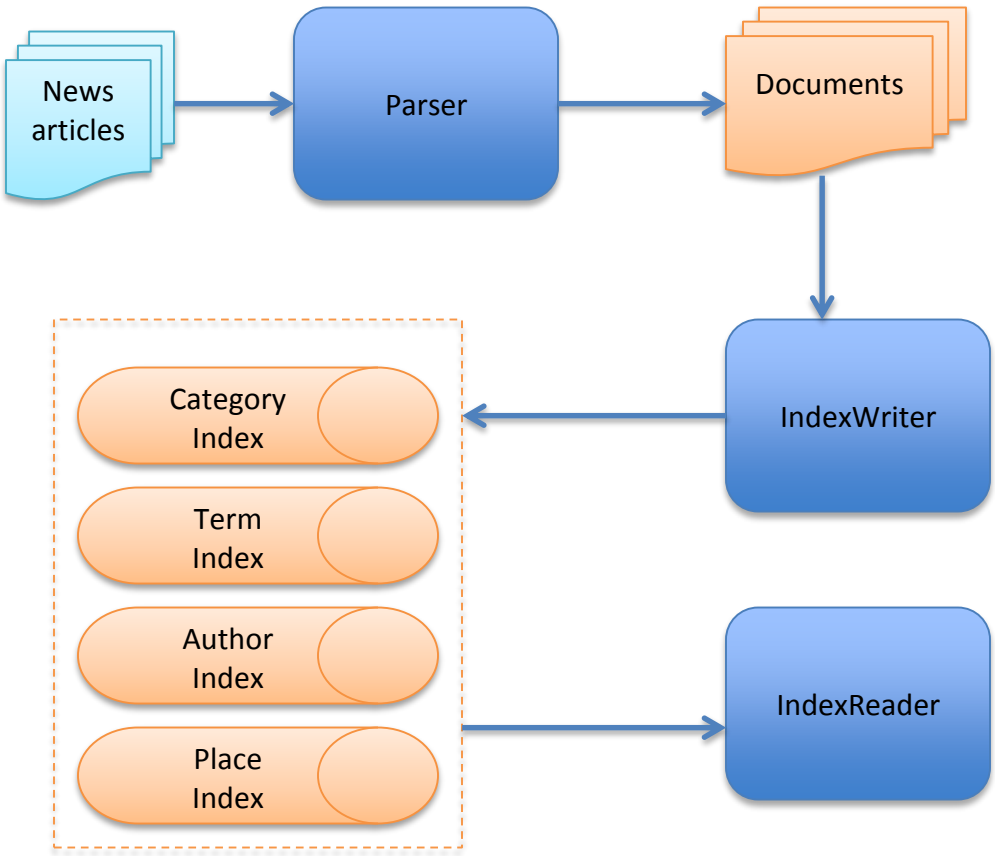
1. Project Description

This project aims to build a News indexer with the following goals:

- Parse simple news articles with minimal markup
- Index a decent sized subset of the given news corpus.
- Create multiple indexes on the news articles as well as metadata.
- Provide an index introspection mechanism that can later be built upon to support queries.

2. System components

The system consists of two main components: a parser and an indexer. An overall system diagram is shown below.



2.1. Parser

This component is responsible for converting a given text file into a Document representation. A Document is nothing but a collection of fields. Each field can have its own indexing strategy that would be applied by the IndexWriter.

2.2. IndexWriter

Once a given file has been converted into a Document, the IndexWriter is responsible for writing the fields to the corresponding indexes and dictionaries. Representative snapshots of the indexes and dictionaries follow.

- **Term index**: An index that maps different terms to documents. This is the standard index on which you would perform retrieval.
- **Author index**: An author to document index, stores the different documents written by a given author.
- **Category index**: A category to document index, stores the different documents classified by a given category.
- **Place index**: A place to document index, stores the different documents as referenced by a given place.

3. Parsing

This stage consists of mainly two classes:

3.1.1. Parser

This class is responsible for parsing the given file into a Document instance.

Methods to be implemented:

Document	parse	This is a static method that parses the given file into a fully populated Document instance. If any exception occurs, the method should throw a ParseException.
1.	String filename	This is the name of the news file to be parsed. It is a fully qualified filename

3.1.2. Document

This class is simply a container. The Parser must call the setField method with the correct values. The IndexWriter should call getField to read the values.

3.2. Tokenization

This stage is responsible for transforming the given Document class instances into Tokens (technically TokenStream instances but read on).

3.2.1. Token

This is the smallest logical element that would be indexed. At the very least, each Token would have some text associated with it. The termText field as a string and the termBuffer field as a character array represent this.

3.2.2. TokenStream

This is an iterable stream of Tokens.

3.2.3. Tokenizer

This class is responsible for converting strings into TokenStream objects. A Tokenizer is instantiated without any arguments or a given delimiter. The former merely implies space-delimited tokenization.

3.2.4. Analyzer

This is an interface that defines operations over TokenStream objects. Multiple Analyzer instances can be chained to allow successive operations on a given TokenStream. There are two levels of usage – either as a single Analyzer as a TokenFilter instance or as a chained Analyzer applicable for a given FieldName.

3.2.5. TokenFilter and TokenFilterType

A TokenFilter is a single Analyzer instance denoted by a given TokenFilterType.

3.3. Indexing

There are only two methods in this class that serve as entry points, addDocument that adds a document and close that indicates completion of all documents being added.

3.4. Index reader

This is the final component. All this class does is give utility methods to read an index (which means all including referenced dictionaries). The different methods are listed below:

- Constructor: It takes two arguments: the index directory and the field on which the index was constructed.
- getTotalKeyTerms and getTotalValueTerms: This is just the size of the underlying dictionaries, the former for the key field and latter is for the value field.
- getPostings: Method to retrieve the postings list for a given term. Apart from the corresponding reverse lookups (for both keys and values), the expected result is only a map with the value field as the key of the map and the number of occurrences as the value of the map.
- getTopK: This returns the key dictionary terms that have the k largest postings list. The return type is expected to be an ordered string in the descending order of result postings, i.e., largest in the first position, and so on.
- query: This emulates an evaluation of a multiterm Boolean AND query on the index. Implement this only for the bonus. The order of the entries in the map is again defined by the cumulative sum of the number of occurrences.