**INDEXER (Assignment-01)**



Session: 2021 – 2025

**Submitted by:**

Muhammad Yaqoob        2021-CS-118

**Supervised by:**

Dr Syed Khaldoon Khurshid

Department of Computer Science

**University of Engineering and Technology UET Lahore**

Table of Contents

[1. Introduction 4](#_Toc182463345)

[2. Objective 4](#_Toc182463346)

[3. Document Collection 4](#_Toc182463347)

[4. Indexing Approaches 4](#_Toc182463348)

[5. Steps to Create Index 5](#_Toc182463349)

[5.1. Meta Data Extraction 5](#_Toc182463350)

[5.1.1. Workflow 5](#_Toc182463351)

[5.1.2. Code 5](#_Toc182463352)

[5.2. Case Folding 5](#_Toc182463353)

[5.2.1. Workflow 6](#_Toc182463354)

[5.2.2. Code 6](#_Toc182463355)

[5.3. Tokenization 6](#_Toc182463356)

[5.3.1. Workflow 6](#_Toc182463357)

[5.3.2. Code 6](#_Toc182463358)

[5.4. Remove Stopwords 6](#_Toc182463359)

[5.4.1. Workflow 6](#_Toc182463360)

[5.4.2. Code 6](#_Toc182463361)

[5.5. Creating Index 6](#_Toc182463362)

[5.5.1. Workflow 7](#_Toc182463363)

[5.5.2. Code 7](#_Toc182463364)

[6. Word Based Search 8](#_Toc182463365)

[6.1. Workflow 8](#_Toc182463366)

[6.2. Code 8](#_Toc182463367)

[6.3. Screenshot 8](#_Toc182463368)

[7. Title Based Search 8](#_Toc182463369)

[7.1. Workflow 9](#_Toc182463370)

[7.2. Code 9](#_Toc182463371)

[7.3. Screenshot 9](#_Toc182463372)

[8. Data Flow Diagram 9](#_Toc182463373)

[9. Conclusion 10](#_Toc182463374)

Table Of Figures

[Figure 1: Metadata Extraction 5](#_Toc182463375)

[Figure 2: Case Folding 6](#_Toc182463376)

[Figure 3: Remove Stopwords 6](#_Toc182463377)

[Figure 4: Creating Index 7](#_Toc182463378)

[Figure 5: Document IDs and Titles alongside Indexing 7](#_Toc182463379)

[Figure 6: Word Based Search workflow 8](#_Toc182463380)

[Figure 7: Word Based Search execution 8](#_Toc182463381)

[Figure 8: Title Based Search workflow 9](#_Toc182463382)

[Figure 9: Title Based Search execution 9](#_Toc182463383)

[Figure 10: Dataflow Diagram 10](#_Toc182463384)

# Introduction

Information retrieval is a critical component of modern search systems, enabling users to quickly locate relevant content from vast datasets. Whether in search engines, document management systems, or data analysis tools, the ability to index and query data plays a pivotal role. This assignment focuses on implementing an Index, a data structure commonly used in information retrieval to map terms to the documents in which they appear, along with the frequency of their occurrence.

# Objective

There were two objectives for this assignment:

* **Word-based Searches:** To implement a search engine capable of finding all occurrences of a given word across multiple documents and returning the relevant document titles along with their term frequencies.
* **Title-based Searches:** To enable the search engine to locate a specific document based on its title from a given directory and retrieve its content efficiently.

# Document Collection

Names of the documents collected are written below. Each document has content.

* Health and Fitness.txt
* Nature and Wildlife.txt
* Space Exploration.txt
* Technology and Programming.txt
* Travel and Adventure.txt

# Indexing Approaches

There were two possible approaches for building the indexer:

* Forward Index
* Inverted Index

|  |  |  |
| --- | --- | --- |
| Aspect | Forward Index | Inverted Index |
| Definition | Maps a document ID to its contents, listing all the terms present in the document. | Maps terms to the list of document IDs where they occur, along with additional details (e.g., frequency). |
| Structure | Document ID → [Terms in the document] | Term → [Document IDs, frequencies, positions, etc.] |
| Storage | Requires less storage space | Consumes more storage |
| Time | O(n) | O (1) |

I have chosen the inverted index approach for this assignment due to its:

* Fast search operations suitable for word-based and title-based queries.
* Scalability for handling large document collections.
* Ability to efficiently support the assignment's objectives, despite slightly higher insertion and storage costs compared to the forward index.

# Steps to Create Index

* Metadata Extraction
* Case Folding
* Tokenization
* Remove Stopwords
* Creating Index

## Meta Data Extraction

Extract important information such as document titles and unique identifiers (IDs). This ensures each document is uniquely identifiable in the indexing process.

### Workflow

Text documents (individual or from a directory) are provided as input for indexing.

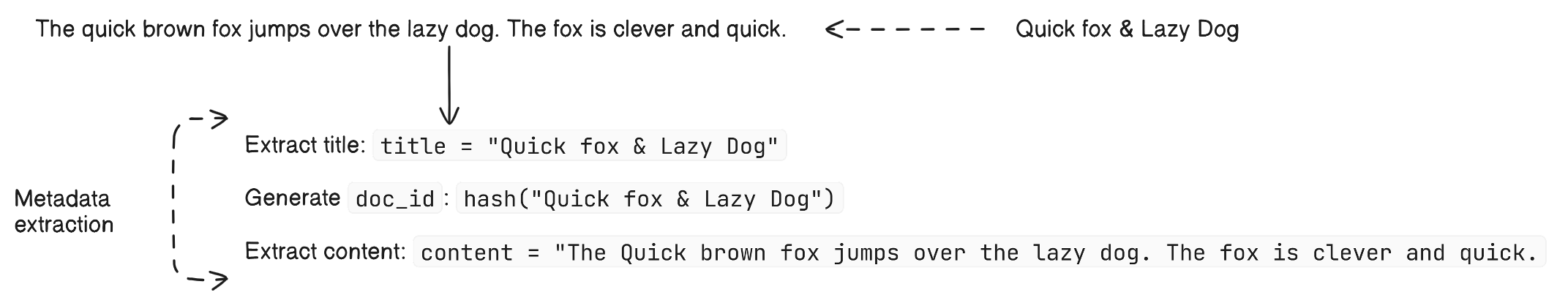


Figure 1: Metadata Extraction

### Code

Code Reference:

add\_single\_document(file\_path) processes a single document.

add\_documents\_from\_directory(directory\_path) processes multiple documents.

|  |
| --- |
| def add\_single\_document(self, file\_path):      """Add a single document from a file."""      with open(file\_path, 'r', encoding='utf-8') as file:          title = os.path.splitext(os.path.basename(file\_path))[0]  # Extract title from file name          content = file.read()          doc\_id = hash(file\_path)  # Use a hash of the file path as a unique ID          # Indexer          self.create\_indexer(doc\_id, title, content)  def add\_documents\_from\_directory(self, directory\_path):      """Adds all text files from a directory to the index."""      for file\_name in os.listdir(directory\_path):          if file\_name.endswith(".txt"):              file\_path = os.path.join(directory\_path, file\_name)              self.add\_single\_document(file\_path) |

## Case Folding

Convert all text to lowercase to ensure uniformity during processing. For example, "Fox" and "fox" will be treated as the same term.

### Workflow

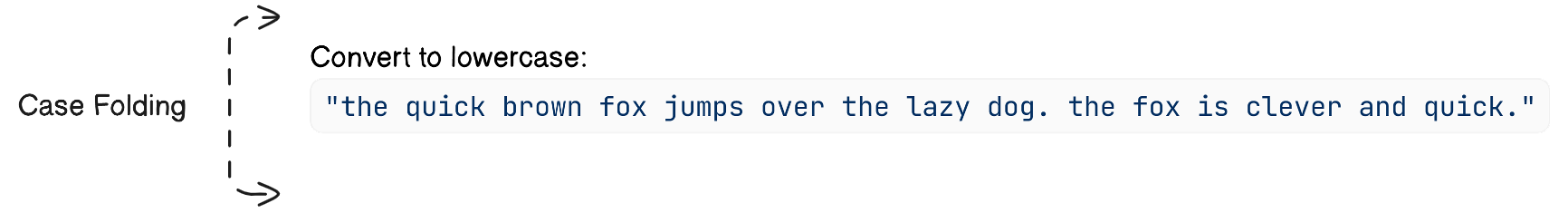


Figure 2: Case Folding

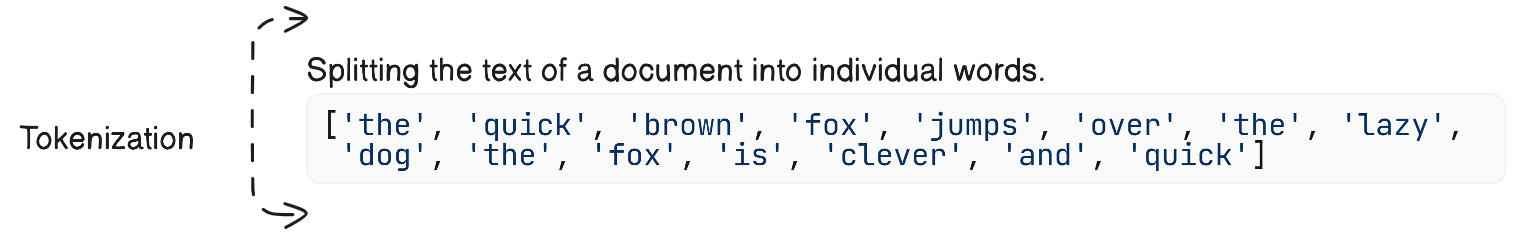
### Code

|  |
| --- |
| def \_case\_folding(self, text):      """Convert text to lowercase."""      return text.lower() |

## Tokenization

Split the document text into individual words or tokens. For instance, the sentence "The fox jumped over the dog" becomes tokens: ["the", "fox", "jumped", "over", "the", "dog"].

### Workflow



### Code

|  |
| --- |
| def \_tokenization(self, text):          """Split text into words."""          return re.findall(r'\b\w+\b', text) |

## Remove Stopwords

Eliminate common and less meaningful words like "the," "is," and "and" to reduce index size and improve efficiency. This step retains only the significant terms.

### Workflow

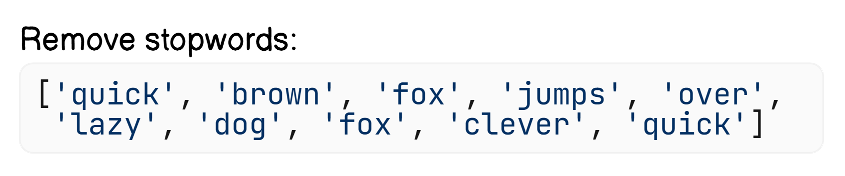


Figure 3: Remove Stopwords

### Code

|  |
| --- |
| def \_remove\_stopwords(self, words):       """Remove stopwords from a list of words."""       return [word for word in words if word not in self.stopwords] |

## Creating Index

Map each token to the documents where it appears, along with its frequency and other details.

### Workflow

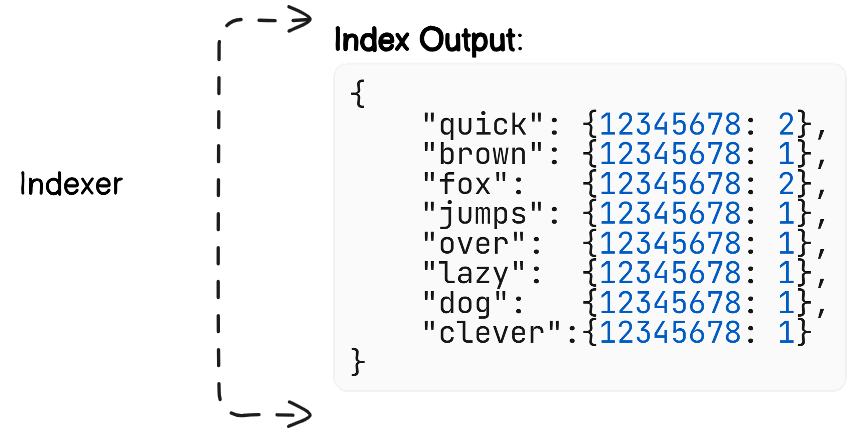


Figure 4: Creating Index

It also keeps track of documents and document titles with their Ids.

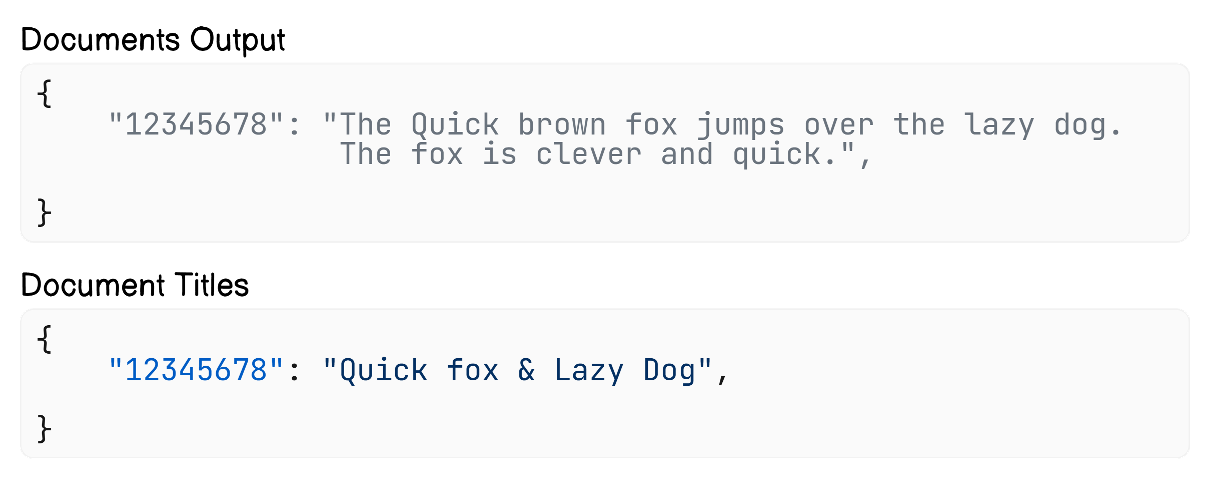


Figure 5: Document IDs and Titles alongside Indexing

### Code

|  |
| --- |
| def create\_indexer(self, doc\_id, title, text):      """Add or update a document in the index."""      # CASE FOLDING AND TOKENIZATION      tokens = self.\_tokenize(text)      if doc\_id in self.documents:          self.remove\_document(doc\_id)      self.documents[doc\_id] = text      self.document\_titles[doc\_id] = title      # CREATING INDEXER      for token in tokens:          if token not in self.index:              self.index[token] = {}          if doc\_id not in self.index[token]:              self.index[token][doc\_id] = 0          self.index[token][doc\_id] += 1 |

# Word Based Search

The word-based search functionality allows users to retrieve a list of documents containing a specific word. This is accomplished by querying the index, which maps terms to the documents in which they appear.

## Workflow

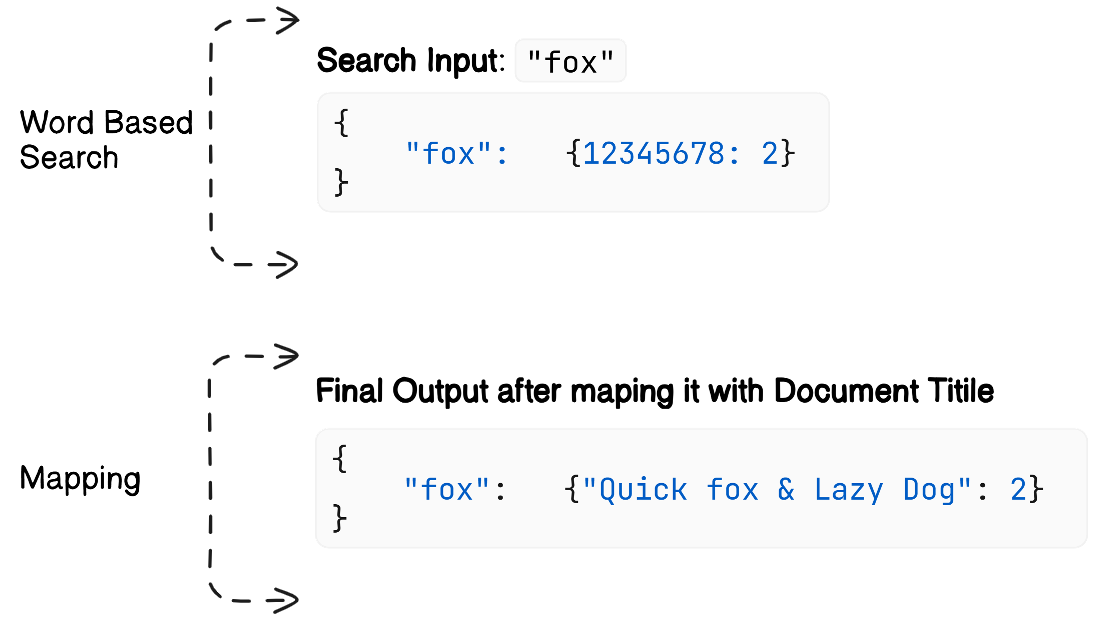


Figure 6: Word Based Search workflow

## Code

|  |
| --- |
| def search(self, word):      """Search for a word in the index."""      word = word.lower()      if word in self.index:          results = self.index[word]          return {self.document\_titles[doc\_id]: count for doc\_id, count in results.items()}      return {} |

## Screenshot

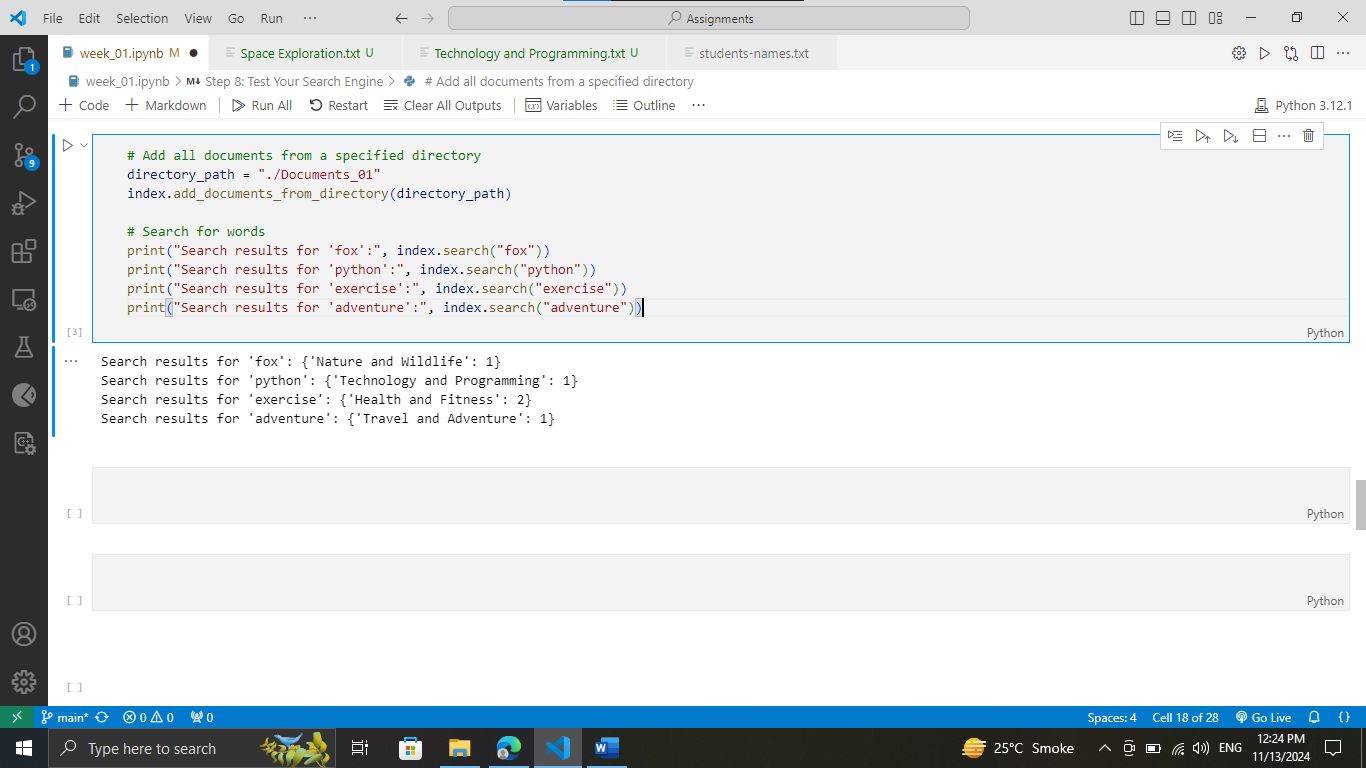


Figure 7: Word Based Search execution

# Title Based Search

The title-based search allows users to retrieve the full content of a document by specifying its title. This is useful when the user knows the exact document they are looking for.

## Workflow

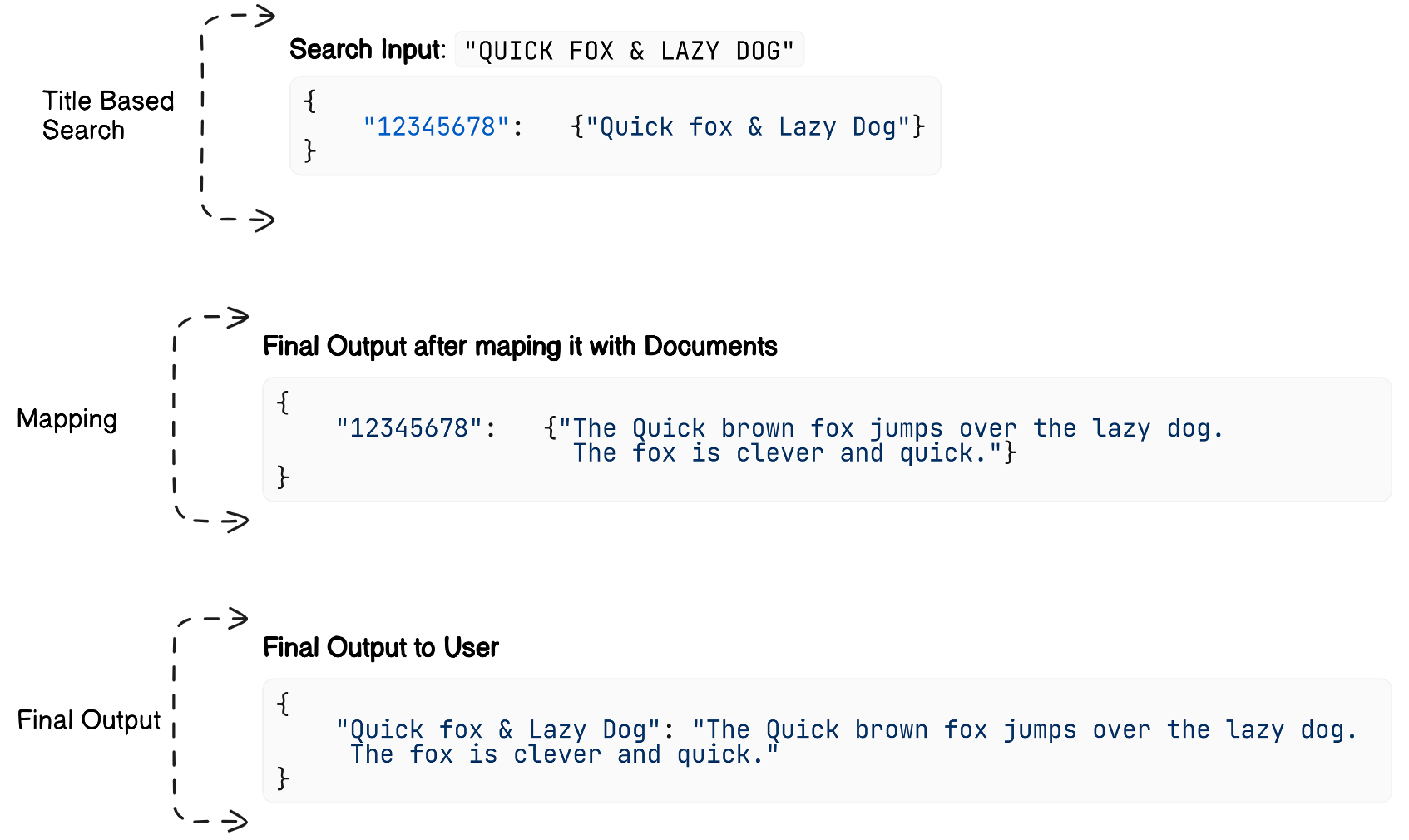


Figure 8: Title Based Search workflow

## Code

|  |
| --- |
| def search\_document\_by\_title(self, title):      """Search and return the content of a document by its title."""      for doc\_id, doc\_title in self.document\_titles.items():          if doc\_title.lower() == title.lower():              return self.documents[doc\_id]      return f"Document with title '{title}' not found." |

## Screenshot

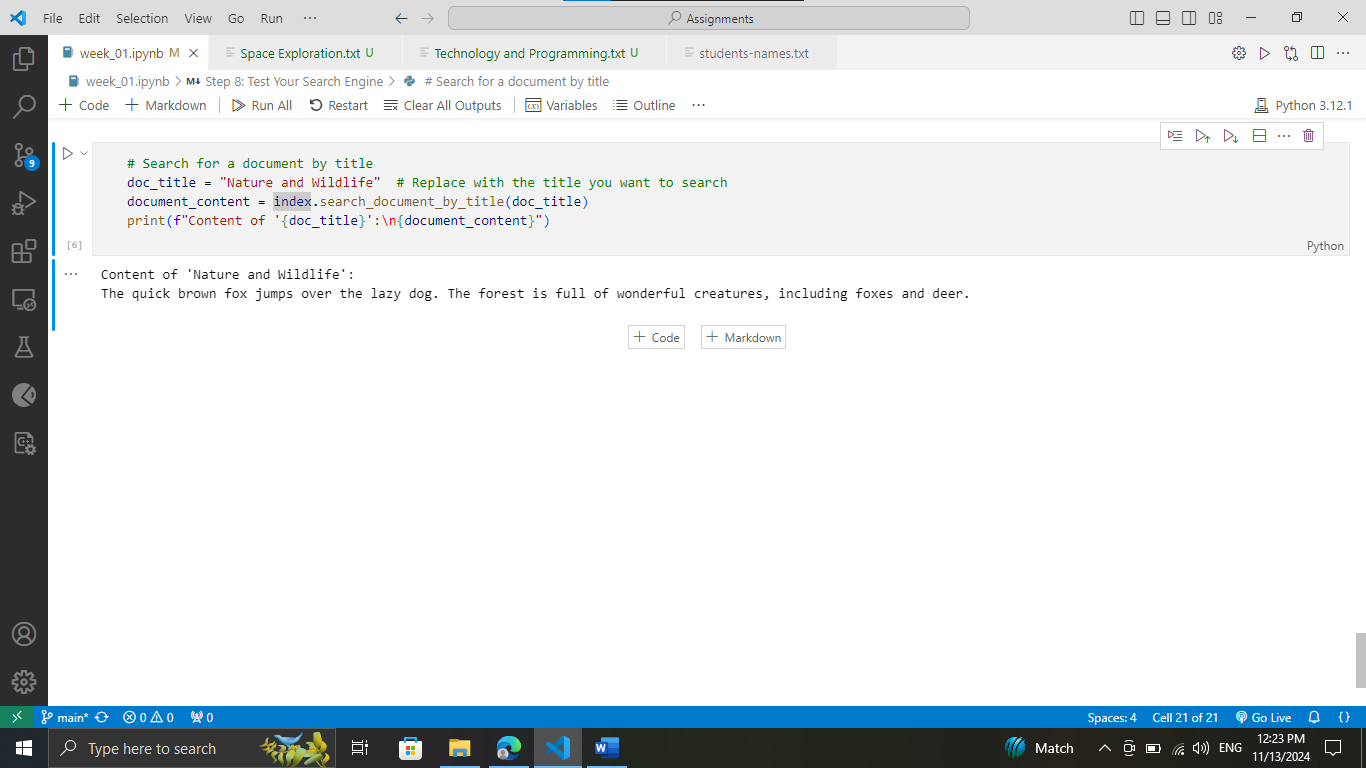


Figure 9: Title Based Search execution

# Data Flow Diagram

The Data Flow Diagram (DFD) provides a high-level representation of the data movement within the system. It captures the flow of information during the indexing and searching processes. Here is the dataflow diagram for the Index.

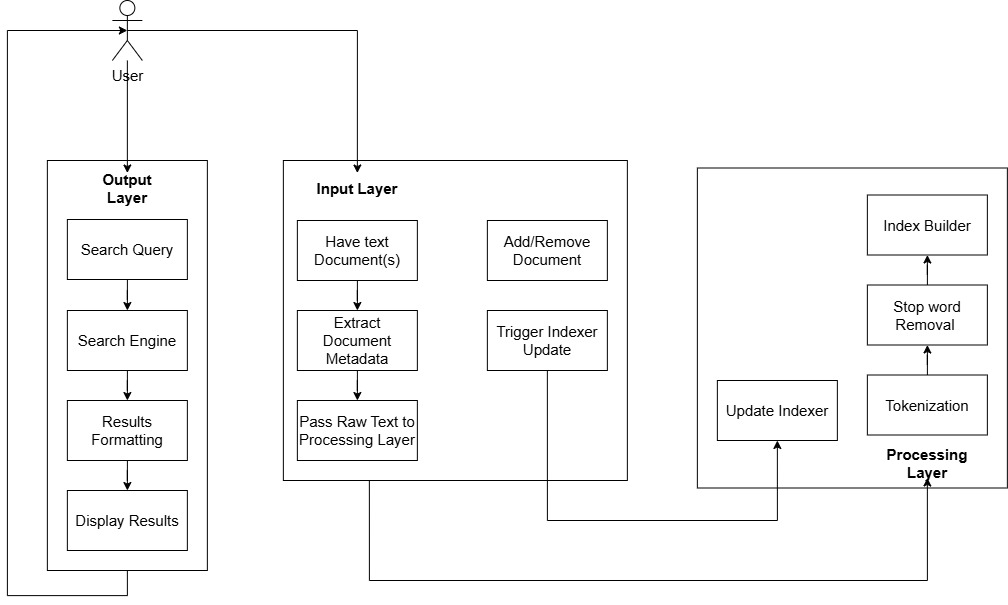


Figure 10: Dataflow Diagram

# Conclusion

In this project, I have implemented an index-based search engine capable of performing efficient word-based and title-based searches across a collection of documents. By using the inverted index approach, I have optimized the search process, ensuring quick retrieval of relevant documents and their frequencies.

The system is designed to handle dynamic updates, allowing new documents to be added at runtime without compromising performance. Key features like stop-word removal, tokenization, and case normalization were included to improve the accuracy and relevance of search results.

Overall, this project demonstrates the importance of indexing in information retrieval systems and highlights how an indexer can provide a fast and scalable solution for handling large document collections. The implementation not only meets the project objectives but also lays a strong foundation for extending the system with advanced features like phrase searches and ranking mechanisms in the future.