Legacy Search Creation Documentation

This documentation provides a comprehensive step-by-step explanation of the Legacy Search creation process, covering both the frontend and backend aspects. It includes code snippets where necessary, along with detailed explanations of how different components were implemented and integrated, starting from the setup, data ingestion, to the deployment and functionality of the legacy search interface.

Server Implemented on:
(http://ec2-54-79-231-211.ap-southeast-2.compute.amazonaws.com)

(Site: http://54.79.231.211/)

★ Project Structure Overview

➔ Key Directories and Files:

- **danswer/legacy_search/**

➔ app.py (Frontend: Streamlit application)
➔ mongo_utils.py (Backend: MongoDB connection and search logic)
➔ Dockerfile (Docker setup for the legacy search)
➔ requirements.txt (Python dependencies)
➔ .env (Environment variables for MongoDB connection)
➔ venv/ (Virtual environment for Python dependencies)
➔ start_streamlit.sh (Streamlit start script)

1) (mongo_utils.py) Backend Setup:

Path: cd danswer/legacy_search/mongo_utils.py

Summary: The **backend** of the **Legacy Search system leverages MongoDB Atlas** to **store** and **manage legal case data**. **First**, a **free MongoDB Atlas cluster** was **created**, and its **connection string** was **stored securely** in a .**env file** within the **project directory**. The **Python-based mongo_utils.py file** was **responsible** for **handling database connections**, **CSV data ingestion**, and **executing full-text searches**. **Key data fields** such as **case title**, **date of judgment**, **judge**(**s**) **names**, and **all_text** were **extracted** from the **provided CSV**, and **additional fields** were **ignored** as **specified**. The **backend functionality** includes an **exact match search** and a **refined search** that **allows** users to **narrow down results** by **specific keywords** or **filters** such as **judge names**, **case titles**, and **date ranges**. Additionally, the **backend setup** involved writing a **script** that **automates** the **insertion** of **data** into **MongoDB** and **configuring** a **Docker container** for **deployment.**

❖ **Key Modifications:**

- **MongoDB Atlas Cluster Creation**: **We created** a **MongoDB Atlas cluster** to **store** the **legal case data**. The **steps involved**:

- **Cluster Deployment: Set** up a **free M0 cluster** using **MongoDB Atlas**.
- **User Creation: Added** a **database user** with a **username** and **password (currently my username and password)**, and **allowed access** from all **IP (0.0.0.0/0)**.
- **Connection URL: Retrieved** the **connection string** in the **following format**:

  mongodb+srv://ibrahimsultan4705:y4Zn89Jxxl1stb4k@cluster0.22eho.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

- **Environment Variables: Stored** this **connection string** in a .**env file** for **use** in the **Python code**.

  CONNECTION_URL=mongodb+srv://ibrahimsultan4705:y4Zn89Jxxl1stb4k@cluster0.22eho.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

2) Data Ingestion into MongoDB:

- **CSV File: Received** a **CSV** file (**fixedddd.csv**) **containing** the **following fields**:

- **case title**
- **date of judgment**
- **judges name(s)**
- **all_text**

- **Data Insertion Script:** A **Python script** in **mongo_utils.py** was **created** to **handle data ingestion**. **Important** parts of the **script**:

- **MongoDB Connection: connect_mongo() establishes** the **connection using** the **environment variable**.

- **Data Ingestion: insert_data**(**filepath**) **inserts** the **filtered** data (**selected fields**) **from the CSV** file into the **MongoDB collection**.

- **Code:**

```
def insert_data(filepath):
    collection = connect_mongo()
    data = pd.read_csv(filepath, usecols=['case title', 'judges name(s)', 'date of judgment', 'all_text'])
```

```
    data_dict = data.to_dict("records")
    collection.insert_many(data_dict)
    print("Data inserted successfully")
```

- **Execution:** **The data** was **ingested** into the **database using** the **following command**:

  python -c "from mongo_utils import insert_data; insert_data('fixedddd.csv')"

3) MongoDB Search Functionality:

- **Implemented search logic** in **mongo_utils.py**:

- **Full-text search: text_search() function** that **searches** within the **all_text field** in **MongoDB**.
- **Refined Search: Functionality** to **refine search** results by **keywords**.
- **Advanced Filters: Filter** results **by judge name**, **case title**, and **date range**.

- **Code:**

```
def text_search(search_text):
    collection = connect_mongo()
    res = list(collection.find({"$text": {"$search": search_text}}).limit(100))
    res.sort(key=lambda doc: doc["all_text"].count(search_text), reverse=True)
    return res
```

4) (app.py) Streamlit Frontend Setup:

Path: cd danswer/legacy_search
Path: cd danswer/web/src/components/search/SearchResultsDisplay.tsx
Path: cd danswer/web/src/app/legacysearch

Summary: The **frontend** for the **Legacy Search** was **built** using **Streamlit**, **providing** a **clean** and **responsive user interface**. The **core functionality includes** a **search bar** where **users** can **input queries** and **receive paginated results** from the **MongoDB backend**. The **frontend** also **allows users** to **refine their search** through **advanced filters** like **case title**, **judge name**, and **date**. **Additionally**, a **theme toggle feature** was **added by streamlit**, **enabling** users to **switch** between **light** and **dark modes**. A **"Return to Search" button** was placed **prominently** in the **UI** to **allow users** to **easily navigate back** to the **main search system (Currently Commented out)**. The **front end also uses** a simple **layout** for **displaying search results**, with **improved accessibility features** like **real-time feedback** on **searches** and **smooth transitions between pages**. The **application** was **containerized** using **Docker** to **ensure smooth deployment alongside** the **backend components**.

- **The code built the frontend of the legacy search using Streamlit. The key components are:**

- **Search Input:** A **basic search bar** allowing **users** to **input queries**.
- **Refine Search:** A **section allowing** users to **further refine results** based on **additional keywords**.
- **Advanced Filters: Filters** for **judge name(s)**, **date range**, and **case titles**.

- **Code:**

```
query = st.text_input("Enter your search query")
if st.button("Search"):
   st.session_state.res = base_search(query)
```

- **Results Display and Pagination:**

- **Pagination:** The **results** are **paginated** with **navigation buttons** for easy **browsing**.
- **Result Display:** For **each document**, **relevant fields** such as **case title**, **judges name**(**s**), and **date** of **judgment** are **displayed**.

- **Code:**

```
page = st.session_state.get("page", 1)
per_page = 10
start_index = (page - 1) * per_page
end_index = start_index + per_page
```

5) Integration and Enhancements:

- **Button Integration with Main Search:**

- **Legacy Search Integration:** We **implemented** a **button** in the **main search system** to **redirect** users to the **legacy search** when no **results** were **found**.
- The button **redirects** to /**legacysearch**.

- **Code:**

```
          <button
            className="btn-46"
            onClick={() => {
              window.location.href = "/legacysearch";
            }}
          >
            Still looking for answers? Click here to explore our legacy search.
          </button>
```

- **Redirects to /legacysearch which renders Page.tsx:**

- **Code:**

```
import React from 'react';

const LegacySearch = () => {
  return (
    <div style={{ width: '100%', height: '100vh' }}>
      <iframe
        src="http://54.79.231.211:8501"
        style={{ width: '100%', height: '100%', border: 'none' }}
        title="Legacy Search"
      />
    </div>
  );
};

export default LegacySearch;
```

- **Back to Search Button: Added** a **"Return to Search" button** in the **Streamlit as st.button**. This **button allows users** to **return** to the **main search page (Currently commented out)**.

- **The URL for this is: http://54.79.231.211/search.**

- **Code:**

```
# Add button to return to standard search
# if st.button("Return to Standard Search"):
#     st.session_state.page = 1
#     st.session_state.res = ""
#     st.rerun()
```

- **Light/Dark Mode Toggle: Streamlit provides** a **toggle button allowing users** to switch **between light** and **dark modes**.

- **Light Mode: Light background** with **dark text**.
- **Dark Mode: Dark background** with **light text**.

6) Deployment:

- **Docker Setup and Deployment for Legacy Search:**

- **Dockerfile: Created** a **Dockerfile** for the **legacy search application**. This **Dockerfile builds** the **Streamlit app** and **installs** the **necessary dependencies** from **requirements.txt**.

- **Code:**

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install the required packages
RUN pip install --no-cache-dir -r requirements.txt

# Make port 8501 available to the world outside this container
EXPOSE 8501

# Run the streamlit app
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

- **Docker-Compose Integration:**

- **Service Definition: Added** the **streamlit_app service** to **docker-compose.yml** to **run** the **legacy search** in **Docker**. **Mapped** port **8501** to **expose** the **Streamlit interface**.

- **Code:**

```
 streamlit_app:
    build:
      context: ../../legacy_search
      dockerfile: Dockerfile
    ports:
      - "8501:8501"
    restart: always
    volumes:
      - ../../legacy_search:/app
```

- **Testing and Validation:**

- **Container Restart: Ensured** that the **Streamlit app** restarts **upon failures** and **runs consistently** within the **Docker container**.

- **Port Check: Verified** that **no process** was **using port 8501** before **launching** the **application**.

  To access the Danswer application as an admin, please log in as:
  Email: noelshallum@gmail.com
  Password: 12345

- **Ibrahim Sultan**