# Lucene Review

As Lucene is a 20 year old project, widely known and used. Having never personally used it, I don't think I can really offer anything new or insightful on the topic. I will, instead, simply focus on the features of Lucene I am likely to need to complete my project.

My project is essentially just a search engine over internet browsing history, and the specific things that I need from Lucene are:

- Process new pages/documents as they come in
- index them with their url as the key
- Store/load this index
- Query this index

In this document I'll go through the Lucene documentation [1], picking out things that are likely to be useful. This should all (hopefully) prove rather simple, considering Lucene *is* a search library.

Starting from the top, I need to be able to append documents to the index in real time (something I couldn't find any mechanism for within MeTA [2], which I initially thought to use, leading me to Lucene). As the user browses, I want to be able to index the contents of all pages they open. Ideally, I wouldn't need to save the pages to disk before indexing them.

The following snippet of code, from the Lucene documentation [3], shows how I might want to generate an index:

```
Analyzer analyzer = new StandardAnalyzer();

Path indexPath = Files.createTempDirectory("tempIndex");
Directory directory = FSDirectory.open(indexPath);
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter iwriter = new IndexWriter(directory, config);
Document doc = new Document();
String text = "This is the text to be indexed.";
doc.add(new Field("fieldname", text, TextField.TYPE_STORED));
iwriter.addDocument(doc);
iwriter.close();
```

Breaking this down:

- `Analyzer`?

  > An Analyzer builds TokenStreams, which analyze text. ***It thus represents a policy for extracting index terms from text.*** [4]

  `StandardAnalyzer` appears to be general purpose, but there are also one that specialize in, for example, Japanese text. For this project, `StandardAnalyzer` is likely to be sufficient, though I would like to see at some point if multiple analyzers could be combined or chained together.

- `IndexWriter`?

  > An IndexWriter creates and maintains an index. [5]

  The index is saved to disk at the provided directory. Based on the `IndexFiles` demo[6] having an `-update` flag, it seems that it should also be possible to make changes to an existing index, instead of regenerating the entire thing.

[1] https://lucene.apache.org/core/9_4_1/index.html
[2] https://meta-toolkit.org/
[3] https://javadoc.io/doc/org.apache.lucene/lucene-core/latest/index.html
[4] https://javadoc.io/static/org.apache.lucene/lucene-core/9.4.1/org/apache/lucene/analysis/Analyzer.html
[5] https://javadoc.io/static/org.apache.lucene/lucene-core/9.4.1/org/apache/lucene/index/IndexWriter.html
[6] https://lucene.apache.org/core/9_4_1/demo/src-html/org/apache/lucene/demo/IndexFiles.html

- `Document`

  > Documents are the unit of indexing and search. [7]

  Before So, as I build the documents myself, I don't need to read them off disk! Additionally, "fieldname" is set manually, and isn't some random/automatic id, so I can stick source URLs there.

- `TextField.TYPE_STORED`?

  > A field that is indexed and tokenized, without term vectors. For example this would be used on a 'body' field, that contains the bulk of a document's text. [8]

  It seems I may be able to index pages *without* storing the contents by using `TextField.TYPE_NOT_STORED`, though I'm not sure how that works. Look into this. I don't particularly need or want to store the pages themselves

## Store [9]

I obviously need the index to persist beyond a single browsing session. What index storage mechanisms are available in Lucene? What is the most efficient way to do this? This appears to happen "by default". See the `indexPath` and `directory` lines above. Though, I imagine there are some more things to look at (or there wouldn't be a storage package), I won't go into that right now.

## Querying

More from the example on the index[10]:

```
// Now search the index:
DirectoryReader ireader = DirectoryReader.open(directory);
IndexSearcher isearcher = new IndexSearcher(ireader);
// Parse a simple query that searches for "text":
QueryParser parser = new QueryParser("fieldname", analyzer);
Query query = parser.parse("text");
ScoreDoc[] hits = isearcher.search(query, 10).scoreDocs;
assertEquals(1, hits.length);
// Iterate through the results:
for (int i = 0; i < hits.length; i++) {
  Document hitDoc = isearcher.doc(hits[i].doc);
  assertEquals("This is the text to be indexed.", hitDoc.get("fieldname"));
}
ireader.close();
directory.close();
IOUtils.rm(indexPath);
```

### What is all that?

- What is `IndexSearcher`?

  > Implements search over a single IndexReader. Applications usually need only call the inherited search(Query,int) method. For performance reasons, if your index is unchanging, you should share a single IndexSearcher instance across multiple searches instead of creating a new one per-search. If your index has changed and you wish to see the changes reflected in searching, you should use DirectoryReader.openIfChanged(DirectoryReader) to obtain a new reader and

[7] https://javadoc.io/doc/org.apache.lucene/lucene-core/latest/org/apache/lucene/document/Document.html
[8] https://lucene.apache.org/core/9_4_1/core/org/apache/lucene/document/TextField.html
[9] https://lucene.apache.org/core/9_4_1/core/org/apache/lucene/store/package-summary.html
[10] https://javadoc.io/doc/org.apache.lucene/lucene-core/latest/index.html

then create a new IndexSearcher from that. Also, for low-latency turnaround it's best to use a near-real-time reader [11]

May require some work here, as my index will receive updates pretty regularly (every page load)

- What is `QueryParser`?

  QueryParser parses the user query string and constructs a Lucene Query object [...] The first parameter to the QueryParser constructor specifies the default search field, which is content field in this case. This default field is used if the query string does not specify the search field. The second parameter specifies the Analyzer to be used when the QueryParser parses the user query string. [12]

- What is `Query`? Representation of a user query.
- What is `ScoreDoc`? List of documents and their scores. It appears that you can choose the retrieval model used for scoring [13] Doesn't actually return the documents themselves, but contains the documents "number" in the index, score, etc The `10` it's taking is the number of results we want to get back.
- What is being returned in `hitDoc`? Gets the actual document referred to in ScoreDoc

## PyLucene

Having trouble setting up a java environment?

[11] https://javadoc.io/static/org.apache.lucene/lucene-core/9.4.1/org/apache/lucene/search/IndexSearcher.html
[12] http://web.cs.ucla.edu/classes/winter15/cs144/projects/lucene/index.html
[13] https://lucene.apache.org/core/9_4_1/core/org/apache/lucene/search/package-summary.html