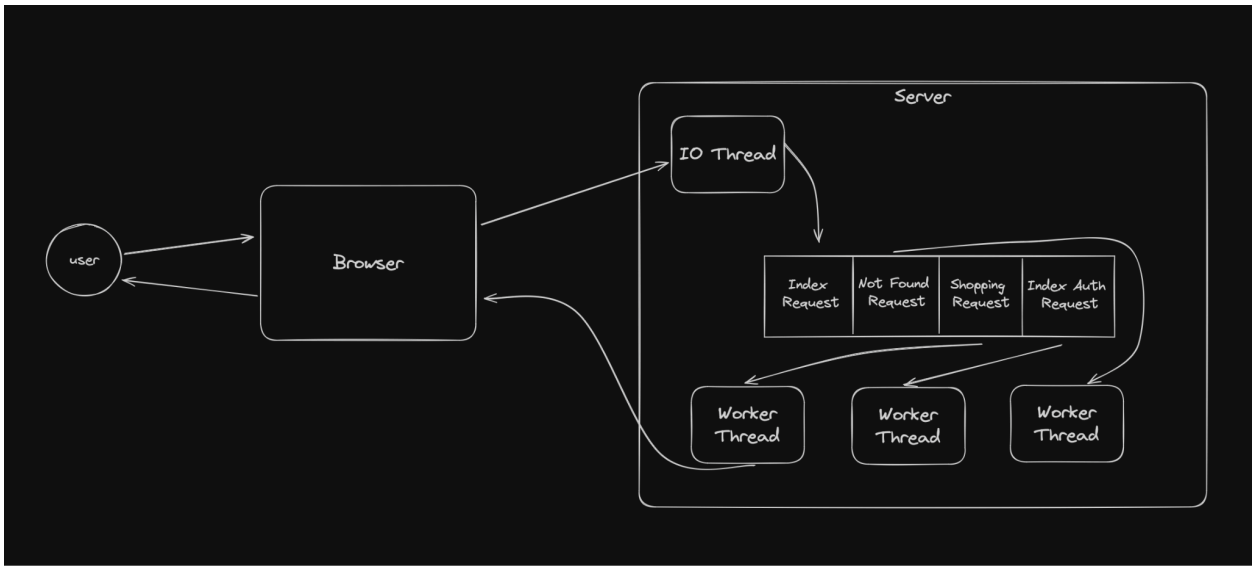
Scenario

You are a hoodie salesman working on his next big project: hoodie.com. For this project, your web server (written in the coolest most rad language of the time: Java), needs to be able to handle numerous requests asynchronously. So, it's your job to write the request handler for hoodie.com. It needs to be able to serve requests with the following ordering of preference: shopping-related requests, requests with authorization, and then based on request time. Also, write a method so that when the index is queried, it shows all pending requests in sorted form based on the ordering above.



Assignment

- Implement the following
    - The RequestQueue Class
        - `public void put()` →
            - Push a request to the request queue after parsing.
        - `public Request take()` →
            - Wait until a request is available from the queue if the queue is empty. Then, remove the request from the said queue and return it.
        - `public List<Request> getQueue()` →
            - Return a List of Request remaining objects of the request queue in the order of which they would be popped from the queue.
    - The Request Class
        - `public int compareTo(Request other)`
            - Implement a compareTo method to order, implement a comparison scheme between two requests based on the ordering mentioned above. Note that you can check a user is

authorized if they have the “Authorization” header in their request.

- Please ensure to include comments with your solution

Grading Scheme (out of 50)

- +1 for each compareTo test passed (5)
- +1 for each Heap Sort test passed (5)
- +1 for each Request Queue test passed (5)
- put() method (9)
    - +1 for usage of synchronized correctly
    - +2 to add to end of heap
    - +5 for correct addition implementation
    - +1 for use of notify correctly
- take() method (9)
    - +1 for usage of synchronized correctly
    - +1 for blocking using wait correctly
    - +7 for correct retrieval implementation (either recursive or non-recursive)
- getQueue() method (9)
    - +1 for usage of synchronized correctly
    - +2 for using a copy (not modifying original heap)
    - +6 for correct heap sort implementation
- compareTo() method (6)
    - +2 for implementing shopping priority correctly
    - +2 for implementing authorization priority correctly
    - +2 for implementing time priority correctly
- +2 points for commented code

Issues

- One of the main issues I had with this lab was debugging the parsing of HTTP requests. While the actual structure of HTTP requests is quite simple, the parsing of it in Java, can be prone to a lot of errors. Normally, I would use a debugger to find the issue at hand, but due to the nature of input, a debugger wouldn’t save anything from the client input stream. Furthermore, since requests are finished fast, simulating a full queue was quite challenging when testing the display of the index page.

Future Improvements

- A full authentication system
    - At the moment, this backend system estimates authorized request solely based on the existence of an authentication header on the request. This doesn’t implement the actual lookup and storage of the bearer token on the server side, nor actually store any sort of user data. To take this project to

the next logical step, implementing a user authentication system on both the front end and the backend would allow for developing more effective use into this website

- A proper frontend web app
    - Adding on to the above, serving a proper web application with user sign in options, actual shopping data and shopping options, alongside a more well designed website in general would allow for this app to actually be viable for selling hoodies.
- Heap Visualisation
    - Since the data structure of choice for this lab is to implement a heap manually, adding a visualiser into the website of the request heap would allow for a more visual form of learning, therefore not only improving the product with graphics on the website but also improving the learning experience of the lab alongside it.