Python Developer Test Project

Prepared by:

Görkem Kurter

General Information About Application:

In general, the application uses "Scrapy Framework" every day at 00:00 to scrape the information of the books on sale in the history category of the relevant websites and transfer them to MongoDB.

The information it conveys is the name of the book, the author, the publisher, the price and the discount amount.

Detailed Information:

Spiders:

While developing the application, I designed two separate spiders for "kitapyurdu" and "kitapsepeti". These two spiders first scrape the information of all the products in the history books of the sites and store them in .json format.

The structures of the two spiders are quite similar except for the response codes. The only difference is that the old price values are also stored in the "Kitapyurdu", so I calculated the percentage discount with the old price and new price information.

Spider Code Explanation:

name: The name attribute of the QuotesSpider class represents the name of the Spider. In this case, it is set to "kitapyurdu".

book_count: The book_count attribute of the QuotesSpider class serves as a counter for each book. It is used to assign a number to each book.

start_urls: It is a list that contains the starting URLs for the Spider. In this example, the starting URL is set to a category page on "https://www.kitapyurdu.com". It is used to navigate through the page and scrape book information.

parse Function: It is the main function of the Spider that performs the scraping task and parses the web page. The response parameter represents the response of the web page.

CSS Selectors: CSS selectors are used with the response object to select elements from the web page using the response.css() method. For example:

title: Selects the book titles using response.css("div.name.ellipsis a span::text").getall().

publisher: Selects the publishers using response.css("div.publisher span a span::text").getall().

author: Selects the authors using response.css("div.author span a span::text").getall().

old_price: Selects the old prices using response.css("div.price div.price-old.price-passive span.value::text").getall().

new_price: Selects the new prices using response.css("div.price div.price-new span.value::text").getall().

yield Statements: The selected elements are returned using the yield statement. A dictionary is created for each item, and the fields are filled accordingly. For example:

"list_number": Assigned the value of the book counter, book_count.

"title": Assigned the book title.

"publisher": Assigned the publisher.

"author": Assigned the author.

"old_price": Assigned the old price.

"new_price": Assigned the new price.

"discount": Calculated using the old price and new price to determine the discount percentage.

Page Navigation: The "next_url" variable selects the URL of the next page using the CSS selector "a.next::attr(href)". If there is a next page, it sends a request to that page using scrapy.Request() and calls the parse function again with callback=self.parse. This allows for navigating through the pages and scraping book information from multiple pages.

Discount Calculation:

old_price_str and **new_price_str**: These variables store the respective values from the **old_price** and **new_price** lists for the current iteration. If the index **x** is within the length of the list, the corresponding value is assigned. Otherwise, an empty string is assigned.

old_price_float and **new_price_float**: These variables are used to convert the price strings to floating-point numbers. The **float()** function is applied to the **old_price_str** and **new_price_str** values after removing commas and whitespace characters using the **replace(',', '').strip()** operations. If the string values are empty (indicating missing price information), a default value of 0.0 is assigned.

discount: This variable represents the discount percentage for a book. The calculation is based on the **old_price_float** and **new_price_float** values. If the **old_price_str** is not empty (indicating a valid old price), the discount is calculated as ((old_price - new_price) / old_price) * 100 and rounded to one decimal place using the **round()** function. Otherwise, a discount of 0.0 is assigned.

yield statement: The calculated values, including the discount, are yielded in a dictionary format along with other book details. The **discount** value is included only if both **old_price_str** and **new_price_str** are not empty. The book details are then returned as output for further processing.

These operations ensure that the discount is calculated correctly based on the old and new prices, and the relevant values are assigned and included in the output dictionary for each book.

I hope this explanation clarifies the discount calculation and the related variable operations in the code.

Main Part:

Libraries:

pymongo: This library is used for interacting with MongoDB database.

json: This library is used for working with JSON data.

subprocess: This library is used to run the Scrapy crawl command as a subprocess.

os: This library provides a way to interact with the operating system and perform operations like file removal.

time: This library is used for adding delays and working with time-related operations.

schedule: This library is used for scheduling recurring tasks.

Function: scrape_update()

This function performs the web scraping and updates the MongoDB database.

It uses subprocess.run() to run the Scrapy crawl commands as subprocesses.

After scraping, it establishes a connection to the MongoDB database using pymongo.MongoClient().

It selects the desired database and collections.

The function loads the scraped data from the JSON files (kitapyurdu.json and kitapsepeti.json) using json.load().

It deletes the existing documents in the collections using collection1.delete_many({}) and collection2.delete_many({}).

Then, it inserts the new scraped data into the collections using collection1.insert_one() and collection2.insert_one().

Finally, it removes the JSON files using os.remove().

Scheduling: schedule.every().day.at("00:00").do(scrape_update)

This schedules the scrape_update() function to run every day at midnight (00:00).

Main Loop:

The code enters a continuous loop using while True.

schedule.run_pending() checks if there are any scheduled tasks to run.

time.sleep(1) adds a small delay to the loop to prevent excessive CPU usage.

This code sets up a scheduled task using schedule to run the scrape_update() function every day at midnight. The function performs web scraping, updates the MongoDB database, and removes the JSON files. The main loop continuously checks for scheduled tasks and executes them.

Screenshots:

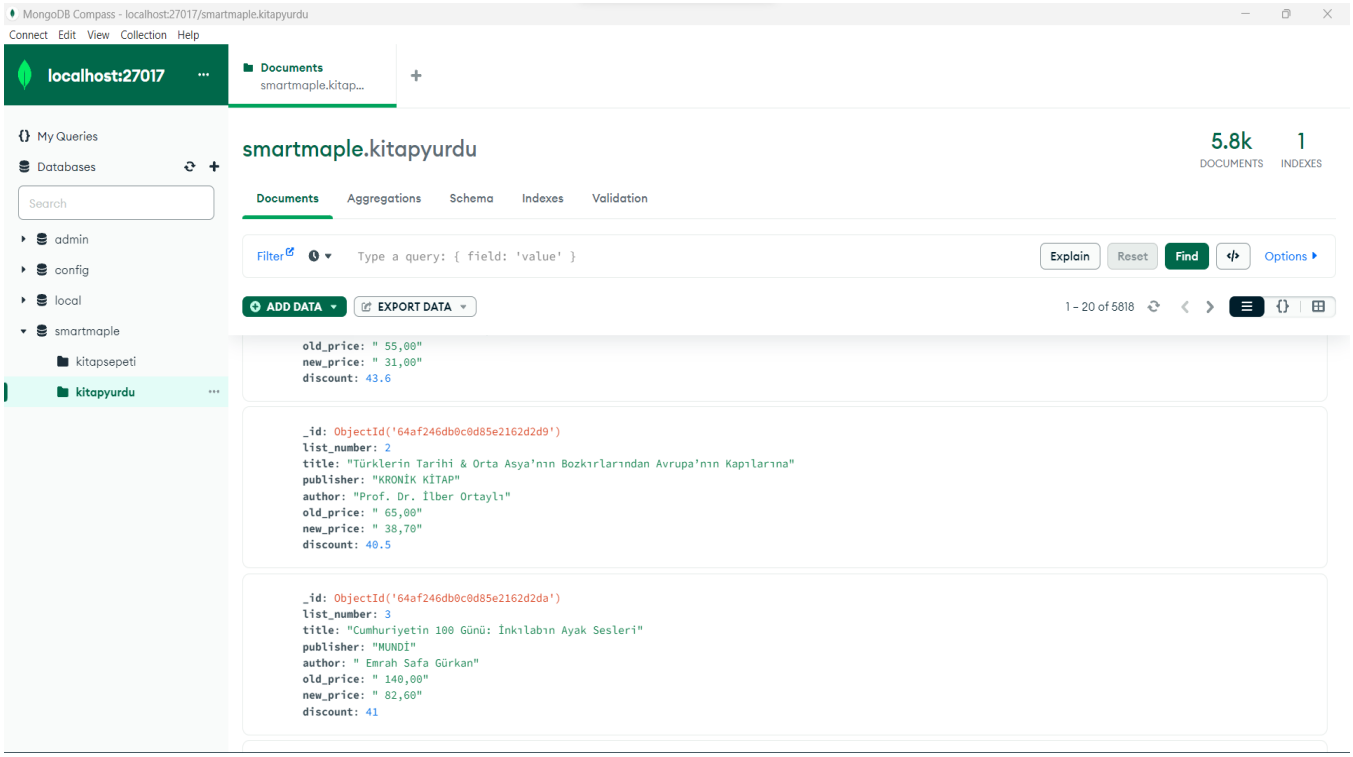

Figure 1:Display of smartmaple database's kitapyurdu collection

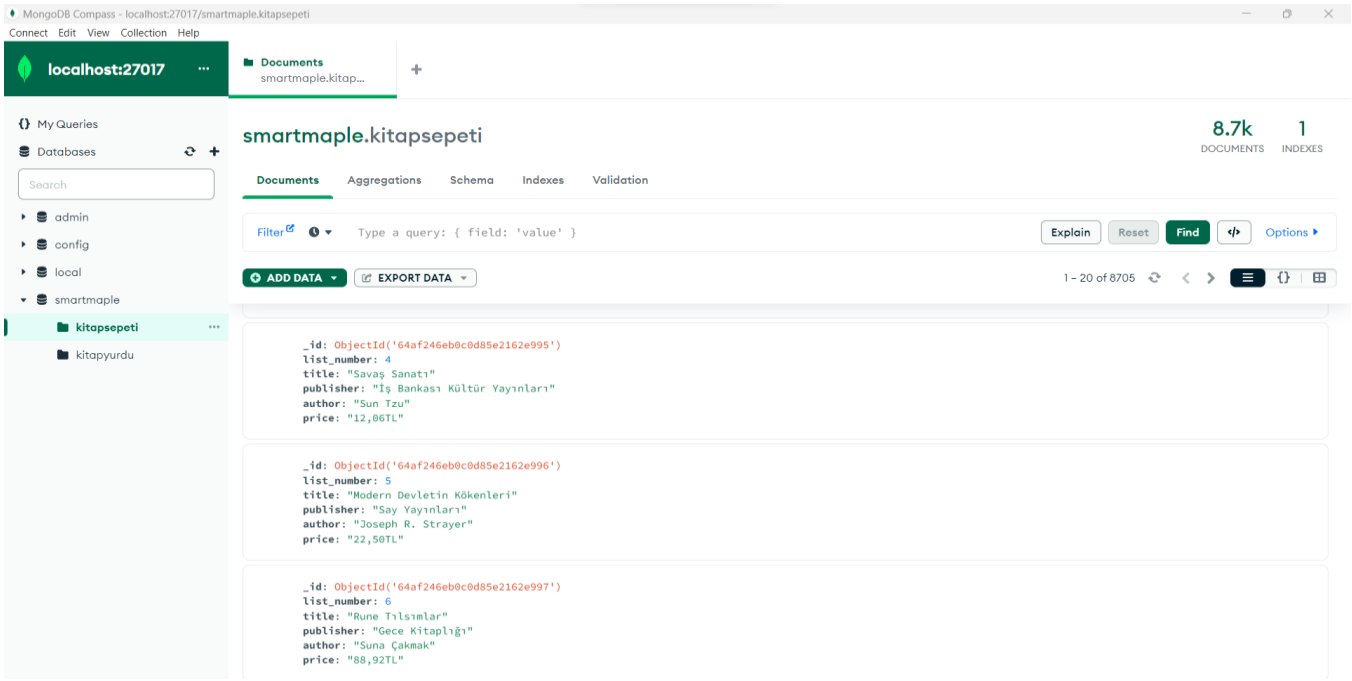

Figure 2:Display of smartmaple database's kitapsepeti collection

Source Codes:

kitapsepeti.py(Spider):

```
import scrapy



class QuotesSpider(scrapy.Spider):
    name = "kitapsepeti"
    book_count = 1
    start_urls = [
        "https://www.kitapsepeti.com/arastirma-inceleme-716?pg=1&stock=1"
    ]


    def parse(self, response):
        # Extract the book titles from the web page
```

```
        title = response.css("div.box.col-12.text-center a.fl.col-12.text-description.detailLink::text").getall()

        # Extract the publishers from the web page
        publisher = response.css("div.box.col-12.text-center a.col.col-12.text-title.mt::text").getall()

        # Extract the authors from the web page
        author = response.css("div.box.col-12.text-center a.fl.col-12.text-title::text").getall()

        # Extract the prices from the web page
        price = response.css("div.col.col-12.currentPrice::text").getall()
        for x in range(len(title)):
            yield {
                # Assign a list number to the book based on the book count
                "list_number": self.book_count,

  # Assign the current title, or an empty string if the index is out of range
                "title": title[x].replace("\n","") if x < len(title) else "",

# Assign the current publisher, or an empty string if the index is out of range
                "publisher": publisher[x] if x < len(publisher) else "",

 # Assign the current author, or an empty string if the index is out of range
                "author": author[x] if x < len(author) else "",

    # Assign the price, or an empty string if the index is out of range
                "price": price[x].replace("\n","") if x < len(price) else "",


            }
```

```
            # Increment the book count for the next book
            self.book_count += 1

            # Extract the URL of the next page, if available
        next_url = response.css("a.next::attr(href)").get()
        # If a next page URL exists, send a new request to that URL and call the 'parse' method recursively
        if next_url is not None:
            yield scrapy.Request(response.urljoin(next_url), callback=self.parse)
```

kitapyurdu.py(Spider):

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "kitapyurdu"
    book_count = 1
    start_urls = [
"https://www.kitapyurdu.com/index.php?route=product/category&page=1&filter_category_all=true&path=1_41&filter_in_stock=1&filter_in_shelf=1&sort=purchased_365&order=DESC&limit=20"
    ]

    def parse(self, response):
```

```
        # Extract the book titles from the web page
        title = response.css("div.name.ellipsis a span::text").getall()

        # Extract the publishers from the web page
        publisher = response.css("div.publisher span a span::text").getall()

        # Extract the authors from the web page
        author = response.css("div.author span a span::text").getall()

        # Extract the old prices from the web page

        old_price = response.css("div.price div.price-old.price-passive
span.value::text").getall()

        # Extract the new prices from the web page
        new_price = response.css("div.price div.price-new
span.value::text").getall()

        for x in range(len(title)):
            # Get the current new&old prices as a string, or an empty string
if the index is out of range
            old_price_str = old_price[x] if x < len(old_price) else ""
            new_price_str = new_price[x] if x < len(new_price) else ""


 # Convert the new&old price to a float, removing commas and stripping
whitespace, or set it to 0.0 if it's an empty string
            old_price_float = float(old_price_str.replace(',', '').strip()) if
old_price_str else 0.0
            new_price_float = float(new_price_str.replace(',', '').strip()) if
new_price_str else 0.0

            # Calculate the discount percentage if the old price is available,
otherwise set it to 0
```

```
            discount = 0
            if old_price_str:
                discount = round(((old_price_float - new_price_float) /
old_price_float) * 100, 1)


            yield {
                # Assign a list number to the book based on the book count
                "list_number": self.book_count,

                # Assign the current title, or an empty string if the index is
out of range
                "title": title[x] if x < len(title) else "",

                # Assign the current publisher, or an empty string if the
index is out of range
                "publisher": publisher[x] if x < len(publisher) else "",

                # Assign the current author, or an empty string if the index
is out of range
                "author": author[x] if x < len(author) else "",

                # Assign the current old price, or an empty string if the
index is out of range
                "old_price": old_price[x] if x < len(old_price) else "",

                # Assign the current new price, or an empty string if the
index is out of range
                "new_price": new_price[x] if x < len(new_price) else "",

                # Assign the discount percentage if both the old and new
prices are available, otherwise set it to 0.0
                "discount": discount if old_price_str and new_price_str else
0.0,
```

```
            }
            # Increment the book count for the next book
            self.book_count += 1

        # Extract the URL of the next page, if available
        next_url = response.css("a.next::attr(href)").get()

        # If a next page URL exists, send a new request to that URL and call the 'parse' method recursively
        if next_url is not None:
            yield scrapy.Request(response.urljoin(next_url), callback=self.parse)
```

main.py(Spider):

```
import pymongo
import json
import subprocess
import os
import time
import schedule


def scrape_update():
```

```
    # Run Scrapy crawl command for 'kitapsepeti' spider and save the output to
'kitapsepeti.json' file
    subprocess.run(['scrapy', 'crawl', 'kitapsepeti','-
o','kitapsepeti.json'],cwd='booksdatabase')



    # Delay for 10 seconds
    time.sleep(10)



    # Run Scrapy crawl command for 'kitapyurdu' spider and save the output to
'kitapyurdu.json' file
    subprocess.run(['scrapy', 'crawl', 'kitapyurdu','-
o','kitapyurdu.json'],cwd='booksdatabase')



    # Connect to MongoDB database
    client = pymongo.MongoClient("mongodb://localhost:27017/")



    # Select the 'smartmaple' database
    database = client["smartmaple"]



    # Select the 'kitapyurdu' and 'kitapsepeti' collections
    collection1 = database["kitapyurdu"]
    collection2 = database["kitapsepeti"]


    # Changing working direction for files
    os.chdir('booksdatabase')


    # Load data from 'kitapyurdu.json' file
    with open('kitapyurdu.json',encoding="utf-8") as file:
        data = json.load(file)
```

```
    # Delete existing documents in 'kitapyurdu' collection
    collection1.delete_many({})

    # Insert new data into 'kitapyurdu' collection
    for books in data:
        collection1.insert_one(books)

    # Load data from 'kitapsepeti.json' file
    with open('kitapsepeti.json',encoding="utf-8") as file:
        data2 = json.load(file)

    # Delete existing documents in 'kitapsepeti' collection
    collection2.delete_many({})

    # Insert new data into 'kitapsepeti' collection
    for books in data2:
        collection2.insert_one(books)
    # Remove the temporary JSON files
    os.remove("kitapsepeti.json")
    os.remove("kitapyurdu.json")
# Schedule the 'scrape_update' function to run every day at midnight
schedule.every().day.at("00:00").do(scrape_update)

# Continuous loop to check for scheduled tasks and execute them
while True:
    schedule.run_pending()
    time.sleep(1)
```