import webbrowser

import random

from tkinter import Tk, Button, Frame, Label, StringVar, Toplevel

from datetime import datetime

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

# Set up Chrome options

chrome\_options = Options()

chrome\_options.add\_argument('--ignore-certificate-errors')

# Initialize the Chrome WebDriver with options

driver = webdriver.Chrome(options=chrome\_options)

# Function to open a URL in the web browser

def open\_url(url):

webbrowser.open(url)

# Function to fetch ESPN headlines

def fetch\_espn\_headlines():

driver.get('https://www.espn.com/')

wait = WebDriverWait(driver, 10)

headline\_links = driver.find\_elements(By.CSS\_SELECTOR, 'a[data-mptype="headline"]')

visible\_headlines = [(link.text.strip(), link.get\_attribute('href'))

for link in headline\_links if link.text.strip() and link.get\_attribute('href')]

return visible\_headlines[:10] # Limit to 10 headlines

# Function to fetch specific Investing.com links

def fetch\_investing\_headlines():

driver.get('https://www.investing.com/')

wait = WebDriverWait(driver, 10)

# Type 1 links (general finance headlines)

type\_1\_links = driver.find\_elements(By.CSS\_SELECTOR, 'a.line-clamp-3.text-base.font-semibold.leading-7.hover\\:underline.sm\\:line-clamp-2.md\\:line-clamp-3.md\\:leading-6')

# Type 2 links (small-cap stocks analysis)

type\_2\_links = driver.find\_elements(By.CSS\_SELECTOR, 'a.text-inv-blue-500.hover\\:text-inv-blue-500.hover\\:underline.focus\\:text-inv-blue-500.focus\\:underline.mb-2.text-base\\/\\[28px\\].font-semibold.\\!text-warren-gray-900')

# Collect text and limit the number of results for each type

visible\_headlines = []

# Combine type 1 and type 2 links

for link in type\_1\_links + type\_2\_links:

if link.text.strip():

visible\_headlines.append((link.text.strip(), link.get\_attribute('href')))

return visible\_headlines[:10] # Limit to 10 combined headlines

def fetch\_bbc\_headlines():

driver.get('https://www.bbc.com/')

wait = WebDriverWait(driver, 10)

headline\_links = driver.find\_elements(By.CSS\_SELECTOR, 'a[href^="/news/"], a[href\*="bbc.com/news/"]')

visible\_headlines = []

for link in headline\_links:

href = link.get\_attribute('href')

# If href starts with "https", it is already a full URL

if href.startswith("https"):

url = href

else:

url = f"https://www.bbc.com{href}"

if link.text.strip() and url:

visible\_headlines.append((link.text.strip(), url))

return visible\_headlines[:6] # Limit to 6 headlines for Politics

# Function to fetch Health.com headlines

def fetch\_health\_headlines():

driver.get('https://www.health.com/')

wait = WebDriverWait(driver, 10)

headline\_links = driver.find\_elements(By.CSS\_SELECTOR, 'a.mntl-card-list-items') # Updated selector for health links

visible\_headlines = []

for link in headline\_links:

headline\_text = link.find\_element(By.CSS\_SELECTOR, '.card\_\_title-text').text.strip()

url = link.get\_attribute('href')

if headline\_text and url:

visible\_headlines.append((headline\_text, url))

return visible\_headlines[:10] # Limit to 10 headlines

# Function to dynamically fetch headlines based on category

def fetch\_and\_assign\_details(source):

headlines = []

if source == "ESPN":

headlines = fetch\_espn\_headlines()

elif source == "Investing":

headlines = fetch\_investing\_headlines()

elif source == "Politics":

headlines = fetch\_bbc\_headlines()

elif source == "Health":

headlines = fetch\_health\_headlines()

return assign\_random\_views\_and\_sort(headlines)

# Function to update the display of headlines

def display\_headlines(headlines, root, frame, category\_name):

for widget in frame.winfo\_children():

widget.destroy()

current\_datetime = datetime.now().strftime("%m.%d.%Y %I:%M %p")

title\_label = Label(frame, text=f"Latest {category\_name} Headlines - {current\_datetime}",

font=("Helvetica", 18, "bold"), fg="white", bg="#2C3E50") # Dark brown background

title\_label.pack(pady=10)

sort\_by\_views\_button = Button(frame, text="Sort by Views", command=lambda: sort\_and\_display(headlines, 'views', root, frame),

bg="#1ABC9C", fg="white", font=("Arial", 10, "bold")) # Light salmon color

sort\_by\_views\_button.pack(pady=5)

sort\_by\_rating\_button = Button(frame, text="Sort by Rating", command=lambda: sort\_and\_display(headlines, 'rating', root, frame),

bg="#1ABC9C", fg="white", font=("Arial", 10, "bold")) # Light salmon color

sort\_by\_rating\_button.pack(pady=5)

refresh\_button = Button(frame, text="Refresh", command=lambda: refresh\_headlines(root, frame),

bg="#E74C3C", fg="white", font=("Arial", 10, "bold")) # Red color

refresh\_button.pack(pady=5)

for index, (headline, url, views, rating, stars, rng\_number) in enumerate(headlines, start=1):

formatted\_views = f"{views:,}"

headline\_text = f"#{index} ({formatted\_views} views, Rating: {stars} ({rng\_number})): {headline}"

headline\_button = Button(frame, text=headline\_text,

fg="white", bg="#1ABC9C", cursor="hand2", font=("Arial", 10, "bold"),

activebackground="#16A085", activeforeground="white", # Tomato color for active button

bd=0,

command=lambda url=url: open\_url(url))

headline\_button.pack(anchor="w", pady=5, padx=5, fill="x")

# Function to sort headlines based on views or rating and display them

def sort\_and\_display(headlines, sort\_by, root, frame):

if sort\_by == 'views':

sorted\_headlines = sorted(headlines, key=lambda x: x[2], reverse=True)

elif sort\_by == 'rating':

sorted\_headlines = sorted(headlines, key=lambda x: x[3], reverse=True)

display\_headlines(sorted\_headlines, root, frame, selected\_source.get())

# Function to convert a rating into a star representation

def generate\_star\_rating(rating):

full\_stars = int(rating)

half\_star = 1 if rating - full\_stars >= 0.5 else 0

empty\_stars = 5 - full\_stars - half\_star

return '★' \* full\_stars + '½' \* half\_star + '☆' \* empty\_stars

# Function to assign random views, ratings, and sort headlines

def assign\_random\_views\_and\_sort(headlines):

headlines\_with\_details = []

for headline, url in headlines:

views = random.randint(1000000, 10000000)

rating = round(random.uniform(0.5, 5.0), 1)

stars = generate\_star\_rating(rating)

rng\_number = random.randint(10, 10000)

headlines\_with\_details.append((headline, url, views, rating, stars, rng\_number))

return headlines\_with\_details

# Function to refresh headlines

def refresh\_headlines(root, frame):

source = selected\_source.get()

if source: # Ensure source is set

headlines\_with\_details = fetch\_and\_assign\_details(source)

display\_headlines(headlines\_with\_details, root, frame, source)

# Function to update the display based on selected category

def update\_category(source):

selected\_source.set(source) # Update the selected source

headlines\_with\_details = fetch\_and\_assign\_details(source)

# Use the selected category name in the display

category\_name = "ESPN.com" if source == "ESPN" else \

"Investing.com" if source == "Investing" else \

"BBC.com" if source == "Politics" else \

"Health.com"

display\_headlines(headlines\_with\_details, root, frame, category\_name)

# Timer logic in a separate window

def open\_timer\_window():

timer\_window = Toplevel(root)

timer\_window.title("Timer")

timer\_window.configure(bg="#34495E") # Dark brown background

running = False

start\_time = None

def update\_clock(clock\_label):

if running:

elapsed\_time = datetime.now() - start\_time

clock\_label.config(text=str(elapsed\_time).split(".")[0]) # Show elapsed time without microseconds

clock\_label.after(1000, update\_clock, clock\_label)

def start\_timer(clock\_label):

nonlocal running, start\_time

if not running:

running = True

start\_time = datetime.now()

update\_clock(clock\_label)

def stop\_timer():

nonlocal running

running = False

def reset\_timer(clock\_label):

nonlocal start\_time

start\_time = None

clock\_label.config(text="00:00:00")

# Timer display

timer\_label = Label(timer\_window, text="00:00:00", font=("Helvetica", 48, "bold"), fg="white", bg="#34495E")

timer\_label.pack(pady=20)

# Buttons to control the timer

start\_button = Button(timer\_window, text="Start", command=lambda: start\_timer(timer\_label),

font=("Helvetica", 16), fg="white", bg="#1ABC9C")

start\_button.pack(side="left", padx=10)

stop\_button = Button(timer\_window, text="Stop", command=stop\_timer,

font=("Helvetica", 16), fg="white", bg="#E74C3C")

stop\_button.pack(side="left", padx=10)

reset\_button = Button(timer\_window, text="Reset", command=lambda: reset\_timer(timer\_label),

font=("Helvetica", 16), fg="white", bg="#1ABC9C")

reset\_button.pack(side="left", padx=10)

# Create the main window

root = Tk()

root.title("Headlines and Timer")

root.configure(bg="#2C3E50") # Dark brown background

# Create a variable to hold the selected source

selected\_source = StringVar(root)

# Create a frame for category selection

category\_frame = Frame(root, bg="#2C3E50") # Dark brown background

category\_frame.pack(padx=10, pady=10)

# Add a label for the category selection

category\_label = Label(category\_frame, text="Category:", font=("Helvetica", 16), fg="white", bg="#2C3E50") # Dark brown background

category\_label.grid(row=0, column=0, padx=5)

# Add buttons for selecting categories

categories = [("Sports", "ESPN"), ("Finance", "Investing"), ("Politics", "Politics"), ("Health", "Health")]

for index, (text, source) in enumerate(categories):

category\_button = Button(category\_frame, text=text, font=("Helvetica", 14), fg="white", bg="#1ABC9C",

command=lambda source=source: update\_category(source))

category\_button.grid(row=0, column=index+1, padx=5)

# Create a button to open the timer window

timer\_button = Button(category\_frame, text="Open Timer", font=("Helvetica", 14), fg="white", bg="#1ABC9C",

command=open\_timer\_window)

timer\_button.grid(row=0, column=len(categories) + 1, padx=5)

# Create a frame for displaying headlines

frame = Frame(root, bg="#34495E") # Dark brown background

frame.pack(padx=10, pady=10, fill="both", expand=True)

# Add a placeholder label instructing the user to select a category

placeholder\_label = Label(frame, text="Please select a category to view headlines.",

font=("Helvetica", 18, "bold"), fg="white", bg="#2C3E50")

placeholder\_label.pack(pady=20)

# Start the main event loop

root.mainloop()

# Quit the driver after closing the application

driver.quit()