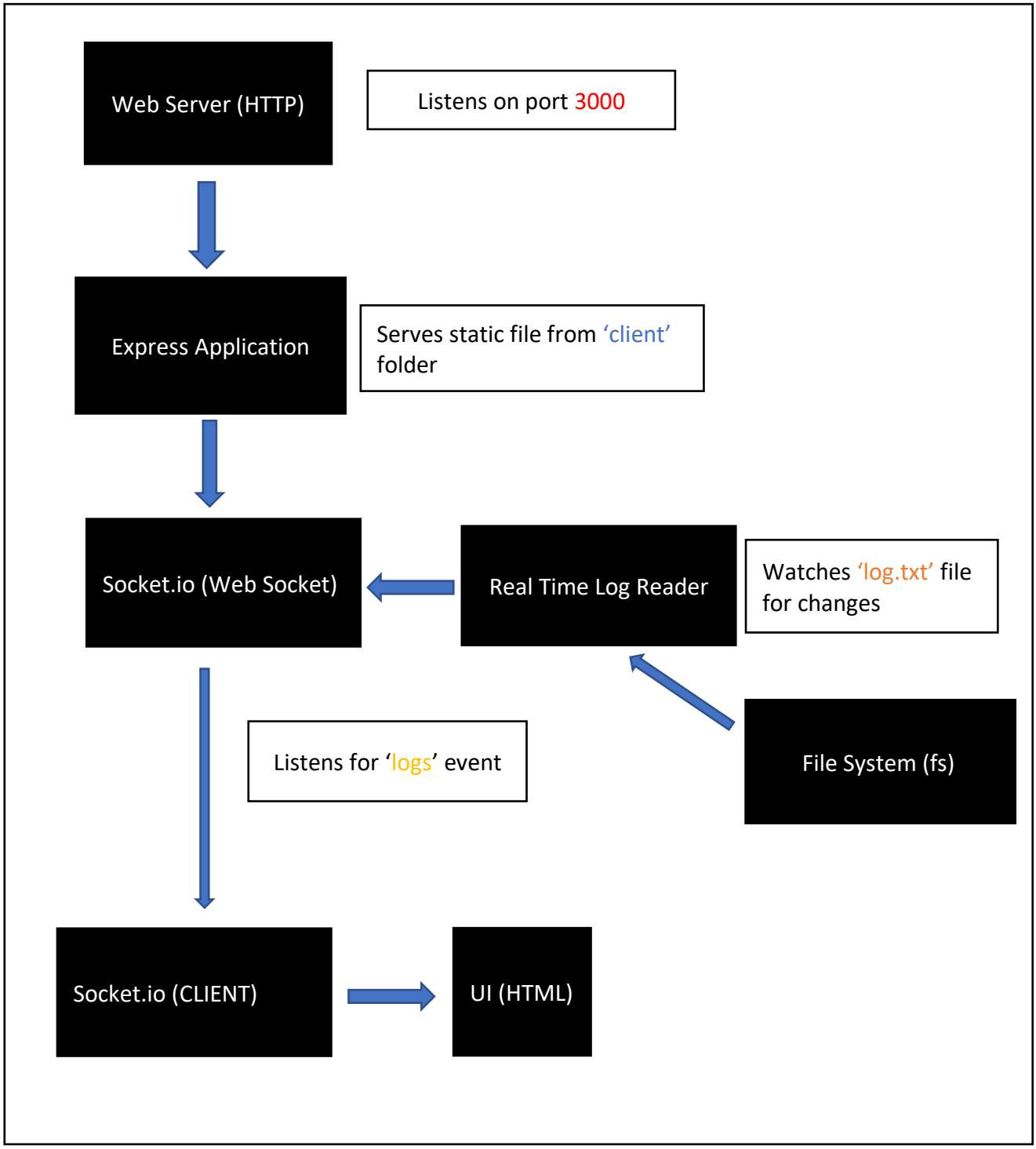
A basic flow diagram stating the architecture of the system :



The web server is an HTTP server created using the Node.js http module, with an Express application added to it. The Express application serves static files from the 'client' folder.

The Socket.io server is created using the Server class and is added to the HTTP server. It listens for incoming connections and broadcasts 'logs' events to all connected clients.

The Real Time Log Reader uses the fs module to watch the 'log.txt' file for changes. When a change is detected, it creates a read stream that starts from the previous end position and reads the file to the current end position. The data is then broadcast to all connected clients via the Socket.io server.

A write up on solving the 3 major technical issues :

There are three major technical challenges that have been addressed in this system:

1. *Real-time connection and broadcasting* - The first technical challenge was to establish a real-time connection between the server and the clients, and broadcast data to all connected clients. This was addressed by using the Socket.io library, which provides an easy-to-use API for real-time communication between the server and the clients. The Server class from the Socket.io library was used to create a **WebSocket server**, which allows for bidirectional communication between the server and the clients. The server listens for incoming connections and emits 'logs' events to all connected clients using the emit method.

2. *Watching a file for changes* - The second technical challenge was to watch the 'log.txt' file for changes and read new data when it is added to the file. This was addressed using the **fs.watchFile** method from the fs module. This method watches a file for changes and triggers a call-back function when the file is modified. The call-back function creates a read stream using the **fs.createReadStream** method and reads the new data from the file. The data is then emitted to all connected clients using the Socket.io server.

3. *Reading a file from the last read position* - The third technical challenge was to read the 'log.txt' file from the last read position. This was addressed by passing an options object to the **fs.createReadStream** method with a start property that specifies the byte position to start reading from. The start property is set to the previous size of the file (i.e., the position where the last read ended). The end property is set to the current size of the file (i.e., the end of the file). This ensures that only new data added to the file is read, and data that has already been read is not read again.

In summary, the technical challenges were solved by using the ***Socket.io library*** *for real-time communication*, **the** ***fs.watchFile method*** *to watch the file for changes*, and the ***fs.createReadStream method*** *to read the file from the last read position.*

Web Socket Server