

FastAPI Solr Integration

how to use solr in my persimon fast api python app

To use Apache Solr in your FastAPI Python app, you'll need to interact with the Solr server using a Python client. The `pysolr` library is a popular choice for this.

Here are the steps to integrate Solr into your FastAPI application:

1. Install Required Libraries

You will need the `pysolr` library to communicate with Solr.

```
bash

pip install pysolr
```

2. Set Up Solr Server

Ensure that your Solr server is running. You can download and set up Solr from the [official site](#), or use an existing Solr instance.

Once Solr is running, create a core or collection for your data.

For example, to create a core:

```
bash

bin/solr create_core -c mycore
```

3. Initialize Solr Client in Your FastAPI App

In your FastAPI app, you can initialize the Solr client using `pysolr`.

Here's an example setup for using Solr with FastAPI:

Example FastAPI App with Solr

```
python

import pysolr
from fastapi import FastAPI, HTTPException

# Connect to Solr
SOLR_URL = "http://localhost:8983/solr/mycore" # Replace with your Solr instance
```

URL and core name

```
solr = pysolr.Solr(SOLR_URL, always_commit=True)

app = FastAPI()

@app.get("/search")
async def search(query: str):
    """Search Solr for a query"""
    try:
        results = solr.search(query)
        return {"num_found": results.hits, "docs": results.docs}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Solr error: {str(e)}")

@app.post("/add_document")
async def add_document(doc: dict):
    """Add document to Solr"""
    try:
        solr.add([doc])
        return {"status": "success", "doc": doc}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error adding document: {str(e)}")

@app.delete("/delete_document")
async def delete_document(id: str):
    """Delete a document from Solr by ID"""
    try:
        solr.delete(id=id)
        return {"status": "success", "id": id}
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error deleting document: {str(e)}")
```

4. Interacting with Solr

- **Search Query:** You can use the `/search` endpoint to query Solr.
 - Example query: `http://localhost:8000/search?query=job`
- **Add Document:** You can add documents to Solr using the `/add_document` endpoint. A document might look like:

```
json
```

```
{
  "id": "123",
  "title": "Software Developer",
  "description": "Looking for a talented developer",
  "location": "New York"
}
```

- **Delete Document:** Use `/delete_document` to delete a document from Solr using the document `id`.

5. Example Solr Document Structure

When adding a document to Solr, ensure that the structure aligns with your Solr schema. For example, the schema in Solr might define fields like `id`, `title`, `description`, etc. Ensure these field names match the schema configuration in Solr.

6. Testing and Troubleshooting

- Ensure Solr is running properly by accessing the Solr Admin UI at `http://localhost:8983/solr/`.
- Verify your core or collection is set up properly and contains the fields you want to search or store.
- Test queries through the FastAPI endpoints.

With this setup, your FastAPI app can interact with Solr for search and indexing operations.

'bin' is not recognized as an internal or external command,
operable program or batch file.

The error message `'bin' is not recognized as an internal or external command` typically occurs because you're running the command from a Windows environment where the Solr `bin/` directory path is not included in the system's environment variables.

Here's how you can fix the issue and run Solr on Windows:

Step-by-Step Solution

1. Download and Install Solr

If you haven't already, download and install Solr from the [official website](#). Once downloaded, extract the archive to a directory on your system.

2. Start Solr on Windows

To start Solr on Windows, you need to use the `solr` command from the Solr installation directory. Here's how you can do it:

1. **Navigate to the Solr directory:** Open a command prompt (CMD) or PowerShell window, and navigate to the folder where Solr was extracted, for example:

```
bash  
  
cd C:\path\to\solr
```

2. **Start Solr:** In Windows, you can start Solr using `solr` batch script instead of the `bin/` directory commands.

Use this command to start Solr:

```
bash  
  
java -Dsolr.solr.home=./server/solr -jar start.jar
```

This will start Solr on `http://localhost:8983/solr`.

Alternatively, if you want to run Solr in the background, you can use the `start` command:

```
bash  
  
start java -Dsolr.solr.home=./server/solr -jar start.jar
```

3. Creating a Core in Solr

Once Solr is running, you can create a core for your data.

- To create a core, open a new command window (leave the Solr server running) and run the following:

```
bash  
  
java -Dsolr.solr.home=./server/solr -jar solr-core-<version>.jar create -c  
mycore
```

This will create a new core named `mycore` (you can change `mycore` to whatever name you prefer). If this still doesn't work, make sure that the Solr installation directory has been added to the `PATH` environment variable for easier access from anywhere in the terminal.

4. Verify Solr is Running

To verify that Solr is running, open your web browser and navigate to:

```
bash

http://localhost:8983/solr/
```

If Solr is running correctly, you'll see the Solr Admin UI, where you can manage cores, collections, and search configurations.

5. Interact with Solr in FastAPI

After starting Solr, you can use `pysolr` to interact with it from your FastAPI app as previously discussed. Make sure your Solr server is up and running when you make requests from your app.

Let me know if you run into any issues following these steps!

where can i find path to solar

To find the path to Solr on your system, follow these steps based on your operating system.

On Windows:

1. **If you installed Solr manually:** If you've downloaded and extracted Solr manually (e.g., `solr-8.x.x.zip`), you should already know where you extracted it. The directory path you extracted it to is your Solr home.

For example, if you extracted Solr to `C:\solr\solr-8.x.x`, then your Solr home directory is `C:\solr\solr-8.x.x`.

2. **Using Environment Variables:** If Solr is installed globally, and you have set the `SOLR_HOME` environment variable, you can find the Solr path as follows:

- Open Command Prompt (CMD) and run:

```
bash
```

```
echo %SOLR_HOME%
```

This will display the path to your Solr home directory if the environment variable is set.

3. **Finding Solr Using Search:** If you're unsure where Solr is installed, you can search for `solr` using the File Explorer or by running a search command:

- In File Explorer, search for `solr` to find the folder containing Solr.
- Or, in Command Prompt, search for the Solr executable:

```
bash  
  
where solr
```

On Linux or macOS:

1. **If Solr is Installed Using a Package Manager:**

- On **Linux (e.g., Ubuntu)**, if Solr was installed via `apt` or similar, the default installation path is usually `/opt/solr`. You can check by running:

```
bash  
  
whereis solr
```

Or:

```
bash  
  
find / -name solr 2>/dev/null
```

2. **If Solr is Installed Manually:** If you manually installed Solr (e.g., by downloading and extracting the `.tar.gz` file), the path is wherever you extracted the Solr files. For example, if you extracted it to `/home/user/solr`, then that would be the Solr path.

3. **Using Environment Variables:** If you've set the `SOLR_HOME` environment variable, you can run:

```
bash  
  
echo $SOLR_HOME
```

This will return the path to your Solr installation.

Summary

- If you installed Solr manually, the Solr path is wherever you extracted the Solr archive.
- If you installed it using a package manager, common paths are `/opt/solr` (Linux) or `C:\solr` (Windows).
- Use `whereis solr`, `where solr`, or `find / -name solr` on your terminal to locate Solr.

Let me know if you need further clarification!

```
PS C:\Users\SESPL\bhargav-workspace\Day 34-35
exploratory da and persimon app\Persimon App> bin/solr
create_core -c mycore
bin/solr : The term 'bin/solr' is not recognized as the name
of a cmdlet, function, script file, or operable program.
Check the spelling of the name, or if a path was included,
verify that the path is correct and try again.
At line:1 char:1
+ bin/solr create_core -c mycore
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (bin/solr:String) [],
CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

The error you're encountering is because you're trying to run the Solr command as if it's available globally (i.e., using `bin/solr`), but on Windows, this is not the correct approach. You need to directly reference the `solr` script from the Solr installation directory or run the correct command to create a core.

Steps to Fix the Issue

1. Navigate to the Solr Installation Directory

You need to navigate to the directory where Solr was extracted and run the command from there.

Assuming you've downloaded Solr and extracted it to `C:\solr`, you can try the following steps:

1. Open PowerShell or Command Prompt and navigate to the Solr folder (replace this path with the actual directory where Solr is located):

```
bash
```

```
cd C:\path\to\solr
```

For example, if you extracted Solr to `C:\solr\solr-8.x.x`, you would do:

```
bash
```

```
cd C:\solr\solr-8.x.x
```

2. **Run the Solr Command from the Correct Directory:** From within the Solr directory, use the full path to the Solr command script. Instead of `bin/solr`, you can run:

```
bash
```

```
.\bin\solr create_core -c mycore
```

In this case, `.\bin\solr` ensures you're correctly referencing the `solr` script within the `bin` directory.

2. Using the `start.jar` Command to Start Solr (if Solr is not running)

If Solr is not running yet, you need to start it first:

1. Navigate to the Solr directory where the `start.jar` is located (e.g., `C:\solr\solr-8.x.x`).
2. Run the following command to start Solr:

```
bash
```

```
java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

This will start Solr on `http://localhost:8983/solr`.

3. Create the Core Once Solr is Running

Once Solr is running, you can proceed with creating the core:

1. From the Solr directory, run:

```
bash
```

```
.\bin\solr create_core -c mycore
```

This will create a new core named `mycore` that you can use for your data.

Summary:

- Make sure you're inside the Solr installation directory.
- Use `.\bin\solr` to execute Solr commands from PowerShell or Command Prompt.
- If Solr is not running, start it using `java -Dsolr.solr.home=./server/solr -jar start.jar` before creating the core.

Let me know if you encounter any further issues!

```
C:\Users\SESPL\Solr>cd C:\Users\SESPL\Solr\solr-9.7.0-  
src.tgz  
The directory name is invalid.  
  
C:\Users\SESPL\Solr>
```

It looks like you're trying to navigate to a `.tgz` file, which is a compressed archive, not a directory. You need to first extract the contents of the `.tgz` file before you can navigate to it and work with Solr.

Steps to Fix the Issue

1. Extract the `.tgz` File:

Since you're working with a `.tgz` file (which is a compressed archive), you need to extract it before you can run any Solr commands. You can use a tool like 7-Zip, WinRAR, or the built-in Windows Subsystem for Linux (WSL) to extract it.

Using 7-Zip (if installed):

- Right-click the `solr-9.7.0-src.tgz` file.
- Select `7-Zip` > `Extract Here` (or choose another extraction location).

This will create a folder containing the Solr files.

2. Navigate to the Solr Directory:

After extracting, you should have a folder named `solr-9.7.0` (or something similar). Navigate to that directory.

For example, if you extracted the files to `C:\Users\SESPL\Solr\solr-9.7.0`, run the following command in PowerShell or Command Prompt:

```
bash
```

```
cd C:\Users\SESPL\Solr\solr-9.7.0
```

3. Start Solr:

Once you're in the correct directory, you can start Solr by running the following command:

```
bash

java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

This will start Solr, and you should be able to access the Solr admin UI at <http://localhost:8983/solr>.

4. Create a Core:

After Solr is running, you can create a core. Run the following command to create a new core (e.g., `mycore`):

```
bash

.\bin\solr create_core -c mycore
```

Summary:

- Extract the `.tgz` file first using an extraction tool like 7-Zip.
- Navigate to the extracted Solr directory.
- Start Solr with the `java -Dsolr.solr.home=.\server\solr -jar .\start.jar` command.
- Then, create a core with `.\bin\solr create_core -c mycore`.

Let me know if you need further assistance!

```
C:\Users\SESPL\Solr\solr-9.7.0-src>java -
Dsolr.solr.home=.\server\solr -jar .\start.jar
Error: Unable to access jarfile .\start.jar
```

The error message `Error: Unable to access jarfile .\start.jar` indicates that the Solr directory structure might not be correct or the required files are missing. Here's how to resolve the issue:

Steps to Resolve:

1. Verify the Extraction Process:

Ensure that you have fully extracted the Solr `.tgz` file and that the directory structure contains all the necessary files. Typically, the Solr distribution should have the following structure:

- `solr-9.7.0/`
 - `bin/`
 - `server/`
 - `lib/`
 - `start.jar`

If the `start.jar` file is missing, it's possible that the version you're using is a source version (`solr-9.7.0-src`), which requires you to build Solr from the source before you can use it. If you're working with a source version, follow the steps below to build Solr.

2. Check the Folder Structure:

Look inside the `solr-9.7.0-src` folder and verify that the `start.jar` file is present in the folder where you're trying to run the `java -jar .\start.jar` command.

If the `start.jar` file is not there, then you're likely working with the source version of Solr, and you need to build it.

Option 1: Use a Prebuilt Solr Distribution

If you don't need the source code and just want a working Solr setup, I recommend downloading the prebuilt Solr distribution.

1. Download a Prebuilt Solr Release:

- Go to the [Solr Downloads page](#).
- Download the "Solr 9.x.x (pre-built)" ZIP or TAR file.
- Extract the contents to a directory (e.g., `C:\Users\SESPL\Solr\solr-9.x.x`).

2. Start Solr with the Prebuilt Version:

Once extracted, navigate to the `solr-9.x.x` directory and start Solr by running the following command:

```
bash
```

```
java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

This should start Solr without issues.

Option 2: Build Solr from Source

If you still want to build Solr from the source (which is what you have with the `solr-9.7.0-src` directory), follow these steps:

1. **Ensure Java is Installed:** Solr requires Java to be installed on your machine. You can check if Java is installed by running:

```
bash

java -version
```

If Java is not installed, download and install it from [here](#).

2. **Build Solr from Source:**

- Navigate to the `solr-9.7.0-src` directory:

```
bash

cd C:\Users\SESPL\Solr\solr-9.7.0-src
```

- If you have Apache Maven installed, you can use Maven to build Solr from source:

```
bash

mvn clean install -DskipTests
```

- This will build the `start.jar` file and other necessary components. After building, you can follow the steps to start Solr again.

Conclusion:

- If you want an easier setup, download the **prebuilt Solr distribution**.
- If you're building from source, ensure **Maven** is installed and build Solr following the steps mentioned above.

Let me know if you need further clarification or assistance!

If you're unable to download the Solr file, here are some alternative steps and troubleshooting options:

1. Use a Mirror Link to Download Solr

Sometimes the main download page may be slow or inaccessible. You can try using alternative mirror links:

- Visit the **Apache Solr download mirrors** page: [Solr Mirrors](#).
- Choose a mirror close to your location and select the **pre-built Solr distribution** (e.g., `solr-9.x.x.zip` or `solr-9.x.x.tgz`).
- Once you click on the link, it should redirect you to the download page of the mirror.

2. Check Your Internet Connection

- Ensure your internet connection is stable and fast enough for large file downloads.
- If you're behind a corporate or school firewall, it might be blocking large downloads. You could try a different network (like a mobile hotspot).

3. Use wget or curl (If Command Line Tools Are Available)

If you have **wget** or **curl** installed, you can try downloading the file via the command line.

For **wget**:

```
bash

wget https://downloads.apache.org/lucene/solr/9.7.0/solr-9.7.0.tgz
```

For **curl**:

```
bash

curl -O https://downloads.apache.org/lucene/solr/9.7.0/solr-9.7.0.tgz
```

4. Download from a CDN

You can try downloading the Solr release from a CDN or using an official mirror. Here's an example using **Solr 9.7.0**:

1. Go to <https://archive.apache.org/dist/lucene/solr/>.

2. Find and download the `solr-9.7.0.zip` or `solr-9.7.0.tgz` file.

5. Download Solr Using Docker (Alternative Method)

If you can't download Solr directly, you can use **Docker** to run Solr. This is especially useful if you want to test Solr without installing it locally.

Steps to Run Solr via Docker:

1. **Install Docker:** If you don't have Docker installed, follow the instructions [here](#).
2. **Run Solr Container:** After installing Docker, run the following command to pull and run Solr as a container:

```
bash

docker pull solr:9.7
docker run -d -p 8983:8983 --name solr solr:9.7
```

This will download Solr, start it, and map port 8983 on your local machine to Solr's port. You can access Solr by navigating to `http://localhost:8983/solr`.

6. Alternative: Try a Previous Version of Solr

If Solr 9.7.0 is not available for download or is causing issues, you can download a previous stable release like Solr 8.11.2:

- Visit <https://archive.apache.org/dist/lucene/solr/>, and download a previous version (e.g., `solr-8.11.2.zip` or `.tgz`).

Let me know if you encounter further issues, and I can help you troubleshoot further!

Solr 9.7.0

Solr 9.7.0 is the most recent Apache Solr release.

Source release: `solr-9.7.0-src.tgz` [PGP] [SHA512]

Binary releases: `solr-9.7.0.tgz` [PGP] [SHA512], `solr-9.7.0-slim.tgz` [PGP] [SHA512]

OpenAPI specification: `solr-openapi-9.7.0.json` [PGP] [SHA512]

Docker: `solr:9.7.0`, `solr:9.7.0-slim`

Change log

Solr 8.11.4

Solr 8.11.4 is the last release in the 8.x series.

Source release: solr-8.11.4-src.tgz [PGP] [SHA512]

Binary releases: solr-8.11.4.tgz [PGP] [SHA512] / solr-8.11.4.zip [PGP] [SHA512]

Docker: solr:8.11.4

Change log

Solr Reference Guide

where to click to download

To download **Solr 9.7.0** or any other version from the Apache Solr official site, follow these steps:

1. Go to the Apache Solr Download Page:

- Visit the [official Apache Solr downloads page](#).

2. Choose the Version You Want:

- You'll see a list of available releases.
- For Solr 9.7.0, look for "**Binary releases**" and find the option that suits your needs:
 - `solr-9.7.0.tgz` (Full release).
 - `solr-9.7.0-slim.tgz` (A slim version with fewer components).
- You can also download **Docker images** (`solr:9.7.0`).

3. Download Binary Release (Recommended):

- For most users, it's easiest to download the **binary release** (either `.tgz` or `.zip`).

Here's how:

- Look for the "**solr-9.7.0.tgz**" file under the **Binary releases** section and click on it to start downloading.

If the direct link isn't working or you prefer a mirror, you can use the alternative mirror links that may appear after clicking the download link or in case of slow download speeds.

Direct Download Links:

Alternatively, you can use the following direct download links for Solr 9.7.0:

- [solr-9.7.0.tgz](#)

- [solr-9.7.0-slim.tgz](#)

Just click the link, and it should begin downloading.

Let me know if you encounter any further issues!

download .tgz file now what to do

Great! Now that you've downloaded the `.tgz` file for Solr, follow these steps to set it up and get Solr running:

1. Extract the `.tgz` File

You need to extract the contents of the `.tgz` file before you can use Solr. Here's how to do that:

On Windows:

1. Using 7-Zip (if installed):

- Right-click on the `solr-9.7.0.tgz` file.
- Select **7-Zip > Extract Here** (or **Extract to solr-9.7.0**).
- This will extract the Solr files to a folder called `solr-9.7.0` (or something similar).

2. Using Command Prompt (if you prefer):

- Open Command Prompt and navigate to the directory where the `.tgz` file is saved.
- You can use a tool like `tar` in Git Bash or Windows Subsystem for Linux (WSL). If you have `tar` installed, run:

```
bash

tar -xvzf solr-9.7.0.tgz
```

- This will extract the Solr files to a folder named `solr-9.7.0`.

On macOS or Linux:

1. Using Terminal:

- Open the **Terminal** and navigate to the directory where the `.tgz` file is saved:

```
bash
```



```
cd /path/to/solr-9.7.0.tgz
```

- Use the `tar` command to extract the `.tgz` file:

```
bash

tar -xvzf solr-9.7.0.tgz
```

- This will extract the Solr files into a `solr-9.7.0` folder.

2. Navigate to the Solr Directory

Once extracted, navigate to the folder where Solr is located. In the command line (Command Prompt or Terminal), run:

```
bash

cd path_to_solr_folder/solr-9.7.0
```

For example, if you extracted Solr to `C:\Users\SESPL\Downloads\solr-9.7.0`, run:

```
bash

cd C:\Users\SESPL\Downloads\solr-9.7.0
```

3. Start Solr

To start Solr, you need to run the `start.jar` file, which starts the Solr service.

On Windows:

1. Open **Command Prompt** or **PowerShell** and navigate to the Solr directory (if you're not already there).
2. Run the following command to start Solr:

```
bash

java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

This command tells Solr where its data directory is and runs it on the default port 8983.

On macOS or Linux:

1. Open **Terminal** and navigate to the Solr directory.
2. Run the same command to start Solr:

```
bash
```

```
java -Dsolr.solr.home=./server/solr -jar ./start.jar
```

4. Access the Solr Admin UI

Once Solr starts, you can access the **Solr Admin UI** in your web browser. Go to:

```
bash
```

```
http://localhost:8983/solr
```

This will bring up the Solr dashboard where you can manage cores, query data, and configure Solr.

5. Create a Core (Optional)

To create a Solr core (a storage unit for indexing data), use the following command in the Solr directory:

```
bash
```

```
.\bin\solr create_core -c mycore
```

This creates a core named `mycore`. You can replace `mycore` with any name you prefer.

Summary:

1. **Extract** the `.tgz` file to a folder.
2. **Navigate** to the Solr folder.
3. **Run** Solr with the command `java -Dsolr.solr.home=./server\solr -jar .\start.jar`.
4. **Access** the Solr Admin UI at `http://localhost:8983/solr`.
5. Optionally, **create a core** using the command `.\bin\solr create_core -c mycore`.

Let me know if you run into any issues!

It sounds like after extracting the `.tgz` file, you're seeing another `.tar` file inside the extracted folder. This is normal because `.tgz` files are compressed TAR files, so the extraction of the `.tgz` file could leave you with a `.tar` archive that still needs to be extracted.

Steps to Extract the `.tar` File

You need to extract the `.tar` file, which contains the actual Solr files. Here's how you can do that:

On Windows:

1. Using 7-Zip:

- Right-click the `.tar` file.
- Select **7-Zip > Extract Here** or **Extract to "solr-9.7.0"** (depending on the option).
- This will extract the contents of the `.tar` file into a folder.

2. Using Command Prompt (if you have tar or Git Bash):

- Open **Command Prompt** or **Git Bash** and navigate to the folder where the `.tar` file is located.
- Use the following command to extract the `.tar` file:

```
bash

tar -xvf solr-9.7.0.tar
```

- This will extract the contents into a folder.

On macOS or Linux:

1. Open **Terminal** and navigate to the directory where the `.tar` file is located.
2. Use the `tar` command to extract the `.tar` file:

```
bash

tar -xvf solr-9.7.0.tar
```

After Extracting the `.tar` File:

Once you've extracted the `.tar` file, you should now see a folder containing the Solr files, such as:

- `bin/`
- `server/`
- `lib/`
- `start.jar`

Continue with Starting Solr:

Now, follow these steps to start Solr:

1. **Navigate to the Solr Folder:** Go into the folder where the Solr files were extracted.
2. **Start Solr:**

- On **Windows** (Command Prompt or PowerShell):

```
bash

java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

- On **macOS/Linux**:

```
bash

java -Dsolr.solr.home=./server/solr -jar ./start.jar
```

3. **Access Solr** in your browser at `http://localhost:8983/solr`.

Let me know if this helps, or if you face any further issues!

```
C:\Users\SESPL\solr-9.7.0>java -
Dsolr.solr.home=.\server\solr -jar .\start.jar
Error: Unable to access jarfile .\start.jar
```

The error message `Unable to access jarfile .\start.jar` indicates that Solr is not able to find or access the `start.jar` file. This could happen due to a few reasons, such as incorrect directory structure or missing files. Let's walk through the troubleshooting steps:

1. Verify the Solr Directory Structure

Ensure that you're in the correct directory and that the `start.jar` file exists in the expected location.

- Navigate to the directory where you extracted Solr (`C:\Users\SESPL\solr-9.7.0`).
- Check the folder for the `start.jar` file.

You should have a folder structure like this:

```
python

solr-9.7.0/
├─ bin/
├─ server/
├─ lib/
├─ start.jar
└─ ...
```

If you **don't** see the `start.jar` file in the root of the `solr-9.7.0` directory, then something might have gone wrong with the extraction, and the file might be missing.

2. Check if the `start.jar` File Exists

Make sure the `start.jar` file exists in the folder. You can do this by:

1. Navigate to the Solr folder:

```
bash

cd C:\Users\SESPL\solr-9.7.0
```

2. List the files (in Command Prompt or PowerShell):

```
bash

dir
```

Look for the `start.jar` file in the listing. If it's not there, you might have extracted the files incorrectly.

3. Use the Correct Path to `start.jar`

If the `start.jar` file is in the `solr-9.7.0\bin` folder, you need to adjust your command to point to the correct location.

Try running this command instead:

```
bash
```

```
java -Dsolr.solr.home=.\server\solr -jar .\bin\start.jar
```

This tells Solr to look in the `bin` folder for the `start.jar` file.

4. Re-extract the `.tgz` or `.tar` File

If you don't see the `start.jar` file, there may have been an issue with extracting the archive. Try re-extracting the `.tgz` or `.tar` file:

- Delete the current folder (`solr-9.7.0`).
- Re-download the `.tgz` or `.tar` file.
- Extract it again using a tool like **7-Zip** or **tar** to ensure all files are correctly extracted.

After re-extracting, check again for the presence of `start.jar` in the `solr-9.7.0` folder.

5. Try Using a Full Path to `start.jar`

If you're still having issues, you can try specifying the full path to `start.jar` . For example:

```
bash
```

```
java -Dsolr.solr.home=.\server\solr -jar C:\Users\SESPL\solr-9.7.0\start.jar
```

This ensures that the system looks for the `start.jar` file directly at the given path.

Next Steps:

- Check if `start.jar` exists in the `solr-9.7.0` directory.
- Use the correct path to `start.jar` if it's in a subfolder.
- If the file is still missing, re-extract the archive to make sure the files are complete.

Let me know if you need further assistance!

how to use solr in fast api

To use **Solr** in a **FastAPI** project, you will need to integrate Solr as a search engine and communicate with it using HTTP API calls. Solr exposes a rich set of REST APIs for managing and querying data, which you can interact with from your FastAPI application.

Here's a step-by-step guide on how to set it up:

1. Set Up Solr

Ensure Solr is running and accessible on your machine (or on a remote server).

- You can start Solr with the following command (assuming Solr is already installed and you've navigated to the correct directory):

```
bash

java -Dsolr.solr.home=.\server\solr -jar .\start.jar
```

- Once Solr is running, you can access the Solr Admin UI in your browser at `http://localhost:8983/solr`.

2. Install Required Libraries for FastAPI

You will need to install the `requests` library to interact with Solr's HTTP API from FastAPI.

Run the following command to install `requests`:

```
bash

pip install requests
```

Alternatively, you can use `httpx` (which is asynchronous and more appropriate for FastAPI):

```
bash

pip install httpx
```

3. Create a Solr Client in FastAPI

You'll be making HTTP requests to Solr to perform operations like adding documents, querying, etc.

Example with `httpx`:

Here's how you can create a simple Solr client using `httpx` in FastAPI:

1. Create a FastAPI app and a Solr client:

```
python
```

```

from fastapi import FastAPI
import httpx

app = FastAPI()

SOLR_URL = "http://localhost:8983/solr" # Solr base URL
CORE_NAME = "mycore" # Name of the Solr core you want to use

# Example function to search Solr
async def solr_search(query: str):
    async with httpx.AsyncClient() as client:
        # Construct Solr query URL
        url = f"{SOLR_URL}/{CORE_NAME}/select?q={query}&wt=json&rows=10"
        response = await client.get(url)
        response.raise_for_status() # Raise an exception if the request fails
        return response.json()

# Example function to add a document to Solr
async def add_to_solr(doc: dict):
    async with httpx.AsyncClient() as client:
        url = f"{SOLR_URL}/{CORE_NAME}/update?commit=true"
        response = await client.post(url, json=[doc])
        response.raise_for_status() # Raise an exception if the request fails
        return response.json()

@app.get("/search/")
async def search(query: str):
    # Search Solr with the given query string
    result = await solr_search(query)
    return result

@app.post("/add/")
async def add_document(doc: dict):
    # Add a document to Solr
    result = await add_to_solr(doc)
    return result

```

Explanation:

- `solr_search(query)` : Sends a search query to Solr and returns the results in JSON format. This uses Solr's `/select` endpoint.

- `add_to_solr(doc)` : Adds a document to Solr using Solr's `/update` endpoint.
- **FastAPI routes:**
 - `/search/` : Accepts a query parameter and returns search results from Solr.
 - `/add/` : Accepts a POST request with a document (in dictionary format) and adds it to Solr.

4. Testing the FastAPI Solr Integration

1. **Start Solr:** Make sure Solr is running and the core `mycore` is created in Solr. You can create a core using the Solr Admin UI or the command line:

```
bash

bin/solr create_core -c mycore
```

2. **Run FastAPI:** Save the Python code to a file (e.g., `app.py`), then run your FastAPI app:

```
bash

uvicorn app:app --reload
```

3. **Test Search Endpoint:**

- Open a browser or use `curl` to test the `/search/` endpoint:

```
bash

http://127.0.0.1:8000/search/?query=your_search_term
```

- This will send the search query to Solr and return the results.

4. **Test Add Document Endpoint:**

- To add a document to Solr, use a POST request with the document's data:

```
bash

curl -X 'POST' \
  'http://127.0.0.1:8000/add/' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": "1",
    "title": "Solr Example",
```

```
"content": "This is an example document."
}'
```

This will send the document to Solr for indexing.

5. Solr Schema Configuration

Ensure that your Solr core schema (usually located in `solr-9.7.0/server/solr/mycore/conf/schema.xml`) is configured to handle the fields you're sending in the documents.

For example, for a document with fields like `id`, `title`, and `content`, your `schema.xml` should have definitions for these fields.

xml

```
<field name="id" type="string" indexed="true" stored="true"/>
<field name="title" type="text_general" indexed="true" stored="true"/>
<field name="content" type="text_general" indexed="true" stored="true"/>
```

You can also use Solr's **dynamic fields** if you want to automatically handle new fields that are added in the future.

Summary

- **Install** `httpx` (or `requests` for synchronous operations).
- **Create an async Solr client** using FastAPI's `httpx` library.
- **Define search and add document endpoints.**
- **Configure Solr core schema** to handle the data you're sending.

This setup allows FastAPI to interact with Solr and use it as a search backend for your application. Let me know if you need further assistance!

```
java -Dsolr.solr.home=.\server\solr -jar .\start.jar
C:\Users\SESPL\solr-9.7.0>java -
Dsolr.solr.home=.\server\solr -jar .\start.jar
Error: Unable to access jarfile .\start.jar
```

The error `Unable to access jarfile .\start.jar` typically means that the `start.jar` file is either missing or the path to the file is incorrect. Let's go through a few troubleshooting steps to resolve this issue:

1. Verify the File Location

Ensure that the `start.jar` file exists in the Solr directory.

1. Navigate to the `solr-9.7.0` directory where you extracted the Solr files.
2. Check if the `start.jar` file exists in the root of the directory. You can do this by running the following command in Command Prompt:

```
bash
```

```
dir
```

Look for `start.jar` in the output. If it's missing, you may need to re-extract the Solr files properly.

2. Check the Correct Path

Make sure you're in the correct directory when running the command. The Solr directory should have the following structure:

```
python
```

```
solr-9.7.0/  
├─ bin/  
├─ server/  
├─ lib/  
├─ start.jar  
└─ ...
```

- If the `start.jar` file is in the `bin/` folder, modify your command to reflect the correct path. For example:

```
bash
```

```
java -Dsolr.solr.home=.\server\solr -jar .\bin\start.jar
```

This tells Solr to look in the `bin` folder for the `start.jar` file.

3. Use the Full Path to `start.jar`

If you're still encountering issues, you can try using the full path to the `start.jar` file. For example:

```
bash

java -Dsolr.solr.home=C:\Users\SESPL\solr-9.7.0\server\solr -jar
C:\Users\SESPL\solr-9.7.0\start.jar
```

Replace `C:\Users\SESPL\solr-9.7.0` with the correct path to your Solr folder.

4. Check Java Installation

Ensure that **Java** is correctly installed on your system and available in the system's PATH. You can check if Java is installed by running:

```
bash

java -version
```

If Java is not recognized, you may need to install it or add it to your system's PATH environment variable.

5. Re-extract Solr Files

If none of the above steps work, it's possible that the `start.jar` file is missing or corrupted. In this case, re-extract the Solr `.tgz` or `.tar` file to a new directory to ensure the files are complete:

1. Delete the current Solr directory.
2. Re-download the Solr `.tgz` or `.tar` file.
3. Extract it again, ensuring that all files are extracted correctly.

6. Use Windows PowerShell or Command Prompt

Sometimes, there could be issues with the shell you're using. If you're running the command in **PowerShell**, try using **Command Prompt** (or vice versa) to see if that makes a difference.

Summary:

1. Verify that the `start.jar` file exists in the correct location.
2. Use the correct path to `start.jar` (either `bin/start.jar` or root directory).
3. Check your Java installation using `java -version`.
4. Re-extract Solr if the `start.jar` file is missing.

Let me know how it goes or if you run into any further issues!

C:\Users\SESPL\solr-9.7.0\server
i can see in this folder

If you see the `start.jar` file inside the `server` folder (e.g., `C:\Users\SESPL\solr-9.7.0\server\start.jar`), you need to adjust the path in your command accordingly.

Steps to Correct the Command:

Since the `start.jar` file is inside the `server` folder, you should run the following command:

```
bash

java -Dsolr.solr.home=.\server\solr -jar .\server\start.jar
```

Or, you can use the **full path** to the `start.jar` file:

```
bash

java -Dsolr.solr.home=C:\Users\SESPL\solr-9.7.0\server\solr -jar
C:\Users\SESPL\solr-9.7.0\server\start.jar
```

Explanation:

- `-Dsolr.solr.home=.\server\solr`: Specifies the Solr home directory, which is where Solr stores its configuration and data.
- `-jar .\server\start.jar`: Specifies the path to the `start.jar` file, which is inside the `server` folder.

Things to Check:

1. **Ensure Correct Directory:**
 - Open Command Prompt and navigate to the directory where `start.jar` is located:

```
bash
```

```
cd C:\Users\SESPL\solr-9.7.0
```

2. Verify the `start.jar` Path:

- Use `dir` to make sure that `start.jar` is indeed inside the `server` folder.

3. Java Version:

- Make sure that Java is installed correctly by running:

```
bash
```

```
java -version
```

- If Java isn't recognized, you'll need to install it or add it to the PATH.

Once you run the correct command, Solr should start up. You can then access the Solr Admin UI by navigating to:

```
bash
```

```
http://localhost:8983/solr
```

Let me know if you encounter any other issues!

```
C:\Users\SESPL\solr-9.7.0>java -
Dsolr.solr.home=C:\Users\SESPL\solr-9.7.0\server\solr -jar
C:\Users\SESPL\solr-9.7.0\server\start.jar
ERROR : No enabled jetty modules found!
INFO : ${jetty.home} = C:\Users\SESPL\solr-9.7.0\server
INFO : ${jetty.base} = C:\Users\SESPL\solr-9.7.0
ERROR : Please create and/or configure a ${jetty.base}
directory.
```

```
Usage: java -jar $JETTY_HOME/start.jar [options]
[properties] [configs]
    java -jar $JETTY_HOME/start.jar --help # for more
information
```

The error message `No enabled jetty modules found!` indicates that Solr is having trouble finding the necessary Jetty configuration files to start the server. Jetty is the web server that Solr uses to serve requests, and it requires some additional configuration to work properly.

Steps to Resolve:

1. **Check Solr Installation:** The issue might arise if the Solr installation is incomplete or corrupted. Ensure that you have extracted the full Solr package and that all necessary files are present.
2. **Use the Correct Command to Start Solr:** Solr uses Jetty as the web server, and it seems that Jetty is not properly configured. Try using the Solr `bin/solr` script (which sets up the environment properly) instead of directly running `start.jar`.

Instead of running:

```
bash

java -Dsolr.solr.home=C:\Users\SESPL\solr-9.7.0\server\solr -jar
C:\Users\SESPL\solr-9.7.0\server\start.jar
```

Use the `bin/solr` command to start Solr. For example, in Windows, you can use the following:

```
bash

C:\Users\SESPL\solr-9.7.0\bin\solr.cmd start
```

This script takes care of the Jetty setup and environment variables automatically.

3. **Verify Jetty Configuration:** Solr uses Jetty to serve web requests, and the error suggests that it is missing Jetty configurations. Usually, these configurations should be automatically included in the Solr installation.

Check the following:

- `solr-9.7.0\server\etc\jetty.xml` : This file contains Jetty's configuration. It should be present.
 - `solr-9.7.0\server\lib\jetty-*.jar` : These JAR files should be present to run Jetty.
4. **Check Permissions:** Sometimes, Windows permissions might cause issues with accessing files. Ensure that the user running Solr has sufficient permissions to access the necessary files.
 5. **Re-extract Solr:** If none of the above steps work, there might be a problem with your Solr installation. Try re-extracting the Solr `.tgz` or `.tar` file and ensure all files are extracted correctly. Make sure to follow the installation instructions properly:
 - Download the correct version.

- Use a reliable extraction tool like 7-Zip or WinRAR.

6. **Start Solr with Jetty Module Configuration:** If you need to manually configure the Jetty modules, you can create a `jetty.base` directory and configure it, but this is usually handled automatically.

For advanced configurations:

- Create a `jetty.base` directory inside the Solr directory.
- Copy necessary Jetty configuration files (like `jetty.xml`) into it.
- Update the `start.jar` command to point to the new `jetty.base` directory.

In Summary:

- Use `solr.cmd start` to start Solr (rather than using `start.jar` directly).
- Ensure that all necessary Solr files, including Jetty configurations, are present.
- Verify that you're using the correct Solr installation by checking the Solr home and directories.
- If the issue persists, try re-extracting Solr and reinstalling it.

Let me know if you continue to face issues!

```
C:\Users\SESPL\solr-9.7.0>C:\Users\SESPL\solr-9.7.0\bin\solr.cmd start
Java 17 detected. Enabled workaround for SOLR-16463
Java HotSpot(TM) 64-Bit Server VM warning: JVM cannot use large page memory because it does not have enough privilege to lock pages in memory.
WARNING: A command line option has enabled the Security Manager
WARNING: The Security Manager is deprecated and will be removed in a future release
Failed to parse command-line arguments due to:
Unrecognized option: --max-wait-secs

usage: bin/solr status [-maxWaitSecs <SECS>] [--solr-url <URL>]

List of options:
-maxWaitSecs <SECS>  Wait up to the specified number of
```


seconds to see Solr running.

`--solr-url <URL>` Address of the Solr Web application, defaults to: `http://localhost:8983`.

Please see the Reference Guide for more tools documentation:

<https://solr.apache.org/guide/solr/latest/deployment-guide/solr-control-script-reference.html>

ERROR: Solr did not start or was not reachable. Check the logs for errors.

The error message you're seeing indicates a few things:

1. Java Warning:

- The warning `JVM cannot use large page memory because it does not have enough privilege to lock pages in memory` is generally not critical but suggests that the JVM cannot use large memory pages, which is a performance optimization. This is often seen when running Solr on Windows without administrator privileges.

2. Unrecognized option: `--max-wait-secs`:

- The error `Unrecognized option: --max-wait-secs` is more critical. It suggests that there's an invalid option passed during the Solr startup process.

3. Solr Not Starting:

- `ERROR: Solr did not start or was not reachable` means that Solr failed to start up correctly.

Possible Solutions:

1. Check Solr Version Compatibility with Java 17:

- Solr 9.x is compatible with Java 11 and higher, but you should verify if the version you're using is fully compatible with Java 17.
- Solr 9.x might have specific configurations needed for Java 17, but the warning about `SOLR-16463` suggests that Solr has already enabled a workaround for that.

2. Fix `--max-wait-secs` Argument Issue:

- The error regarding `--max-wait-secs` indicates an invalid argument or an incompatible command-line option.

- Check the exact command you're running. For example, in some cases, the `solr status` command might be incorrectly invoked, or an older configuration file might be causing this issue. To start Solr, you should typically use:

```
bash

C:\Users\SESPL\solr-9.7.0\bin\solr.cmd start
```

- Do **not** manually add the `--max-wait-secs` argument unless explicitly required by your setup.

3. Check the Solr Logs:

- Solr logs can provide more information about why it is not starting. Check the Solr log file located in:

```
bash

C:\Users\SESPL\solr-9.7.0\server\logs
```

- Look for any exceptions or errors that could provide more insight into what's failing.

4. Run with Administrator Privileges:

- Given the warning about large page memory and the security manager, running Solr with administrator privileges can resolve potential permission issues.
- Right-click the **Command Prompt** or **PowerShell** window and select **Run as Administrator**, then try starting Solr again.

5. Verify Ports and Dependencies:

- Ensure that no other processes are using the port `8983`, which is the default Solr port. You can check this by running:

```
bash

netstat -ano | findstr :8983
```

- If another process is using the port, stop it or change the Solr port by modifying `solr.xml` or using the `-p` option when starting Solr.

6. Reinstall Solr:

- If none of the above work, you might have an incomplete or corrupted installation. Try re-downloading and extracting Solr again to make sure all files are intact.

Next Steps:

- **Run Solr with Administrator Privileges** to avoid permission issues.
- **Check the Solr logs** for further details on why it's not starting.
- **Check for conflicting command-line arguments** or configurations.

Let me know what you find in the logs or if you continue to face issues!

what is add in json file for add document

To add a document to Solr using JSON format, you need to format the document correctly in JSON syntax and send it to Solr's `/update` endpoint. Here's an example of how you can structure a JSON file to add documents to Solr.

Example of a JSON Document to Add to Solr:

1. **Basic Document Format:** Solr typically requires a structure where each document is wrapped in a `add` element, and each document is defined with fields and values.

```
json

{
  "add": {
    "doc": {
      "id": "1",
      "title": "Sample Document",
      "content": "This is a sample document content.",
      "author": "John Doe",
      "published_date": "2024-11-23T12:00:00Z"
    }
  }
}
```

2. **Multiple Documents:** If you want to add multiple documents in one request, you can pass them as an array of objects.

```
json

{
  "add": {
    "doc": [
      {
```

```

    "id": "1",
    "title": "Sample Document 1",
    "content": "This is the first document.",
    "author": "John Doe",
    "published_date": "2024-11-23T12:00:00Z"
  },
  {
    "id": "2",
    "title": "Sample Document 2",
    "content": "This is the second document.",
    "author": "Jane Smith",
    "published_date": "2024-11-24T12:00:00Z"
  }
]
}
}

```

3. **Sending the JSON to Solr:** After creating the JSON document, you need to send it to Solr's `/update` endpoint. You can use tools like `curl` or HTTP libraries to do this.

Example using `curl` to send a JSON document to Solr:

```
bash
```

```
curl http://localhost:8983/solr/mycore/update?commit=true -d @document.json -H
'Content-Type: application/json'
```

Where:

- `localhost:8983` is the address of your Solr instance.
- `mycore` is the Solr core you want to update.
- `document.json` is the JSON file containing the document(s) to be added.

Key Points:

- **Fields:** Each document in Solr can have multiple fields. Ensure that the field names you use match the field names defined in your Solr schema (`schema.xml` or dynamic fields).
- **Commit:** After adding the document(s), you may need to commit the changes by either using the `commit=true` query parameter (as shown above) or issuing a separate commit command to ensure the changes are persisted.

Example for `commit=true`:

This is used to immediately commit the changes (add the document) to the Solr index.

bash

```
curl http://localhost:8983/solr/mycore/update?commit=true -d '{"add": {"doc": {"id": "1", "title": "Sample Title", "content": "Sample content"}}}' -H 'Content-Type: application/json'
```

Troubleshooting:

- Ensure that your Solr core (`mycore`) is correctly set up with the appropriate fields.
- If you're using SolrCloud, you might need to specify the Solr collection name in the URL.
- Make sure your Solr instance is running before attempting to send the request.

Let me know if you need further clarification or run into any issues!

where to add my pdf file

To add a PDF file to Solr, the file must first be **indexed** by Solr. Solr doesn't index the content of a file directly by just uploading it; instead, you need to **extract the content** from the PDF and then send that extracted content to Solr in a structured format (like JSON).

Here's how you can index a PDF file into Solr:

Steps to Index a PDF File into Solr:

1. **Use Solr's Tika-based Data Import Handler (DIH):** Solr uses **Apache Tika** to extract content from different file types, including PDFs. You need to configure Solr to use the Data Import Handler to process the PDF and send the extracted content to Solr.
2. **Configure Tika in Solr:** Ensure that Solr is set up to use Tika for file content extraction. You may need to configure the `tika` field and create a data import configuration (if you want Solr to automatically extract content from files).

Basic Process for Adding a PDF File:

Option 1: Using Solr's ExtractingRequestHandler

1. **Send the PDF to Solr's ExtractingRequestHandler:** Solr has a built-in handler for extracting content from files, called the `extracting` request handler. You can send a PDF file to Solr for content extraction.

The endpoint is typically available at:

```
bash

http://localhost:8983/solr/<your_core>/update/extract
```

2. **Use cURL or HTTP Post:** Use `curl` or another HTTP tool to send the PDF to Solr. Here's an example using `curl` :

```
bash

curl http://localhost:8983/solr/mycore/update/extract -F
"file=@path_to_your_pdf_file.pdf" -F "literal.id=your_unique_document_id" -F
"commit=true"
```

Where:

- `path_to_your_pdf_file.pdf` is the path to your PDF file.
- `mycore` is your Solr core name.
- `your_unique_document_id` is a unique ID for the document you're indexing.
- `commit=true` ensures that the changes are committed immediately.

The `literal.id` field is a Solr field that you define to hold a unique identifier for the document. You can add other fields to capture metadata or additional information from the PDF as needed.

3. **Solr ExtractingRequestHandler Configuration:** Make sure your `solrconfig.xml` file includes the following for the extracting handler:

```
xml

<requestHandler name="/update/extract"
class="solr.extraction.ExtractingRequestHandler">
  <lst name="defaults">
    <str name="fmap.content">text</str>
  </lst>
</requestHandler>
```

This configuration tells Solr to map the extracted text content to the `text` field.

Option 2: Data Import Handler (DIH)

You can use Solr's **Data Import Handler (DIH)** to index PDFs by configuring DIH to use Tika for content extraction.

1. **Configure DIH in Solr:** You need to set up a DIH configuration file (`data-config.xml`) to specify how Solr should process the PDF files.

Example `data-config.xml` :

```
xml

<dataConfig>
  <document>
    <entity name="pdf_files" processor="FileListEntityProcessor"
baseDir="C:/path/to/pdf/files">
      <field column="id" source="fileName"/>
      <field column="content" source="content" />
    </entity>
  </document>
</dataConfig>
```

2. **Run DIH to Index PDFs:** Once your configuration is set up, you can use the following `curl` command to run the Data Import Handler:

```
bash

curl "http://localhost:8983/solr/mycore/dataimport?command=full-import"
```

This will import all PDFs from the specified directory (`baseDir`) and send their extracted content to Solr.

Final Notes:

- **Tika Extraction:** Tika is the content extraction library used by Solr to extract text and metadata from files like PDFs. Solr has built-in integration with Tika, so you don't have to worry about manually parsing the content.
- **Fields:** You will need a field in your Solr schema to store the extracted text (e.g., `text`). You can also define fields for metadata (e.g., `author` , `title` , etc.).
- **Commit:** Make sure to commit the changes after indexing the PDF content, or Solr may not display the indexed content immediately.

Example Schema Update:

In your `schema.xml` or through Solr's schema API, you might want to define a `text` field for storing extracted content from the PDF:

```
xml
```

```
<field name="text" type="text_general" indexed="true" stored="true"
multiValued="false"/>
<field name="id" type="string" indexed="true" stored="true" required="true"/>
```

This would allow Solr to store and index the extracted text and other metadata.

Let me know if you need further assistance with configuring Solr for PDF indexing!

i will give a document in fast api persimon app whre you need to use document to convert to text and then to json using solr

To implement the feature in your FastAPI application (Persimon App) where a document (like a PDF) is uploaded, its content is extracted and converted into JSON, and then indexed into Solr, you'll need to follow these steps:

Step-by-Step Implementation:

1. **Install Dependencies:** First, ensure you have the required libraries installed for extracting content from the document and interacting with Solr.

You'll need:

- `requests` : To interact with Solr.
- `python-magic` : To detect file types.
- `PyPDF2` or `pdfplumber` : If you are extracting text from PDFs (alternatively, you could use Solr's Tika for PDF extraction).

Install the dependencies using:

```
bash
```

```
pip install requests python-magic PyPDF2 pdfplumber
```

2. **Create FastAPI Endpoint for Document Upload:** You'll need an endpoint in your FastAPI application where users can upload a document. Once the document is uploaded, extract its content and convert it to JSON.

3. **Text Extraction from Document:** Depending on the file type (PDF, Word, etc.), you'll need to use different methods to extract text. Solr can extract text from PDFs using Tika, but here, I'll use a simpler approach for PDFs with **PyPDF2** or **pdfplumber**.
4. **Send Extracted Content to Solr:** Once the content is extracted, format it as JSON and send it to Solr using its **ExtractingRequestHandler**.

FastAPI Example Code:

python

```
from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel
import requests
import PyPDF2
import pdfplumber
import json

app = FastAPI()

# Define a pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str
    published_date: str

# Function to extract text from PDF using PyPDF2
def extract_text_from_pdf(pdf_file):
    try:
        pdf_reader = PyPDF2.PdfFileReader(pdf_file)
        text = ""
        for page_num in range(pdf_reader.numPages):
            page = pdf_reader.getPage(page_num)
            text += page.extract_text()
        return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None

# Function to extract text from PDF using pdfplumber (alternative)
```

```

def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
                text += page.extract_text()
            return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url = "http://localhost:8983/solr/mycore/update/extract"

    # Convert the document content to JSON
    json_data = {
        "add": {
            "doc": {
                "id": document.id,
                "title": document.title,
                "author": document.author,
                "content": document.content,
                "published_date": document.published_date
            }
        }
    }

    headers = {'Content-Type': 'application/json'}

    # Send the document content to Solr
    response = requests.post(solr_url, data=json.dumps(json_data), headers=headers)

    if response.status_code == 200:
        print("Document successfully indexed in Solr!")
    else:
        print(f"Error indexing document: {response.status_code} - {response.text}")

# Endpoint to upload PDF document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):

```

```

try:
    # Save the uploaded file temporarily
    file_location = f"temp_files/{file.filename}"
    with open(file_location, "wb") as f:
        f.write(await file.read())

    # Extract text from the uploaded PDF file
    with open(file_location, "rb") as f:
        text_content = extract_text_from_pdf_plumber(f) # Use PyPDF2 or
pdfplumber

    # Create the document JSON structure
    document = Document(
        id=file.filename,
        title=title,
        author=author,
        content=text_content,
        published_date=published_date
    )

    # Send the document to Solr
    send_to_solr(document)

    return {"message": "Document uploaded and indexed successfully!"}

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

```

Explanation:

1. Upload File Endpoint (/upload_document/):

- This endpoint allows the user to upload a PDF document. The uploaded file is saved temporarily.

2. Text Extraction:

- The `extract_text_from_pdf_plumber` function uses **pdfplumber** to extract the text from the uploaded PDF. You can use **PyPDF2** or any other library based on your requirements.

3. Creating Document:

- Once the text is extracted, we create a `Document` Pydantic model that includes metadata like `title`, `author`, `content`, and `published_date`.

4. Sending Data to Solr:

- After the document is prepared, it's sent to Solr using the `ExtractingRequestHandler`. We use `requests.post()` to send the document to Solr's `/update/extract` endpoint in JSON format.

5. Solr Configuration:

- Make sure your Solr core (`mycore`) is correctly configured to accept the `content` field (and other fields like `id`, `title`, etc.). This is done in Solr's `schema.xml`.

Notes:

- **Solr Core Setup:** Ensure your Solr core (`mycore`) is configured with fields like `id`, `title`, `author`, `content`, and `published_date`.
- **File Types:** You can extend this solution to support other file types (Word, Excel, etc.) by integrating additional libraries for content extraction, or by using Solr's **Data Import Handler** with Tika.
- **Error Handling:** Proper error handling is added to ensure that the application responds gracefully in case of any failures during text extraction or Solr indexing.

Testing:

- **Run the FastAPI app:**

```
bash

uvicorn app:app --reload
```

- **Test with cURL or Postman:**

- Upload a PDF via POST request to `/upload_document/` with the required metadata.

Let me know if you need further assistance with implementation!

are you using solr

Yes, the solution I provided assumes you're using **Solr** to index and store document content. Specifically, it's designed to send extracted content from uploaded PDF files to a Solr instance, where it can be indexed and searched.

In the provided example, the steps involve:

- Extracting text from a PDF file uploaded via the FastAPI application.
- Creating a document object that includes metadata like title, author, and content.
- Sending this document to Solr, which is running locally on `http://localhost:8983/solr/mycore`.

Solr is used to index the text and metadata so you can later search, filter, or retrieve the documents based on your queries.

If you are unsure about your Solr setup or need help integrating Solr, I can guide you through configuring Solr for your FastAPI app.

Would you like more detailed instructions on setting up Solr, or do you need assistance with any specific part of the integration?

```
SyntaxError: (unicode error) 'unicodeescape' codec can't
decode bytes in position 2-3: truncated \UXXXXXXXX escape
PS C:\Users\SESPL\bhargav-workspace\Day 34-35
exploratory da and persimon app\Persimon App>
```

The error you're seeing, `SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes`, typically occurs when you use a Windows-style file path in a Python script and the backslashes (`\`) are interpreted as escape sequences. In Python, backslashes are escape characters (for example, `\n` for newline, `\t` for tab), and this causes the issue.

How to Fix This:

You have a few options to resolve this issue:

1. **Use Raw Strings:** Prefix your string with `r` to tell Python to treat it as a raw string, where backslashes are not interpreted as escape characters. This is the easiest solution for file paths.

Example:

```
python

file_path = r"C:\Users\SESPL\bhargav-workspace\Day 34-35 exploratory da and
persimon app\Persimon App"
```

2. **Double Backslashes:** You can manually escape the backslashes by using double backslashes (`\\`).

Example:

```
python
```

```
file_path = "C:\\Users\\SESPL\\bhargav-workspace\\Day 34-35 exploratory da and  
persimon app\\Persimon App"
```

3. **Use Forward Slashes:** Python on Windows can also handle file paths with forward slashes (/), which avoids the escape sequence problem.

Example:

```
python
```

```
file_path = "C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da and  
persimon app/Persimon App"
```

Summary:

The root cause is the interpretation of backslashes in the file path. By using one of the methods above, you can fix this issue. I recommend using raw strings (r"...") for simplicity.

Let me know if you have any further questions!

```
file_path = r"C:\Users\SESPL\bhargav-workspace\Day 34-35  
exploratory da and persimon app\Persimon App\Solr.docx"
```

Your file path looks correct as a raw string (r"..."), but if you're still encountering issues, it might be due to spaces or other special characters in the path. Here's a checklist to make sure everything is properly set up:

1. **Ensure the File Path is Correct:** Verify that the file exists at the specified location (C:\Users\SESPL\bhargav-workspace\Day 34-35 exploratory da and persimon app\Persimon App\Solr.docx). You can do this by checking the folder manually or printing the file path in your script to confirm.
2. **Use Raw String Correctly:** You've already done this by using the r before the string, which tells Python to interpret the string as a raw string and prevent escape sequence issues. This part looks good.

Example:

```
python
```

```
file_path = r"C:\Users\SESPL\bhargav-workspace\Day 34-35 exploratory da and  
persimon app\Persimon App\Solr.docx"
```

3. **Check for File Access Permissions:** Ensure that your script has permission to access the file. Sometimes, files in specific directories require special permissions.
4. **Test File Path:** If you're still running into issues, try printing the file path in your script to ensure it's what you expect.

```
python  
  
print(file_path)  # This will print the raw string path
```

5. **Check for File Existence:** Before accessing the file, you can use Python's `os` module to check if the file exists.

```
python  
  
import os  
  
if os.path.exists(file_path):  
    print("File found")  
else:  
    print("File not found")
```

6. **Potential Issues with the File:** If you're trying to open or process a `.docx` file (Microsoft Word file), ensure you're using the correct libraries to handle `.docx` files (e.g., `python-docx` or `docx2txt`), as the content extraction from `.docx` files is different from PDFs.

Example for reading `.docx` file:

If you're working with a `.docx` file, here's how you might extract text from it using the `python-docx` library:

1. Install the library:

```
bash  
  
pip install python-docx
```

2. Extract text from the `.docx` file:

```
python
```

```

from docx import Document

# Load the document
doc = Document(file_path)

# Extract text
full_text = []
for para in doc.paragraphs:
    full_text.append(para.text)

# Join paragraphs with a newline
text_content = '\n'.join(full_text)
print(text_content)

```

This will extract and print the text from the `.docx` file. If you want to send it to Solr, you can then proceed to create a document with this text and send it to Solr using the method described earlier.

Let me know if you face any further issues!

Response body

Download

```

{
  "error": "An error occurred: [Errno 2] No such file or
directory: 'C:/Users/SESPL/bhagav-workspace/Day 34-35
exploratory da and persimon app/Persimon App/Solr.docx'"
}

```

The error message `"No such file or directory: 'C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da and persimon app/Persimon App/Solr.docx'"` indicates that the file path you're providing does not match the actual location of the file, or there might be an issue with the way the path is written or interpreted.

Here's how to troubleshoot and resolve this issue:

1. Verify the File Path

Make sure the file exists at the specified location (`C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da and persimon app/Persimon App/Solr.docx`).

- Open File Explorer and navigate to this path.

- Ensure that the file `Solr.docx` is located in that directory.

2. Test the Path Programmatically

You can use Python's `os.path.exists()` function to check if the file exists at the given path.

```
python

import os

file_path = r"C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da and persimon app/Persimon App/Solr.docx"

if os.path.exists(file_path):
    print("File exists!")
else:
    print("File not found.")
```

This will confirm whether the path is correct or not. If it prints `"File not found"`, the issue is with the path itself.

3. Check for Typos in the Path

Sometimes, spaces or typos in the folder or file names can cause the file path to be incorrect. Double-check the path for any discrepancies:

- Ensure that there are no extra spaces, especially at the beginning or end of the file name or directory names.
- Ensure the file name is exactly `Solr.docx` (case-sensitive).

4. Handle File Access Issues (Permissions)

Ensure your script has permission to access the file. You may want to test this by trying to manually open the file from the same location using a Python script:

```
python

try:
    with open(file_path, "rb") as file:
        print("File opened successfully!")
except Exception as e:
    print(f"Error opening file: {e}")
```

If the file opens successfully, this confirms that the path and permissions are correct. If it throws an error, it could be a permissions issue.

5. Correct Usage of Raw String for Path

Ensure you're using the `r""` notation for raw strings when defining file paths on Windows. This avoids interpreting backslashes (`\`) as escape characters.

Example:

```
python

file_path = r"C:\Users\SESPL\bhargav-workspace\Day 34-35 exploratory da and persimon
app\Persimon App\Solr.docx"
```

6. Testing with Different Path Formats

You can also try using forward slashes instead of backslashes in the file path.

Example:

```
python

file_path = "C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da and persimon
app/Persimon App/Solr.docx"
```

7. Path Length Limitation (if applicable)

Windows has a limit on file path length (260 characters), which can sometimes cause issues if the path is too long. To check if this is the case, you can shorten the path temporarily and test again.

8. Alternative: Use a File Upload in FastAPI

If you are uploading the file through an API request, ensure that the file is being properly saved before accessing it. The file might not be saved correctly on the server.

For example, when using FastAPI to upload files:

```
python

from fastapi import FastAPI, File, UploadFile

app = FastAPI()
```

```

@app.post("/upload/")
async def upload_file(file: UploadFile = File(...)):
    try:
        file_location = f"temp_files/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())
        return {"filename": file.filename}
    except Exception as e:
        return {"error": str(e)}

```

This ensures that the file is saved in a known location before you attempt to access it. Once the file is saved, you can proceed to extract its content.

Next Steps:

1. **Check if the file exists** with `os.path.exists()` and print the result.
2. **Fix any path issues** and confirm the file is accessible.
3. **Test file upload** if applicable.

Let me know if any of these steps help or if you encounter any other issues!

```

from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel
import requests
import PyPDF2
import pdfplumber
import json

app = FastAPI()

# Define a pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str
    published_date: str

# Function to extract text from PDF using PyPDF2
def extract_text_from_pdf(pdf_file):
    try:

```

```

pdf_reader = PyPDF2.PdfFileReader(pdf_file)
text = ""
for page_num in range(pdf_reader.numPages):
    page = pdf_reader.getPage(page_num)
    text += page.extract_text()
return text
except Exception as e:
    print(f"Error extracting text from PDF: {e}")
    return None

# Function to extract text from PDF using pdfplumber
(alternative)
def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
                text += page.extract_text()
            return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url =
"http://localhost:8983/solr/mycore/update/extract"

    # Convert the document content to JSON
    json_data = {
        "add": {
            "doc": {
                "id": document.id,
                "title": document.title,
                "author": document.author,
                "content": document.content,
                "published_date": document.published_date
            }
        }
    }

    headers = {'Content-Type': 'application/json'}

```

```

# Send the document content to Solr
response = requests.post(solr_url,
data=json.dumps(json_data), headers=headers)

if response.status_code == 200:
    print("Document successfully indexed in Solr!")
else:
    print(f"Error indexing document:
{response.status_code} - {response.text}")

# Endpoint to upload PDF document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title:
str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"C:/Users/SESPL/bhagav-workspace/Day
34-35 exploratory da and persimon app/Persimon
App/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded PDF file
        with open(file_location, "rb") as f:
            text_content = extract_text_from_pdf_plumber(f) #
Use PyPDF2 or pdfplumber

        # Create the document JSON structure
        document = Document(
            id=file.filename,
            title=title,
            author=author,
            content=text_content,
            published_date=published_date
        )

        # Send the document to Solr
        send_to_solr(document)

    return {"message": "Document uploaded and indexed
successfully!"}

```

```
except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}
check my code
```

Your code looks well-structured, but I see a few areas that might need attention to ensure smooth execution. Here's a review with some suggestions and fixes:

Review & Suggestions:

1. File Path Issues:

- Ensure that `file_location` is a valid path. As previously discussed, it's always a good idea to use forward slashes (`/`) or raw string literals (`r"..."`) for Windows file paths. However, since you are saving the file, make sure the path is valid and the directories exist.

2. Handling PDF Extraction:

- The `extract_text_from_pdf` and `extract_text_from_pdf_plumber` functions are designed to handle PDF extraction. You're currently using `pdfplumber` in your upload endpoint, which is good for more accurate extraction from PDFs with tables or complex layouts.
- Ensure that both PyPDF2 and pdfplumber are installed. You can install them using:

```
bash

pip install PyPDF2 pdfplumber
```

3. Solr Update API:

- Solr has a specific method for handling document extraction via the `extract` API, which you are using correctly. However, make sure that Solr is configured to accept `application/json` requests for document indexing.

4. Solr Core:

- You are posting to Solr at `http://localhost:8983/solr/mycore/update/extract`. Make sure that the Solr core (`mycore`) exists and is configured to index documents.

5. File Cleanup:

- Consider deleting the temporary file after processing it to avoid unnecessary accumulation of files.

6. Additional Considerations:

- If you're expecting large PDFs, consider adding error handling for oversized files or timeouts.

Modified Version of the Code:

Here are some changes and fixes based on the above review:

```
python

from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel
import requests
import PyPDF2
import pdfplumber
import json
import os

app = FastAPI()

# Define a pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str
    published_date: str

# Function to extract text from PDF using PyPDF2
def extract_text_from_pdf(pdf_file):
    try:
        pdf_reader = PyPDF2.PdfFileReader(pdf_file)
        text = ""
        for page_num in range(pdf_reader.numPages):
            page = pdf_reader.getPage(page_num)
            text += page.extract_text()
        return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None

# Function to extract text from PDF using pdfplumber (alternative)
def extract_text_from_pdf_plumber(pdf_file):
    try:
```

```

    with pdfplumber.open(pdf_file) as pdf:
        text = ""
        for page in pdf.pages:
            text += page.extract_text()
    return text
except Exception as e:
    print(f"Error extracting text from PDF: {e}")
    return None

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url = "http://localhost:8983/solr/mycore/update/extract"

    # Convert the document content to JSON
    json_data = {
        "add": {
            "doc": {
                "id": document.id,
                "title": document.title,
                "author": document.author,
                "content": document.content,
                "published_date": document.published_date
            }
        }
    }

    headers = {'Content-Type': 'application/json'}

    # Send the document content to Solr
    response = requests.post(solr_url, data=json.dumps(json_data), headers=headers)

    if response.status_code == 200:
        print("Document successfully indexed in Solr!")
    else:
        print(f"Error indexing document: {response.status_code} - {response.text}")

# Endpoint to upload PDF document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily

```



```

file_location = f"C:/Users/SESPL/bhagav-workspace/Day 34-35 exploratory da
and persimon app/Persimon App/{file.filename}"
with open(file_location, "wb") as f:
    f.write(await file.read())

# Extract text from the uploaded PDF file
with open(file_location, "rb") as f:
    text_content = extract_text_from_pdf_plumber(f) # Use pdfplumber

# Create the document JSON structure
document = Document(
    id=file.filename,
    title=title,
    author=author,
    content=text_content,
    published_date=published_date
)

# Send the document to Solr
send_to_solr(document)

# Optionally delete the temporary file after processing
os.remove(file_location)

return {"message": "Document uploaded and indexed successfully!"}

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

```

Key Fixes:

1. **Temporary File Cleanup:** After processing the file, it deletes the temporary file using `os.remove(file_location)`.
2. **Ensure Proper Solr Core:** Make sure that the Solr core (`mycore`) is properly set up and ready to accept documents.
3. **Error Handling:** Added better exception handling and printed error messages for easier debugging.
4. **Extract Text:** Uses `pdfplumber` to extract text, but you can switch to `PyPDF2` if preferred.

5. **Ensure Solr Availability:** If Solr is running on a different port or host, adjust the `solr_url`.

Next Steps:

1. Ensure that Solr is running locally and accessible at `http://localhost:8983/solr/mycore`.
2. Test the file upload and Solr indexing by sending a POST request with a PDF file to `/upload_document/`.
3. If you encounter errors while posting to Solr, check the Solr logs for more details and confirm that the core `mycore` exists and is correctly configured.

Let me know if you face any further issues!

do i need to install solar in server

Yes, you need to install and run Apache Solr on a server (either your local machine or a remote server) for your FastAPI application to interact with it.

Here are the basic steps to install Solr on a server:

1. Download and Install Solr:

- You need to download the Solr binary from the [official website](#).

For instance, on a Linux machine, you can use the following steps:

- Download the Solr archive file:

```
bash

wget https://archive.apache.org/dist/lucene/solr/9.7.0/solr-9.7.0.tgz
```

- Extract the archive:

```
bash

tar xvf solr-9.7.0.tgz
```

- Install Solr:

```
bash
```

```
cd solr-9.7.0
sudo ./bin/solr start
```

For Windows, as mentioned earlier, ensure you have a JDK installed and then use the `start.jar` or `solr.cmd` to start Solr.

2. Run Solr:

After installation, you can start Solr as follows (for example):

- On Linux/macOS:

```
bash

./bin/solr start
```

- On Windows (run `solr.cmd`):

```
bash

solr.cmd start
```

Solr will now be running on `http://localhost:8983` by default.

3. Create a Solr Core:

Solr stores documents in *cores*, which are like databases in Solr. You need to create a core to store documents. You can create one via the Solr command:

```
bash

./bin/solr create -c mycore
```

- This will create a Solr core named `mycore`. Make sure to replace `mycore` with your desired core name.

After creating the core, you can start adding documents to it.

4. Configure Solr for FastAPI:

Ensure your FastAPI application is pointing to the correct Solr core URL. For example:

```
python
```

```
solr_url = "http://localhost:8983/solr/mycore/update/extract"
```

Make sure that `mycore` is the Solr core you've created.

5. Testing:

Once Solr is running, test that it's working by navigating to Solr's admin interface at `http://localhost:8983/solr/`. You can check the status of your core and make sure Solr is running properly.

6. Solr as a Service (Optional):

On a production server, you might want to configure Solr to run as a service, so it automatically starts when the server is rebooted. You can refer to the [Solr Service Setup documentation](#) for more details.

Conclusion:

- **Yes, you need to install and run Solr** either on your local machine or a dedicated server for your FastAPI app to interact with it.
- Once Solr is installed and running, you can configure your FastAPI app to send documents to Solr via the appropriate API.

Let me know if you need more specific help with the installation process or configuring Solr!

```
C:\Users\SESPL\solr-9.7.0>solr.cmd start
'solr.cmd' is not recognized as an internal or external
command,
operable program or batch file.
```

The error message `'solr.cmd' is not recognized as an internal or external command` suggests that the Solr executable is not in your system's PATH or you're running the command in the wrong directory.

Here are the steps you can follow to fix the issue:

1. Navigate to the Solr Directory:

You need to make sure you're running the command from the Solr directory where `solr.cmd` is located. Typically, the Solr binary scripts are located in the `bin` folder of the Solr installation.

Follow these steps:

1. **Open Command Prompt.**
2. **Navigate to the Solr `bin` directory:**

```
bash  
  
cd C:\Users\SESPL\solr-9.7.0\bin
```

This ensures you're in the directory where `solr.cmd` is located.

2. Run Solr from the `bin` Directory:

Once you're inside the `bin` directory, try running the following command:

```
bash  
  
solr.cmd start
```

3. If the Command Still Doesn't Work:

- **Check if `solr.cmd` exists:** Open the `bin` folder (`C:\Users\SESPL\solr-9.7.0\bin`) and confirm that the `solr.cmd` file is present. If it is missing, the Solr installation may be incomplete, and you may need to re-download Solr.
- **Use the Full Path:** If `solr.cmd` exists in the `bin` directory but you're still unable to run it, try using the full path:

```
bash  
  
C:\Users\SESPL\solr-9.7.0\bin\solr.cmd start
```

4. Check `JAVA_HOME`:

Solr requires Java to be installed and properly configured. Ensure that your Java environment is set up correctly.

- **Verify Java is installed:** Run this command to check if Java is installed and its version:

```
bash  
  
java -version
```

- **Set JAVA_HOME environment variable:** Make sure that the `JAVA_HOME` environment variable is set to the Java installation directory. For example:

```
bash
```

```
set JAVA_HOME=C:\Program Files\Java\jdk-17
```

5. Solr Installation Issues:

If you're still unable to run Solr after trying the above steps, I recommend re-downloading and extracting Solr:

- Delete the current Solr folder.
- Re-download the Solr binary from the official website and re-extract it.

6. Windows-Specific Troubleshooting:

If you continue to face issues, ensure your command prompt is running with **Administrator privileges**. Right-click on Command Prompt and select **Run as Administrator**.

After following these steps, you should be able to start Solr with `solr.cmd start`. Let me know if you encounter further issues!

```
Java 17 detected. Enabled workaround for SOLR-16463
Java HotSpot(TM) 64-Bit Server VM warning: JVM cannot use
large page memory because it does not have enough
privilege to lock pages in memory.
WARNING: A command line option has enabled the Security
Manager
WARNING: The Security Manager is deprecated and will be
removed in a future release
Failed to parse command-line arguments due to:
Unrecognized option: --max-wait-secs

usage: bin/solr status [-maxWaitSecs <SECS>] [--solr-url
<URL>]

List of options:
-maxWaitSecs <SECS>  Wait up to the specified number of
```

seconds to see Solr running.

`--solr-url <URL>` Address of the Solr Web application, defaults to: `http://localhost:8983`.

Please see the Reference Guide for more tools documentation:

<https://solr.apache.org/guide/solr/latest/deployment-guide/solr-control-script-reference.html>

ERROR: Solr did not start or was not reachable. Check the logs for errors.

The error message you're seeing indicates a couple of issues related to Solr and Java. Here's a breakdown of the problems and how to fix them:

1. Java 17 Warning and Security Manager:

- The warning about `Security Manager` being deprecated in Java 17 is informational. It indicates that Solr uses the Java Security Manager, which is deprecated and will be removed in future versions of Java.
- **No immediate action is required** unless you need to address this for a production environment. You can ignore this warning for now.

2. Unrecognized Option: `--max-wait-secs`:

This error seems to be related to a command-line option that is not recognized in the version of Solr you are using. This typically happens when there's an invalid or outdated configuration in the Solr startup command.

Solution Steps:

1. Correcting the Command to Start Solr:

The `--max-wait-secs` option in the error seems to be an invalid or misconfigured option. Ensure that you're using the correct command to start Solr. Typically, you should use:

```
bash
```

```
solr.cmd start
```

Do not add `--max-wait-secs` or other options unless explicitly required for your specific configuration.

2. Check Solr Log Files:

To understand more about why Solr is not starting, you can check the log files. The Solr log files are typically located in the `logs` directory within your Solr installation directory:

- **Windows:** `C:\Users\SESPL\solr-9.7.0\server\logs`

Look for any errors in the log files (like `solr.log` or `solr.out`) to pinpoint what might be causing the issue.

3. Check the Java Version:

- Solr 9.7.0 requires Java 8 or newer. Since you're using Java 17, Solr should be compatible, but it's important to ensure that your `JAVA_HOME` environment variable is correctly set.
- If the issue persists with Java 17, try using Java 11 or 8 if possible, as these are commonly recommended versions for Solr 9.x.

4. Solr Core Not Starting:

Sometimes Solr fails to start because it can't initialize the core. Try creating a core manually before starting Solr:

```
bash

solr.cmd create_core -c mycore
```

This will create a core named `mycore` that Solr can use. After that, run:

```
bash

solr.cmd start
```

5. System Permissions:

- **Large Page Memory Warning:** This is a JVM warning that can happen on Windows when Solr is trying to use large memory pages but lacks sufficient privileges.

If you are not running Solr with sufficient permissions, it can be avoided by running the command as an **Administrator**. Right-click on Command Prompt and select **Run as Administrator** before trying to start Solr.

Conclusion:

- **Ignore the Security Manager warning** unless it affects your use case.

- Remove or fix the `--max-wait-secs` option from the startup command.
- Check the logs for specific errors related to the startup.
- Ensure Solr is correctly configured and the Java version is compatible.
- Manually create a Solr core if necessary.

Once you've fixed these issues, try starting Solr again and see if it works. Let me know if you need further assistance!

```
C:\Users\SESPL\solr-9.7.0\bin>solr.cmd create_core -c
mycore
WARNING: Using _default configset with data driven
schema functionality. NOT RECOMMENDED for production
use.
    To turn off: bin\solr config -c mycore -p 8983 -action
set-user-property -property update.autoCreateFields -value
false
create_core is not a valid command!
```

The error message `'create_core is not a valid command!'` suggests that the `solr.cmd` script is not recognizing the `create_core` command properly. This could be due to a few reasons, including the command structure, or an issue with the Solr version you're using.

Here's how to resolve this and correctly create a core in Solr:

1. Check Solr Command Syntax:

You should be using the correct command to create a core. For Solr 9.7.0, the command to create a core is typically:

```
bash

solr.cmd create -c mycore
```

Ensure you are in the `bin` directory and use the correct command to create the core (`create` instead of `create_core`).

2. Make Sure Solr is Running:

Before creating a core, make sure Solr is running. If Solr isn't started, the core creation command will fail. You can start Solr with:

```
bash
```

```
solr.cmd start
```

Then try running the `create` command again:

```
bash
```

```
solr.cmd create -c mycore
```

3. Correct the Config Set Warning:

The warning you're seeing about `_default configset` is fine for development, but not recommended for production. If you want to turn off automatic field creation in your schema (which can prevent unwanted fields), you can execute the following command:

```
bash
```

```
bin\solr config -c mycore -p 8983 -action set-user-property -property  
update.autoCreateFields -value false
```

This is optional and only necessary for production-grade configuration.

4. Use Full Path:

If you continue to encounter issues, try using the full path to the Solr executable and explicitly specify the core creation:

```
bash
```

```
C:\Users\SESPL\solr-9.7.0\bin\solr.cmd create -c mycore
```

5. Creating a Solr Core with a Specific Config:

If you want to use a custom configuration instead of the default, you can specify the configuration directory as part of the `create` command:

```
bash
```

```
solr.cmd create -c mycore -d <path-to-config-set>
```

Replace `<path-to-config-set>` with the location of your custom config set, if any.

Once Solr is running, and the core is successfully created, you should see the `mycore` core listed when you check Solr's admin panel (`http://localhost:8983/solr`).

Let me know if you need further clarification or if the problem persists!

give me what to add in swagger

To add the functionality for uploading documents and indexing them in Solr through Swagger (using FastAPI), you need to ensure that the relevant endpoints are properly defined and annotated in your FastAPI application. FastAPI automatically generates the Swagger UI documentation based on the defined routes and models.

Here's an updated version of your FastAPI application with correct Swagger annotations for uploading documents:

FastAPI Code with Swagger Annotations

python

```
from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel
import requests
import pdfplumber
import json
from typing import List

app = FastAPI()

# Define a Pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str
    published_date: str

# Function to extract text from PDF using pdfplumber (alternative method)
def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
```

```

        text += page.extract_text()
    return text
except Exception as e:
    print(f"Error extracting text from PDF: {e}")
    return None

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url = "http://localhost:8983/solr/mycore/update/extract"

    # Convert the document content to JSON
    json_data = {
        "add": {
            "doc": {
                "id": document.id,
                "title": document.title,
                "author": document.author,
                "content": document.content,
                "published_date": document.published_date
            }
        }
    }

    headers = {'Content-Type': 'application/json'}

    # Send the document content to Solr
    response = requests.post(solr_url, data=json.dumps(json_data), headers=headers)

    if response.status_code == 200:
        print("Document successfully indexed in Solr!")
    else:
        print(f"Error indexing document: {response.status_code} - {response.text}")

# Endpoint to upload a document and index it in Solr
@app.post("/upload_document/", summary="Upload and index document in Solr",
description="This endpoint allows users to upload a document (PDF) and its metadata, which will be indexed in Solr.")
async def upload_document(
    file: UploadFile = File(..., description="The document to be uploaded"),
    title: str = "",
    author: str = "",
    published_date: str = ""

```

```

):
    """
    Upload a document, extract text from it, and index it in Solr.

    - file: The document file (PDF).
    - title: Title of the document (optional).
    - author: Author of the document (optional).
    - published_date: Published date of the document (optional).
    """
    try:
        # Save the uploaded file temporarily
        file_location = f"C:/Users/SESPL/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded document
        with open(file_location, "rb") as f:
            text_content = extract_text_from_pdf_plumber(f)

        # Create the document JSON structure
        document = Document(
            id=file.filename,
            title=title,
            author=author,
            content=text_content,
            published_date=published_date
        )

        # Send the document to Solr for indexing
        send_to_solr(document)

        return {"message": "Document uploaded and indexed successfully!"}

    except Exception as e:
        return {"error": f"An error occurred: {str(e)}"}

```

Key Points for Swagger Documentation:

1. Endpoint Summary and Description:

- The `@app.post` decorator now includes a `summary` and `description` to enhance Swagger UI. These fields are displayed in the Swagger interface to explain what the

endpoint does.

Example:

```
python
```

```
@app.post("/upload_document/", summary="Upload and index document in Solr",  
description="This endpoint allows users to upload a document (PDF) and its  
metadata, which will be indexed in Solr.")
```

2. Parameter Descriptions:

- The `UploadFile` parameter `file` is annotated with a description using `File(..., description="The document to be uploaded")` to describe the file parameter in Swagger.
- Other parameters (`title`, `author`, `published_date`) are optional and can be described to indicate they are part of the document metadata.

3. Endpoint Documentation:

- The docstring inside the `upload_document` function provides additional documentation about the parameters expected in the request. This is shown in the Swagger UI for better understanding.

Example:

```
python
```

```
"""  
Upload a document, extract text from it, and index it in Solr.  
  
- **file**: The document file (PDF).  
- **title**: Title of the document (optional).  
- **author**: Author of the document (optional).  
- **published_date**: Published date of the document (optional).  
"""
```

Swagger UI:

Once you run your FastAPI app, you can access the Swagger UI by navigating to:

```
bash
```

`http://localhost:8000/docs`

In this Swagger UI, you will see:

- **POST /upload_document/**: An interactive form where you can upload the document (PDF file) along with metadata fields like `title`, `author`, and `published_date`.
- You can test the functionality of uploading the document and see the response message directly in the UI.

Final Notes:

- Make sure Solr is running and correctly configured for indexing the documents.
- Ensure that your FastAPI app is correctly serving at `http://localhost:8000`, and that the endpoint is accessible.
- You can also test the **POST** endpoint directly from Swagger UI after launching the FastAPI server.

This setup will provide a clean and easy-to-use API with automated documentation through Swagger. Let me know if you need further help!

```
{
  "error": "An error occurred: 1 validation error for
Document\ncontent\n Input should be a valid string
[type=string_type, input_value=None,
input_type=NoneType]\n For further information visit
https://errors.pydantic.dev/2.10/v/string_type"
}
```

The error you're encountering indicates that the `content` field in the `Document` model is receiving `None` instead of a valid string. This happens when the PDF file's text extraction function (`extract_text_from_pdf_plumber`) fails or returns `None`, and the `content` field in the `Document` model is not getting a valid string.

Solution:

1. **Check PDF Text Extraction:** Ensure that the text extraction functions (`extract_text_from_pdf_plumber`) are correctly extracting text. If the function fails, you need to handle it more gracefully.

2. **Ensure content is always a valid string:** You should ensure that the `content` is always a valid string, even if the extraction process fails.

Updated Code to Handle Errors and Provide Default Values:

python

```
from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel, Field
import requests
import pdfplumber
import json
from typing import Optional

app = FastAPI()

# Define a Pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str = Field(..., min_length=1) # Ensure content is always a valid
string
    published_date: Optional[str] = None # Optional field for published_date

# Function to extract text from PDF using pdfplumber
def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
                text += page.extract_text() or "" # Ensure text is never None
            return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return "" # Return empty string if extraction fails

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url = "http://localhost:8983/solr/mycore/update/extract"

    # Convert the document content to JSON
```



```

json_data = {
    "add": {
        "doc": {
            "id": document.id,
            "title": document.title,
            "author": document.author,
            "content": document.content,
            "published_date": document.published_date
        }
    }
}

headers = {'Content-Type': 'application/json'}

# Send the document content to Solr
response = requests.post(solr_url, data=json.dumps(json_data), headers=headers)

if response.status_code == 200:
    print("Document successfully indexed in Solr!")
else:
    print(f"Error indexing document: {response.status_code} - {response.text}")

# Endpoint to upload a document and index it in Solr
@app.post("/upload_document/", summary="Upload and index document in Solr",
description="This endpoint allows users to upload a document (PDF) and its metadata,
which will be indexed in Solr.")
async def upload_document(
    file: UploadFile = File(..., description="The document to be uploaded"),
    title: str = "",
    author: str = "",
    published_date: str = ""
):
    """
    Upload a document, extract text from it, and index it in Solr.

    - **file**: The document file (PDF).
    - **title**: Title of the document (optional).
    - **author**: Author of the document (optional).
    - **published_date**: Published date of the document (optional).
    """
    try:
        # Save the uploaded file temporarily

```

```

file_location = f"C:/Users/SESPL/{file.filename}"
with open(file_location, "wb") as f:
    f.write(await file.read())

# Extract text from the uploaded document
with open(file_location, "rb") as f:
    text_content = extract_text_from_pdf_plumber(f)

# Ensure content is not empty (set a fallback if needed)
if not text_content:
    text_content = "No content extracted from the document."

# Create the document JSON structure
document = Document(
    id=file.filename,
    title=title,
    author=author,
    content=text_content,
    published_date=published_date
)

# Send the document to Solr for indexing
send_to_solr(document)

return {"message": "Document uploaded and indexed successfully!"}

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

```

Key Updates:

1. Default Value for `content` Field:

- I updated the `content` field in the `Document` model to ensure it is always a valid string. It now uses the `Field(..., min_length=1)` annotation to enforce that it's never empty.

2. Handling Empty Text Extraction:

- If the PDF text extraction fails or returns `None`, I explicitly return an empty string `""` instead of `None` inside the `extract_text_from_pdf_plumber` function.

3. Fallback Content for Missing Text:

- If the extracted text is empty or not valid, the `content` field will default to `"No content extracted from the document."`.

Now:

- **Validation Error:** The error should no longer occur because the `content` field will always have a valid string, even if text extraction fails.
- **Swagger UI:** You can now test uploading the document in Swagger UI and check if it successfully processes the file and indexes it in Solr.

Try running this updated version of your FastAPI app, and it should solve the issue! Let me know if any new errors appear.

indexed successfully

Great to hear that the document was indexed successfully! ☑

Now, your document is available in Solr, and you can use Solr to search and retrieve it based on its indexed content. Here are a few next steps you might want to take:

1. Search Documents in Solr:

You can use Solr's search endpoint to query the indexed documents. For example, you can search for documents by title, author, or any other field you indexed. Here's how you can do it using the Solr query URL:

- **Basic Solr Search URL:**

```
bash
```

```
http://localhost:8983/solr/mycore/select?q=title:"YourTitle"
```

Replace `"YourTitle"` with the actual title you're searching for. You can also use other fields (like `author`, `content`, etc.) in the query.

2. Integrating Solr Search in FastAPI:

You can create a new endpoint in your FastAPI app to search documents from Solr. Here's an example:

```
python
```

```

import requests

@app.get("/search_document/")
async def search_document(query: str):
    solr_url = f"http://localhost:8983/solr/mycore/select?q={query}"

    response = requests.get(solr_url)

    if response.status_code == 200:
        return response.json()
    else:
        return {"error": f"Failed to search Solr: {response.status_code}"}

```

3. Monitor Solr:

You can check the Solr admin UI at `http://localhost:8983/solr` to view the core (`mycore`), and you can see how many documents are indexed, search them, and manage the core settings.

4. Improve Document Indexing:

Consider adding more metadata to your documents or other fields that will help with more effective searching. You can also fine-tune your Solr schema to make full-text searching more efficient.

If you have any more questions or want to expand functionality, feel free to ask!

```

http://localhost:8983/solr/mycore/select?q=title:Solr
{
  "responseHeader": {
    "status": 400,
    "QTime": 0,
    "params": {
      "q": "title:Solr"
    }
  },
  "error": {
    "metadata": [
      "error-class",
      "org.apache.solr.common.SolrException",
      "root-error-class",
      "org.apache.solr.common.SolrException"
    ]
  }
}

```

```
],  
  "msg": "undefined field title",  
  "code": 400  
}  
}
```

The error you're seeing, `undefined field title`, indicates that Solr does not recognize the `title` field in your Solr schema. This typically happens when the schema for your Solr core (`mycore`) does not include the `title` field, or it is not defined correctly.

To resolve this issue, follow these steps:

1. Define the Field in the Solr Schema:

You need to define the `title` field in your Solr schema, so Solr knows how to index and query it.

Option 1: Use the Managed Schema

If you're using Solr's managed schema (which is the default), you need to ensure that the `title` field is included in your schema. Here's how:

1. Navigate to the **Solr Admin UI** at `http://localhost:8983/solr/`.
2. Select your core (`mycore`).
3. Go to the **Schema** tab and click on **Add Field**.
4. Add the field definition for the `title` field:

```
xml  
  
<field name="title" type="string" indexed="true" stored="true"/>
```

This ensures that the `title` field will be indexed and stored.

5. After saving, **reload** the core to apply the changes.

Option 2: Modify `schema.xml` Manually

If you're not using Solr's managed schema, you can modify the `schema.xml` directly in your core's configuration files:

1. Go to the `mycore` directory, and find the `schema.xml` file in `server/solr/mycore/conf/`.
2. Add the field definition for `title` inside the `<fields>` section:

```
xml
```

```
<field name="title" type="string" indexed="true" stored="true"/>
```

3. Save the file and reload the core.

2. Reload Solr Core:

Once you've added the `title` field to your schema, you need to reload the core so that Solr picks up the changes.

- You can reload the core via the Solr Admin UI or by running the following Solr command:

```
bash

bin/solr restart -c mycore
```

3. Re-index the Documents:

After modifying the schema, you need to re-index any documents that have already been uploaded. This ensures that the documents include the `title` field in the index.

4. Retry the Query:

Once the `title` field is correctly defined and the core is reloaded, retry the query:

```
bash

http://localhost:8983/solr/mycore/select?q=title:Solr
```

It should work now, returning documents that have the `title` field with the value "Solr".

If you want to avoid manually modifying the schema, you could also use Solr's **dynamic fields** to automatically create fields on the fly, but for specific fields like `title`, it's best to define them explicitly. Let me know if you need further assistance!

```
http://localhost:8983/solr/mycore/select?
q=title:%22Solr%22
{
  "responseHeader": {
    "status": 400,
```

```
"QTime": 0,
"params": {
  "q": "title:\"Solr\""
},
"error": {
  "metadata": [
    "error-class",
    "org.apache.solr.common.SolrException",
    "root-error-class",
    "org.apache.solr.common.SolrException"
  ],
  "msg": "undefined field title",
  "code": 400
}
```

The error you're seeing, `undefined field title`, indicates that Solr does not recognize the `title` field in your core's schema. Here's how to resolve the issue:

1. Verify the Schema Definition

You need to make sure that the `title` field is correctly defined in the schema for your core (`mycore`).

Steps to Check or Add the `title` Field:

1. Go to the Solr Admin UI:

- Navigate to `http://localhost:8983/solr`.
- Select the `mycore` core from the list.

2. Check the Schema:

- Go to the **Schema** tab on the Solr Admin UI.
- Search for the `title` field. If it's not there, you need to add it.

3. Add the `title` Field: If the `title` field is missing, you need to add it. Here's how:

- Click on **Add Field**.
- Define the `title` field as follows:

```
xml
```

```
<field name="title" type="string" indexed="true" stored="true"/>
```

This will ensure that the `title` field is indexed and stored, making it searchable.

4. **Reload the Core:** After adding the field, make sure to **reload** the `mycore` core to apply the changes.

- You can reload the core from the Admin UI or run the following command from the Solr directory:

```
bash

bin/solr reload -c mycore
```

2. Modify `schema.xml` Manually (if needed)

If you're not using the Solr Admin UI, you can directly edit the `schema.xml` file:

1. Go to the `mycore` directory, and find the `schema.xml` file located in `server/solr/mycore/conf/`.
2. Add the following field definition inside the `<fields>` section of the `schema.xml` file:

```
xml

<field name="title" type="string" indexed="true" stored="true"/>
```

3. After making the changes, save the file and **reload** the core.

3. Re-index Documents

Once the `title` field is defined and the core is reloaded, you may need to **re-index** your documents if they were previously indexed before the field was defined.

If the documents are already indexed, you should be able to search them. If they haven't been indexed yet, you'll need to re-upload the documents to Solr.

4. Retry the Query

After making sure the field is defined in your schema, try your query again:

```
text

http://localhost:8983/solr/mycore/select?q=title:"Solr"
```


This should now work correctly and return documents containing the `title` field with the value `"Solar"`.

Let me know if this resolves your issue, or if you need further help!

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "Solar"
    }
  },
  "response": {
    "numFound": 0,
    "start": 0,
    "numFoundExact": true,
    "docs": []
  }
}
```

The response you're getting indicates that the query was successful (status 0), but no documents were found that match the query `"Solar"`. This could be due to a few reasons:

1. No Matching Documents in Solr

The most likely explanation is that there are no documents in the Solr index that match the query `"Solar"` in the `title` field. Here's how to troubleshoot this:

- **Check the Documents in Solr:**
 - Go to the Solr Admin UI (`http://localhost:8983/solr`).
 - Select your core (`mycore`).
 - Go to the **Query** tab.
 - Try running a simple query like `*:*` to retrieve all documents in the index.

text

```
q=*:*&rows=10
```

This will show you the documents that are currently indexed in your Solr core.

- **Verify Document Indexing:**

- Ensure that documents have been successfully indexed in Solr, and the `title` field contains the expected data.
- If you're indexing documents through your FastAPI app, make sure the `title` field is populated when sending the document to Solr.

2. Check for Correct Field Mapping

Make sure the `title` field is indexed correctly and that the documents you're adding contain the value `"Solar"` in the `title` field.

- Go to your Solr Admin UI, select `mycore`, and navigate to the **Schema** tab to verify the `title` field definition.
- Ensure that the `title` field is indexed and stored (as per your previous configuration):

```
xml
```

```
<field name="title" type="string" indexed="true" stored="true"/>
```

3. Check for Tokenization or Case Sensitivity

Solr may be tokenizing or treating your search terms in ways that could affect results.

- **Tokenization:** If you're using a `text_general` field type for `title`, Solr might tokenize the content into individual words, which means querying `"Solar"` might not find documents if the text is tokenized.
- **Case Sensitivity:** If your `title` field is a `string` type, Solr's default behavior is case-sensitive. Ensure that `"Solar"` exactly matches the case in the indexed data.

4. Verify the Document Structure

Ensure that the documents you're uploading to Solr have the `title` field populated with the expected value. You can check the documents that are being indexed by logging the content you're sending to Solr.

5. Re-index Documents

If you suspect that the documents are not correctly indexed or the `title` field was not populated during indexing, you can re-upload the documents and re-index them.

For example, in your FastAPI code, verify that the `title` is passed correctly when creating the `Document` and that it's part of the document sent to Solr.

Example Query:

To perform the query on the `title` field, ensure you're searching like this:

text

```
http://localhost:8983/solr/mycore/select?q=title:"Solar"
```

If no documents match, try querying with a wildcard (`*:*`) to see what documents are in the index:

text

```
http://localhost:8983/solr/mycore/select?q=*:*&rows=10
```

Let me know if you need further assistance in debugging or re-indexing the documents!

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "Solr"
    }
  },
  "response": {
    "numFound": 0,
    "start": 0,
    "numFoundExact": true,
    "docs": []
  }
}
```

The response indicates that the query `"Solr"` did not match any documents in your Solr index:

json

```
{
  "numFound": 0,
  "docs": []
}
```

This means that Solr did not find any documents with the term `"Solr"`, which suggests a couple of possible reasons why the query is returning no results:

Troubleshooting Steps

1. Ensure Documents Are Indexed Correctly

- First, let's confirm that documents are actually being indexed in your Solr core.
- Open the Solr Admin UI at `http://localhost:8983/solr` and select your core (`mycore`).
- Go to the **Query** tab and run the following query to retrieve all documents:

```
text

q=*&rows=10
```

This will show you a list of all the documents indexed in Solr. If this query returns results, you can be sure that documents are being indexed but there might be issues with how they're indexed (such as missing the `title` field or incorrect field mapping).

2. Check Field Mappings

- In the Solr Admin UI, navigate to the **Schema** tab for your `mycore` core.
- Ensure that the `title` field is defined correctly as `indexed="true"` and `stored="true"` in the schema. The schema should look something like this:

```
xml

<field name="title" type="string" indexed="true" stored="true"/>
```

- If the `title` field is not indexed or stored correctly, you may need to modify the schema or re-index your documents.

3. Verify Document Upload

Check that the documents you're uploading to Solr have the `title` field populated with the expected value (in this case, `"Solr"`). You can add debug logging in your FastAPI code to ensure that the `title` field is being passed correctly in the document to Solr.

For example, log the document content before sending it to Solr:

```
python

# Print the document content before sending to Solr
print(f"Document being sent to Solr: {document.dict()}")
```

4. Re-index Documents

If necessary, re-index the documents to ensure they are being added correctly. You can do this by uploading the documents again using your FastAPI endpoint or using Solr's Data Import Handler (DIH) or post tool.

5. Test Case Sensitivity

If the `title` field is indexed as a `string` type and you are doing case-sensitive searches, ensure that the value of `title` in Solr exactly matches `"Solr"`. For case-insensitive searches, you may need to use a `text_general` field type or adjust your query to handle case sensitivity.

You can try querying using `q=title:"solr"` or with wildcard searches:

```
text

q=title:Solr*
```

6. Check for Errors or Warnings in Solr Logs

If documents are still not being indexed, check the Solr logs for any errors or warnings during indexing. You can find Solr logs in the `logs` directory within your Solr installation path.

Let me know if you're able to find the documents with the `*:*` query or if you need further help debugging the issue.

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload_document/?
  title=Solr&author=bhargav&published_date=15-12-94' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
```

```
-F 'file=@Solr.docx;type=application/vnd.openxmlformats-officedocument.wordprocessingml.document'
```

The `curl` command you've provided looks mostly correct, but there are a few things to check:

Key Components of the `curl` Command

1. HTTP Method:

- You're sending a `POST` request, which is correct for uploading a file.

2. URL:

- The URL you're posting to is `http://127.0.0.1:8000/upload_document/`, which should match the FastAPI endpoint you've set up for file uploading.

3. Headers:

- You're using the correct headers: `accept: application/json` and `Content-Type: multipart/form-data`. This is also correct for uploading files and expecting JSON responses.

4. Form Data:

- You're uploading the file (`Solr.docx`) as part of the request with the `-F` flag. You're correctly specifying the file type (`type=application/vnd.openxmlformats-officedocument.wordprocessingml.document`).

Things to Check

1. Path to the File:

- Ensure that the `Solr.docx` file is in the same directory where you are running the `curl` command. If the file is located somewhere else, you need to provide the full path to the file.
- For example, if the file is located at `C:/Users/SESPL/Documents/Solr.docx`, your `curl` command should be:

```
bash

-F
'file=@C:/Users/SESPL/Documents/Solr.docx;type=application/vnd.openxmlformats-officedocument.wordprocessingml.document'
```

2. Ensure FastAPI Is Running:

- Make sure your FastAPI application is running and accessible at `http://127.0.0.1:8000`. You should be able to open this URL in your browser and get a response from your FastAPI app.

3. File Size:

- If the file you're uploading is too large, you may need to adjust the FastAPI configuration to allow for larger file uploads (via `max_content_length` or increasing the `upload_max_size` if needed).

Example of the Correct Command

```
bash

curl -X 'POST' \
  'http://127.0.0.1:8000/upload_document/?
title=Solr&author=bhargav&published_date=15-12-94' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@Solr.docx;type=application/vnd.openxmlformats-
officedocument.wordprocessingml.document'
```

Possible Responses

If everything is set up correctly, you should see a response like this (in JSON):

```
json

{
  "message": "Document uploaded and indexed successfully!"
}
```

If there's an issue with the file upload or other parts of the code, you'll get an error response with more details about the issue.

Let me know how it goes or if you face any specific errors!

how to see txt file

To view the contents of a `.txt` file, you can use several methods depending on your operating system and preferences. Below are different ways to open and view `.txt` files.

On Windows

1. Using Notepad:

- Right-click on the `.txt` file.
- Select **Open with** and choose **Notepad** (or any text editor).
- Alternatively, you can simply double-click the file, and it will open in Notepad by default.

2. Using Command Prompt (for quick viewing):

- Open Command Prompt (`cmd`) and use the `type` command to print the file content to the screen:

```
cmd

type "C:\path\to\your\file.txt"
```

- This will display the contents of the file in the terminal.

3. Using PowerShell:

- You can also use PowerShell to read the file:

```
powershell

Get-Content "C:\path\to\your\file.txt"
```

On macOS

1. Using TextEdit:

- Right-click on the `.txt` file and select **Open with > TextEdit**.
- Alternatively, you can double-click the file, and it will open in TextEdit by default.

2. Using Terminal:

- Open **Terminal** and use the `cat` command to view the file content:

```
bash

cat /path/to/your/file.txt
```


3. Using `less` or `more` :

- You can use `less` or `more` to scroll through large text files:

```
bash

less /path/to/your/file.txt
```

On Linux

1. Using Text Editors (e.g., Gedit, Nano, Vim):

- Open the terminal and use any text editor to open the file:

```
bash

gedit /path/to/your/file.txt  # Gedit for GUI-based
nano /path/to/your/file.txt   # Nano for terminal-based
vim /path/to/your/file.txt    # Vim for terminal-based
```

2. Using `cat`, `more`, or `less` :

- For quick viewing of the file's contents, you can use the `cat`, `more`, or `less` commands:

```
bash

cat /path/to/your/file.txt    # Outputs the entire file
less /path/to/your/file.txt   # Allows scrolling
more /path/to/your/file.txt   # Allows scrolling (older command)
```

Opening `.txt` Files in Programming (Python Example)

If you're working with Python and want to read the contents of a `.txt` file, you can do so using the following code:

```
python

# Python code to open and read a .txt file

file_path = "C:/path/to/your/file.txt"

with open(file_path, 'r') as file:
```

```
content = file.read()
print(content)
```

This will read the entire file and print its contents.

Let me know if you need help with a specific method or platform!

how to see solr txt file indexed properly know

To view the contents of a document indexed in Apache Solr, you need to perform a search query to retrieve the data. Here's a step-by-step guide on how to do this:

1. Verify Solr is Running

Ensure that your Solr instance is up and running. You can check this by accessing the Solr admin UI at:

```
bash

http://localhost:8983/solr/
```

You should see the Solr dashboard. If you can access it, Solr is running properly.

2. Check the Solr Core

Make sure that your core (e.g., `mycore`) is correctly set up. To verify this:

- Go to the Solr admin panel (`http://localhost:8983/solr/`).
- Click on the "Core Selector" dropdown at the top left to make sure your core (`mycore`) is selected.

3. Search for the Indexed Document

Use Solr's `select` API to query the index and view the document. If your document contains a `title` field (as in your example), you can search by it.

Example of a Search Query

To search for a document with the title `Solr`, use this URL in your browser or send a `GET` request via `curl` or Postman:

```
bash
```

```
http://localhost:8983/solr/mycore/select?q=title:Solr&wt=json
```

- **Explanation:**

- `q=title:Solr` – This query searches for documents where the `title` field matches "Solr".
- `wt=json` – This specifies the response format as JSON.

Example Response:

The response from Solr will look like this:

```
json

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "title:Solr"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "numFoundExact": true,
    "docs": [
      {
        "id": "Solr.docx",
        "title": "Solr",
        "author": "bhargav",
        "published_date": "15-12-94",
        "content": "This is the extracted content of the Solr document..."
      }
    ]
  }
}
```

- **Explanation of fields in the response:**

- `"id"` – This is the unique ID of the document in the index.

- `"title"`, `"author"`, `"published_date"` – These are fields you indexed with the document.
- `"content"` – This is the content of the document that was extracted and indexed by Solr.

4. Troubleshooting No Results

If you get an empty response (`numFound: 0`), it could mean:

- The document wasn't indexed properly.
- You may need to check Solr's logs or check if the fields are mapped correctly in the Solr schema.

Example of Empty Response:

```
json

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {
      "q": "title:Solr"
    }
  },
  "response": {
    "numFound": 0,
    "start": 0,
    "numFoundExact": true,
    "docs": []
  }
}
```

In this case, you would want to verify:

- The document was correctly sent to Solr.
- The correct field names are being used in the Solr schema.

5. Check Solr Logs

If you're having trouble and the document is still not showing up:

- Check Solr's log files (`logs/solr.log` or `logs/*` in the Solr directory) to see if there were any errors when indexing the document.
- Look for error messages like "field not found" or "indexing error".

6. Solr Query via curl (Alternative)

You can also use `curl` to test the Solr query from the command line:

```
bash

curl "http://localhost:8983/solr/mycore/select?q=title:Solr&wt=json"
```

This will return a JSON response with all the documents that match the query.

If you're still not able to see the indexed document, you may need to check the schema configuration, confirm the data is indexed correctly, or ensure that Solr is properly configured to handle the data format you're working with.

```
itle
string
(query)
Solr
author
string
(query)
bhargav
published_date
string
(query)
15-12-94
Request body

multipart/form-data
file *
string($binary)
The document to be uploaded
```

It looks like you're trying to perform a search query on Solr using fields such as `title`, `author`, and `published_date`. From your request, it seems like you're uploading documents and querying Solr with those fields. Let me walk you through this step by step to make sure everything is working correctly.

1. Making Sure the Data is Indexed Correctly

Your Solr schema should be set up to handle fields such as `title`, `author`, `published_date`, and `content`. If the fields are not defined in the schema, Solr will not be able to index or search by them.

Example of Solr Schema Setup

In the Solr schema (`managed-schema`), you need to define fields like this:

xml

```
<field name="title" type="string" indexed="true" stored="true"/>
<field name="author" type="string" indexed="true" stored="true"/>
<field name="published_date" type="string" indexed="true" stored="true"/>
<field name="content" type="text_general" indexed="true" stored="true"/>
```

- `title`, `author`, `published_date` are `string` fields.
- `content` is a `text_general` field to allow full-text search on the document content.

2. Correct Query Syntax

In Solr, when you search for a document, you should be using the correct query syntax. Based on your example:

Example Query

If you're searching for a document by `title`, `author`, or `published_date`, the query would look like this:

http

```
http://localhost:8983/solr/mycore/select?q=title:"Solr"&wt=json
```

or for searching by multiple fields:

http

```
http://localhost:8983/solr/mycore/select?q=title:"Solr" AND author:"bhargav"&wt=json
```

You can replace `Solr` and `bhargav` with the values you're searching for.

- `q=title:"Solr"` : This searches for documents where the `title` field contains "Solr".
- `wt=json` : This specifies the response format as JSON.

3. Indexing a Document (Multipart Request)

You're using FastAPI to upload a document. To ensure the document is properly indexed, the content is extracted and sent to Solr in a format that Solr can understand. You're also making sure that fields like `title`, `author`, and `published_date` are passed correctly.

Here is an example of how to send a document to Solr from FastAPI:

FastAPI Endpoint

python

```
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"C:/Users/SESPL/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded file
        with open(file_location, "rb") as f:
            text_content = extract_text_from_pdf_plumber(f) # Use PyPDF2 or pdfplumber

        # Prepare document metadata for indexing
        document = Document(
            id=file.filename,
            title=title,
            author=author,
            content=text_content,
            published_date=published_date
        )

        # Send the document to Solr
```

```
send_to_solr(document)

return {"message": "Document uploaded and indexed successfully!"}
except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}
```

Here, you're passing `title`, `author`, and `published_date` as parameters in the API request. Make sure you send the data as `multipart/form-data` from Postman or `curl`.

4. Search Query via `curl`

Once the document is uploaded and indexed, you can query it with `curl` like so:

```
bash

curl "http://localhost:8983/solr/mycore/select?q=title:%22Solr%22&wt=json"
```

This will return the document with the title "Solr" in the `title` field, provided the document was indexed correctly.

5. Troubleshooting

If your query is still returning no results (`numFound: 0`), here are some things to check:

- **Check the Solr logs** for any indexing issues.
- **Verify the schema:** Ensure that the fields (`title`, `author`, `published_date`) are defined in Solr's schema and that they are correctly indexed.
- **Check field names:** Ensure that the field names in the schema exactly match the names in your document (e.g., `title`, `author`).
- **Reindex the data:** If the fields weren't indexed correctly or if the Solr core was misconfigured, you may need to reindex the documents.

6. Use the Solr Admin UI

To manually inspect the documents indexed in Solr:

1. Go to `http://localhost:8983/solr/`.
2. Select your core (`mycore`).
3. Use the "Query" tab to run queries against your indexed documents.

By doing this, you should be able to view and search the documents properly. Let me know if you run into any further issues!

use only title

To search using **only the** `title` field in Solr, here's how you can proceed:

1. Indexing Document with Title Field

Ensure that your schema is set up correctly to index the `title` field. For example, in the `managed-schema` file:

```
xml

<field name="title" type="string" indexed="true" stored="true"/>
```

This ensures that Solr is able to index the `title` field and make it searchable.

2. Index the Document with Title Field

In your FastAPI code, you're already passing the `title` to Solr. When you upload the document, make sure the `title` field is being correctly indexed.

Example FastAPI Endpoint to Upload and Index a Document with Title:

```
python

@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"C:/Users/SESPL/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded PDF file
        with open(file_location, "rb") as f:
            text_content = extract_text_from_pdf_plumber(f) # Use PyPDF2 or pdfplumber

        # Prepare document metadata for indexing
        document = Document(
            id=file.filename,
```

```

        title=title,
        author=author,
        content=text_content,
        published_date=published_date
    )

    # Send the document to Solr
    send_to_solr(document)

    return {"message": "Document uploaded and indexed successfully!"}
except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

```

3. Search by Title Field Only

Now that the document is indexed with the `title` field, you can search for it using **only the title**.

You can do this by using the Solr `select` API with a query like:

Example Search Query (using `curl`):

```

bash

curl "http://localhost:8983/solr/mycore/select?q=title:%22Solr%22&wt=json"

```

Explanation:

- `q=title:"Solr"` : This query searches for documents where the `title` field contains the word `Solr`.
- `wt=json` : Specifies that the response should be returned in JSON format.

Response Example:

If the document with the title `Solr` is indexed correctly, Solr will return the document in the response:

```

json

{
  "responseHeader": {
    "status": 0,
    "QTime": 1,
    "params": {

```

```

    "q": "title:\"Solr\""
  }
},
"response": {
  "numFound": 1,
  "start": 0,
  "numFoundExact": true,
  "docs": [
    {
      "id": "Solr.docx",
      "title": "Solr",
      "author": "bhargav",
      "published_date": "15-12-94",
      "content": "Extracted text from the document content..."
    }
  ]
}
}

```

4. Troubleshooting

If the document is not found when querying the `title`, here are a few things to check:

- **Ensure the `title` field is indexed:** Verify that Solr's schema is configured to index the `title` field.
- **Check for errors:** Review Solr's logs to check if there were any issues while indexing the document.
- **Check the document content:** Make sure that the document was actually indexed with the correct `title` value.

Summary:

To search only by `title` in Solr, you can:

1. Ensure the `title` field is defined and indexed in Solr.
2. Index documents with a `title` field.
3. Use the Solr `select` query to search documents by `title` (e.g., `q=title:"Solr"`).

Let me know if you need more details!

It seems like the extracted text from the document isn't being returned correctly, possibly due to how the text extraction is being handled. Let's go over some possible solutions for extracting text from your uploaded document (e.g., Word, PDF) and ensuring that it's properly extracted and indexed.

Step 1: Ensure Correct Text Extraction

You're currently using `pdfplumber` for PDF extraction. However, if you're uploading a Word document (`.docx`), you should use a library like `python-docx` to extract text from Word documents instead.

Let's update your code to handle both **PDF** and **Word** document extraction properly.

Example: Extract Text from Word Document and PDF

1. Install the necessary libraries:

```
bash

pip install python-docx pdfplumber
```

2. Update the text extraction code:

```
python

import pdfplumber
from docx import Document as DocxDocument

# Function to extract text from Word Document
def extract_text_from_word(docx_file):
    try:
        doc = DocxDocument(docx_file)
        text = ""
        for para in doc.paragraphs:
            text += para.text + "\n"
        return text
    except Exception as e:
        print(f"Error extracting text from Word document: {e}")
        return None

# Function to extract text from PDF using pdfplumber
```

```
def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
                text += page.extract_text()
            return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None
```

Step 2: Update FastAPI to Handle File Upload and Text Extraction

Let's modify your FastAPI endpoint to detect the file type and extract text accordingly.

```
python

from fastapi import FastAPI, File, UploadFile
from pydantic import BaseModel
import requests
from docx import Document as DocxDocument
import pdfplumber
import json

app = FastAPI()

# Define a Pydantic model for document metadata
class Document(BaseModel):
    id: str
    title: str
    author: str
    content: str
    published_date: str

# Function to extract text from Word Document
def extract_text_from_word(docx_file):
    try:
        doc = DocxDocument(docx_file)
        text = ""
        for para in doc.paragraphs:
            text += para.text + "\n"
        return text
    except Exception as e:
```

```

        print(f"Error extracting text from Word document: {e}")
        return None

# Function to extract text from PDF using pdfplumber
def extract_text_from_pdf_plumber(pdf_file):
    try:
        with pdfplumber.open(pdf_file) as pdf:
            text = ""
            for page in pdf.pages:
                text += page.extract_text()
            return text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
        return None

# Function to send document content to Solr
def send_to_solr(document: Document):
    solr_url = "http://localhost:8983/solr/mycore/update/extract"
    json_data = {
        "add": {
            "doc": {
                "id": document.id,
                "title": document.title,
                "author": document.author,
                "content": document.content,
                "published_date": document.published_date
            }
        }
    }

    headers = {'Content-Type': 'application/json'}

    # Send the document content to Solr
    response = requests.post(solr_url, data=json.dumps(json_data), headers=headers)

    if response.status_code == 200:
        print("Document successfully indexed in Solr!")
    else:
        print(f"Error indexing document: {response.status_code} - {response.text}")

# Endpoint to upload PDF/Word document
@app.post("/upload_document/")

```

```

async def upload_document(file: UploadFile = File(...), title: str = "", author: str
= "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"C:/Users/SESPL/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text based on the file type (Word or PDF)
        text_content = ""
        if file.filename.endswith(".pdf"):
            with open(file_location, "rb") as f:
                text_content = extract_text_from_pdf_plumber(f)
        elif file.filename.endswith(".docx"):
            with open(file_location, "rb") as f:
                text_content = extract_text_from_word(f)

        if not text_content:
            return {"error": "Failed to extract text from the document."}

        # Prepare document metadata for indexing
        document = Document(
            id=file.filename,
            title=title,
            author=author,
            content=text_content,
            published_date=published_date
        )

        # Send the document to Solr
        send_to_solr(document)

        return {"message": "Document uploaded and indexed successfully!"}
    except Exception as e:
        return {"error": f"An error occurred: {str(e)}"}

```

Explanation:

1. Text Extraction:

- For PDFs, it uses `pdfplumber`.
- For Word Documents, it uses `python-docx`.

2. Document Upload:

- The document is uploaded, saved locally, and the correct extraction method is used based on the file extension (`.pdf` or `.docx`).
- The extracted text is then sent to Solr for indexing.

Step 3: Testing the Document Upload

1. **Upload a PDF or Word document** via the FastAPI endpoint `/upload_document/` using Swagger or `curl` .
2. **Check the Extracted Content:**
 - After uploading the document, ensure that Solr correctly indexes the extracted content.

Search Query Example

Once indexed, you can search by `title` to verify:

```
bash
```

```
curl "http://localhost:8983/solr/mycore/select?q=title:\"Solr\"&wt=json"
```

This should return the document, including the extracted text content, if everything is set up correctly.

Let me know if you encounter any issues!

```
File "C:\Users\SESPL\bhargav-workspace\Day 34-35
exploratory da and persimon app\Persimon App\main2.py",
line 4, in <module>
    from docx import Document as DocxDocument
File "C:\Users\SESPL\bhargav-workspace\Day 34-35
exploratory da and persimon app\Persimon
App\myenv\Lib\site-packages\docx.py", line 30, in
<module>
    from exceptions import PendingDeprecationWarning
ModuleNotFoundError: No module named 'exceptions'
```

The error you're encountering, `ModuleNotFoundError: No module named 'exceptions'` , typically occurs when the `python-docx` package tries to import `PendingDeprecationWarning` from the `exceptions` module, which is no longer available in Python 3.x.

This issue arises from the `docx` package that was originally designed for Python 2, and it is not fully compatible with Python 3. The proper package to use for Word document handling in Python 3 is `python-docx`. However, it seems you're using a version of `python-docx` that is outdated or incompatible with Python 3.

Solution

1. Ensure the Correct Package Installation:

- You need to make sure that the correct `python-docx` library is installed in your environment. If you're using a virtual environment, activate it first, and then install the correct version.

To install the correct version of `python-docx`, run:

```
bash

pip uninstall docx
pip install python-docx
```

2. Check Your Code:

- After ensuring that the correct package is installed, check your imports. It should look like this:

```
python

from docx import Document  # Correct import for python-docx
```

Note: Do not use `from docx import Document as DocxDocument`, as this might cause conflicts.

3. Re-run the Application:

After updating the package, try re-running your FastAPI application, and it should resolve the issue.

Let me know if you continue to encounter any issues!

uploaded sucessfully now get text

Great! Since you've successfully uploaded the document, now the next step is to extract the text from the uploaded file. Depending on the type of document you're uploading (e.g., DOCX, PDF), you can use appropriate libraries to extract the text.

Here's an approach for extracting text based on the file format. I'll cover both DOCX (Word) and PDF file extraction.

1. Extracting Text from DOCX (Word) Files

For DOCX files, you can use the `python-docx` library. Here's how to extract text from a DOCX file:

```
python

from docx import Document

def extract_text_from_docx(file_path):
    doc = Document(file_path)
    text = ""
    for para in doc.paragraphs:
        text += para.text + "\n"
    return text
```

2. Extracting Text from PDF Files

For PDF files, you can use `PyPDF2` or `pdfplumber`. Here's how to use `pdfplumber`:

```
python

import pdfplumber

def extract_text_from_pdf(file_path):
    with pdfplumber.open(file_path) as pdf:
        text = ""
        for page in pdf.pages:
            text += page.extract_text() + "\n"
    return text
```

Example FastAPI Endpoint to Handle Text Extraction

Let's modify your FastAPI endpoint to handle text extraction based on the file type:

```
python

from fastapi import FastAPI, File, UploadFile
import os
from docx import Document
```

```

import pdfplumber

app = FastAPI()

# Function to extract text from DOCX
def extract_text_from_docx(file_path):
    doc = Document(file_path)
    text = ""
    for para in doc.paragraphs:
        text += para.text + "\n"
    return text

# Function to extract text from PDF
def extract_text_from_pdf(file_path):
    with pdfplumber.open(file_path) as pdf:
        text = ""
        for page in pdf.pages:
            text += page.extract_text() + "\n"
    return text

# Endpoint to upload and extract text
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...)):
    try:
        # Save the uploaded file temporarily
        file_location = f"./{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text based on file type (DOCX or PDF)
        file_extension = file.filename.split(".")[1].lower()
        if file_extension == "docx":
            text_content = extract_text_from_docx(file_location)
        elif file_extension == "pdf":
            text_content = extract_text_from_pdf(file_location)
        else:
            return {"error": "Unsupported file type"}

        # Return extracted text (or process further, such as indexing)
        return {"message": "Text extracted successfully", "content": text_content}

```

```
except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}
```

Explanation:

1. Text Extraction:

- For DOCX files, it uses `python-docx` to read the text from the document.
- For PDF files, it uses `pdfplumber` to extract text from each page.

2. File Handling:

- The uploaded file is saved temporarily.
- Based on the file extension, the appropriate text extraction function is called.

3. Error Handling:

- If the file is not a DOCX or PDF, it returns an error message.

Testing the Endpoint:

You can now test this endpoint by uploading a DOCX or PDF file. The response will return the extracted text.

Let me know if you need help with any additional functionality!

uploaded successfully now how to get txt format

Once you've uploaded the file successfully and extracted the text, you can return the extracted text in the response in plain text format, or optionally, save it as a `.txt` file on the server or provide it for download.

1. Returning Extracted Text in the Response

You can simply return the extracted text in the response as plain text:

```
python

@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...)):
    try:
        # Save the uploaded file temporarily
        file_location = f"./{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())
```

```

# Extract text based on file type (DOCX or PDF)
file_extension = file.filename.split(".")[1].lower()
if file_extension == "docx":
    text_content = extract_text_from_docx(file_location)
elif file_extension == "pdf":
    text_content = extract_text_from_pdf(file_location)
else:
    return {"error": "Unsupported file type"}

# Return extracted text in plain text
return {"message": "Text extracted successfully", "content": text_content}

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

```

In this case, the response will look something like this:

```

json

{
  "message": "Text extracted successfully",
  "content": "This is the extracted text from the document."
}

```

2. Saving Extracted Text as a .txt File and Providing a Download Link

If you want to save the extracted text as a .txt file and allow users to download it, you can write the text content to a .txt file and return a download URL in the response.

Here's an updated version of the code that saves the extracted text to a .txt file and provides a download link:

```

python

import os

@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...)):
    try:
        # Save the uploaded file temporarily
        file_location = f"./{file.filename}"
        with open(file_location, "wb") as f:

```

```

        f.write(await file.read())

    # Extract text based on file type (DOCX or PDF)
    file_extension = file.filename.split(".")[1].lower()
    if file_extension == "docx":
        text_content = extract_text_from_docx(file_location)
    elif file_extension == "pdf":
        text_content = extract_text_from_pdf(file_location)
    else:
        return {"error": "Unsupported file type"}

    # Save the extracted text to a .txt file
    text_file_location = f"./{file.filename}.txt"
    with open(text_file_location, "w") as text_file:
        text_file.write(text_content)

    # Return a download link to the .txt file
    return {"message": "Text extracted successfully", "download_link":
f"/download/{file.filename}.txt"}

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

@app.get("/download/{file_name}")
async def download_file(file_name: str):
    try:
        # Provide the .txt file for download
        file_path = f"./{file_name}"
        if os.path.exists(file_path):
            return FileResponse(file_path)
        else:
            return {"error": "File not found"}
    except Exception as e:
        return {"error": f"An error occurred: {str(e)}"}

```

Explanation:

1. **Extracting Text:** Same as before, the text is extracted from the document based on the file type (DOCX or PDF).
2. **Saving Text to a .txt File:** The extracted text is written to a .txt file with the same name as the uploaded document.

3. **Returning a Download Link:** After saving the text to a `.txt` file, the response includes a link (`/download/{file_name}.txt`) where the user can download the extracted text file.
4. **Download Endpoint:** A new endpoint `/download/{file_name}` is created to serve the `.txt` file when the user clicks the download link.

Example Workflow:

1. Upload the file via the `/upload_document/` endpoint.
2. The server extracts the text and saves it to a `.txt` file.
3. The response includes a link to download the `.txt` file.
4. The user clicks the download link to get the extracted text in `.txt` format.

Testing:

You can test it by uploading a DOCX or PDF file and then using the provided download link to get the `.txt` file.

Let me know if you have further questions!

where i can see extracted txt

The extracted text file (`.txt`) will be saved on the server in the directory where the FastAPI application is running. The file will have the same name as the uploaded file but with a `.txt` extension.

For example:

- If you upload a file named `document.docx` , the extracted text will be saved as `document.docx.txt` .
- You can access this text file through the download URL provided in the API response, or you can check the server's file system to locate the `.txt` file.

How to access the extracted `.txt` file:

1. **Via API (Download Link):** After the file is uploaded and the text is extracted, the response will contain a URL like this:

```
json

{
  "message": "Text extracted successfully",
```

```
"download_link": "/download/document.docx.txt"
}
```

You can access the extracted `.txt` file by visiting this URL in your browser or making a `GET` request to it:

```
bash

http://localhost:8000/download/document.docx.txt
```

2. **File System:** If you are running the FastAPI application locally, you can find the extracted `.txt` file in the same directory where the FastAPI app is located. For example, if your app is running from the directory `C:/Users/SESPL/Persimon App`, you should see the extracted text file there with the name `document.docx.txt` (or whatever the uploaded file name is, with the `.txt` extension).

Example File Structure:

Assuming your FastAPI app is running from the directory `C:/Users/SESPL/Persimon App`, the structure might look like this:

```
graphql

C:/Users/SESPL/Persimon App
|-- main2.py    # Your FastAPI app script
|-- document.docx # The uploaded document
|-- document.docx.txt # The extracted text file
```

Once the file is extracted and saved as `.txt`, you can either download it using the link provided by the FastAPI response or find it directly in your app's directory.

Let me know if you need further assistance!

```
INFO: 127.0.0.1:54565 - "POST /upload_document/?
title=Solr&author=bhargav&published_date=15-12-1994
HTTP/1.1" 200 OK
INFO: 127.0.0.1:54567 - "POST /upload_document/?
title=Solr&author=bhargav&published_date=15-12-1994
HTTP/1.1" 200 OK
INFO: 127.0.0.1:54584 - "GET
/download/document.docx.txt HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:54585 - "GET /download/Solar.docx.txt
```



```
HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:54585 - "GET /download/Solr.docx.txt
HTTP/1.1" 404 Not Found
```

The `404 Not Found` error indicates that FastAPI cannot find the `.txt` file at the specified download URL. This could be happening for a few reasons:

1. File Saving Path Issue:

The `.txt` file might not be getting saved in the expected location, or FastAPI isn't serving the file from the correct directory.

Solution:

Make sure you're saving the extracted `.txt` file in a directory that FastAPI can serve. For example, you can save the text file in a directory like `./static/` or `./downloads/` where FastAPI can access and serve it.

Here's how you can modify the code to save the text file and make it accessible via a download endpoint.

2. Serving Static Files in FastAPI:

FastAPI can serve static files using `StaticFiles` if you configure it to do so. Here's an updated version of the code to save the extracted text and serve it correctly:

Update `main.py`:

1. Add the static file serving setup in your FastAPI application.
2. Save the `.txt` file in a specific directory (e.g., `./downloads/`).
3. Provide a URL to download the `.txt` file.

Here is an example of how to do it:

```
python

from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
import os
from docx import Document as DocxDocument

app = FastAPI()

# Ensure a directory to store the .txt files
```

```

if not os.path.exists('downloads'):
    os.makedirs('downloads')

# Function to extract text from a .docx file
def extract_text_from_docx(docx_file):
    doc = DocxDocument(docx_file)
    text = ""
    for paragraph in doc.paragraphs:
        text += paragraph.text + '\n'
    return text

# Endpoint to upload document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"downloads/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded document
        extracted_text = extract_text_from_docx(file_location)

        # Save the extracted text as a .txt file
        txt_file_location = f"downloads/{file.filename}.txt"
        with open(txt_file_location, "w") as f:
            f.write(extracted_text)

        return {"message": "Document uploaded and text extracted successfully!",
                "download_link": f"/download/{file.filename}.txt"}

    except Exception as e:
        return {"error": f"An error occurred: {str(e)}"}

# Endpoint to download extracted text file
@app.get("/download/{filename}")
async def download_file(filename: str):
    file_path = f"downloads/{filename}"
    if os.path.exists(file_path):
        return FileResponse(file_path)
    return {"error": "File not found"}

```

Changes made:

1. **Save .txt File:** The extracted text is saved to a file with the same name as the uploaded file but with the `.txt` extension, inside a `downloads/` directory.
2. **Serve .txt File:** A new route `/download/{filename}` is created that will serve the `.txt` file when accessed.

3. Testing the File Download:

1. Upload a document via the `/upload_document/` API.
2. Once the document is uploaded, the response will contain a link to download the extracted `.txt` file.

For example, if you uploaded `Solr.docx`, the response will look like this:

```
json

{
  "message": "Document uploaded and text extracted successfully!",
  "download_link": "/download/Solr.docx.txt"
}
```

3. You can now access the extracted text file by visiting the URL:

```
arduino

http://127.0.0.1:8000/download/Solr.docx.txt
```

4. Check Your Logs:

Ensure there are no errors when uploading the document. If everything is working fine, the `.txt` file should be created and downloadable.

Let me know if you encounter any other issues!

now change text to json file

To convert the extracted text into a JSON file instead of a `.txt` file, we will modify the code slightly to:

1. Convert the extracted text into a JSON object.
2. Save the JSON object as a `.json` file.

3. Serve the `.json` file for download.

Here's how you can modify your code:

Updated Code to Save Text as JSON:

python

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
import os
import json
from docx import Document as DocxDocument

app = FastAPI()

# Ensure a directory to store the .json files
if not os.path.exists('downloads'):
    os.makedirs('downloads')

# Function to extract text from a .docx file
def extract_text_from_docx(docx_file):
    doc = DocxDocument(docx_file)
    text = ""
    for paragraph in doc.paragraphs:
        text += paragraph.text + '\n'
    return text

# Endpoint to upload document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"downloads/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded document
        extracted_text = extract_text_from_docx(file_location)

        # Create a JSON object
        document_data = {
```

```

        "title": title,
        "author": author,
        "published_date": published_date,
        "content": extracted_text
    }

    # Save the document data as a .json file
    json_file_location = f"downloads/{file.filename}.json"
    with open(json_file_location, "w") as json_file:
        json.dump(document_data, json_file, indent=4)

    return {
        "message": "Document uploaded and text extracted successfully!",
        "download_link": f"/download/{file.filename}.json"
    }

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

# Endpoint to download the .json file
@app.get("/download/{filename}")
async def download_file(filename: str):
    file_path = f"downloads/{filename}"
    if os.path.exists(file_path):
        return FileResponse(file_path)
    return {"error": "File not found"}

```

Explanation:

1. **Extract Text from DOCX:** The `extract_text_from_docx` function reads the `.docx` file and extracts its text.
2. **Create JSON Object:** After extracting the text, we create a JSON object containing the document's metadata (title, author, published date) along with the extracted content.
3. **Save as JSON:** The `json.dump()` function saves the JSON object into a `.json` file.
4. **Download Endpoint:** The `/download/{filename}` route serves the saved `.json` file.

Test the Flow:

1. **Upload a Document:**
 - Use the `/upload_document/` API to upload a `.docx` file.

- The document metadata (title, author, published date) and the extracted text will be converted into a JSON file.

2. **Response:** Once the document is uploaded and processed, you will receive a response like this:

```
json

{
  "message": "Document uploaded and text extracted successfully!",
  "download_link": "/download/Solr.docx.json"
}
```

3. **Download the JSON:** You can download the resulting JSON file by navigating to:

```
arduino

http://127.0.0.1:8000/download/Solr.docx.json
```

4. **Sample JSON Output:** After downloading, the JSON file will look like this:

```
json

{
  "title": "Solr",
  "author": "bhargav",
  "published_date": "15-12-1994",
  "content": "This is the extracted text from the Solr.docx file."
}
```

This will give you the text extracted from the `.docx` file in a structured JSON format, which is now ready for use. Let me know if you need further assistance!

i want text and json format

If you want both the text and the JSON format for the extracted content, you can modify the code to:

1. Save the extracted text as a `.txt` file.
2. Save the same extracted content as a `.json` file with the same metadata.

Here is an updated version of the code to save both `.txt` and `.json` files:

Updated Code to Save Both `.txt` and `.json` Files:

python

```
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import FileResponse
import os
import json
from docx import Document as DocxDocument

app = FastAPI()

# Ensure a directory to store the .txt and .json files
if not os.path.exists('downloads'):
    os.makedirs('downloads')

# Function to extract text from a .docx file
def extract_text_from_docx(docx_file):
    doc = DocxDocument(docx_file)
    text = ""
    for paragraph in doc.paragraphs:
        text += paragraph.text + '\n'
    return text

# Endpoint to upload document
@app.post("/upload_document/")
async def upload_document(file: UploadFile = File(...), title: str = "", author: str = "", published_date: str = ""):
    try:
        # Save the uploaded file temporarily
        file_location = f"downloads/{file.filename}"
        with open(file_location, "wb") as f:
            f.write(await file.read())

        # Extract text from the uploaded document
        extracted_text = extract_text_from_docx(file_location)

        # Create JSON object
        document_data = {
            "title": title,
            "author": author,
            "published_date": published_date,
```

```

        "content": extracted_text
    }

    # Save the extracted text as a .txt file
    txt_file_location = f"downloads/{file.filename}.txt"
    with open(txt_file_location, "w") as txt_file:
        txt_file.write(extracted_text)

    # Save the document data as a .json file
    json_file_location = f"downloads/{file.filename}.json"
    with open(json_file_location, "w") as json_file:
        json.dump(document_data, json_file, indent=4)

    return {
        "message": "Document uploaded, text extracted, and saved as both .txt
and .json!",
        "download_txt_link": f"/download/{file.filename}.txt",
        "download_json_link": f"/download/{file.filename}.json"
    }

except Exception as e:
    return {"error": f"An error occurred: {str(e)}"}

# Endpoint to download the .txt or .json file
@app.get("/download/{filename}")
async def download_file(filename: str):
    txt_file_path = f"downloads/{filename}"
    if os.path.exists(txt_file_path):
        return FileResponse(txt_file_path)
    return {"error": "File not found"}

```

Explanation of Changes:

1. **Extract Text from DOCX:** We use the `extract_text_from_docx` function to extract text from the uploaded `.docx` file.
2. **Save as TXT:** The extracted text is saved in a `.txt` file with the same filename as the uploaded document but with a `.txt` extension.
3. **Save as JSON:** The extracted text, along with document metadata (title, author, published date), is saved as a `.json` file.
4. **Download Links:** After uploading, the response will provide two download links:

- One for the `.txt` file with the extracted text.
- One for the `.json` file with the metadata and content.

Test the Flow:

1. Upload a Document:

- Use the `/upload_document/` API to upload a `.docx` file.
- The document's metadata (title, author, published date) and the extracted content will be converted into both `.txt` and `.json` files.

2. Response: After successfully uploading the document, you will receive a response like this:

```
json

{
  "message": "Document uploaded, text extracted, and saved as both .txt and .json!",
  "download_txt_link": "/download/Solr.docx.txt",
  "download_json_link": "/download/Solr.docx.json"
}
```

3. Download the TXT and JSON Files:

- **TXT File:** Download the extracted text as a `.txt` file by navigating to:

```
arduino

http://127.0.0.1:8000/download/Solr.docx.txt
```

- **JSON File:** Download the extracted content along with metadata as a `.json` file by navigating to:

```
arduino

http://127.0.0.1:8000/download/Solr.docx.json
```

4. Sample Output:

TXT File Content:

```
vbnet
```

This **is** the extracted **text from** the Solr.docx file.
It includes multiple paragraphs **of text** that were extracted **from** the DOCX.

JSON File Content:

```
json

{
  "title": "Solr",
  "author": "bhargav",
  "published_date": "15-12-1994",
  "content": "This is the extracted text from the Solr.docx file.\nIt includes multiple paragraphs of text that were extracted from the DOCX."
}
```

Conclusion:

This implementation allows you to extract the text from a `.docx` file, save it both as a `.txt` file and as a `.json` file, and then provide download links for both formats. Let me know if you need any further adjustments!

how to stop solr server

To stop the Solr server, you can use the `bin/solr` command with the `stop` option. Here's how you can do it:

Steps to Stop Solr Server

1. **Navigate to the Solr Directory:** Open your terminal or command prompt and navigate to the directory where Solr is installed (e.g., `C:\Users\SESPL\solr-9.7.0`).
2. **Run the Stop Command:** Use the following command to stop the Solr server:

```
bash

bin\solr.cmd stop
```

For Linux or macOS, use:

```
bash

bin/solr stop
```

3. Optional Parameters:

- If you are running Solr on a specific port, you can specify it:

```
bash  
  
bin\solr.cmd stop -p 8983
```

Replace `8983` with the port number Solr is running on.

4. **Check if Solr Stopped:** You can verify if the Solr server has stopped by accessing the web interface:

```
bash  
  
http://localhost:8983/solr
```

If the server is stopped, this URL will not load.

Stopping Solr When Running in the Background

If Solr is running in the background or as a service:

- **Windows:** Use Task Manager to stop the Java process associated with Solr.
- **Linux/macOS:** Use the `ps` and `kill` commands to find and stop the Solr process:

```
bash  
  
ps aux | grep solr  
kill -9 <PID>
```

Replace `<PID>` with the process ID of the Solr server.

Let me know if you encounter any issues!