Assignment-2-Distributed-Systems-a1911465

Building an aggregation server with consistency management and a RESTful API.

Aggregation Server Enhancements

Updated the Aggregation Server to handle GET and PUT requests more effectively, returning appropriate HTTP status codes. Implemented persistent storage for weather data that survives server crashes. The server can now clean up stale connections after 30 seconds of inactivity**.**

Lamport Clock Implementation

Implemented a Lamport clock in all components (Aggregation Server, Content Server, GET Client) to maintain synchronization across distributed components. The clock increments on each request to ensure logical consistency.

GET Client Enhancements

Enhanced the GET Client to send requests to the Aggregation Server and handle responses. It now displays weather data in a user-friendly format and includes error handling for invalid responses.

Testing Improvements

Implemented unit tests for key functionalities, including the process Weather Data method, and conducted tests for edge cases and integration scenarios. This ensures robustness and reliability of the system**.**

Running and Testing Instructions

1. ***Running the Aggregation Server***: Navigate to the project directory and run:

```bash

java Aggregation Server 4567

```

2. **_Running the Content Server_**: Run the Content Server with the server URL and feed file path:

```bash
java Content Server localhost:4567 /path/to/weather_data.txt
```

3. **_Running the GET Client_**: Use the following command to retrieve weather data:

```bash
java GET Client localhost:4567 [stationid]
```

4. **_Testing_**: Unit tests can be run using your IDE or build tool. Ensure all tests pass for validation.

Design Sketch

A design sketch illustrating the architecture of the distributed weather data system will be included in the final submission. This sketch outlines the interactions between the Aggregation Server, Content Server, and GET Client, highlighting the flow of data and the use of Lamport clocks.