Project Structure for `manual_scraper_ext`

Create new scrapy project **`manual_scraper_ext`** where each spider resides in its respective subfolder:

```
manual_scraper_ext/
├── manual_scraper_ext/
│   ├── __init__.py
│   ├── items.py
│   ├── middlewares.py
│   ├── pipelines.py
│   ├── settings.py
│   └── spiders/
│       ├── SubzeroWolfCom/
│       │   ├── __init__.py
│       │   └── SubzeroWolfCom.py
│       ├── Spider2/
│       │   ├── __init__.py
│       │   └── Spider2.py
│       └── SpiderN/
│           ├── __init__.py
│           └── SpiderN.py
└── scrapy.cfg
```

Explanation:

1. **Subfolder for Each Spider**
   Each spider has its own directory (e.g., `SubzeroWolfCom/`) containing the spider implementation file and an `__init__.py` file to mark the directory as a package.
2. **Main Directory**
   - `manual_scraper_ext` serves as the top-level folder containing `scrapy.cfg`.
   - The subfolder `manual_scraper_ext/` contains project-specific settings, items, pipelines, etc

1. **Schema Compliance**:
    - Ensure that the example item provided in the schema is present in the output file and matches exactly as given.
2. **File_url Field Validation**:
    - Check that all entries in the **file_url** field contain URLs that end with **.pdf**.
    - Ensure that each URL is complete, including the necessary http:// or https:// prefix.
3. **Field Uniqueness**:
    - Verify there are no repetitions where:
        - **brand** does not appear in the **model** field.
        - **product** does not appear in the **model** field.
        - **model** does not appear in the **product** field.
        - **type** does not appear in the **model** field.
4. **URL Completeness**:
    - All URLs, whether **file_url**, **thumb**, or others, should be fully qualified URLs with appropriate protocols.
5. **Data Cleaning for Model and Product Fields**:
    - Clean and structure the **model** and **product** fields by removing irrelevant characters and standardizing formatting.
6. **Product Field Accuracy**:
    - Ensure that the **product** field entries are meaningful and precise.
        - For example, choose "washing machine" over "laundry" when both are applicable, to provide specific clarity.
7. **Product Language Specification**:
    - The **product_lang** field should always be two lowercase letters (e.g., "en").
8. **Thumbnail Image Handling**:
    - The **thumb** should always be the largest image available.
    - If the thumbnail path is relative (like "/thumb.png"), ensure it is converted to a full URL by joining it with the base response URL.
        - Implement this by if thumb:= response.css("example"):

- thumb = response.urljoin(thumb)

9. **Spider Naming Convention**
   Ensure the spider class name and name attribute match the source name from the schema:

```
class SubzeroWolfComSpider(scrapy.Spider):
            name = "subzero-wolf.com"
```

10. **Yield Each Item Individually**
    For scraping PDFs, ensure each file is yielded as a separate item:

```
for pdf_sel in response.css('a[href*=".pdf"]'):
    manual = Manual()
    rfile = pdf_sel.css("::attr(href)").get()
    manual["file_urls"] = [response.urljoin(rfile)]
    yield manual
```

11. **Dynamic Brand Handling**

    If there's only one brand, hard-code it:

```
manual["brand"] = "Sub-Zero Wolf"
```

    If multiple brands are present, dynamically scrape:

```
manual["brand"] = response.css("brand-selector::text").get()
```

12. **File URL as List**
    Use a list for the file_urls field to handle multiple URLs properly:

```
manual["file_urls"] = [response.urljoin(rfile)]
```

13. **Data Cleaning**

Standardize the model and product fields:

```
manual["model"] = clean_model(manual["model"])
manual["product"] = clean_product(manual["product"])
```

14. **Dynamic Type Matching**

The `type` field should dynamically correspond to the relevant PDF:

```
for pdf_sel in response.css('a[href*=".pdf"]'):
    manual = Manual()
    rfile = pdf_sel.css("::attr(href)").get()

    # Avoid duplicates
    if rfile in self.rfiles:
        continue
    self.rfiles.add(rfile)

    # Extract and clean type
    rtype = self.clean_type(pdf_sel.css("::text").get())
    if not rtype:
        continue

    manual["file_urls"] = [response.urljoin(rfile)]
    manual["type"] = rtype
    yield manual
```

15. Dynamic Language Extraction

Set `product_lang` dynamically based on the `lang` attribute of the HTML tag:

```
manual["product_lang"] = (
    response.css("html::attr(lang)").get(default="en").split("-")[0]
)
```

Example Spider Implementation For the spider located in `SubzeroWolfCom/SubzeroWolfCom.py`:

```
import scrapy
from manual_scraper_ext.items import Manual

class SubzeroWolfComSpider(scrapy.Spider):
    name = "subzero-wolf.com"

    def __init__(self):
        self.rfiles = set()

    def clean_type(self, raw_type):
        # Add logic to clean and return a valid type
        return raw_type.strip() if raw_type else None

    def parse(self, response):
        for pdf_sel in response.css('a[href*=".pdf"]'):
            manual = Manual()
            rfile = pdf_sel.css("::attr(href)").get()

            # Skip duplicates
            if rfile in self.rfiles:
                continue
            self.rfiles.add(rfile)

            # Dynamic type extraction
            rtype = self.clean_type(pdf_sel.css("::text").get())
            if not rtype:
                continue

            manual["file_urls"] = [response.urljoin(rfile)]
            manual["type"] = rtype
            manual["product_lang"] = (

response.css("html::attr(lang)").get(default="en").split("-")[0]
            )
            yield manual
```