California State Polytechnic University, Pomona

CS 5180 Information Retrieval

Project Report
Team 1

CPP Search Engine for International Business and Marketing Department

Student 1	Student 2	Student 3	Student 4	Student 5
Amir Mohideen Basheer Khan	Abdullah Irfan Siddiqui	Fidelis Prasetyo	Roberto Rafael Reyes	Sean Archer

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PROJECT PLAN

Introduction

This project aims to develop a topical search engine for the International Business and Marketing Department at Cal Poly Pomona. Currently, students and faculty must manually navigate individual professor web pages to explore faculty research work, a process that is time-consuming and inefficient. To address this issue, we propose a comprehensive solution that automates the discovery, extraction, and indexing of faculty research information, enabling quick and seamless searches.

The search engine will utilize a web crawler to systematically navigate and retrieve data from the department's website, starting from a seed URL. By leveraging Python libraries such as urllib, BeautifulSoup, and PyMongo, the crawler will identify, and parse faculty pages based on a standard header template. Extracted data, particularly the "Areas of Research" section, will be stored in a MongoDB database for subsequent indexing and retrieval.

To ensure relevance and efficiency, the search engine will implement a query-based retrieval system using the scikit-learn library. Users will be able to enter search queries and receive paginated results, displaying clickable links to faculty pages along with text snippets. This solution will significantly enhance the usability and accessibility of research information within the department, benefiting both students and faculty.

By automating data retrieval and creating an intuitive search mechanism, this project lays the groundwork for a more streamlined and efficient exploration of academic research at Cal Poly Pomona.

Organization of the Project

Table 1: Organization of the Project

Student Name	Role	Tasks Completed
Amir Mohideen Basheer Khan	Project Manager, Documentation	• Set up communication and project organization over discord • Drafted this project word template, contributing towards documentation • Created the Presentation Slides
Abdullah Irfan Siddiqui	Programmer	Worked on the optimized crawler and parser
Fidelis Prasetyo	Programmer	Coding lemmatizer, blurb getter, and utilities functions
Roberto Rafael Reyes	Documentation, Presentation Slides	Contributed towards Documentation and Slides
Sean Archer	Programmer	Coding the 1st crawler, parser, index design, IR design, ranking

Methods and Techniques

Our project employed a structured and modular approach to developing an efficient information retrieval system tailored for the International Business and Marketing Department. Below are the primary methods and techniques used:

1. **Web Crawler Design**

   A robust web crawler was implemented to systematically navigate the department's website, beginning from a seed URL. Utilizing Python libraries such as urllib and

BeautifulSoup, the crawler identified and processed faculty web pages by parsing their content. The crawler stored relevant data, including HTML content and target status, in a MongoDB database for subsequent processing.

2. **Data Storage and Indexing**

   Extracted data was structured and stored in a document-oriented database (MongoDB). Two primary collections were utilized: v3_test_pages for storing raw page data and v3_inverted_index for term-document mappings. The inverted index was built using TF-IDF scores to enable efficient ranking and retrieval of documents based on search queries.

3. **Text Transformation**

   Data transformation began with cleaning HTML content to isolate significant textual information. Using NLTK and regex, punctuation, and stop words were removed, and the text was normalized to lowercase. Tokens were generated using unigram, bigram, and trigram representations to capture word combinations, enhancing context awareness. Stemming and lemmatization were considered for text normalization; however, lemmatization was excluded from the master implementation due to its computational overhead.

4. **Query Processing and Ranking**

   The Vector Space Model (VSM) was employed to rank documents. Queries and documents were represented as vectors in a high-dimensional space, with weights calculated using the TF-IDF scheme. Cosine similarity was applied to determine the relevance of each document to the user query, ensuring accurate and contextually relevant results.

5. **Optimized Crawling and Scalability**

   An optimized version of the crawler improved performance by navigating 130 pages in just 54 seconds and successfully scaling to crawl 17,091 pages while identifying 1,747 faculty pages. This optimization highlighted the effectiveness of regex-based link filtering in enhancing crawler efficiency and scalability.

Through the integration of these techniques, the project successfully automated data retrieval and indexing, creating an intuitive search engine for faculty research information.

TEXT ACQUISITION

Crawler design

This crawler pseudocode outlines the logic for a web crawler designed to traverse a website, gather HTML data, and identify target links. It starts from a seed URL, generates a list of frontier URLs, and filters, validates, and processes each link. Key features include:

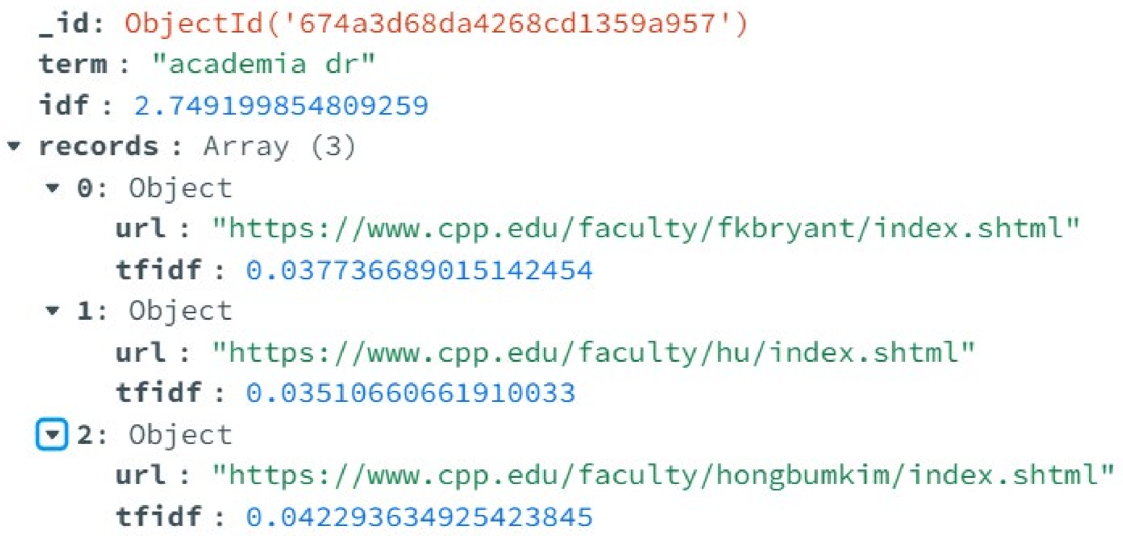
1. **Initialization**: Sets the base URL and the specific target URL to locate.
2. **Link Handling**: Extracts anchor tags, filters links ending in .html or .shtml, and converts relative URLs to absolute ones.
3. **HTML Parsing**: Retrieves and prettifies page content using BeautifulSoup.
4. **Database Storage**: Inserts the crawled data, including page HTML and target status, into a database.
5. **Crawling Process**: Recursively explores links within the same domain, tracking visited pages, and stops after finding a specified number of target links.

The crawler is robust, handling errors gracefully and focusing on a modular design for clarity and extensibility.

Pseudocode:

```
CLASS Crawler:
    VARIABLES:
        base_url // The base domain of the website to crawl
        target_url // The specific target URL or identifier to search for

    METHOD __init__(base_url, target_url):
        SET self.base_url = base_url
        SET self.target_url = target_url

    METHOD visit_link_and_gather_anchor_tags(link_to_visit):
        TRY:
        Open the link using urlopen
        Parse the HTML using BeautifulSoup
        RETURN all anchor tags in the HTML
        CATCH HTTPError, URLError, Exception:
            PRINT appropriate error message
        RETURN an empty list

METHOD create_list_of_raw_links(unfiltered_url_list):
        SET filtered_url_list = []
        FOR each URL in unfiltered_url_list:
                IF URL ends with '.html' or '.shtml':
                        ADD URL to filtered_url_list
        RETURN filtered_url_list

METHOD construct_correct_urls(current_page_url, filtered_url_list):
        SET absolute_urls = []
        FOR each href in filtered_url_list:
                IF href is a relative link:
                        CONVERT to absolute URL using urljoin
                ADD href to absolute_urls
        RETURN absolute_urls

METHOD generate_new_frontier_urls(current_link):
        SET anchor_tags = CALL visit_link_and_gather_anchor_tags(current_link)
        SET raw_list = CALL create_list_of_raw_links(anchor_tags)
        RETURN CALL construct_correct_urls(current_link, raw_list)

METHOD get_html(some_url_link):
        TRY:
                Open the link using urlopen
                Parse and prettify the HTML using BeautifulSoup
                RETURN formatted HTML
        CATCH HTTPError, URLError, Exception:
                PRINT appropriate error message
        RETURN empty string

METHOD is_target_link(current_link, link_to_find):
        RETURN True if link_to_find is in current_link, else False

METHOD is_domain_page(link):
        RETURN True if base_url is in link, else False

METHOD is_department_page(link):
        RETURN True if base_url is in link, else False

METHOD insert_into_database(db_manager, link, page_html, is_target):
        CREATE document object with fields:
                'url': link
                'page_html': page_html
                'is_target': is_target
        CALL db_manager.insert_document(document)

METHOD crawl(seed_url, db_manager):
        SET base_frontier = CALL generate_new_frontier_urls(seed_url)
        SET num_targets = 22
        SET targets_found = 0
SET visited_urls = an empty set

WHILE base_frontier is not empty:
    SET link = base_frontier.pop(0)
    IF link in visited_urls:
        CONTINUE
    ADD link to visited_urls

    IF CALL is_domain_page(link):
        SET is_target = CALL is_target_link(link, target_url)
    IF is_target:
        INCREMENT targets_found
    SET page_html = CALL get_html(link)
    IF page_html is not empty:
        CALL insert_into_database(db_manager, link, page_html, is_target)
    IF targets_found == num_targets:
        CLEAR base_frontier
        RETURN

    SET additional_frontier = CALL generate_new_frontier_urls(link)
    FOR each next_link in additional_frontier:
        IF next_link not in base_frontier:
            ADD next_link to base_frontier
```

Document store design



```
_id: ObjectId('674a3d68da4268cd1359a957')
term : "academia dr"
idf : 2.749199854809259
▾ records : Array (3)
  ▾ 0: Object
      url : "https://www.cpp.edu/faculty/fkbryant/index.shtml"
      tfidf : 0.037736689015142454
  ▾ 1: Object
      url : "https://www.cpp.edu/faculty/hu/index.shtml"
      tfidf : 0.03510660661910033
  ▾ 2: Object
      url : "https://www.cpp.edu/faculty/hongbumkim/index.shtml"
      tfidf : 0.042293634925423845
```

Figure 1: Screenshot of MongoDB Database

The database model schema is using a document-oriented database model implemented using mongoDB, a NoSQL database program. The database is stored locally on the client's PC under the name “project_db”. This database includes two collections: “v3_test_pages” collection and “v3_inverted_index” collection. Below is an explanation of the documents and their respective fields in each collection:

- v3_test_pages

```
{
  "_id": {
"$oid": "674d3ce9f0e3e3714dacd25e"
  },
  "url": "https://www.cpp.edu/privacy.shtml",
  "page_html": "<!DOCTYPE html>…</html>”,
  "is_target": false
}
```

This collection contains information about every website crawled by the crawler. The fields are as follows:

- “_id”: A unique, randomly generated ObjectId assigned to each document.
- “url”: The URL of the crawled website.
- “page_html”: The complete HTML content of the website.
- “is_target”: A boolean flag indicating whether the website is a target for indexing. If its value is true, the document will be included in the indexing process.

- v3_inverted_index

```
{
"_id": {
"$oid": "674d45fe541b587ece72126f"
  },
  "term": "advertising",
  "idf": 1.9858167945227654,
  "records": [
{
          "url": "https://www.cpp.edu/faculty/fkbryant/index.shtml",
          "tfidf": 0.07502262576910462
},
{
          "url": "https://www.cpp.edu/faculty/mcgood/index.shtml",
          "tfidf": 0.012485182802039356
},
…..,
{
          "url": "https://www.cpp.edu/faculty/jroxas/index.shtml",
          "tfidf": 0.06508933049363083
}
  ]
}
```

This collection stores inverted index data generated during the indexing process using TfidfVectorizer. The fields are as follows:

- “_id”: A unique, randomly generated ObjectId assigned to each document.
- “term”: The term being indexed.
- “idf”: The inverse document frequency (IDF) value for the term
- “records”: An array of object, each containing:
  - “url”: The URL of a webpage where the term was found.
  - “tfidf”: The term frequency-inverse document frequency (TF-IDF) value representing the weight of the term in the specific document.

TEXT TRANSFORMATION

Tokenizer definition

The process of tokenizing web documents starts with extracting relevant content from the HTML pages. Using **BeautifulSoup**, the HTML content stored in the **PAGES** collection is parsed to isolate significant text, such as specific sections or division tags that contain useful information. This ensures that only the necessary textual data is processed, reducing the amount of irrelevant data. Once extracted, the text undergoes a thorough cleaning process using a combination of regular expressions and Natural Language Toolkit (**NLTK**) functions. During this process, punctuation and stop words are removed, and the text is normalized by converting all characters to lowercase. The result is a concise and standardized textual representation of the web page, ready for further processing.

The cleaned text is then tokenized into smaller tokens, which are the basis for indexing. Using the NLTK's ngrams function, the text is divided into unigrams, bigrams, and trigrams. This process captures both individual terms and word combinations that provide context, magnifying the relevance of the index. To generate the final index terms, the TfidfVectorizer library is employed, which computes the TF-IDF for each token. This metric evaluates the importance of a term within a specific document relative to its appearance across the entire corpus, assigning higher scores to more unique and significant terms. The resulting index terms, along with their TF-IDF scores and associated document URLs, are stored in the inverted_index collection. This term-document mapping allows for efficient and precise query handling by linking terms to the documents where they appear and ranking them by relevance.

Stopping definition

Stop words are often removed during the text processing to enhance the relevance of index terms. In this project, stop words are addressed during the text cleaning phase, ensuring that only meaningful and content-rich words are tokenized and indexed. The NLTK library is used to identify and filter out these stop words from the extracted text. NLTK provides a list of English stop words, which is applied to the tokenized content to remove words that do not contribute to the semantic understanding of the document.

The filtering process involves tokenizing the cleaned text into individual words using NLTK's word_tokenize function and then comparing each token against the stop word list. Words that are in the stop word list are excluded from further processing. This method ensures that the resulting tokens are both concise and representative of the document's content. By removing stop words early in the pipeline, the system reduces noise in the index and improves the efficiency of text search operations.

Stemming/Lemmatizing definition

Stemming or Lemmatization is performed during the text cleaning phase to standardize words, allowing variations of a word to be treated as a single term. For instance, words such as "running", "runner" and "ran" can be normalized to their root form, "run," to unify their representation in the index. Lemmatization is performed as part of the text cleaning process implemented in the utilities.py file. The clean_text function uses NLTK's tools to tokenize the text into individual words and remove unnecessary characters, such as punctuation.

In this project, lemmatization is applied to both the documents and the query. It is performed before tokenization to ensure that the processed terms are the lemmas of the original terms. The lemmatization process utilizes the WordNetLemmatizer from the NLTK library.

However, we created multiple versions of our implementation, with and without lemmatization. From our observations, lemmatization significantly impacts the program's speed while producing results similar to the versions that omit the lemmatization step. This is potentially because lemmatization relies on predefined dictionaries containing a vast number of English word variations. As a result, we chose not to include lemmatization in the implementation's master branch. Further research and testing are necessary to optimize the lemmatization process for potential inclusion in future versions.

INDEX CREATION

Index design

The project employs an inverted index stored in the **inverted_index** collection of the database to enable efficient retrieval and ranking of documents based on query terms. An inverted index maps terms to the documents where they are from, making it an essential structure for scalable and fast information retrieval. The index creation process begins with extracting and parsing the HTML content from crawled web pages stored in the **pages** collection. Relevant sections of the page are isolated using **BeautifulSoup** to focus on meaningful content. The extracted text is cleaned by removing punctuation, stop words, and unnecessary characters, leaving only significant terms for processing. This text is then tokenized into individual words and n-grams, capturing both single terms and multi-word phrases that add contextual depth.

Once tokenization is complete, the **TF-IDF** weighting scheme is applied to compute the importance of each term. For each unique term, the inverted index stores its **idf** value, along with a list of docs that contain the term. Each document in this list includes the term's TF-IDF score and its URL, creating an efficient mapping from terms to their relevant documents. This data is then stored in the **inverted_index** collection, using MongoDB's **update_one** method with the **upsert=True** option to ensure terms are either added or updated without duplication. By embedding document references and their TF-IDF scores directly into the index, the system enables quick lookups and accurate ranking during query processing. This structured approach enhances retrieval performance, supports large datasets, and ensures a scalable and robust search infrastructure.

Weighting definition

The project employs the **TF-IDF** weighting scheme to evaluate the importance of terms within the corpus and ensure that relevant documents are prioritized during retrieval. TF-IDF consists of two components: TF and IDF. To compute TF-IDF weights, the project uses the Scikit-learn library's TfidVectorizer, which processes the cleaned and tokenized text from each document. For each term in a document, the TF is calculated as the ratio of the term's occurrences to the total terms in the documentm while the IDF is computed as the logarithm of the ratio of the total number of documents to the number of documents containing the term. The resulting TF-IDF scores are then stored in the **inverted_index** collection, with each term linked to its associated documets and their respective TF-IDF values. This weighting method certifies that terms carrying higher informational value significantly impact the document ranking process, enhancing the accuracy and relevance of search results.

RANKING

IR models

The Vector Space Model is employed as the foundation for ranking documents based on their relevance to user queries. In this model, documents and queries are represented as vectors in a high-dimensional space, with each dimension corresponding to a unique term in the corpus. To assign weights to these terms, we utilize the TF-IDF scheme. TF measures how frequently a term appears in a document, indicating its local importance, while IDF adjusts the weight based on the term's rarity across all documents, reducing the influence of common terms and emphasizing unique ones. These TF-IDF weights are used to construct documents vectors, with higher weights for terms that are both locally significant and globally unique.

To rank the documents in response to query, we employ cosine similarity, a measure that calculates the cosine of the angle between the query vector and each document vector. This normalized metric, ranging from 0 which means no similarity to 1 which means perfect match, quantifies the alignment between the query and document. Cosine similarity is computed by dividing the dot product of the vectors by the product of their magnitudes. The query itself is processed into a vector using the same TF-IDF weighting scheme for consistency. Documents are then ranked in descending order of their similarity scores, ensuring that the most relevant results appear at the top. This combination of VSM and cosine similarity enables efficient and accurate retrieval, prioritizing documents with significant term overlap and relevance to the query.

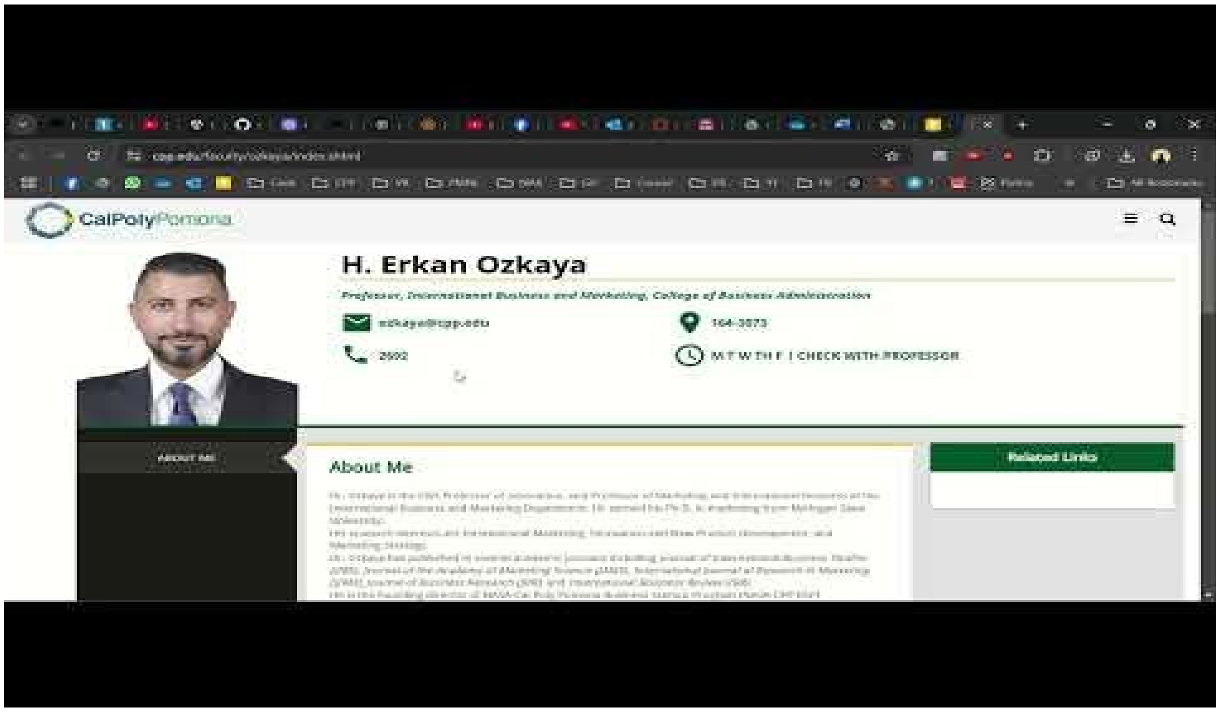
IMPLEMENTATION

Source-Code

https://github.com/Sean1783/CS5180_Group_Project

Video Demo

https://youtu.be/-7uvD2wYCuo


CalPolyPomona
H. Erkan Ozkaya
Professor, International Business and Marketing, College of Business Administration
ozkaya@cpp.edu
164-3073
2502
M T W TH F | CHECK WITH PROFESSOR
ABOUT ME
About Me
Related Links

DISCUSSION

The crawler successfully navigated 130 web pages to locate the 22 faculty target pages within a runtime of 2 minutes and 24 seconds. This result highlights the efficiency of the system in balancing link traversal and page evaluation of the target criteria. Developing and refining the parsing method was key to this success. Early iterations of the parsing method used overly broad criteria, which significantly increased runtime by processing irrelevant pages. Conversely, overly narrow criteria missed many valid pages. Through iterative testing and refinement, we developed a balanced parsing approach that efficiently and accurately identified the target pages.

Additionally, an optimized crawler was built, further enhancing performance. This optimized crawler successfully navigated the same 130 pages to locate 22 target pages in just 54 seconds, showcasing significant improvements in speed and efficiency. When configured to continue crawling from the seed URL, it demonstrated scalability by crawling 17,091 pages and identifying 1,747 faculty pages, underlining its capability to handle larger datasets effectively. The optimized crawler's superior speed may be attributed to the use of regex expressions, which streamlined pattern matching and reduced reliance on multiple if-else statements. This contributed to more efficient parsing and traversal, enabling faster processing of web pages.

Throughout the project, we encountered the challenge of learning advanced concepts, such as crawling, inverted indexing, and cosine similarity, while simultaneously implementing them. One significant milestone was building the inverted index, which involved coordinating information from multiple sources (raw HTML, extracted terms, and TF-IDF calculations) and combining them into a coherent database structure. Each term in the index was associated with its TF-IDF score and a list of URLs where it appeared, allowing for efficient query handling. Another achievement was ranking search results. We adopted a scaffolded approach, starting with basic scoring methods and progressively building toward the use of cosine similarity. This iterative process ensured we had a functional system at every stage while enabling continuous improvement.

One notable issue we faced during text transformation was the impact of lemmatization on performance. While lemmatization enhances text normalization by reducing words to their base forms, its inclusion significantly slowed the system due to its to reliance on predefined word dictionaries. Interestingly, our observations indicated that lemmatization produced results like those without it, which led us to exclude it from the master branch of the implementation. Instead, the final version prioritized lightweight text normalization techniques, such as tokenization and stop word removal, to optimize performance.

Overall, this project emphasized the importance of iterative design, balancing precision and efficiency, and adapting to challenges during implementation. By continuously refining our approach and incorporating more sophisticated methods as our understanding improved, we were able to create a robust crawler and search system capable of efficiently identifying and ranking relevant web pages.

REFERENCES

1. *API Reference*. (n.d.). Scikit-Learn. https://scikit-learn.org/stable/api/index.html
2. Richardson, L. (2019). *Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation*.Crummy.com.https://www.crummy.com/software/BeautifulSoup/bs4/doc/
3. NLTK. (2009). *Natural Language Toolkit — NLTK 3.4.4 documentation*. Nltk.org. https://www.nltk.org/
4. *Getting Started with MongoDB & Mongoose | MongoDB*. (n.d.). Www.mongodb.com. https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/
5. *PyMongo 3.11.0 Documentation — PyMongo 3.11.0 documentation*. (n.d.). Pymongo.readthedocs.io. https://pymongo.readthedocs.io/en/stable/