In this project, we designed and implemented an application to index and search large documents. The implementation is done in two ways: 1) Java-Based Implementation, and 2) Hadoop-based implementation.

**Java-Based Implementation**

This is a server-client based implementation where we have a master server that handles all the connections and request and clients issue an index or a search request. The main components of this system are described below:

**Master Server**

The master server two concurrent threads: 1) MasterServerThread that waits for connection from either a client or a worker node, and 2)Main thread which waits for the arrival of a request in the WorkQueue (this queue holds incoming index/search request).

If the request is an index request, the master server checks if the document has been indexed before, then it skips the indexing, else it proceeds with the indexing request.

It then creates a Job Coordinator thread to verify if the request type of the received request is equal to the previous request. This is done so that no two same requests are handled at the same time (e.g, search request is not handled as the time as the index request).

**Job Coordinator**

The job coordinator creates jobs and sends them to the registered workers and mappers. This class defines number of mapper and number of reducer. These number depends on the size of the file.

The Job Coordinator class sends job to all the mapper/reducer and wait for an acknowledgement that they received the job. This acknowledgement is to avoid deadlock between two workers. The workers are shuffled so that all the works are not passed to the same worker.

**Client**

**Indexing:**

To perform the indexing operation, when we receive an indexing request, we divide the document and send to multiple mappers to do a wordcount on the file. The wordcount object contains the term and total count. The document is preprocessed. All the punctuations and stopwords are removed from the document before sending it off to the mappers. They go through the document and converts each word into a wordcount object. Once done, they save it to AFS space. Reducers dedicated to specific alphabets then go through the wordcount and do a merge of the terms (total count of each item). [talk about the inverted index data structure]

**Hadoob-based implementation**

For the hadoop-based implementation, we implemented an index mapper and index reducer and an query mapper and query indexer. We used StringTokenizer to token the input file and the reducer aggregates the result and creates the index. For querying, the query element is passed through argument. If a match is found the term with filename and its number of occurrence is printed out.