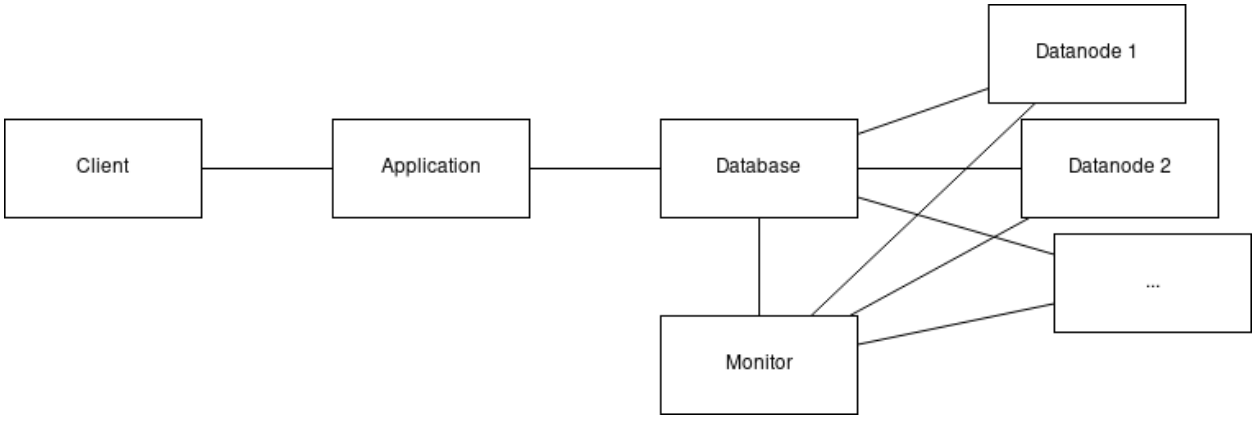
MyYoutube - Entrega 3

Gustavo Ribeiro Moura

The system architecture comprises five core components: Client, Application, Database, Data Nodes, and Monitor.
The Client represents the user's browser used to access the application.
The Application orchestrates all business logic and interfaces with the Database.
The Monitor automatically registers Data Nodes into its service and keeps track of which Data Nodes are alive and sending pings.
The Database abstracts data storage and interfaces with multiple Data Nodes, responsible for actual data storage. A dedicated Database service coordinates and manages access to these nodes, maintaining the distributed index. The Database service interfaces with the Monitor service to get the Data Nodes required to store or retrieve data.



Application

Endpoints:

1. **GET /video/{video_id}**
    - Retrieves a video file based on its `video_id` from the database.
    - Sends a request to the database to get the video file and streams it back to the client.
2. **GET /list**
    - Requests the list of available videos from the database.
    - Generates an HTML page displaying the list of videos and sends it back to the client.
3. **GET /watch/{video_id}**
    - Fetches metadata for a specific `video_id` from the database.
    - Generates an HTML page for watching the video player with its metadata and sends it to the client.
4. **GET /search?query={search_query}**
    - Initiates a search for videos based on the `search_query`.
    - Searches the database for matching videos, generates an HTML list of matched videos, and sends it back to the client.
5. **GET /delete/{video_id}**
    - Deletes the video associated with `video_id` from the database.
    - Initiates a request to delete the video and retrieves an updated list of videos, then generates an HTML list and sends it to the client.
6. **GET /favicon.ico**
    - Retrieves the favicon file.
7. **GET /**
    - Retrieves the index page.
    - Generates an HTML index page and sends it to the client.
8. **POST /upload**
    - Handles video uploads to the server.
    - Expects a `multipart/form-data` request containing the video file.
    - Sends the video file to the database for storage and generates an HTML page for the uploaded video.

Behavior:

- The server listens on a specified port (`PORT = 8080`) for incoming connections.
- Upon connection, a new thread is spawned to handle the HTTP request.
- The server uses a custom HTTP parser to interpret incoming requests and responses.
- Depending on the HTTP method (GET or POST) and the requested URL, different actions are taken:
    - GET requests:
        - Handle video retrieval, list retrieval, video watching, search, deletion, favicon retrieval, or index page retrieval.
    - POST request:
        - Handle video uploads by sending the uploaded video file to the database for storage.

Interaction with Database:

- Communicates with a database service running on rpyc for various operations like getting lists, metadata, file retrieval, deletion, and upload.

Response Handling:

- Responds with appropriate HTTP headers and content types based on the requested operation and data type.
- Generates HTML responses for various operations such as video list, video player, search results, or error (404) pages.

This HTTP API essentially allows clients to interact with the server for video-related operations like uploading, viewing, searching, and deleting videos, leveraging a custom server-to-database communication protocol over HTTP.

Database

- `file(id)`: Allows clients to download a file from the distributed database using the file's unique identifier.
- `list()`: Provides clients with a list of files present in the distributed database along with their metadata.
- `metadata(id)`: Retrieves metadata information for a specific file identified by its unique identifier.
- `delete(id)`: Enables clients to delete a file from the distributed database using the file's identifier.
- `upload(name, size, chunk_generator)`: Allows clients to upload a new file to the distributed database.

Datanode

- `file(id)`: Returns a file.
- `delete(id)`: Deletes a file.
- `upload(id, chunk_generator)`: Saves a file to local storage (currently unused).
- `getWriteFileProxy(id)`: Returns a file proxy for writing to local storage (useful for bulk storage on the client's end).

Monitor

- `register(clientServicePort)`: Registers Node for monitoring.
- `ping(clientServicePort)`: Alerts monitoring service that the Node is still alive
- `list()`: List of all Nodes currently alive.
- `isAlive(address)`: Checks if Node is alive.
- `aliveFromList(list)`: Receives a list of Nodes and returns its subset of all Nodes alive.