**Design changes**

Initial Design

|  |  |  |
| --- | --- | --- |
| Module | Function | Communication |
| GETClient | Send GET request to aggregation server and print response | HTTP comms to aggregation server |
| Aggregation server | Store weather results for GETClient and accept PUT requests from content server | HTTP to content server and GETClient |
| Content server | Send PUT request to aggregation server | HTTP to aggregation server |

Obviously, this design is very simple and low effort. What I needed to consider was error cases and deeper communication between the modules. If we’re sending HTTP messages, we need an HTTP parser. And, as these messages have JSON content we also need a JSON parser. To read content server input and aggregation server file storage we need a file parser as well.

Further design

|  |  |  |
| --- | --- | --- |
| Module | Function | Communication |
| JSON parser | Parse JSON from string to object and object to string | Member variable of GET client and content server |
| HTTP parser | Parse HTTP from request to understandable format | Member variable of all main modules |
| File parser | Used with aggregation server to place data into file, read from file and evict 30 second + old results | Member variable of aggregation server |

Next, I needed to go a little deeper into the dataflow of each module. I’ve made lists for the data flow of each module below.

GETClient:

1. Start with connection port as argument
2. Connect socket to port
3. Send GET request
4. Read HTTP response with HTTP parser
5. Strip of JSON and print with JSON parser

Content server:

1. Start with connection port and input file as argument
2. Read input file to JSON format
3. Connect socket to port
4. Send PUT request with single JSON weather object

Aggregation server:

1. Start with port as argument
2. Start server socket on port
3. On connection of client, start a server thread
4. On PUT request to server thread:
   1. Read with HTTP parser
   2. Write to file storage with File parser
5. On GET request to server thread:
   1. Read with HTTP Parser
   2. Read from file and reply with File parser