VaultCLI: Distributed File Storage

Team Members:
Md Sarim Shamim
Meet Borisagar
Devesh Sharma
Aman Kumar
Muthres Gurjar
Vivek Sawalkar

Scope: A secure distributed storage system with fault tolerance capabilities

Objectives:

- Distributed architecture for fault tolerance
- Data confidentiality through encryption
- Data integrity verification via HMAC
- Data availability/ reconstruction through Reed-Solomon erasure coding
- User authentication and secure data management

System Architecture Overview - Components

- ***Client** :* Handles encryption, chunking, and Reed-Solomon coding
- ***Data Servers (14)***: 10 for storing data chunks, 4 for parity chunks
- ***Metadata Servers (2)**:* Primary and backup storing user credentials and file metadata

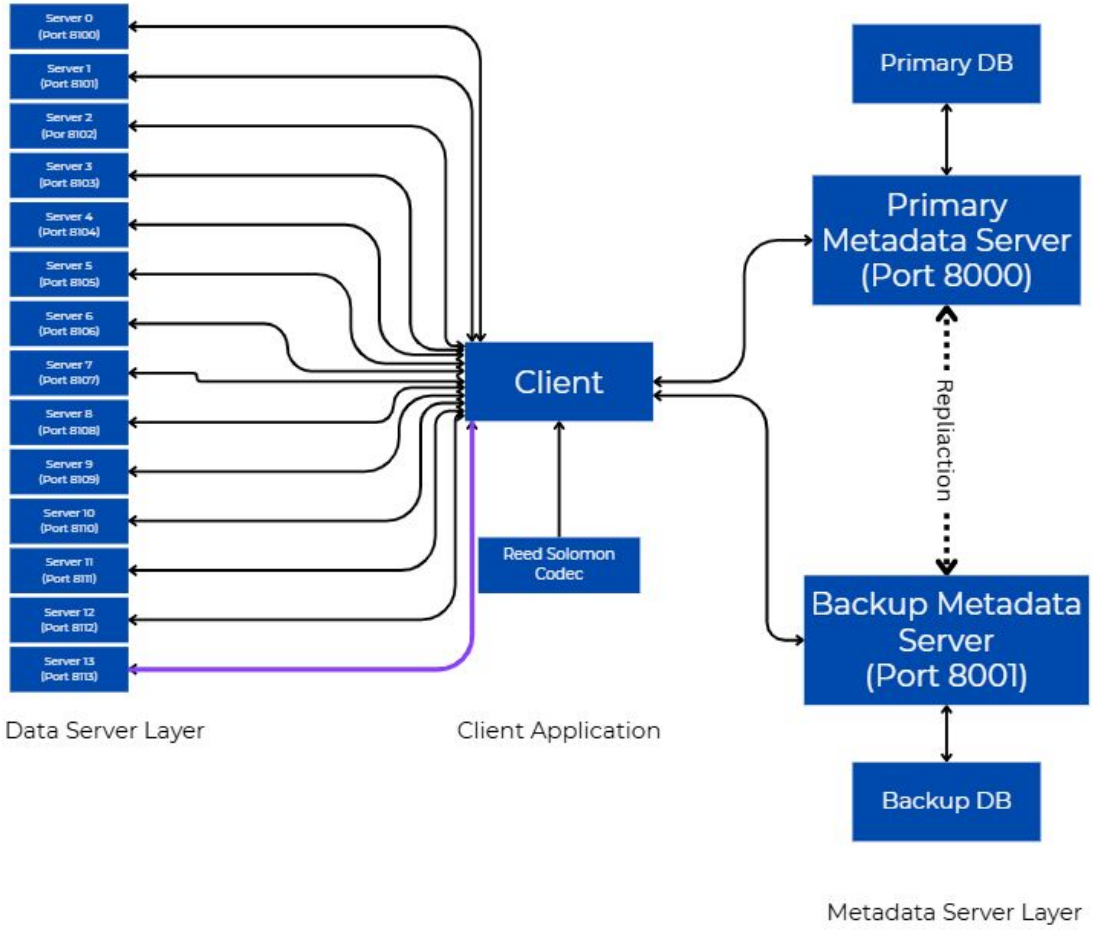
System Architecture Overview - Communication

- **Client ↔ Metadata**: JSON over TCP/IP for authentication, file listing, and metadata management

```
{"operation": "register", "username": ..., "password_hash": ...}
```

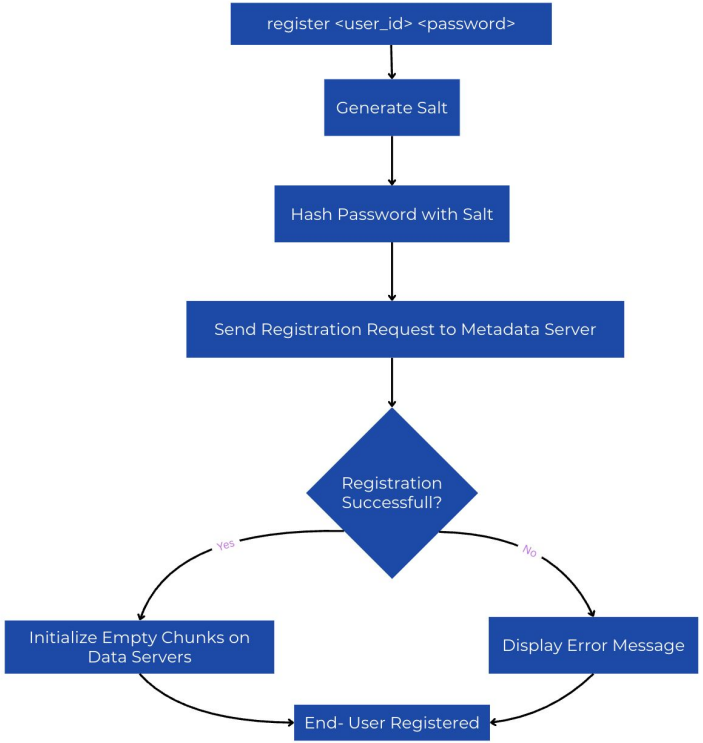
- **Client ↔ Data Servers**: single-letter operation type ('U'/'D'/'X'), JSON headers, and binary data transfer, all over TCP

- **Health Monitoring**: TCP connection tests with 1-second timeout to determine server availability before operations


Server 0 (Port 8100)
Server 1 (Port 8101)
Server 2 (Por 8102)
Server 3 (Port 8103)
Server 4 (Port 8104)
Server 5 (Port 8105)
Server 6 (Port 8106)
Server 7 (Port 8107)
Server 8 (Port 8108)
Server 9 (Port 8109)
Server 10 (Port 8110)
Server 11 (Port 8111)
Server 12 (Port 8112)
Server 13 (Port 8113)
Client
Reed Solomon Codec
Primary DB
Primary Metadata Server (Port 8000)
Repliaction
Backup Metadata Server (Port 8001)
Backup DB
Data Server Layer
Client Application
Metadata Server Layer

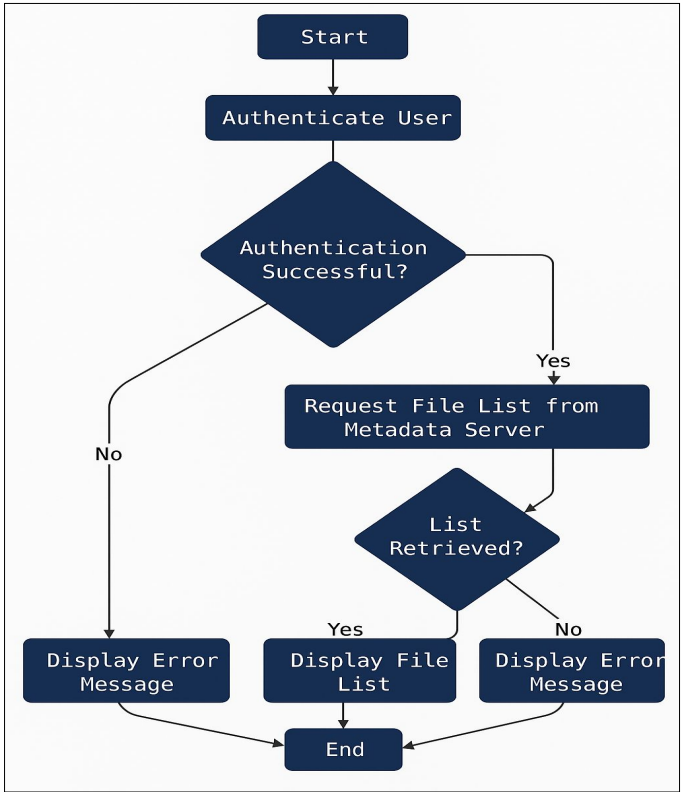
User Registration

- **Salt Generation:**
  Client generates a random salt and hashes the password with it.
- **Secure Registration Request:**
  Sends registration action with user ID, salted password hash, and salt to metadata server via TCP/IP (JSON format).
- **Initialization:**
  On success, initializes empty chunks across available data servers.



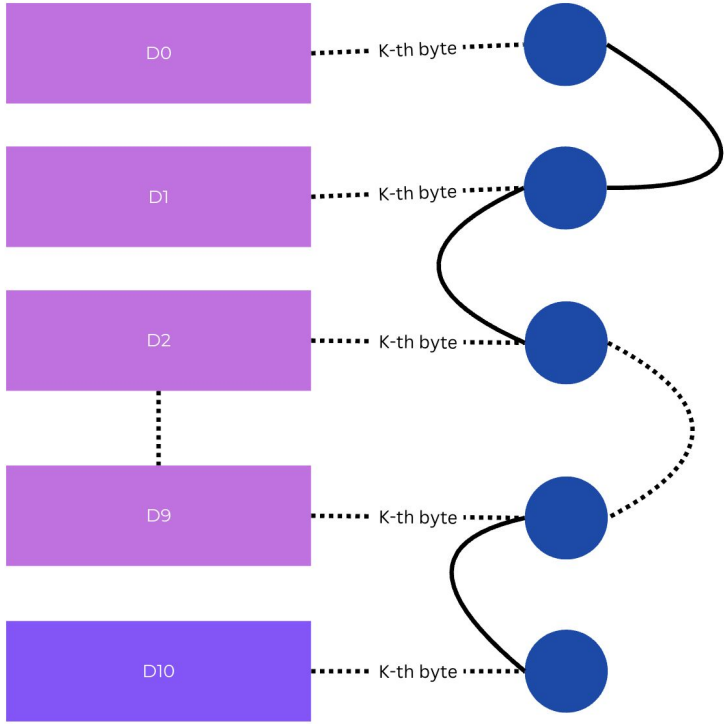
List Files

- **Authentication First**:
  User must authenticate with metadata server using salted password hash.
- **Request File Listing**:
  Client sends a list file action over TCP/IP with a JSON request.
- **Receive and Display**:
  Server responds with a list of user's files or a "no files" message.



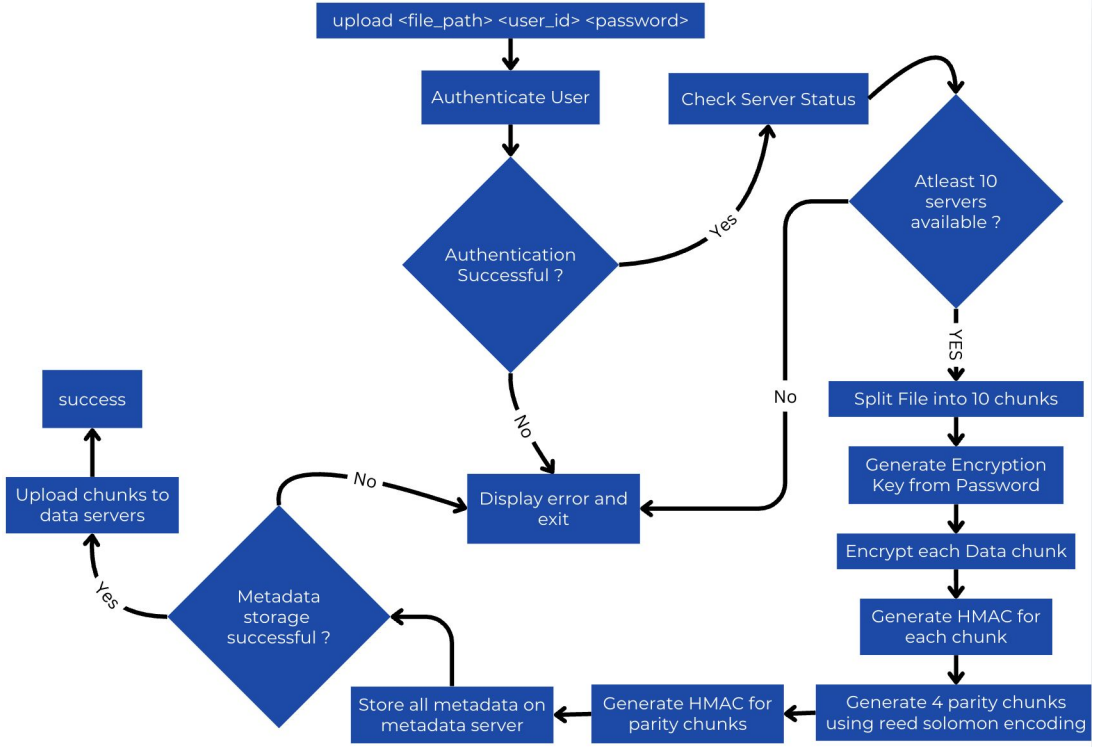
Reed-Solomon Overview

- **Reed-Solomon codes** protect data by adding parity chunks to recover lost or corrupted parts.
- **Column-wise processing** applies error correction on a stripe across all chunks for maximum resilience.
- **With 10 data chunks and 4 parity chunks**, the system can fully recover even if 4 chunks are missing.



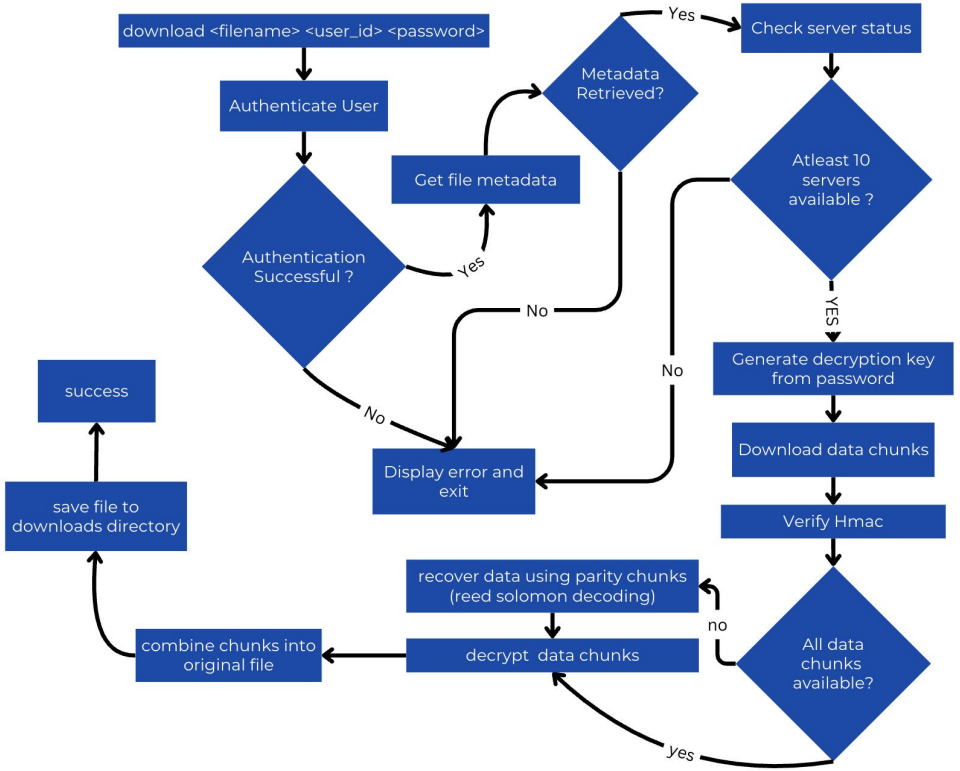
Upload Files

- User authentication with metadata server before file upload begins
- Encrypts file data using AES256 with unique IVs per chunk, with HMAC verification
- Implements Reed-Solomon encoding to generate parity chunks for redundancy
- Distributes encrypted chunks across multiple data servers with chunk metadata stored separately



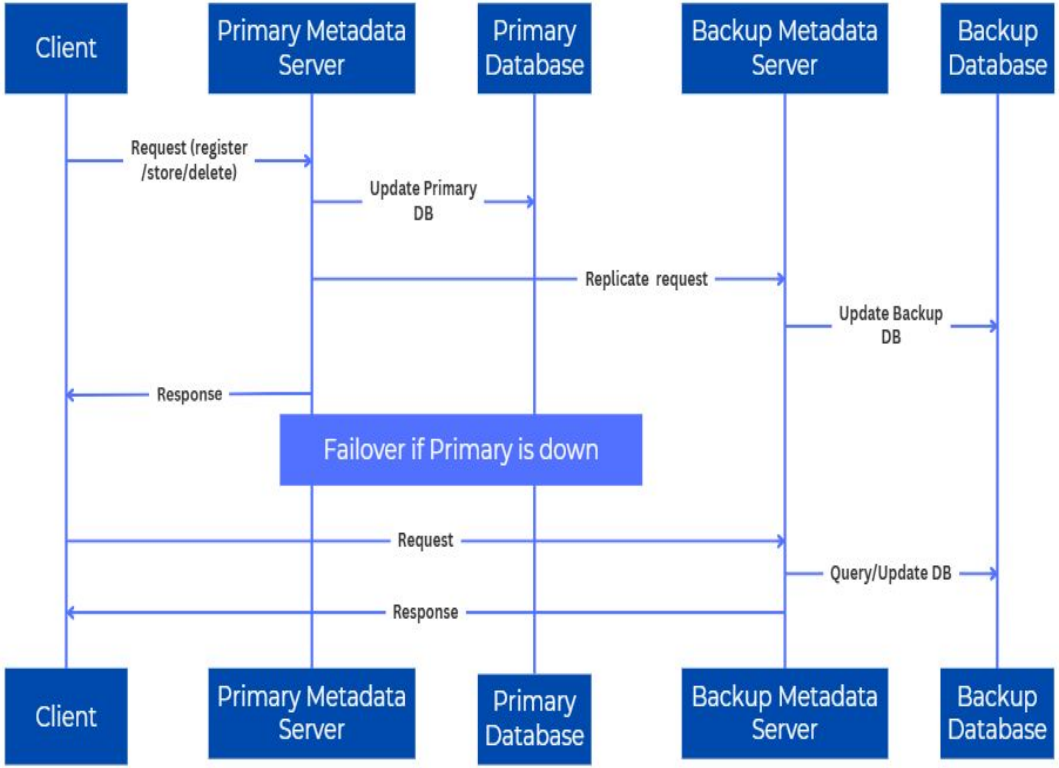
Download Files

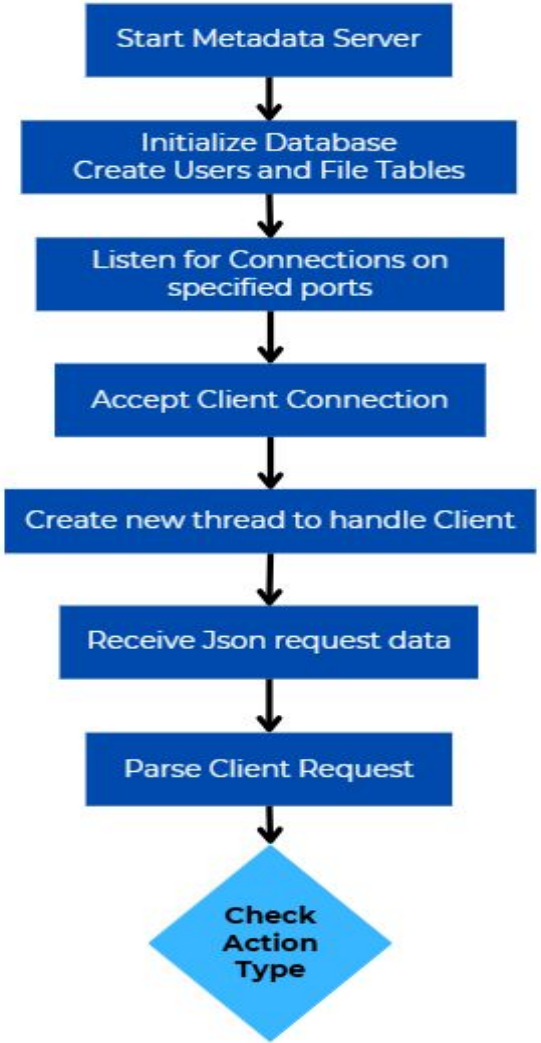
- Retrieves file metadata from metadata server using authenticated request
- Downloads and verifies chunk integrity via HMAC before processing
- Falls back to parity chunks if primary data chunks are unavailable
- Uses Reed-Solomon decoding to reconstruct missing chunks when needed, then decrypts all chunks to reassemble the file



Overview Of Metadata Server

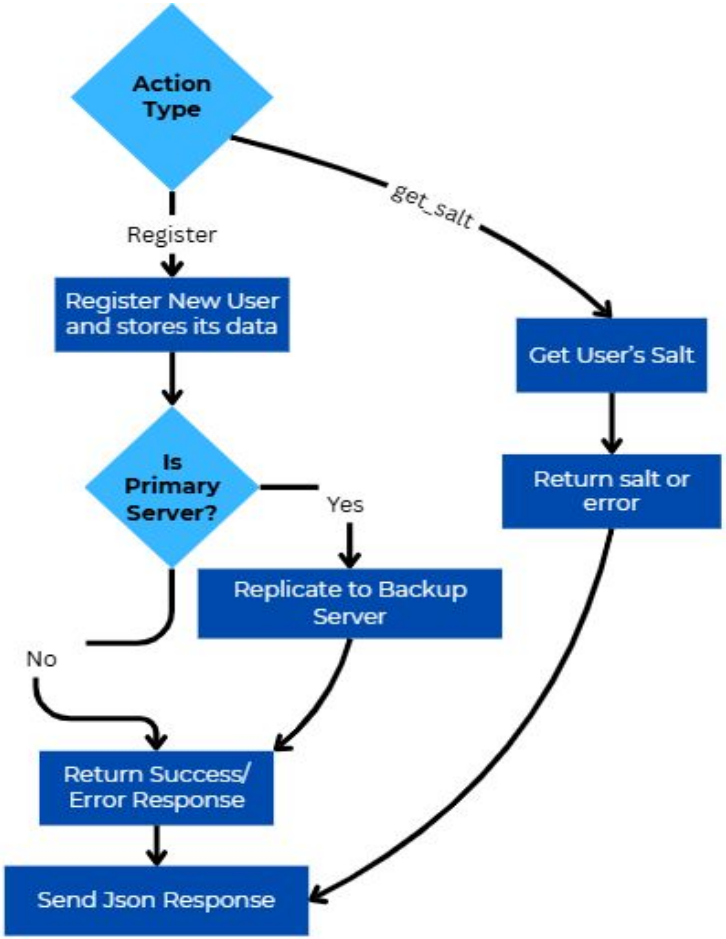
- Primary and Backup Metadata Servers.
- If Primary Metadata Server goes down, then backup will be used to fulfill request.





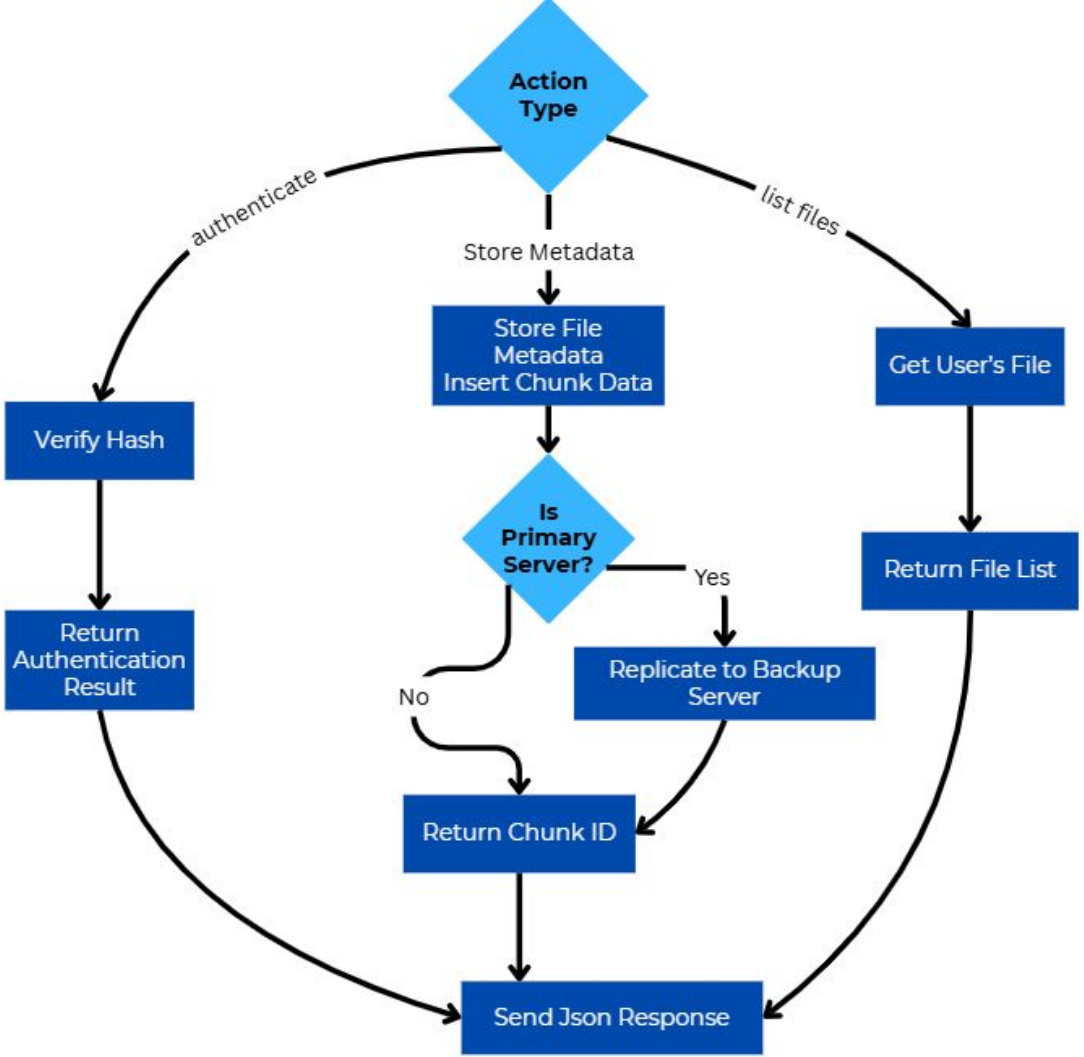
Progress of Request At Metadata Server

- After starting of Metadata Server, Database will be storing information.
- A thread will be created to handle Request of client.
- After this, parsing of Client's Request will be done.



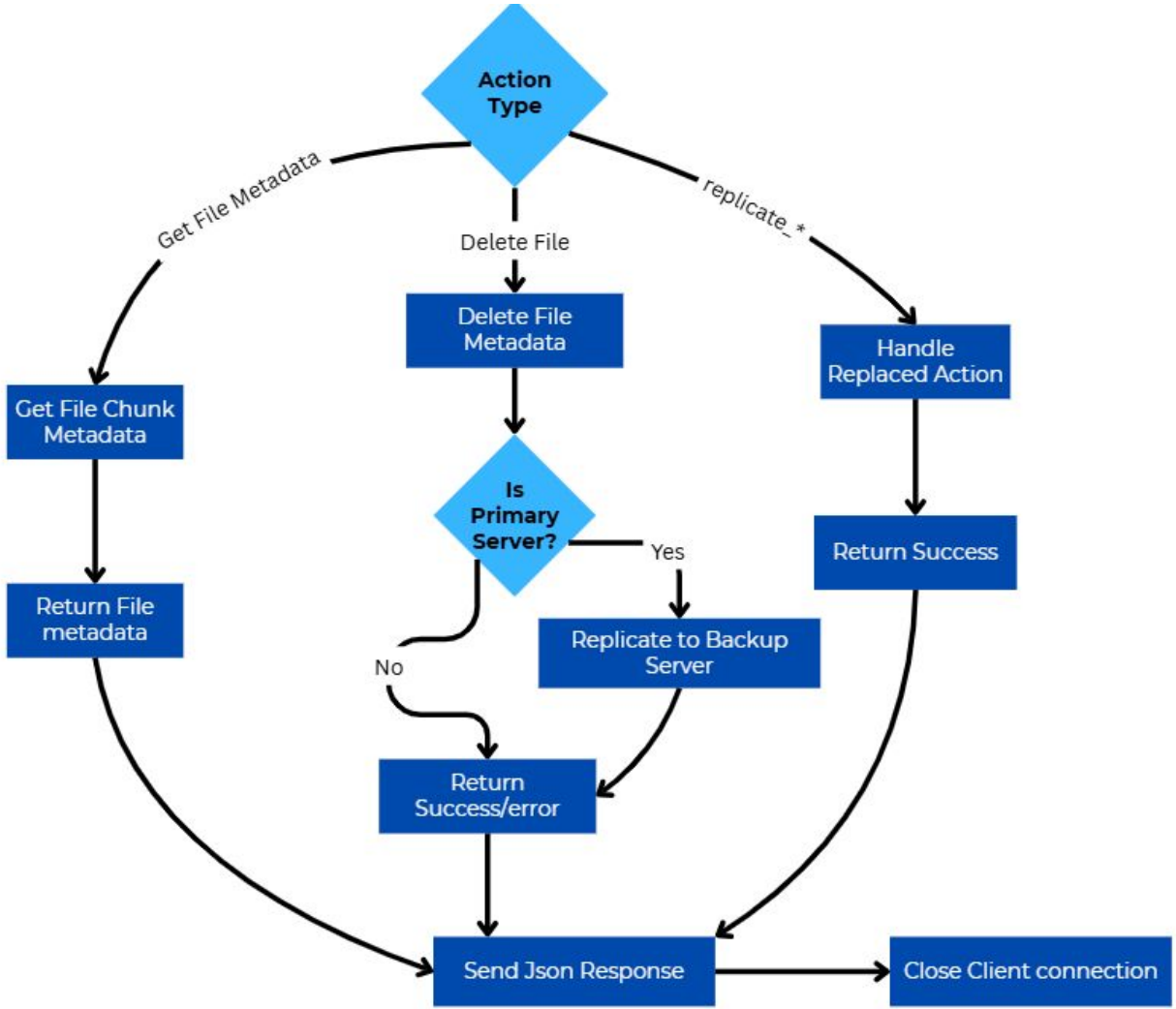
Registration and get_salt

- New users will be registered and their credentials will be saved on both Primary and Backup Servers.
- Salt generated would be checked and error will be returned in case of issue.



Authenticate, Metadata storage and file listing

- Verifies a user's password hash to authenticate them during login.
- Stores or replaces metadata about file chunks uploaded by the user.
- Retrieves a list of unique filenames associated with a user.



Get, delete and Replicate action

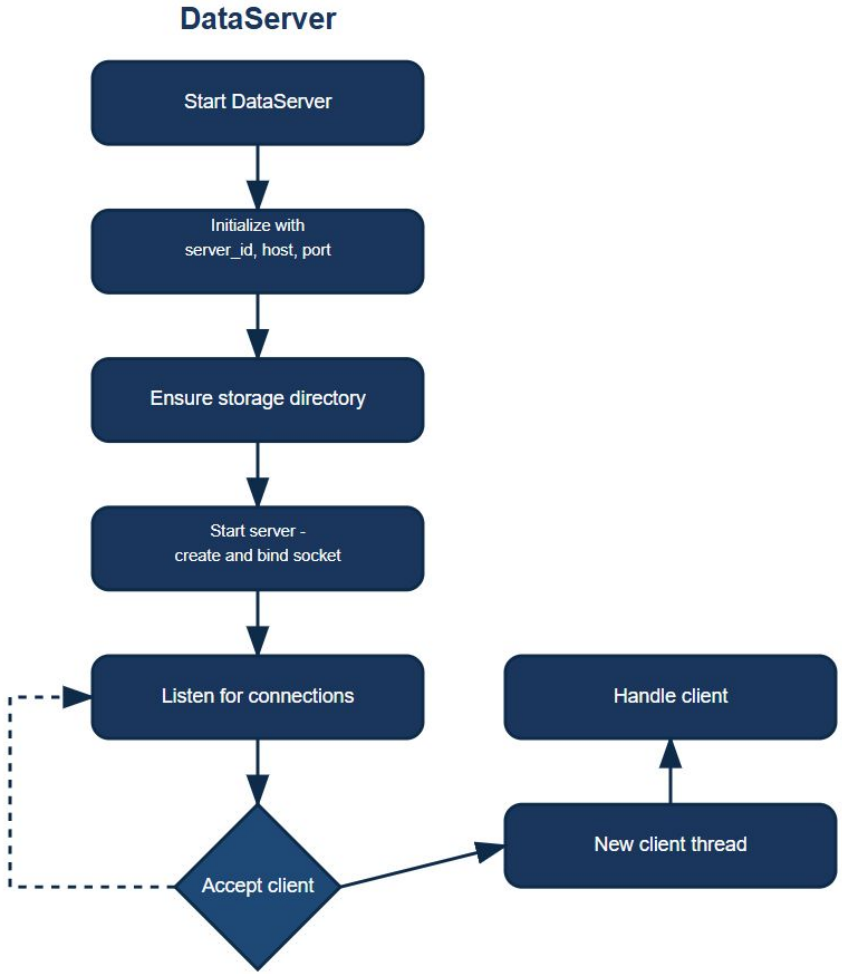
- Fetches detailed metadata (chunk ID, IV, HMAC) for a specific file.
- Deletes all metadata entries for a file belonging to a user.
- Processes different client requests like registration, authentication, metadata operations, and replication.
- Manages the lifecycle of a single client request: receive, process, respond.

Metadata Server: Network Architecture & Communication Flow

- **Socket Communication:** Uses TCP sockets for reliable client-metadata server communication.

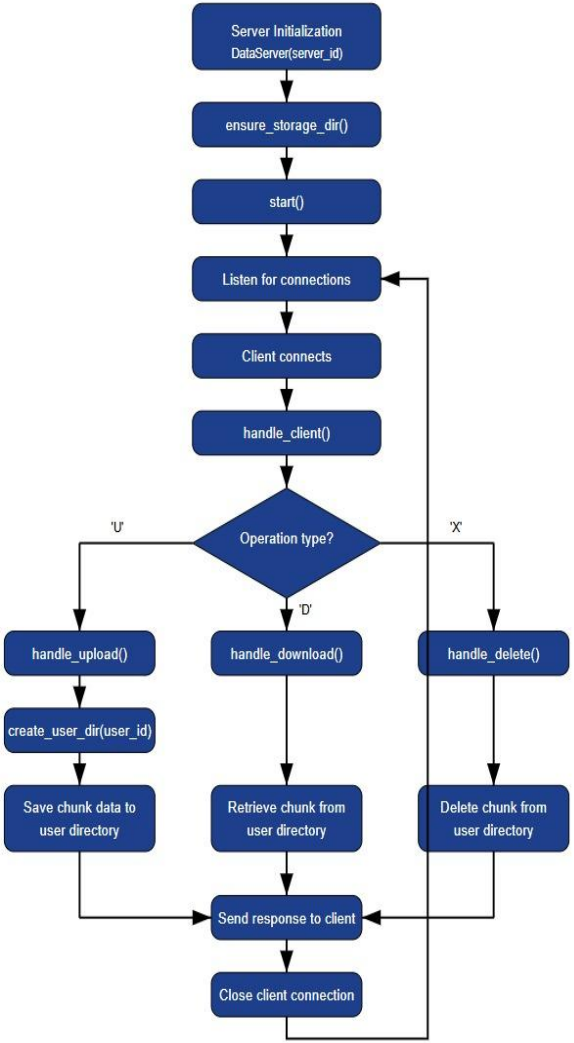
- **Client Handling:** Implements multi-threading for concurrent client connections. Each client connection runs in a separate thread.

- **Server Replication:** Primary-backup architecture for high availability. Replication between servers via local network connections.



- **Main Thread – `accept()`** – Waits for incoming client connection
- **New Thread per Client** – Created using **`threading.Thread()`**
- **Each Client = Independent Thread** – No blocking between clients

- **Parallelism Achieved** – Multiple operations at once



1. Upload Operation (`handle_upload`)

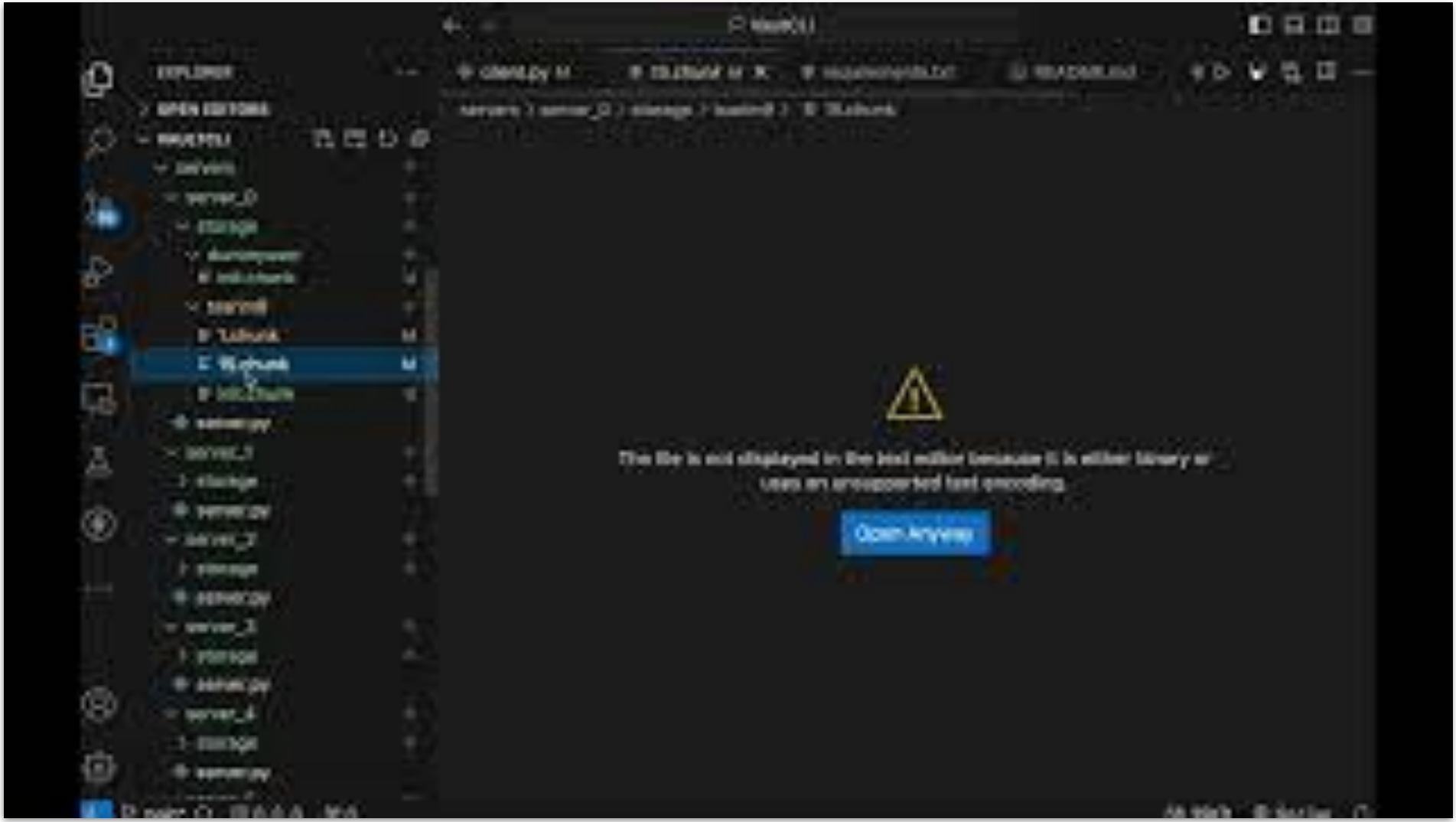
- Receives a 4-byte header size and the JSON header (`user_id, chunk_id`).
- Ensures the user directory exists.
- Receives a 4-byte chunk size and the actual chunk data.
- Saves the chunk as `storage/<user_id>/<chunk_id>.chunk.`

2. Download Operation (`handle_download`)

- Receives header with `user_id` and `chunk_id.`
- Locates the file `storage/<user_id>/<chunk_id>.chunk.`
- If found:
  - Sends 4 bytes for chunk size.
  - Sends the chunk data.
- If not found, sends an error message.

3. Delete Operation (`handle_delete`)

- Receives header with `user_id` and `chunk_id.`
- Locates the chunk file.
- If it exists, deletes it and responds with success.
- If not, sends an error message.


Open Anyway

THANK YOU