

**FOP PROJECT REPORT**

|  |  |
| --- | --- |
| **NAME** | **CMS** |
| Abdul Moiz | 474550 |
| Daniyal Ahmed | 457165 |
| Abeeha Zehra | 470226 |
| Mehwish Nadeem | 455426 |
| Shahzaib Murtaza | 466034 |

**Introduction**

This project involves the development of an RSS feed filter that processes news stories from various sources, applies user-specified triggers to filter relevant news items, and displays the filtered news in a GUI. The system is modular, with distinct classes and functions handling specific tasks such as fetching and parsing RSS feeds, defining triggers based on various criteria, and filtering and displaying news stories. The project is divided among five team members—Abdul Moiz, Abeeha, Daniyal, Shahzaib, and Mehwish—each responsible for different components of the system. This division facilitates parallel development and ensures a comprehensive implementation of the RSS feed filter.

**Division of work and its working**

**Abdul Moiz: RSS Feed Processing and NewsStory Class**

**Responsibilities:**

Handle the processing of RSS feeds and parsing them into NewsStory objects.

Define the NewsStory class to encapsulate the news item details.

**Code:**

import feedparser

from project\_util import translate\_html

from datetime import datetime

import pytz

def process(url):

"""

Fetches news items from the rss url and parses them.

Returns a list of NewsStory-s.

"""

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

guid = entry.guid

title = translate\_html(entry.title)

link = entry.link

description = translate\_html(entry.description)

pubdate = translate\_html(entry.published)

try:

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

pubdate.replace(tzinfo=pytz.timezone("GMT"))

except ValueError:

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

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

ret.append(newsStory)

return ret

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

**Abeeha: Trigger Base Class and Phrase Triggers**

**Responsibilities:**

Define the base Trigger class and the PhraseTrigger class.

Implement methods to check if a phrase is present in a given text.

**Code:**

import string

class Trigger(object):

def evaluate(self, story):

"""

Returns True if an alert should be generated

for the given news item, or False otherwise.

"""

raise NotImplementedError

class PhraseTrigger(Trigger):

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

self.phrase = phrase.lower()

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

text = text.lower()

for punctuation in string.punctuation:

text = text.replace(punctuation, ' ')

phrase\_words = self.phrase.split()

text\_words = text.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

**Daniyal: TitleTrigger, DescriptionTrigger, and TimeTrigger**

**Responsibilities:**

Define the TitleTrigger and DescriptionTrigger classes to evaluate triggers based on title and description.

Implement the TimeTrigger base class and its subclasses for before and after time triggers.

**Code:**

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\_str):

self.time = datetime.strptime(time\_str, "%d %b %Y %H:%M:%S")

self.time = pytz.timezone("EST").localize(self.time)

class BeforeTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate().replace(tzinfo=pytz.timezone('GMT')) < self.time

class AfterTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate().replace(tzinfo=pytz.timezone('GMT')) > self.time

**Shahzaib: Composite Triggers and Filter Stories**

**Responsibilities:**

Implement composite triggers: NotTrigger, AndTrigger, and OrTrigger.

Define the function to filter stories based on a list of triggers.

**Code:**

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)

def filter\_stories(stories, triggerlist):

"""

Takes in a list of NewsStory instances.

Returns: a list of only the stories for which a trigger in triggerlist fires.

"""

filtered\_stories = []

for story in stories:

for trigger in triggerlist:

if trigger.evaluate(story):

filtered\_stories.append(story)

break

return filtered\_stories

**Mehwish: User-Specified Triggers and Main Thread**

**Responsibilities:**

Implement the function to read trigger configurations from a file.

Define the main thread to fetch, filter, and display news stories using the defined triggers.

**Code:**

import time

import threading

from mtTkinter import \*

from datetime import datetime

import pytz

def read\_trigger\_config(filename):

"""

filename: the name of a trigger configuration file

Returns: a list of trigger objects specified by the trigger configuration

file.

"""

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\_dict = {}

trigger\_list = []

def create\_trigger(trigger\_type, args):

if trigger\_type == "TITLE":

return TitleTrigger(\*args)

elif trigger\_type == "DESCRIPTION":

return DescriptionTrigger(\*args)

elif trigger\_type == "AFTER":

return AfterTrigger(\*args)

elif trigger\_type == "BEFORE":

return BeforeTrigger(\*args)

elif trigger\_type == "NOT":

return NotTrigger(trigger\_dict[args[0]])

elif trigger\_type == "AND":

return AndTrigger(trigger\_dict[args[0]], trigger\_dict[args[1]])

elif trigger\_type == "OR":

return OrTrigger(trigger\_dict[args[0]], trigger\_dict[args[1]])

for line in lines:

elements = line.split(",")

name = elements[0]

action = elements[1]

args = elements[2:]

if action == "ADD":

for arg in args:

trigger\_list.append(trigger\_dict[arg])

else:

trigger\_type = action

trigger = create\_trigger(trigger\_type, args)

trigger\_dict[name] = trigger

return trigger\_list

SLEEPTIME = 120 #seconds -- how often we poll

def main\_thread(master):

try:

t1 = TitleTrigger("election")

t2 = DescriptionTrigger("Trump")

t3 = DescriptionTrigger("Clinton")

t4 = AndTrigger(t2, t3)

triggerlist = [t1, t4]

triggerlist = read\_trigger\_config('triggers.txt')

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=root.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 . . .", end=' ')

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("Sleeping...")

time.sleep(SLEEPTIME)

except Exception as e:

print(e)

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

root = Tk()

root.title("Some RSS parser")

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

t.start()

root.mainloop()

**Conclusion**

The RSS feed filter project is a collaborative effort that leverages modular design to efficiently process and filter news stories based on user-defined triggers. By dividing the responsibilities among team members, we have ensured that each aspect of the system, from fetching and parsing RSS feeds to defining and evaluating triggers and ultimately filtering and displaying news stories, is robust and well-implemented. This project not only demonstrates effective teamwork but also showcases our ability to develop a functional and user-friendly application that meets the specified requirements. The resulting system is a powerful tool for users to stay informed about news stories that matter most to them, filtered through a customizable set of triggers.