 K21-4871





Task 2: Distributed Ledger Update Simulation

Objective
The objective of this task is to simulate a distributed ledger system with multiple servers and clients, demonstrating how networked components interact when processing transactions. This task also investigates the effects on consensus when one server intentionally rejects a transaction.

Implementation Details

Key Components
• Ledger File: A JSON file (ledger.json) used to record registered clients and the history of transactions.
• Client Registration: A mechanism that hashes client names using SHA-256 and stores them in the ledger to ensure clients are identifiable.
• Server 1:
  - Listens on port 5001 and receives client messages.
  - Parses incoming messages but intentionally rejects and does not record any valid transactions.
  - Contains placeholder functions for validating, logging transactions, and syncing with another server.
• Server 2:
  - Listens on port 5002 and accepts ledger update transactions sent from other nodes.
  - Processes and logs incoming transactions by printing a confirmation, functioning as a backup ledger node.
• Clients:
  - Client1 and Client2 send formatted messages containing their names and corresponding messages to both servers, testing the multi-server communication.

Client Registration
• The register_clients.py script loads the existing ledger or creates a new one.
• It computes a SHA-256 hash for each client name and updates the ledger if the client is not already registered.
• This ensures that only recognized clients can interact within the system in potential future transactions.

Server 1
• Accepts client connections on port 5001, processes messages by extracting client name and message content.
• Intentionally rejects legitimate messages without updating the ledger or syncing with Server 2, simulating a failed consensus node.

Server 2

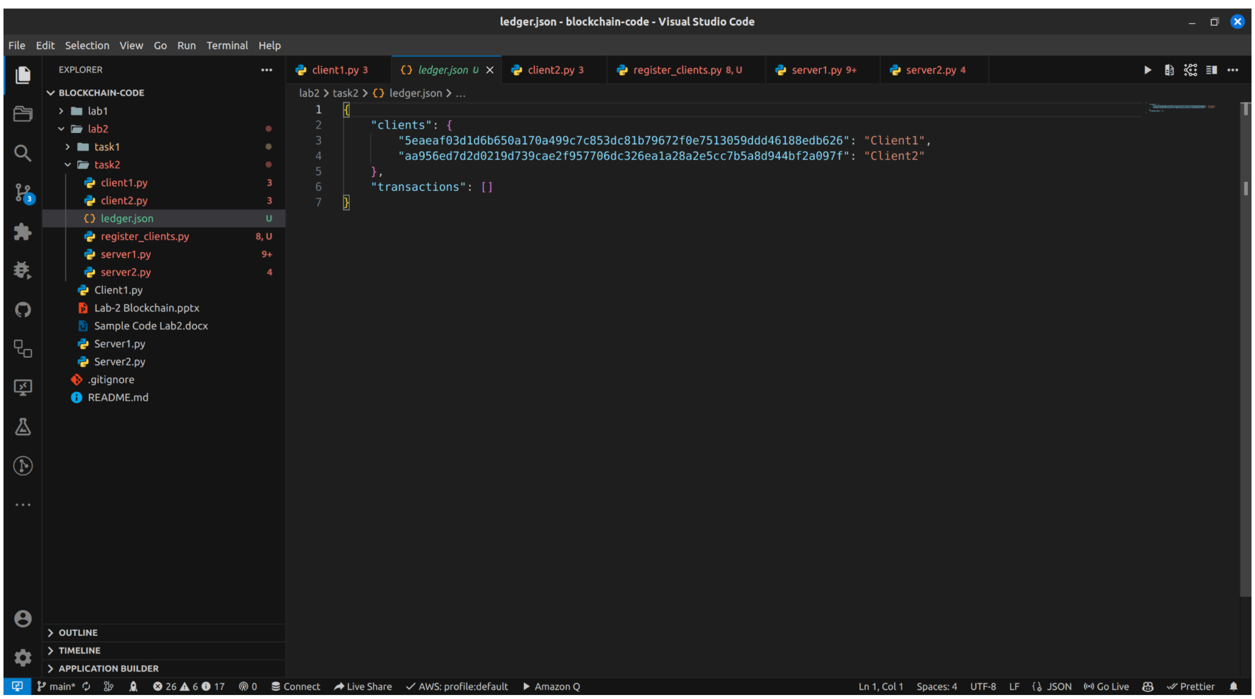
Blockchain & Cryptocurrency Lab 01

K21-4579 K21-4871

• Accepts and logs any ledger update transactions on port 5002 by decoding and printing the received JSON data.
• Functions as the secondary node in the network that could potentially validate or store transactions.

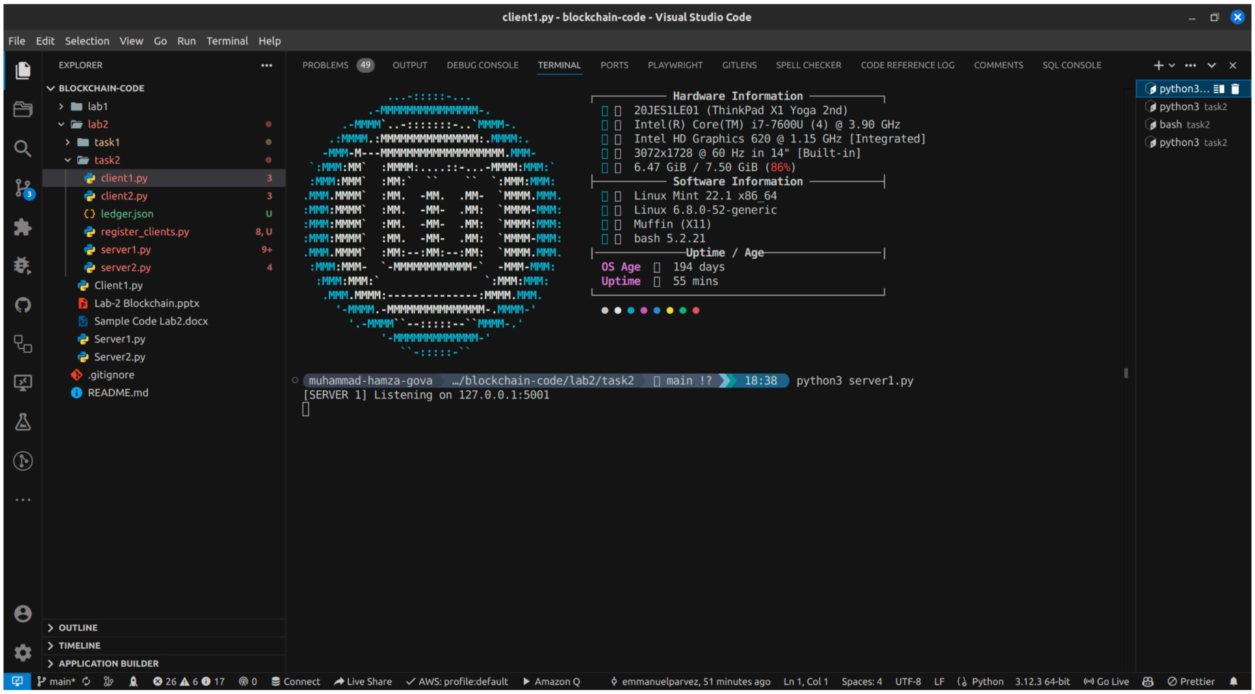
Results
• Client registration successfully hashes and stores the client data in the ledger.
• Both Client1 and Client2 send messages to Servers 1 and 2.
• Server 1 rejects and does not record incoming transactions, while Server 2 logs received ledger update transactions.
• The consensus mechanism fails as the system is unable to achieve agreement between the servers—one server processes the transaction while the other rejects it, leading to an inconsistent state.

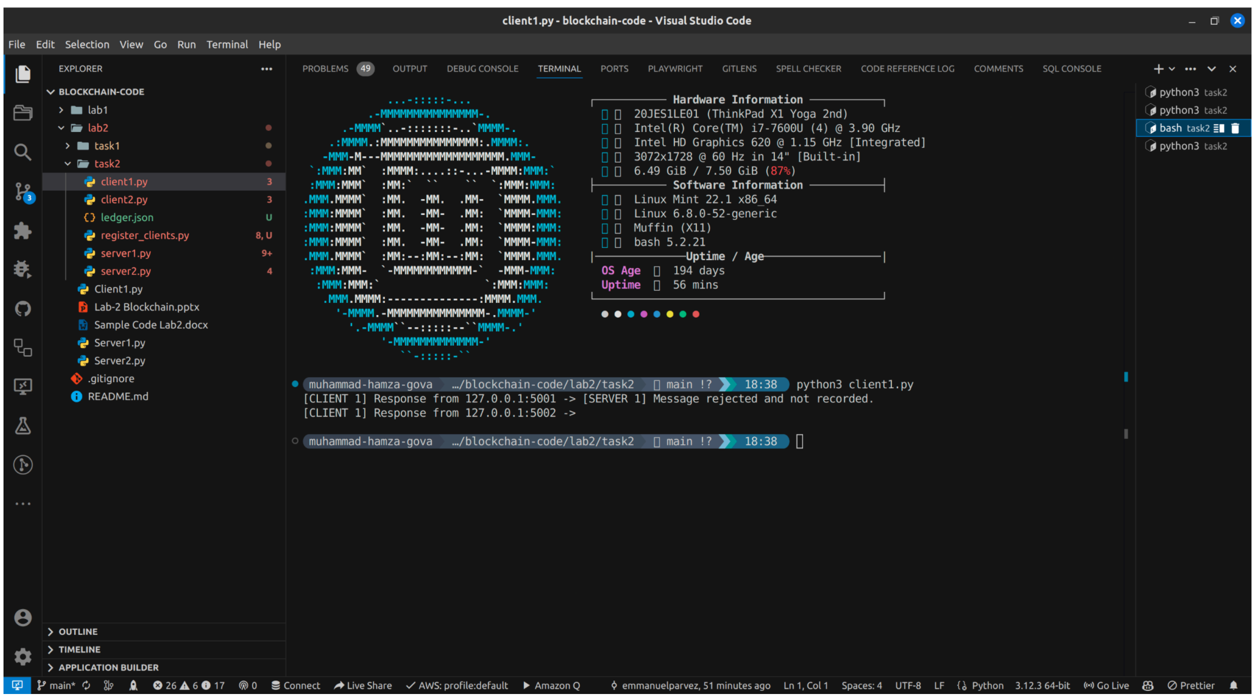
Conclusion
This task demonstrates a basic distributed ledger update simulation using multiple servers and clients. The intentional rejection of transactions by Server 1 exposes a critical issue in consensus mechanisms: if one node in the network refuses to process a transaction, the system cannot reach uniform agreement. This finding highlights the importance of robust consensus protocols in distributed systems to ensure consistency and integrity across all nodes.



Blockchain & Cryptocurrency Lab 01

K21-4579 K21-4871





Blockchain & Cryptocurrency Lab 01
K21-4579 K21-4871