```markdown
# A20565628
# Sai Dhanush Soma

## Abstract
The project aims to develop a web document search system that includes a web
crawler for downloading HTML documents, an indexer for constructing an
inverted index, and a query processor for handling free-text queries. The
system is designed to allow users to search for relevant documents based on
their input query. Next Step is to perform crawling, indexer and flask along
with Tfidf and Cosine Similarity

## Overview
The solution consists of three main components:
1. **Web Crawler (project.py)**:
   - Description: A Scrapy-based crawler for downloading HTML documents from
specified URLs.
   - Features:
     - Initializes with seed URL, max pages, and max depth.
     - Concurrent crawling with AutoThrottle.
     - Distributed crawling with Scrapyd (optional).
     Navigation:- Change your file directory to the spider folder where
project.py is located
   - Usage: Run the crawler using `scrapy crawl project`.

2. **Indexer (indexer.py)**:
   - Description: A Scikit-Learn-based indexer for constructing an inverted
index in pickle format.
   - Features:
     - TF-IDF score representation.
     - Cosine similarity calculation.
     - Optional: Vector embedding representation (word2vec), Neural/Semantic
search kNN similarity (FAISS).
     Navigation:- Change your file directory to the Scrapee folder where
indexer.py is located
   - Usage: Run the indexer using `python indexer.py`.

3. **Query Processor (app.py)**:
   - Description: A Flask-based processor for handling free-text queries in
JSON format.
   - Features:
     - Query validation/error-checking.
     - Top-K ranked results.
     - Optional: Query spelling-correction/suggestion (NLTK), query expansion
(WordNet).
     Navigation:- Change your file directory to the Scrapee folder where
app.py is located
   - Usage: Run the query processor using `python app.py`.
```

## Design
The system is designed to crawl, index, and process queries on web documents efficiently. It uses Scrapy for crawling, Scikit-Learn for indexing, and Flask for query processing. The components interact to provide users with a seamless search experience.

## Architecture
The architecture consists of three main layers: the presentation layer (Flask app), the processing layer (indexer), and the data layer (crawled HTML documents). These layers interact to enable document search functionality.

## Operation
To use the system, follow these steps:
1. Ensure Python and necessary dependencies are installed.
2. Run the web crawler using `scrapy crawl project`.
3. Run the indexer using `python indexer.py`.
4. Run the query processor using `python app.py`.
5. Access the search interface via the provided URL.

## Conclusion
The project successfully implements a web document search system, allowing users to search for relevant documents based on their input queries. However, further enhancements such as semantic search and distributed crawling could be explored in future iterations.

## Data Sources
The system crawls HTML documents from [specified URLs](https://www.scrapethissite.com/pages/).

## Test Cases
Test cases are provided in the `testing.py` file. These test cases validate the functionality of the Flask application, including index page rendering and query processing.

## Source Code
Source code files (`project.py`, `indexer.py`, `app.py`, `index.html`, `testing.py`) are included in the repository. Dependencies are listed in the `requirements.txt` file.

## Bibliography

- Aggarwal, S. (2019). Flask Framework Cookbook: Over 80 proven recipes and techniques for Python web development with Flask. Packt Publishing Ltd.
- Kouzis-Loukas, D. (2016). Learning scrapy. Livery Place: Packt Publishing.
- Koerich, W. V., & dos Santos Mello, R. (2013). A Query-by-Similarity Indexing Strategy for Web Forms. In SBBD (Short Papers) (pp. 16-1).