Họ và tên: Bùi Nguyên Quốc Bảo **MSSV: 24C12002**

Distributed Sytem – LAB 02

I. Checklist:

	Implemented	Note
Client	✓	CLI input, RPC client
Proxy	✓	RPC Server, Middleware to route request from Client to Primary node
Server	✓	RPC Server, FastDB, Primary-Backup node
Leader Election	✓	Implements the Bully Algorithm for leader election among nodes using RPC.
Replication	✓	Manages data replication across nodes to ensure consistency and reliability by copying the state machine (command) from primary node to each backup nodes.
Order of requests	×	(Sequential consistency) -> Timestamp or logic clock implementation.

II. Code overview

Project structure:

Fig 1: Project structure

❖ **Client side**:

```
lab02
├ client
│  ├ commands
│  │  └ handlers.go
│  ├ config
│  │  ├ cli.go
│  │  └ rpc_client.go
│  ├ dispatcher
│  │  └ dispatcher.go
│  └ main.go
```

- Client service has the same functionality as Lab1 but communicating with PROXY server instead of FastDB node directly.

❖ **Proxy server:**

```
├ proxy
│  ├ route
│  │  └ rpc_route.go
│  └ main.go
```

- **router**/: Implements logic for routing commands to the appropriate node (PRIMARY NODE). Keeps track of the current leader node's address (port)

❖ **FastDB Node**

```
├ server
│  ├ config
│  │  ├ fastdb.go
│  │  └ service.go
│  ├ fastdb_rpc
│  │  ├ controller
│  │  │  └ rpc.go
│  │  ├ operation
│  │  │  └ operation.go
│  │  └ utils
│  │  │  └ utils.go
│  ├ proxy_rpc
│  │  └ proxy
│  │  │  └ proxy_client.go
│  ├ replication_rpc
│  │  ├ core
│  │  │  ├ node.go
│  │  │  └ peer.go
│  │  └ event
│  │  │  ├ bus.go
│  │  │  ├ event.go
│  │  │  └ message.go
│  └ main.go
├ go.mod
└ go.sum
```

- **config**/: Contains configuration-related functionalities for both initializing the FastDB database and initializing node in replication cluster. Registering RPC service for the FastDB server to handle remote request and RPC service for each Node to communicate with each others.
- **fastdb/controller**/: Endpoint of the FastDB server. Expose services/methods allow remote clients to perform various FastDB operation like SET, GET, GETALL, DELETE, logging and response back to the client the status of operation. Moreover, to replicate data between other nodes for SET and DELETE command, the leader broadcasts the data and the command to all followers.
- **proxy/proxy_client**: To communicate with proxy server the information of Primary node (Port)

- **replication**: managing nodes, peers, and replication state
  - core/:
    - Node: Represents a single node in the cluster. Handles peer management, replication tasks, node states, and leader election (*HealthCheckLeader, StartLeaderElection, BroadcastToPeers, RegisterWithPeers, IsLeader,…)*
    - Peers: Manages the list of peers in the cluster. Tracks peer statuses, RPC connections, and allows adding or removing peers dynamically.
  - event/:
    - Define the event message to communicate between nodes for leader election or replication updates, or trigger an action like *HealthCheckLeader* using event-driven architecture
- **main.go**: Entry point for the server-side implementation sets up the TCP listener, initializes the in-memory database, registers the RPC service to handle Node communication and User request, trigger leader election and listens for incoming client connections

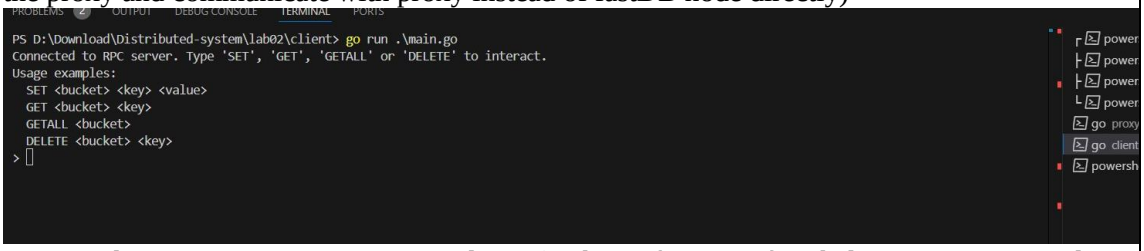
III. Testing result

❖ How to start the application:

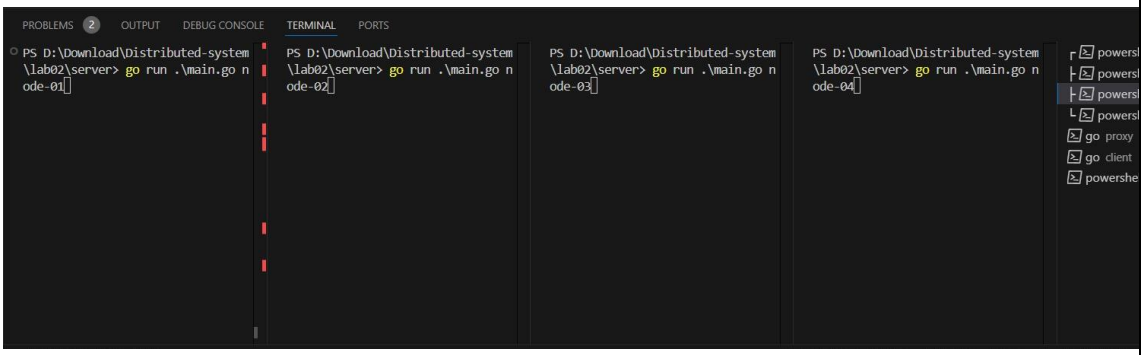
Step 1: Install fastdb package and zerolog for logging using this cmd:
- go get github.com/marcelloh/fastdb
- go get github.com/rs/zerolog

Step 2: cd proxy + go run .\main.go (Must run the proxy server first to avoid starting up issue with fastDB node. This proxy running on port 1234)

Step 3: cd client + go run .\main.go (To start the client, then client will be able to connect to the proxy and communicate with proxy instead of fastDB node directly)

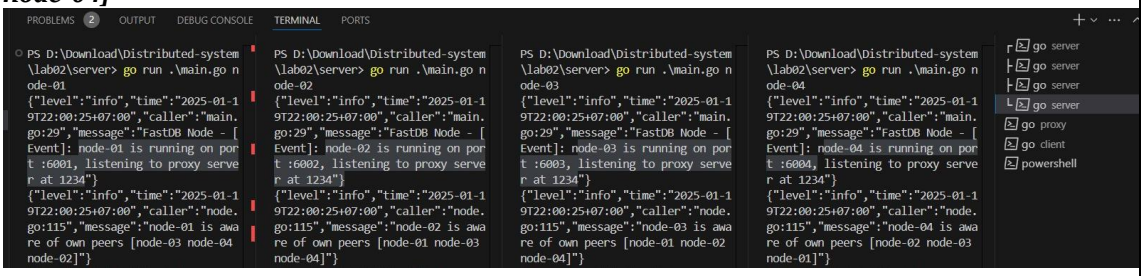


Step 4: cd server + go run .\main.go node-0x (with x = {1, 2, 3, 4} as below picture to simulate multi-node running)

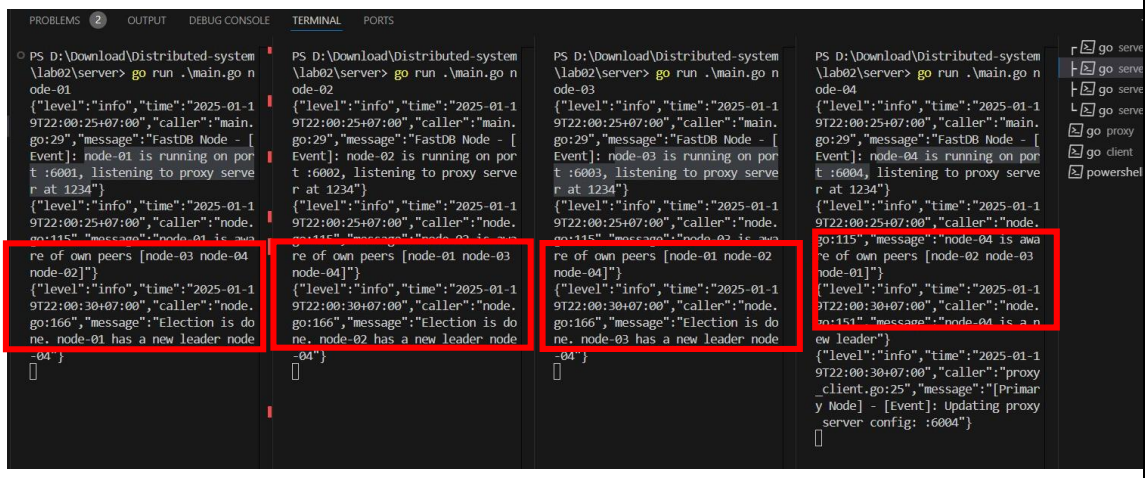


Then we can observe from the logs which port each node is currently running on and how each node know each own peers (by registering to other peer or PING). For instance:

node-01 is running on port :6001
node-02 is aware of own peers [node-01 node-03 node-04]



After a delays (5 secs) 22:00:30. Each node will trigger a leader election event . Node which owns the highest ID (Node-04) will become the leader and notify to other followers so the followers can continously running the leader's healthcheck job.



Proxy server now receives the elected event and update the primary node address to route client request to primary instance (Node-04: port 6004).

❖ **Testing Summary Report:**

✓ TC01: PUT/GET to/from the primary.

Input following commands into the client terminal:

SET city 1 HoChiMinh
SET city 2 HaNoi
SET city 3 DaNang
GETALL city
DELETE city 3
GETALL city

Client logs:

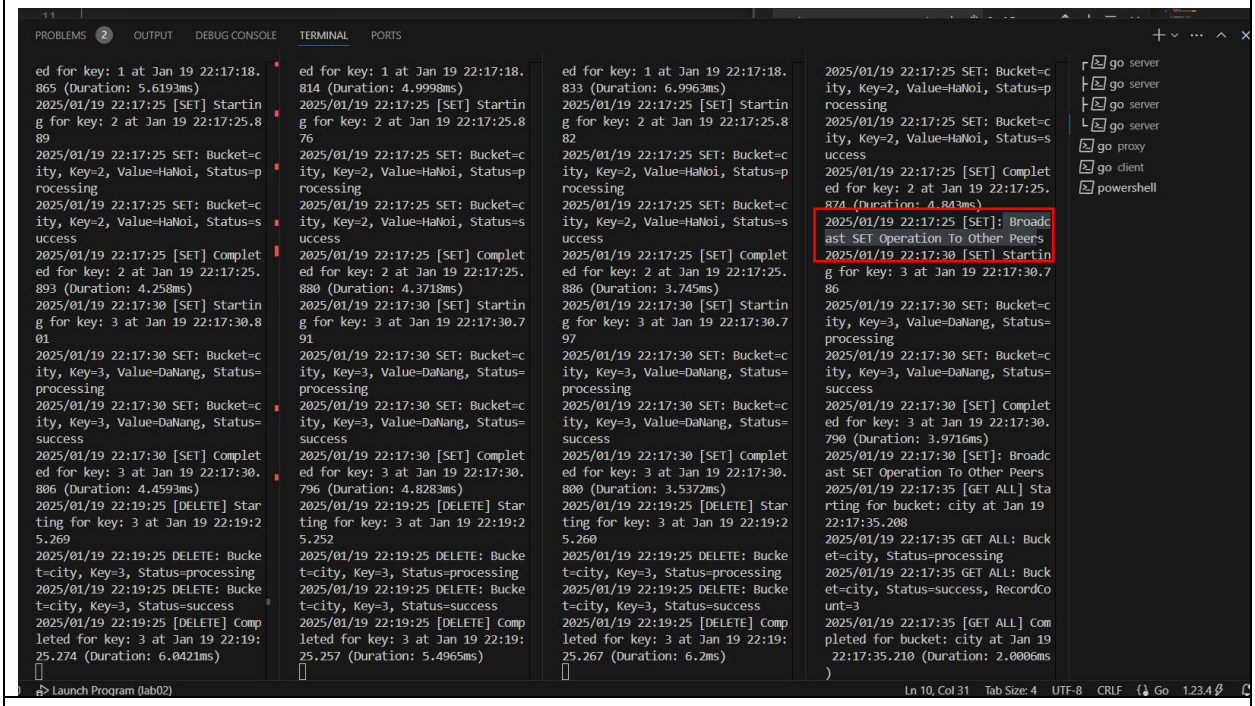
```
PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\Download\Distributed-system\lab02\client> go run .\main.go
Connected to RPC server. Type 'SET', 'GET', 'GETALL' or 'DELETE' to interact.
Usage examples:
  SET <bucket> <key> <value>
  GET <bucket> <key>
  GETALL <bucket>
  DELETE <bucket> <key>
> SET city 1 HoChiMinh
Reply from server: Saved successfully
> SET city 2 HaNoi
Reply from server: Saved successfully
> SET city 3 DaNang
Reply from server: Saved successfully
> GETALL city
Reply from server: Key: 2, Value: "HaNoi"
Reply from server: Key: 3, Value: "DaNang"
Reply from server: Key: 1, Value: "HoChiMinh"
> DELETE city 3
Reply from server: Deleted entry (which owns key: 3) successfully.
> GETALL city
Reply from server: Key: 2, Value: "HaNoi"
Reply from server: Key: 1, Value: "HoChiMinh"
>
```

Proxy server logs:

```
PS D:\Download\Distributed-system\lab02\proxy> go run .\main.go
2025/01/19 21:58:58 message = [Proxy server] is listening on [port: 1234]
{"level":"info","time":"2025-01-19T22:00:30+07:00","message":"[Proxy Server] - [Event]: new primary node has been promoted, updated listener port: :6
004"}
{"level":"info","time":"2025-01-19T22:17:18+07:00","message":"[Proxy Server] - [Event]: Routing SET command to primary node"}
{"level":"info","time":"2025-01-19T22:17:25+07:00","message":"[Proxy Server] - [Event]: Routing SET command to primary node"}
{"level":"info","time":"2025-01-19T22:17:30+07:00","message":"[Proxy Server] - [Event]: Routing SET command to primary node"}
{"level":"info","time":"2025-01-19T22:17:35+07:00","message":"[Proxy Server] - [Event]: Routing GETALL command to primary node"}
{"level":"info","time":"2025-01-19T22:19:25+07:00","message":"[Proxy Server] - [Event]: Routing DELETE command to primary node"}
{"level":"info","time":"2025-01-19T22:19:27+07:00","message":"[Proxy Server] - [Event]: Routing GETALL command to primary node"}
```

Primary (Leader) node logs:

We can observe that the primary node not only handles the command but also broadcasts the command to other peers from below screenshot.



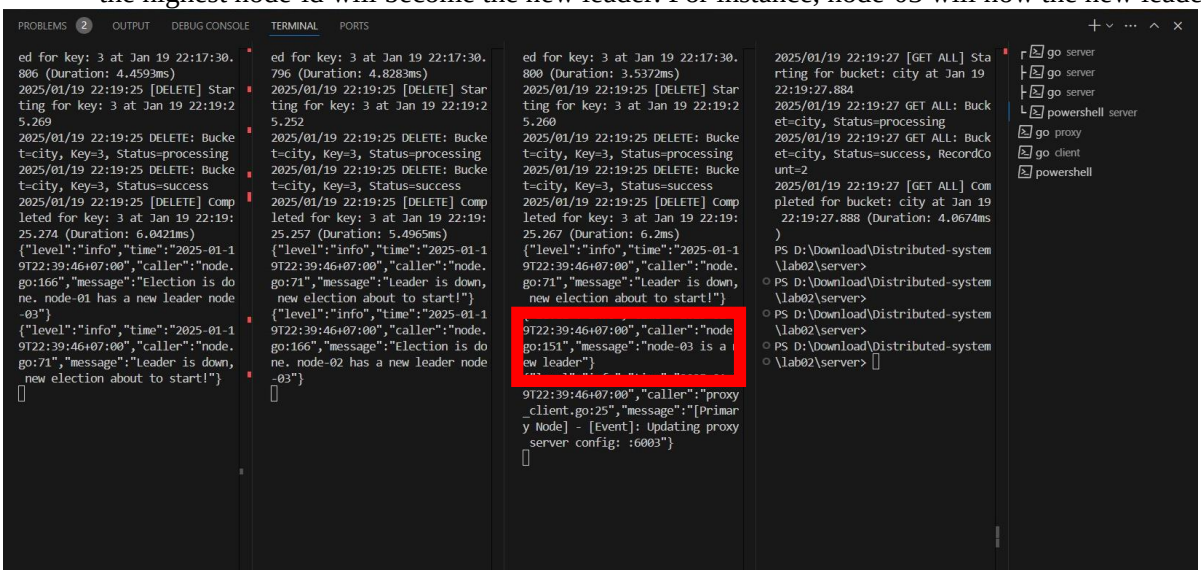
Backup (Follower) node logs:

Executing the same commands as its leader.

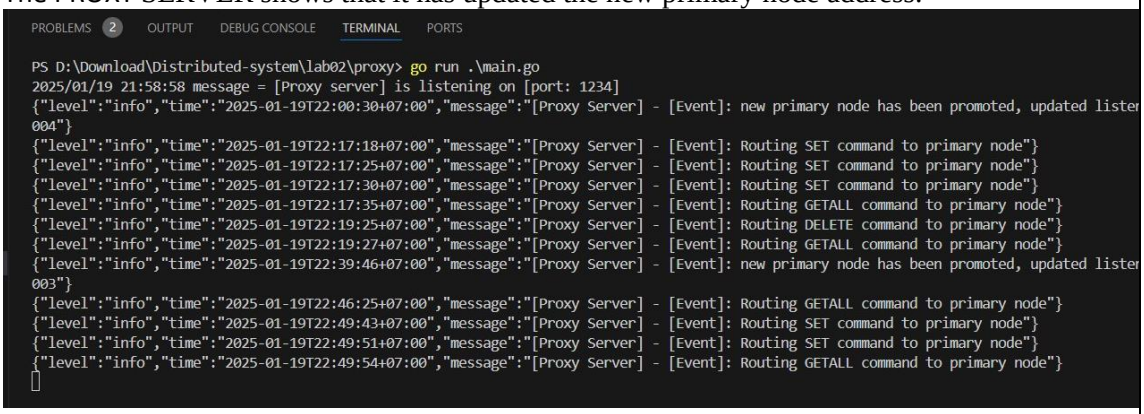
✓ TC02: The primary is dead, GET from backup.

Test Scenarios

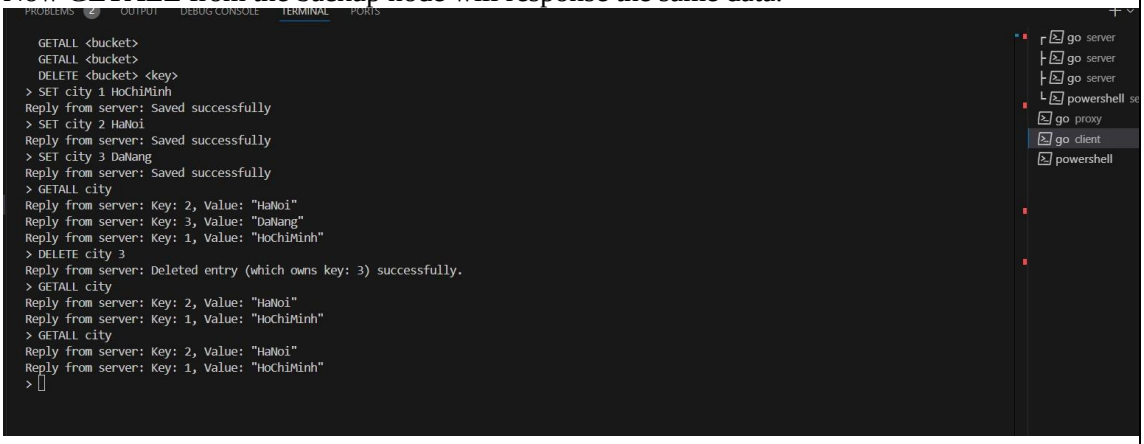
- Terminate node-04 process (Ctrl + C). Now node-04 is down. Followers now checking the leader healthcheck status will trigger the leader ELECTION event. As before, node which have the highest node-id will become the new leader. For instance, node-03 will now the new leader



- The PROXY SERVER shows that it has updated the new primary node address:



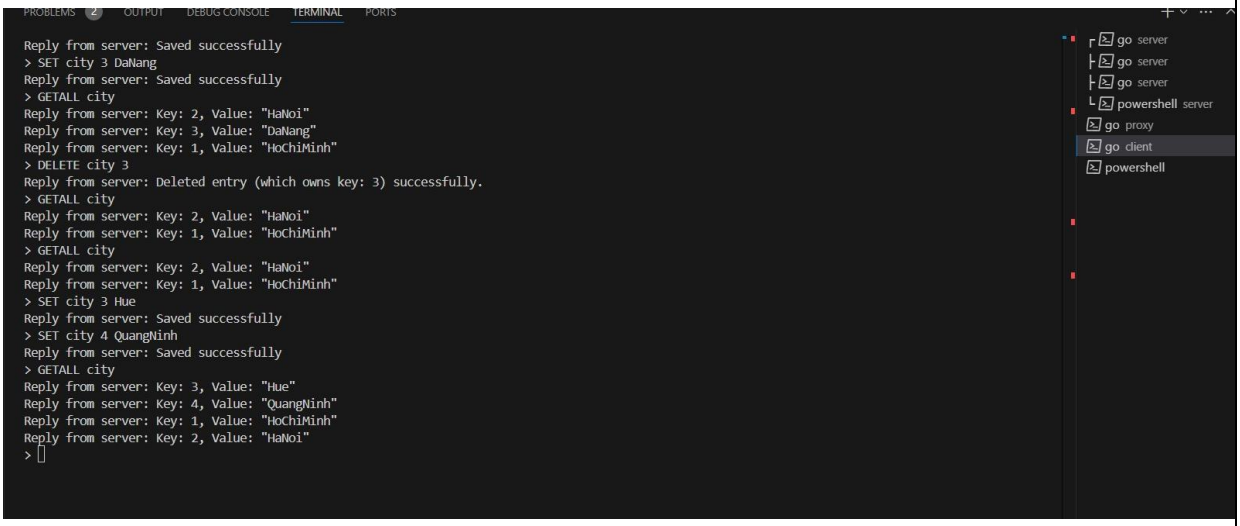
- Now GETALL from the backup node will response the same data:



✓ TC03: The primary is dead, PUT/GET from backup:

Test Scenarios:

SET city 3 Hue
SET city 4 QuangNinh
GETALL



Node-03 now will broadcast all operations like SET or DELETE to all other followers