

School of Mechanical and Manufacturing Engineering

**(SMME)**

**ME-15(A)**

|  |  |
| --- | --- |
| AHMED NASEER(L) | 467276 |
| SADEED ARIF | 455385 |
| ASIM IMRAN | 476434 |
| MUHAMMAD HASSAAN | 460991 |

**“FOP PROJECT”**

**CODE:**

import feedparser

import string

import time

import threading

from project\_util import translate\_html

from tkinter import \*

from datetime import datetime

# -----------------------------------------------------------------------

# ======================

# Code for retrieving and parsing

# Google and Yahoo News feeds

# ======================

def process(url):

"""

Fetches news items from the rss url and parses them.

Returns a list of NewsStory instances.

"""

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

guid = entry.guid

title = translate\_html(entry.title)

link = entry.link

# Check if description field exists

if 'description' in entry:

description = translate\_html(entry.description)

else:

description = ""

# Handling different date formats

if 'published' in entry:

pubdate\_str = entry.published

elif 'published\_parsed' in entry:

pubdate\_str = time.strftime('%a, %d %b %Y %H:%M:%S %Z', entry.published\_parsed)

else:

continue

try:

pubdate = datetime.strptime(pubdate\_str, "%a, %d %b %Y %H:%M:%S %Z")

except ValueError:

pubdate = datetime.strptime(pubdate\_str, "%Y-%m-%dT%H:%M:%SZ")

newsStory = NewsStory(guid, title, description, link, pubdate)

ret.append(newsStory)

return ret

# ======================

# Data structure design

# ======================

class NewsStory:

def \_\_init\_\_(self, guid, title, description, link, pubdate):

self.guid = guid

self.title = title

self.description = description

self.link = link

self.pubdate = pubdate

def get\_guid(self):

return self.guid

def get\_title(self):

return self.title

def get\_description(self):

return self.description

def get\_link(self):

return self.link

def get\_pubdate(self):

return self.pubdate

# ======================

# Triggers

# ======================

class Trigger(object):

def evaluate(self, story):

raise NotImplementedError

class PhraseTrigger(Trigger):

def \_\_init\_\_(self, phrase):

self.phrase = phrase.lower()

def is\_phrase\_in(self, text):

text = text.lower()

for char in string.punctuation:

text = text.replace(char, ' ')

text\_words = text.split()

phrase\_words = self.phrase.split()

for i in range(len(text\_words) - len(phrase\_words) + 1):

if text\_words[i:i + len(phrase\_words)] == phrase\_words:

return True

return False

class TitleTrigger(PhraseTrigger):

def evaluate(self, story):

return self.is\_phrase\_in(story.get\_title())

class DescriptionTrigger(PhraseTrigger):

def evaluate(self, story):

return self.is\_phrase\_in(story.get\_description())

class TimeTrigger(Trigger):

def \_\_init\_\_(self, time):

self.time = datetime.strptime(time, "%Y-%m-%dT%H:%M:%SZ")

class BeforeTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate() < self.time

class AfterTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate() > self.time

class NotTrigger(Trigger):

def \_\_init\_\_(self, trigger):

self.trigger = trigger

def evaluate(self, story):

return not self.trigger.evaluate(story)

class AndTrigger(Trigger):

def \_\_init\_\_(self, trigger1, trigger2):

self.trigger1 = trigger1

self.trigger2 = trigger2

def evaluate(self, story):

return self.trigger1.evaluate(story) and self.trigger2.evaluate(story)

class OrTrigger(Trigger):

def \_\_init\_\_(self, trigger1, trigger2):

self.trigger1 = trigger1

self.trigger2 = trigger2

def evaluate(self, story):

return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)

# ======================

# Filtering

# ======================

def filter\_stories(stories, triggerlist):

filtered\_stories = []

for story in stories:

for trigger in triggerlist:

if trigger.evaluate(story):

filtered\_stories.append(story)

break

return filtered\_stories

# ======================

# User-Specified Triggers

# ======================

def read\_trigger\_config(filename):

trigger\_file = open(filename, 'r')

lines = []

for line in trigger\_file:

line = line.rstrip()

if not (len(line) == 0 or line.startswith('//')):

lines.append(line)

trigger\_file.close()

triggers = {}

trigger\_list = []

for line in lines:

parts = line.split(',')

if parts[0] == 'ADD':

for name in parts[1:]:

if name in triggers:

trigger\_list.append(triggers[name])

else:

trigger\_name = parts[0]

trigger\_type = parts[1]

if trigger\_type == 'TITLE':

triggers[trigger\_name] = TitleTrigger(parts[2])

elif trigger\_type == 'DESCRIPTION':

triggers[trigger\_name] = DescriptionTrigger(parts[2])

elif trigger\_type == 'AFTER':

triggers[trigger\_name] = AfterTrigger(parts[2])

elif trigger\_type == 'BEFORE':

triggers[trigger\_name] = BeforeTrigger(parts[2])

elif trigger\_type == 'NOT':

if parts[2] in triggers:

triggers[trigger\_name] = NotTrigger(triggers[parts[2]])

elif trigger\_type == 'AND':

if parts[2] in triggers and parts[3] in triggers:

triggers[trigger\_name] = AndTrigger(triggers[parts[2]], triggers[parts[3]])

elif trigger\_type == 'OR':

if parts[2] in triggers and parts[3] in triggers:

triggers[trigger\_name] = OrTrigger(triggers[parts[2]], triggers[parts[3]])

return trigger\_list

# ======================

# Main Thread

# ======================

SLEEPTIME = 120 # seconds

def main\_thread(master, keywords):

try:

triggerlist = []

if keywords:

for keyword in keywords:

triggerlist.append(OrTrigger(TitleTrigger(keyword), DescriptionTrigger(keyword)))

frame = Frame(master)

frame.pack(side=BOTTOM)

scrollbar = Scrollbar(master)

scrollbar.pack(side=RIGHT, fill=Y)

t = "Google & Yahoo Top News"

title = StringVar()

title.set(t)

ttl = Label(master, textvariable=title, font=("Helvetica", 18))

ttl.pack(side=TOP)

cont = Text(master, font=("Helvetica", 14), yscrollcommand=scrollbar.set)

cont.pack(side=BOTTOM)

cont.tag\_config("title", justify='center')

button = Button(frame, text="Exit", command=master.destroy)

button.pack(side=BOTTOM)

guidShown = []

def get\_cont(newstory):

if newstory.get\_guid() not in guidShown:

cont.insert(END, newstory.get\_title() + "\n", "title")

cont.insert(END, "\n---------------------------------------------------------------\n", "title")

cont.insert(END, newstory.get\_description())

cont.insert(END, "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", "title")

guidShown.append(newstory.get\_guid())

while True:

print("Polling...")

stories = process("http://news.google.com/news?output=rss")

stories.extend(process("http://news.yahoo.com/rss/topstories"))

stories = filter\_stories(stories, triggerlist)

list(map(get\_cont, stories))

scrollbar.config(command=cont.yview)

print(f"No keywords provided. Continuing to poll...")

time.sleep(SLEEPTIME)

except Exception as e:

print(f"Error occurred: {e}")

def get\_cont(newstory):

if newstory.get\_guid() not in guidShown:

cont.insert(END, newstory.get\_title() + "\n", "title")

cont.insert(END, "\n---------------------------------------------------------------\n", "title")

cont.insert(END, newstory.get\_description())

cont.insert(END, "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", "title")

guidShown.append(newstory.get\_guid())

while True:

print("Polling...")

stories = process("http://news.google.com/news?output=rss")

stories.extend(process("http://news.yahoo.com/rss/topstories"))

stories = filter\_stories(stories, triggerlist)

list(map(get\_cont, stories))

scrollbar.config(command=cont.yview)

print(f"No keywords provided. Continuing to poll...")

time.sleep(SLEEPTIME)

except Exception as e:

print(f"Error occurred: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

root.title("RSS Feed Filter")

keywords = input("Enter keywords (comma-separated): ").strip().split(',')

keywords = [keyword.strip() for keyword in keywords if keyword.strip()]

t = threading.Thread(target=main\_thread, args=(root, keywords))

t.start()

root.mainloop()

SCREENSHOTS:

A black screen with white text

Description automatically generatedA screenshot of a news form

Description automatically generated