For my cache implementation, I used a linked list using the LinkedList class. The head of the list denoted the oldest file and the tail of the list denoted the newest file. Hence, queuing and dequeuing from the list when evicting served as a LRU replacement policy. Additionally, I used a hashmap to map paths to a file wrapper class that I created. The file wrapper class contained a file, random access file, and other important meta data for each file. This served as a cache to obtain information about files in the cache. I also used hashmaps to create a file table that mapped file descriptors to file wrappers. This mapping indicated which files were open for each client.

In order to implement whole file caching with check on use and session semantics, I propagated all of my updates after closing. I handled server updates, cache eviction, and updated the version map in close. There was a lot of necessary communication between the server and proxy. I fetched file content when opening. I also had to create a serializable file wrappers to send metadata about files between the server and proxy.

A large part of the project required version control. Version control was important in order to implement concurrency. When I implement write(), I branch off and create a new version so that writers will not interfere with each other and so that readers can read as well.