INTRODUCTION

In today's digital age, we are inundated with vast amounts of textual information in the form of documents, articles, reports, and more. The ability to quickly and accurately access relevant information from this sea of text is of paramount importance for businesses, researchers, students, and professionals alike. Imagine a web app that allows you to simply type or speak your query, just like you would ask a question to a human, and receive precise and contextually relevant document results within seconds. Whether you are searching for research papers, legal documents, medical records, news articles, or any other textual information, NLP-powered document query systems can make the process seamless and efficient. In this fast-paced world, where information is the key to success, NLP-driven document query systems are empowering individuals and businesses to make data-driven decisions, conduct comprehensive research, and stay informed. As NLP technology continues to advance, we can expect even more sophisticated document query systems that cater to diverse domains and languages, further bridging the gap between humans and machines in understanding and utilizing the wealth of textual knowledge available to us.

CURRENT SYSTEM

Currently Document analysers are available in the market but the significance of this project is that it can Perform Question Answering in natural language to find any answers in your documents. So this can be useful for IT Department during Job Hiring as it can sort and filter the candidates according to the job requirements by querying their CVs And also performs semantic search and retrieve documents according to meaning.

PROPOSED SYSTEM

The proposed system for the "Web App for Document Query in Natural Language Processing (NLP)" will feature a user-friendly web interface allowing users to log in and access personalized features. It will support the ingestion and indexing of various document formats, applying advanced NLP models. The system will employ semantic search techniques to comprehend context and semantics, offering query expansion suggestions and relevance ranking for improved search results.Here the Libraries used are StreamLit and Heystack.Users can apply filters, conduct faceted searches, and expect stringent security measures to protect sensitive data. The architecture will prioritize scalability and performance, while analytics and reporting functionalities will empower users to visualize insights. Continuous learning and user support resources will ensure an adaptive and user-centric platform, making it a valuable tool for information retrieval and analysis in various fields.
The proposed system seeks to create an adaptable and user-friendly platform for NLP-based document query, addressing the need for efficient information retrieval and analysis across diverse domains. Leveraging cutting-edge NLP technologies, it will empower users to effortlessly unlock valuable insights from their document repositories, serving the needs of academia, business intelligence, content curation, and more.

SCOPE OF THE WORK

The main purpose of the project is to allow users to search and retrieve information from a database of documents quickly and efficiently. It could serve as a central hub for businesses to manage and access their internal documents. Employees can search for specific files, contracts, reports, or any other relevant information within the organization. The app could utilize advanced technologies like natural language processing (NLP) and semantic search to offer more accurate and contextually relevant results. The app could incorporate collaboration features, allowing multiple users to work together on shared documents, enabling version control and real-time editing. During Job Recruitment process,This can be useful for filtering the Applicants who are eligible for specific roles by analysing their Resume or CVs. Heystack and NLP(Natural Language Processing) is using here to Perform Question Answering in natural language to find granular answers in your documents. Heystack is an end-to-end NLP framework that enables you to build NLP applications powered by LLMs, Transformer models, vector search and more. Whether you want to perform question answering, answer generation, semantic document search, or build tools that are capable of complex decision making and query resolution, you can use the state-of-the-art NLP models with Haystack to build end-to-end NLP applications solving your use case.

LITERATURE SURVEY

1. "A Survey on Document Retrieval Techniques using Natural Language Processing" by A. Kumar and S. Gupta (2019)

This survey provides an overview of various techniques used in document retrieval using NLP. It covers preprocessing methods, query analysis, document ranking algorithms, and user feedback mechanisms. The paper also discusses the challenges and future directions in this field.

2. "Natural Language Processing for Information Retrieval: Concepts and Techniques" by C. D. Manning, P. Raghavan, and H. Schütze (2008)

This book chapter provides a comprehensive introduction to NLP techniques for information retrieval. It covers topics such as tokenization, stemming, query expansion, and relevance ranking. The chapter also discusses the use of machine learning algorithms for document retrieval.

3. "A Review of Natural Language Processing Techniques for Opinion Mining Systems" by A. Esuli and F. Sebastiani (2009)

This review paper focuses on NLP techniques for opinion mining, which can be useful for sentiment analysis in document retrieval systems. It discusses methods for extracting subjective information from text and sentiment classification algorithms. The paper also highlights the challenges and future directions in opinion mining.

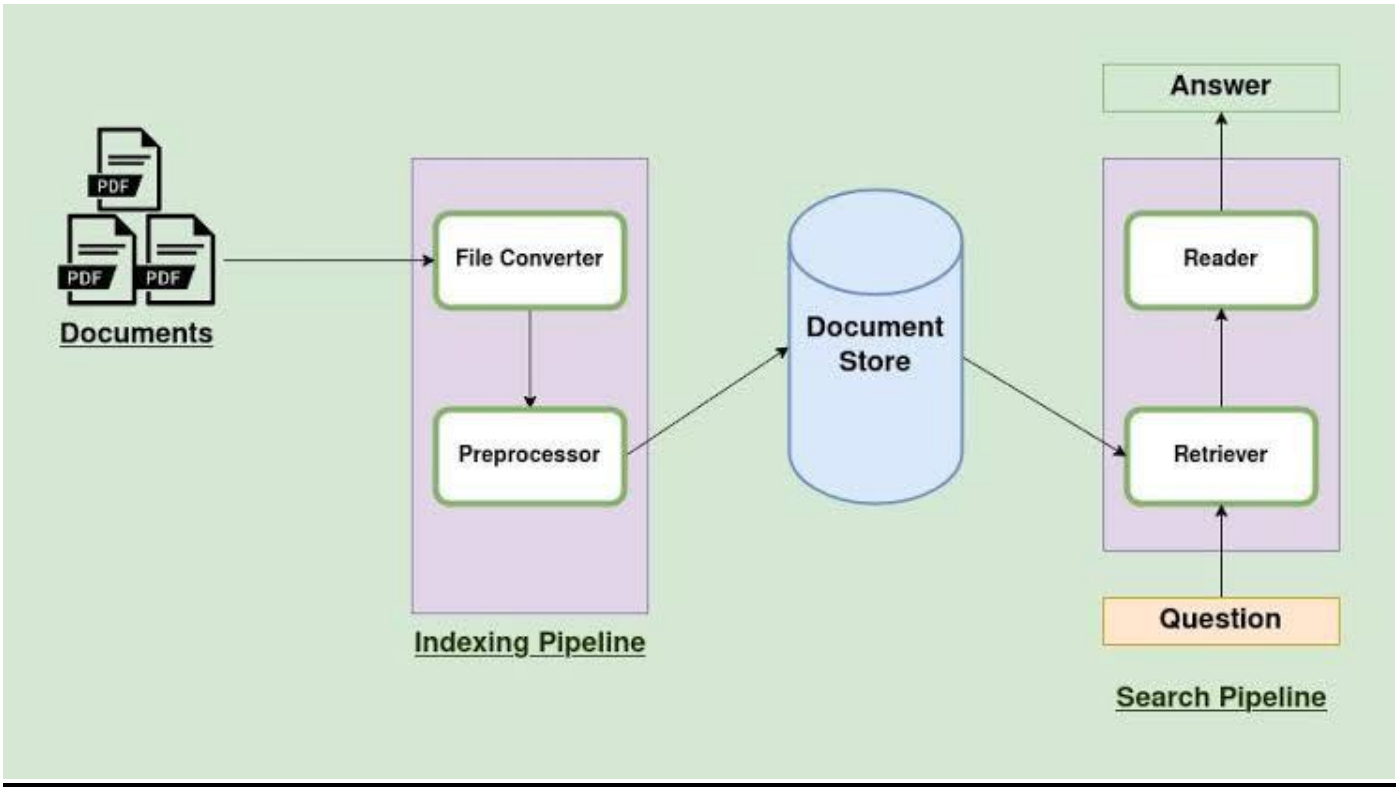
These literature sources provide a solid foundation for understanding the concepts, techniques, and challenges involved in developing a web app for document query using NLP. They offer insights into various preprocessing methods, query analysis techniques, document ranking algorithms, and user feedback mechanisms. Additionally, they highlight the potential applications and future directions in this field.

METHODOLOGY

The following are the steps that form the part of the methodology for developing a web based system for querying document in a question answer format

- **Input Data**: The Web App takes a corpus of text documents as input. These documents could be articles, manuals, web pages, or Resumes
- **Initializing the DocumentStore**: The documents are stored in a document store,which can be a database.Haystack uses this store for fast and efficient retrieval of relevant documents during the question-answering process.
- **Initializing the Retriever Node**: The Retriever component is responsible for quickly finding the most relevant documents related to a given query. It uses BM25 Algorithm or neural retrievers to rank and retrieve the relevant documents from the document store.
- **Initializing the Reader Node**: The Reader component is responsible for processing the retrieved documents and finding the exact answers to the user's questions. It utilizes pretrained transformer-based models to extract answers from the documents.
- **Creating the Retriever-Reader Pipeline**: It connects the Reader and the Retriever. The combination of the two speeds up processing because the Reader only processes the Documents that the Retriever has passed on.
- **Agent**: In some cases, there might be multiple potential answers extracted from different documents. The Agents combines and ranks these answers to provide the most likely and accurate response.
- **User Interface**: The output of the system is presented to the user through a Web Application

WORKFLOW DIAGRAM



REQUIREMENT SPECIFICATIONS

TECHNOLOGIES USED

Technologies used for Feature Implementation	
Programming Language	Python
Toolkit	Django
Framework	API rest
Build Tool	Html and CSS

HARDWARE REQUIREMENTS

Workstation Base Unit
500 GB
HDD DVD +/- RW
16 GB RAM

SOFTWARE REQUIREMENTS

Software installed on SAG Server
Windows 10 Enterprise
Microsoft Office (Word, Excel, PowerPoint)
Notepad
PyCharm

Functional requirements:

1. Document query submission:
- Users should be able to submit natural language queries through the web app's interface.
- The system must process and interpret these queries effectively, including handling complex questions and context-specific requests.

2. Document indexing and retrieval:
- The web app must efficiently index and retrieve documents from a large corpus.
- Users should receive relevant search results based on the query, with options to view documents or summaries.

3. Natural language understanding (NLU):
- The system should employ NLP models to understand and interpret user queries accurately.
- NLU capabilities should include entity recognition, sentiment analysis, and context-aware query processing.

4. Semantic search:
- The application should perform semantic search to understand the context and semantics of documents and queries.
- Results should reflect not just keyword matching but also context-aware relevance.

5. Query refinement and suggestions:
- The web app should provide query expansion suggestions to help users refine their queries for better results.
- Autocomplete and query suggestion features should assist users in formulating effective queries.

6. Document filters and sorting:
- Users must have the ability to apply filters to narrow down search results by date, author, document type, and other attributes.
- The system should support various sorting options, such as relevance, date, or document relevance scores.

7. Security and privacy:
- Stringent security measures should safeguard sensitive documents and user data.
- User data should be handled in compliance with data privacy regulations, ensuring confidentiality.

8. Scalability:
- The web app should be designed to scale efficiently to handle growing volumes of documents and user demands.

Non-functional requirements:

1. Security:
- The application must ensure secure user authentication and authorization.
- Document access controls should be in place to protect sensitive information.

2. Performance:
- The system should provide low-latency responses to user queries.
- It must efficiently handle concurrent search requests and document retrieval.

3. Scalability:
- The web app should be designed to scale horizontally to accommodate the increasing volume of documents and users.

4. Reliability:
- Rigorous testing and monitoring mechanisms must be in place to ensure the reliability of the web app.
- Automatic error handling and recovery features should minimize downtime.

5. Maintainability:
- The project should follow best practices for modular and maintainable software design, allowing for easy updates and enhancements.

6. Usability:
- Both the user interface and the document query functionality should be intuitively designed for user-friendliness.
- User feedback mechanisms should be implemented to enhance usability.

7. Response time:
- The system should minimize query processing and document retrieval times to ensure a responsive user experience.

8. Load testing:
- The web app should support load testing to simulate real-world scenarios and assess its capacity to handle concurrent document queries effectively.

By adhering to these functional and non-functional requirements, Web app for document query aims to deliver a secure, efficient, and user-friendly solution for retrieving information from large document collections using natural language queries.

DETAILED WORK OF PLAN

The below table specifies the list of tasks and deliverables at every milestone of the project.

Serial Number of Task	Tasks or subtasks to be done. (Be precise and specific)	Planned duration in weeks	Specific Deliverable in terms of the project
1	Requirement Analysis	1	Requirement Specification Document Creation
2	Design	2	Design Preparations.
3	Model Selection	1	Trying out different transformer models
3	Coding	7	Coding the system (Python, NLP, Streamlit,,HayStack).
4	Testing	2	Testing the web app.
5	Documentation	2	Documenting the application.
6	Preparation of Project Report	2	Creating the project report for final submission.

CODE

```
"""
URL configuration for docsearchproject project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path,include

from docsearchproject import settings

#from haystack.views import SearchView
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('documentapp.urls')),
   # path('search/',SearchView.as_view(),name='haystack_search')

]
if settings.DEBUG:
    urlpatterns+=static(settings.STATIC_URL,document_root=settings.STATIC_ROOT)
    urlpatterns+=static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)




from django.urls import path,include

from .import views

urlpatterns = [

    path('',views.index,name='index'),
    path('upload/', views.upload_file, name='upload_file'),
    path('upload/query', views.query, name='query'),

]
```

```
import glob


from django.http import HttpResponse
from django.shortcuts import render, redirect
from .models import FileUploadForm
import base64
import PyPDF2
from pathlib import Path
from pypdf import PdfReader
import os
import pandas as pd
from haystack.document_stores import InMemoryDocumentStore
from haystack.pipelines import ExtractiveQAPipeline
from haystack.pipelines.standard_pipelines import TextIndexingPipeline
from haystack.nodes import BM25Retriever
from haystack.nodes import FARMReader
from haystack.nodes.retriever.sparse import TfidfRetriever
from transformers import BertTokenizer, BertForQuestionAnswering
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from django.template import context
from haystack.nodes.reader import TransformersReader



# Create your views here.
def index(request):
    return render(request, "document.html")
```

```
def convert_pdf_to_text(pdf_path):
    text = ""
    try:
        with open(pdf_path, 'rb') as pdf_file:
            pdf_reader = PyPDF2.PdfReader(pdf_file)
            for page_num in range(len(pdf_reader.pages)):
                page = pdf_reader.pages[page_num]
                text += page.extract_text()
    except Exception as e:
        print(f"An error occurred: {e}")
    return text



def upload_file(request):
    uploaded_files=[]
    if request.method == 'POST':
        form = FileUploadForm(request.POST, request.FILES)
        if form.is_valid():
            uploaded_file = form.cleaned_data['file']
            save_folder = './source_documents/'
            save_path = save_folder + uploaded_file.name
            print(save_path)
            print(uploaded_file.name)
            with open(save_path, 'wb+') as w:
                for chunk in uploaded_file.chunks():
```

```
w.write(chunk)

        uploaded_files= [f for f in os.listdir(save_folder) if f.endswith(".pdf")]
        print(uploaded_files)
        for path in Path("./").glob("**/*.pdf"):
            text = convert_pdf_to_text(path)
            txt_path = path.parent / (".".join(path.name.split(".")[:-1]) + ".txt")
            if txt_path.exists():
                print(f"Skip {txt_path} as it already exists")
                continue
            with open(txt_path, "wt", encoding="utf-8") as fp:
                fp.write(text)
        uploadFlag = True
        print(uploadFlag)
        # return render(request, 'success.html', {'file_name': uploaded_file.name})
    else:
        form = FileUploadForm()

    return render(request, 'upload.html', {'form': form,'uploaded_files': uploaded_files})



def query(request):
    if request.method == 'POST':
        query = request.POST['query']
        print(query)
        doc_dir = "./source_documents"
```

```
print(doc_dir)
        files_to_index = [doc_dir + "/" + f for f in os.listdir(doc_dir)]
        print(files_to_index)
        folder_path = os.path.join(os.getcwd(), "source_documents")
        text_files = glob.glob(os.path.join(folder_path, "*.txt"))
        print(text_files)
        document_store = InMemoryDocumentStore(use_bm25=True)
        print(document_store)
        indexing_pipeline = TextIndexingPipeline(document_store)
        indexing_pipeline.run_batch(file_paths=text_files)


        retriever = TfidfRetriever(document_store=document_store)
        retrieved_documents = retriever.retrieve(query)
        # for doc in retrieved_documents:
        # print("Document ID:", doc.id)
        # print("Text:", doc.content)


        reader = FARMReader(model_name_or_path="deepset/roberta-base-squad2",
use_gpu=True)
        pipe = ExtractiveQAPipeline(reader, retriever)


        prediction = pipe.run(
            query=query,
            params={
                "Retriever": {"top_k": 10},
                "Reader": {"top_k": 5}
```

```
}
        )

        answers = prediction["answers"][0]
        result = answers.answer
        print("Query:", query)
        print("Answer:", result)
        print(type(result))

        return render(request, 'upload.html', {'result': result, 'query': query})
        #return redirect('upload_file')
    else:
        result = []

    return render(request, 'upload.html', {'result': result})
```

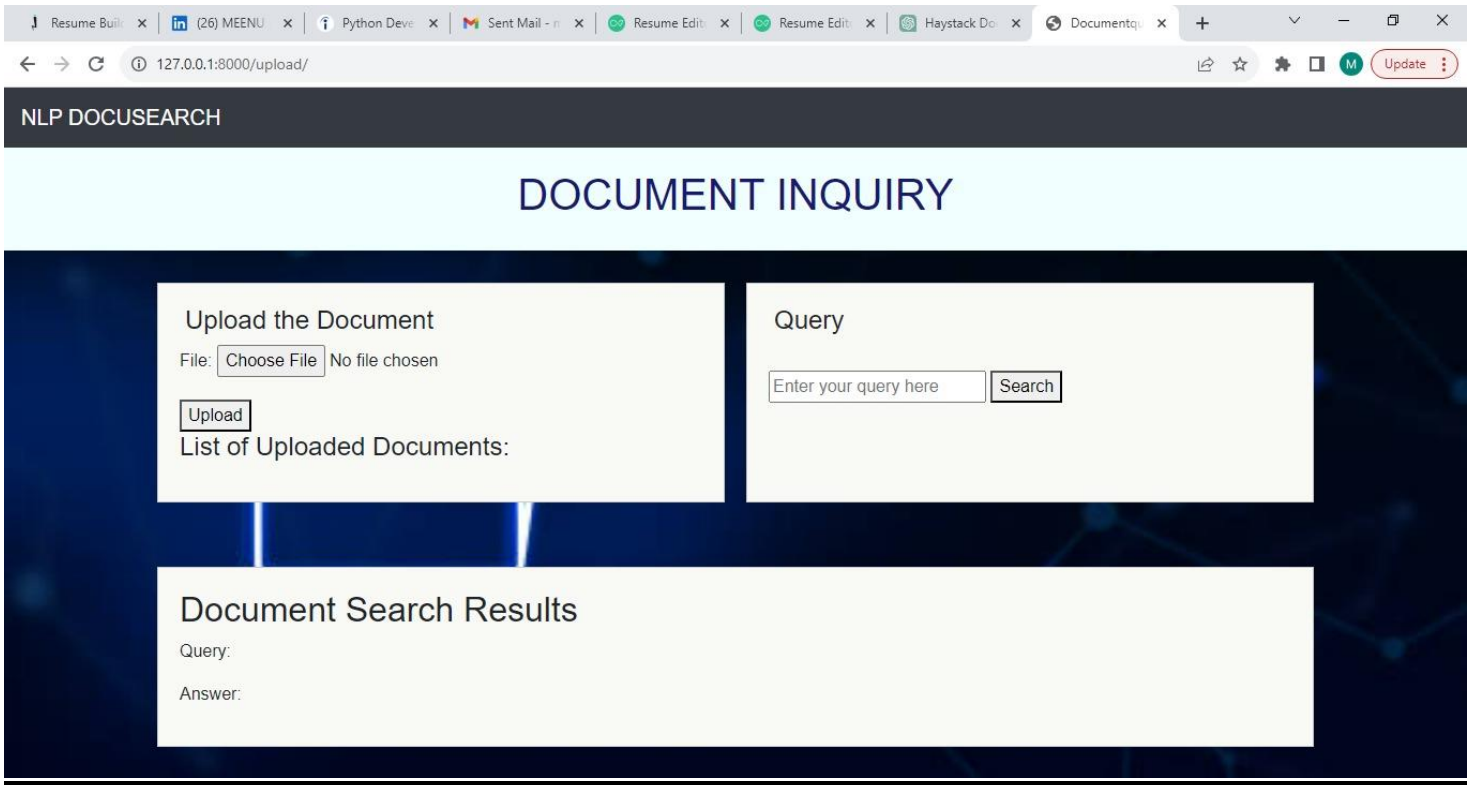
```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Documentquery</title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

    <style>

        body {
            background-image: url("{% static 'images/background1.jpg' %}"); /* Replace 'your-
image.jpg' with the actual image file path or URL */
            background-size: cover; /* Adjust the size to cover the entire viewport */
            background-repeat: no-repeat; /* Prevent the image from repeating */
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
        }
        header {
            background-color:          #F0FFFF;
            color: #191970;
            text-align: center;
```

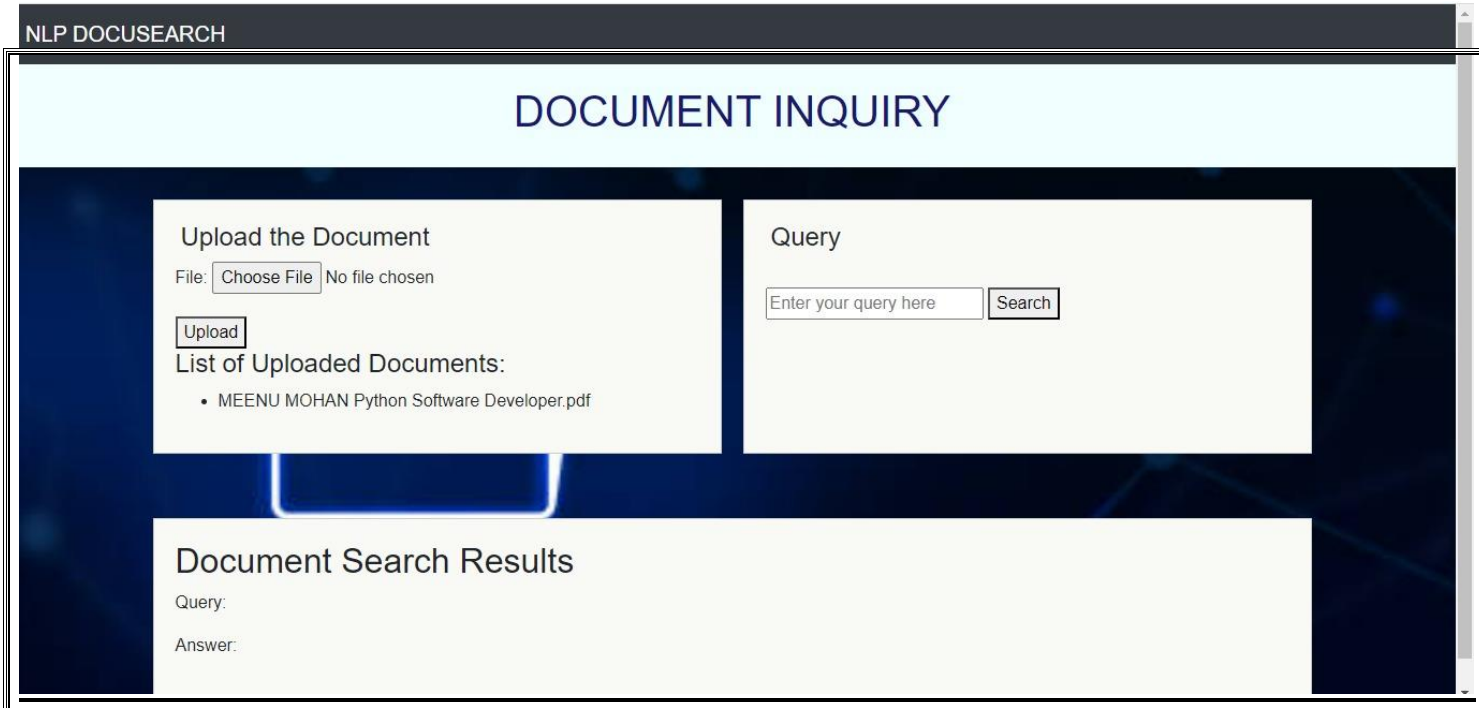
```
 padding: 20px;
      }

 .input-group{
         margin-top: 50px;
         margin-left: 20px;
         }
      .container {
         display: flex;
         justify-content: space-around;
         padding: 20px;
      }
      .section {
         flex: 1;
         margin: 10px;
         padding: 20px;
         border: 1px solid #ccc;
         background-color: #f9f9f6;
      }
   </style>
</head>
<body>
<nav class="navbar navbar-dark bg-dark">
   <a class="navbar-brand" href="#">NLP DOCUSEARCH</a>
</nav>
<header>
```

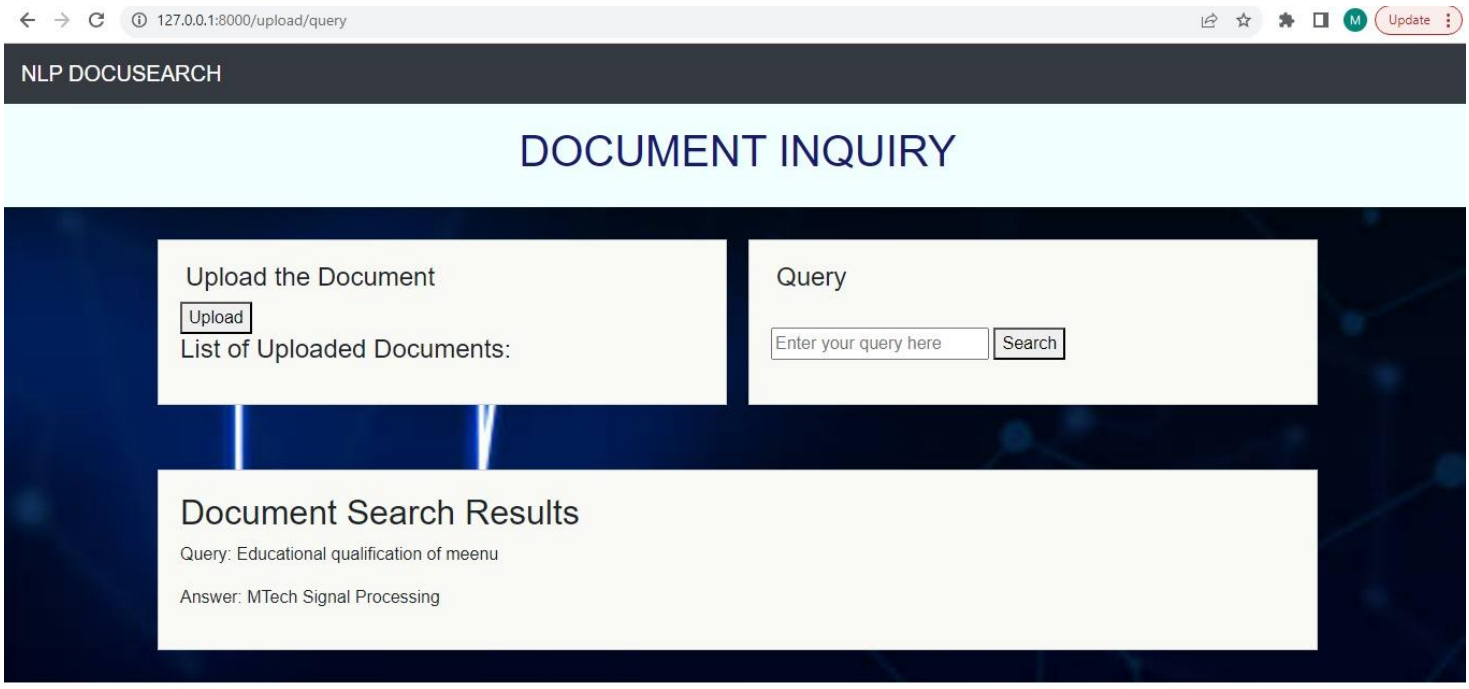
```
 <h1>DOCUMENT INQUIRY</h1> <!-- Header text -->
</header>
<div class="container">
   <div class="section">

 <h4 style="margin-left:5px;">Upload the Document</h4>
      <form method="post" enctype="multipart/form-data">
         {% csrf_token %}
         {{ form.as_p }}
         <button type="submit">Upload</button>
      </form>
      <h4>List of Uploaded Documents:</h4>
      <ul>
         {% for document in uploaded_files %}
            <li>
               {{document}}
            </li>
         {% endfor %}
      </ul>
   </div>
   <div class="section">
      <h4 style="margin-left:5px;">Query</h4>
      <form method="post" action="query" class="query">
         {% csrf_token %}
         <br>
         <input type="text" name="query" placeholder="Enter your query here">
```

```
  <button type="submit">Search</button>
      </form>
   </div>
</div>
<div class="container">
      <div class="section">
         <h2>Document Search Results</h2>
         <div id="answer-display">
            <p>Query: {{ query }}</p>
            <p>Answer: {{result}}</p>




            <!-- The answer will be displayed here -->
         </div>
      </div>
   </div>
</body>
</html>
```

SCREENSHOT


127.0.0.1:8000/upload/
Update
NLP DOCUSEARCH
DOCUMENT INQUIRY
Upload the Document
File: Choose File No file chosen
Upload
List of Uploaded Documents:
Query
Enter your query here
Search
Document Search Results
Query:
Answer:


DOCUMENT INQUIRY
Upload the Document
File: Choose File No file chosen
Upload
List of Uploaded Documents:
MEENU MOHAN Python Software Developer.pdf
Query
Enter your query here
Search
Document Search Results
Query:
Answer:


127.0.0.1:8000/upload/query
Update
NLP DOCUSEARCH
DOCUMENT INQUIRY
Upload the Document
Upload
List of Uploaded Documents:
Query
Enter your query here
Search
Document Search Results
Query: Educational qualification of meenu
Answer: MTech Signal Processing

CONCLUSION

In conclusion, the development of a web app for document query has proven to be a valuable and efficient solution for managing and retrieving information from various documents. The web app provides users with a user-friendly interface that allows them to search for specific content within documents, saving time and effort in manual searching.

By implementing advanced search algorithms and techniques, such as keyword matching, natural language processing, and machine learning, the web app can accurately analyze and index the content of documents, ensuring accurate and relevant search results.

Additionally, the web app offers features like document categorization, filtering, and sorting, making it easier for users to organize and access their documents. The app also allows for easy collaboration and sharing of documents across different users and teams.

Overall, the web app for document query enhances productivity, streamlines document management, and improves information retrieval. By leveraging the power of technology, it simplifies the process of searching and accessing specific information within documents, ultimately leading to increased efficiency and effectiveness in various industries and sectors.