# System Design: (The Code-Explanation file has the complete explanation of the system)

Our system is designed to parse documents, build indexes, process queries, and evaluate system performance. Here's an overview of the key components and design choices:

## Text Parser and Indexer:

The TextParser\_Indexer class supports document parsing, tokenization, stemming, and index creation.

**Tokenization**:

Documents are tokenized into lowercase tokens, numeric tokens are removed, non-alphanumeric letters are separated, and stop words are eliminated using regular expressions and a stop word list.   
**Stemming**: We employ the Porter-Stemmer technique to convert tokens into their root forms.   
**Indexing**: We create forward and inverted indexes to efficiently store and retrieve information about terms and documents.

## Query Processing:

The Query\_Processing class processes user queries, computes TF-IDF weights, compares queries to documents, and assesses system performance.

**TF-IDF Weighting**: We assign TF-IDF weights to terms in documents and queries to indicate their importance.

**Cosine Similarity:** We utilize cosine similarity to compare queries and documents based on their TF-IDF vectors.

**Relevancy Evaluation:** We assess system performance using precision and recall metrics based on relevance judgments.

# Data Structures/Classes:

We utilize classes like Tokenizer, WordDictionary, FileDictionary, and Indexer to organize and manage data efficiently.

Dictionaries are used to map words and documents to unique IDs, while indexes store word-document relationships and frequencies.

We utilize dictionaries for tf-idf-weights storing and then we use pandas dataframe for further cosine similarity computation.

# Term Weighting and Normalization:

**Term Frequency (TF):** We determine the frequency of each term in a document. TF indicates how frequently a term appears in a document.

**Inverse Document Frequency (IDF):** We use IDF to estimate the rarity of a term across all documents in the collection. IDF penalizes common phrases while emphasizing unusual ones.

**TF-IDF Weighting:** We mix TF and IDF to calculate the TF-IDF weight for each phrase. TF-IDF gives larger weights to terms that occur frequently in a document but are uncommon in the collection, suggesting their relevance.

**Euclidean Normalization:** It is computed for each document and query using the tf-idf weights.

# System Performance Comparison:

We evaluate system performance based on precision and recall for different query settings:

## Main Query

### **(Title):**

Precision and recall are calculated when considering only the main query (title) for each topic.

Topic: 352\_title

Precision: 0.2777777777777778

Recall: 1.0

Topic: 353\_title

Precision: 0.9

Recall: 0.8571428571428571

Topic: 354\_title

Precision: 0.75

Recall: 0.8571428571428571

Topic: 359\_title

Precision: 0.07692307692307693

Recall: 1.0

### **Description + Title:**

Precision and recall are computed when considering the description along with the main query (description + title).

Topic: 352\_description\_title

Precision: 0.2222222222222222

Recall: 1.0

Topic: 353\_description\_title

Precision: 0.7142857142857143

Recall: 0.8823529411764706

Topic: 354\_description\_title

Precision: 0.34615384615384615

Recall: 0.9

Topic: 359\_description\_title

Precision: 0.07692307692307693

Recall: 1.0

### **Narrative + Title:**

Precision and recall are evaluated when considering the narrative along with the main query (narrative + title).

Topic: 352\_narrative\_title

Precision: 0.08333333333333333

Recall: 1.0

Topic: 353\_narrative\_title

Precision: 0.4782608695652174

Recall: 1.0

Topic: 354\_narrative\_title

Precision: 0.34615384615384615

Recall: 0.9

Topic: 359\_narrative\_title

Precision: 0.07692307692307693

Recall: 1.0

# Conclusion:

The program effectively handles document parsing, indexing, query processing, and performance evaluation. By implementing TF-IDF weighting and cosine similarity, I provide robust retrieval capabilities. Also, after comparing the performance, all cases have high performance when only the title is used compared to other combinations.