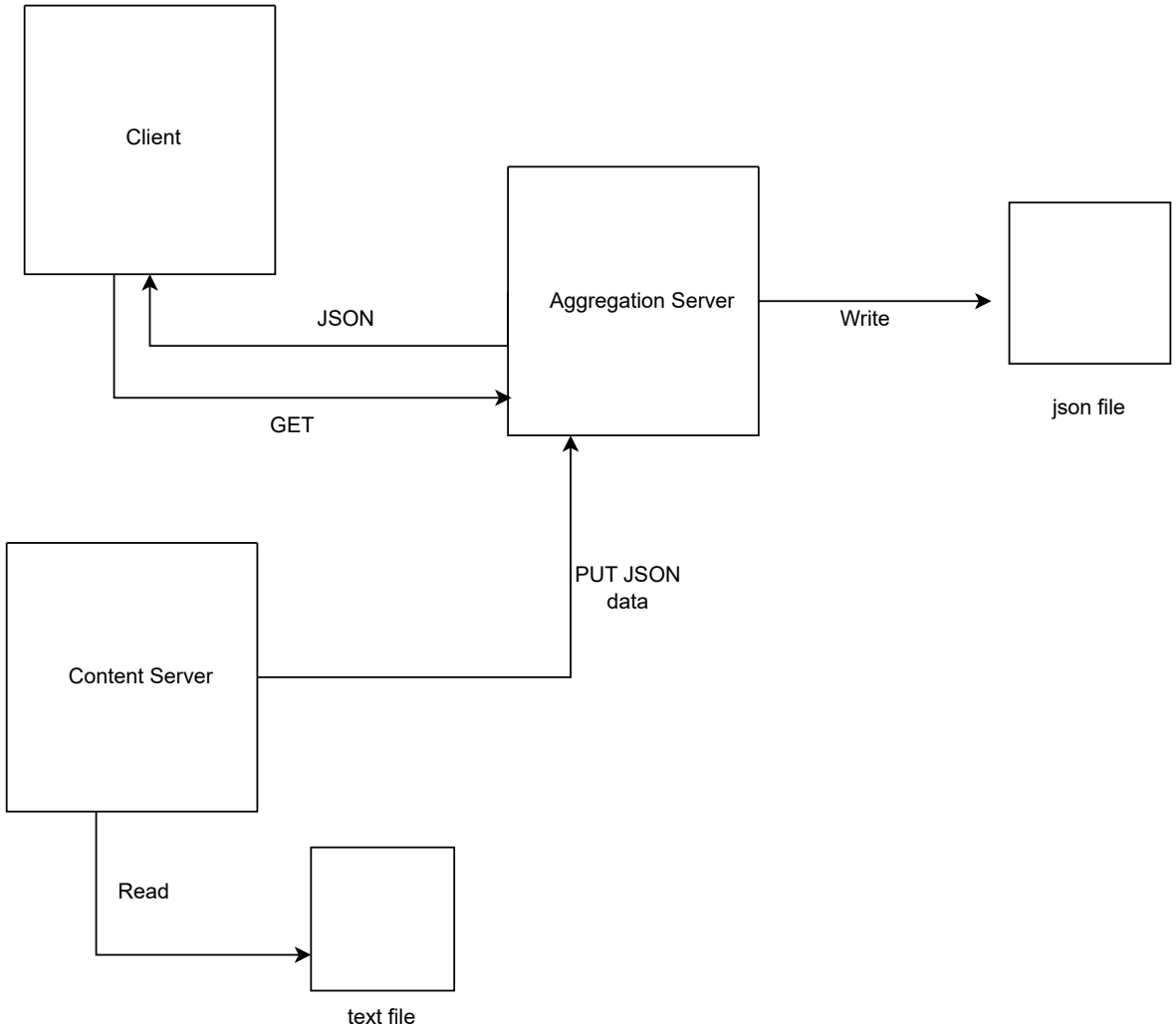
Assignment two Design Sketch



1. **Content Server**:
    - Reads weather data from a text file, formats it into a JSON structure, and sends it to the **Aggregation Server**.
2. **Aggregation Server**:
    - Receives the data via a PUT request, validates and stores it in a file (`weatherInfo.json`), and returns a response with an updated Lamport Clock.
    - When a client (such as the GET Client) requests the data, it responds with the stored weather information in JSON format and updates the Lamport Clock.
3. **GET Client**:
    - Sends a GET request to the **Aggregation Server** to retrieve the stored weather data.
    - The client prints the JSON response (weather data) and updates its internal Lamport Clock based on the server's response.
    -

- **Aggregation Server** serves as a central repository for storing and retrieving weather data.
- **Content Server** is responsible for sending weather data to the Aggregation Server.
- **GET Client** is responsible for retrieving weather data from the Aggregation Server.

Aggregation Server

The **Aggregation Server** acts as a central server responsible for receiving and aggregating weather-related data from clients. It handles incoming requests and stores or serves JSON data accordingly. Here's an overview of its functionality:

- **Port and Lamport Clock**: The server runs on a specified port (default 4567). It also maintains a **Lamport Clock**, which is used for logical time synchronization between the server and clients.
- **Handling Requests**: It supports both **PUT** and **GET** requests:
    - **PUT Requests**:
        - Clients send weather data (in JSON format) to the server using PUT requests.
        - The server validates the JSON data and updates its Lamport Clock based on the client's Lamport Clock (received in the headers).
        - If the JSON is valid, the server writes it to a file (`weatherInfo.json`), and responds with a **201 Created** status, along with the updated Lamport Clock.
        - If the JSON is invalid, the server responds with a **500 Internal Server Error**.
    - **GET Requests**:
        - Clients request the aggregated weather data using GET requests.
        - The server reads the `weatherInfo.json` file, sends its content in the response, and includes the current Lamport Clock in the headers.
- **Concurrency**: The server can handle multiple client requests concurrently by spawning a new thread for each connection.
- **File Operations**: The server reads and writes JSON data to the file system:
    - `WriteToFile()` is used to save incoming JSON data to `weatherInfo.json`.
    - `ReadFromFile()` reads the JSON data from the same file and serves it to clients upon request.
- **JSON Validation**: The server uses the Jackson library to validate whether the received body is valid JSON before writing it to the file.

Content Server

The **Content Server** is responsible for reading weather-related data from a text file (`weather_info.txt`), converting it into JSON format, and sending it to the **Aggregation Server**.

- **Reading Data**:
    - The server reads the `weather_info.txt` file line by line. Each line contains key-value pairs representing weather information.
    - If a line contains valid data, the server adds it as a key-value pair to a JSON object.
- **Building JSON**:
    - The server constructs a JSON object (`JsonObject`) where each key represents a weather parameter (e.g., temperature) and the value is the corresponding data (e.g., 25 degrees).
- **Sending Data**:
    - Once the JSON object is built, the server sends it to the **Aggregation Server** using a **PUT** request over HTTP.
    - The `SendData()` method handles the communication with the Aggregation Server. It sets the content type as JSON and sends the weather data via HTTP.
- **Response Handling**:
    - After sending the data, the server checks the response code from the Aggregation Server to determine whether the operation was successful (201) or failed (500 for invalid JSON).

The **Content Server** acts as a client, sending data to the Aggregation Server for storage and future retrieval.

GET Client

The **GET Client** is a simple client designed to retrieve data from the **Aggregation Server**. It performs **GET** requests to the server and processes the response.

- **Lamport Clock**:
    - Like the Aggregation Server, the GET Client maintains a **Lamport Clock**. The client sends its current Lamport Clock value in the request headers and updates it based on the response from the server.
- **GET Request Handling**:
    - The client sends a GET request to the **Aggregation Server** to fetch the stored weather data.
    - It reads the response, which contains the JSON data (weather information) as well as the server's current Lamport Clock.
    - The client updates its Lamport Clock based on the received value, ensuring that it keeps track of logical time synchronization with the server.
- **Response Processing**:
    - The client reads the JSON response from the server and prints it to the console.
    - If the GET request fails (e.g., if the server is unreachable or an error occurs), the client logs the failure.