import time

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from typing import Union

from bs4 import BeautifulSoup

from selenium.webdriver.firefox.options import Options

class DriverManager:

def \_\_init\_\_(self, driver\_path, log, headless\_mode, driver\_type):

"""

Initialize the web driver for web scraping.

Args:

driver\_path (str): Path to the web driver executable.

log: Logger object for logging messages.

headless\_mode (bool): Set True if you want to run the browser in headless mode, else False.

driver\_type (str): Type of the browser ('firefox' or 'chrome').

Raises:

Exception: If an invalid browser type is provided.

"""

# ... (omitted for brevity)

def initialize\_driver(self, headless\_mode):

"""

Initialize the browser driver based on the user's choice of browser type.

Args:

headless\_mode (bool): Set True if you want to run the browser in headless mode, else False.

Returns:

The initialized browser driver object.

Raises:

Exception: If an error occurs during driver initialization.

"""

# ... (omitted for brevity)

def get\_prdouct\_source\_page(self):

"""

Scroll through the product page to load all content.

Returns:

None

"""

# ... (omitted for brevity)

def scroll\_to\_element(self, element\_xpath, wait\_time):

"""

Scroll the browser window until the specified element is visible in the viewport.

Args:

element\_xpath (str): XPath expression that points to the desired element on the page.

wait\_time (int): Number of seconds to wait before checking if the element is visible.

Returns:

WebElement: The element found after scrolling.

Raises:

TimeoutException: If the element is not found within the specified timeout period.

"""

# ... (omitted for brevity)

def load\_page(self, url):

"""

Load a web page using the Selenium WebDriver and set it as the active page for further actions.

Args:

url (str): The URL of the webpage to be loaded.

Returns:

str: The HTML content of the webpage at the given URL.

"""

# ... (omitted for brevity)

def open\_page(self, url: str):

"""

Open a web page by URL.

Args:

url (str): The URL of the webpage to be opened.

Returns:

None

"""

# ... (omitted for brevity)

def click\_on\_element\_by\_xpath(self, xpath):

"""

Click on a web page element using its XPath.

Args:

xpath (str): XPath expression pointing to the desired element.

Returns:

None

"""

# ... (omitted for brevity)

def scroll\_down(self):

"""

Scroll down the page to make sure all elements are loaded and visible.

Returns:

None

"""

# ... (omitted for brevity)

def scroll\_to\_top(self):

"""

Scroll to the top of the page.

Returns:

None

"""

# ... (omitted for brevity)

def scroll\_page(self, scroll\_count: Union[int, bool]):

"""

Perform scrolling in a loop while a condition is True or for a specified number of loops.

Args:

scroll\_count (bool/int): Number of times to scroll. If True, scroll indefinitely.

Returns:

None

Raises:

ValueError: If an invalid input is provided for scroll\_count.

"""

# ... (omitted for brevity)

def get\_page\_source(self):

"""

Get the current page source as a BeautifulSoup object.

Returns:

BeautifulSoup: Page source code.

"""

# ... (omitted for brevity)

def get\_prdoucts\_on\_page(self, page\_source, return\_value):

"""

Parse the given HTML content and find products on a webpage.

Args:

page\_source (str): HTML content from which products need to be found.

return\_value (str): Attribute of tag to return ('text' or 'tag').

Returns:

list: List of required values from each product tag.

Raises:

KeyError: If an invalid key is provided for return\_value.

"""

# ... (omitted for brevity)

def get\_seller\_id(self):

"""

Extract the seller ID from the product details page.

Returns:

str: Seller ID of the product owner.

"""

# ... (omitted for brevity)

def close\_driver(self):

"""

Close the web driver.

Returns:

None

"""

# ... (omitted for brevity)