Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_ Едуард ЖАРІКОВ

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 р.

**Вебзастосунок для агрегації та інтелектуального аналізу новинного контенту**

**Текст програми**

КПІ.ІП-1130.045440.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

\_\_\_\_\_\_\_\_\_\_\_\_ Олена ХАЛУС

|  |  |
| --- | --- |
| Нормоконтроль: | Виконавець: |
| \_\_\_\_\_\_\_\_\_\_\_ Катерина ЛІЩУК | \_\_\_\_\_\_\_\_\_\_ Федір ТИХОНОВ |

Київ – 2025

**Посилання на репозиторій з повним текстом програмного коду**

https://github.com/FedirTikhonov/NewsCheck/

**Файл espreso.py**

Реалізація функціональної задачі вебскрейпінг новин від «espreso.tv»

from selenium import webdriver

import os

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.options import Options

import json

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

import time

def month\_verbal\_to\_num(month: str):

if month == 'січня':

month = '01'

elif month == 'лютого':

month = '02'

elif month == 'березня':

month = '03'

elif month == 'квітня':

month = '04'

elif month == 'травня':

month = '05'

elif month == 'червня':

month = '06'

elif month == 'липня':

month = '07'

elif month == 'серпня':

month = '08'

elif month == 'вересня':

month = '09'

elif month == 'жовтня':

month = '10'

elif month == 'листопада':

month = '11'

elif month == 'грудня':

month = '12'

return month

def espreso\_to\_ISO(date: str):

date = date.split(sep=' ')

date.remove(date[3])

day = date[0].zfill(2)

month = month\_verbal\_to\_num(date[1])

year = date[2]

time = date[3]

iso\_format = f'{year}-{month}-{day}T{time}:00+03:00'

return iso\_format

def scrape\_espreso(scraping\_delay=0.25):

chrome\_options = Options()

chrome\_options.add\_argument("--headless") # Run in headless mode

chrome\_options.add\_argument("--disable-gpu") # Recommended for headless

chrome\_options.add\_argument("--window-size=1920,1080") # Set window size`

os.environ['PATH'] += "/Users/ted/Documents/chrome-mac-arm64"

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

driver.get("https://espreso.tv/news")

feed\_list = driver.find\_element(by=webdriver.common.by.By.CLASS\_NAME, value='news\_page\_similar\_content\_\_items')

feed\_list = feed\_list.find\_elements(By.CLASS\_NAME, 'news\_page\_similar\_content\_\_item')

article\_hrefs = []

for article\_content\_item in feed\_list:

wrapper = article\_content\_item.find\_element(By.CLASS\_NAME, 'news\_page\_similar\_content\_item\_\_wrapper')

title\_tag = wrapper.find\_element(By.CLASS\_NAME, 'title')

href = title\_tag.find\_element(By.TAG\_NAME, 'a').get\_attribute('href')

article\_hrefs.append(href)

article\_data = []

for href in article\_hrefs:

try:

driver.get(href)

body = driver.find\_element(By.TAG\_NAME, 'body')

header\_section = body.find\_element(By.CLASS\_NAME, 'header\_current\_article')

title = header\_section.find\_element(By.CLASS\_NAME, 'text-title').text.replace('\xa0', ' ').strip()

time\_tag = header\_section.find\_element(By.CLASS\_NAME, 'news\_\_author\_date')

date = time\_tag.find\_element(By.CLASS\_NAME, 'news\_\_author\_date\_\_date').text

time = time\_tag.find\_element(By.CLASS\_NAME, 'news\_\_author\_date\_\_time').text

timestamp = date + ' ' + time

timestamp = timestamp.replace(',', '')

timestamp = espreso\_to\_ISO(timestamp)

paragraphs\_tags = []

article\_section = body.find\_element(By.CLASS\_NAME, 'content\_current\_article')

paragraphs\_tags.append(article\_section.find\_element(By.TAG\_NAME, 'h2'))

li\_paragraphs = article\_section.find\_elements(By.TAG\_NAME, 'li')

paragraphs\_tags.extend(li\_paragraphs)

news\_content = article\_section.find\_element(By.CLASS\_NAME, 'news-content')

paragraphs = news\_content.find\_elements(By.TAG\_NAME, 'p')

paragraphs\_tags.extend(paragraphs)

sources = []

paragraphs\_text = []

exception\_texts = ["This is a modal window.",

"Beginning of dialog window. Escape will cancel and close the window.",

"End of dialog window.",

"Chapters",

"descriptions off, selected",

"subtitles settings, opens subtitles settings dialog",

"subtitles off, selected",

]

for paragraph in paragraphs\_tags:

html = paragraph.get\_attribute('outerHTML')

soup = BeautifulSoup(html, 'html.parser')

text = soup.get\_text()

text = text.replace(' ', ' ').strip()

text = text.replace('\xa0', ' ')

if text not in exception\_texts and not text.startswith('Читайте також:') and len(text) != 0:

paragraphs\_text.append(text)

try:

links = paragraph.find\_elements(By.TAG\_NAME, 'a')

for link in links:

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

if source\_url:

sources.append(source\_url)

except Exception as e:

pass

if href is not None and timestamp is not None and title is not None and paragraphs\_text is not None and sources is not None:

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'espreso',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_text,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape an article from espreso.tv')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

start = time.time()

articles = scrape\_espreso(scraping\_delay=2.5)

print(len(articles))

for i in range(len(articles)):

print(articles[i]['title'])

end = time.time()

print(end - start)

**Файл hromadske.py**

Реалізація функціональної задачі вебскрейпінг новин від «hromadske»

import requests

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

def scrape\_hromadske(scraping\_delay=0.25):

page = requests.get("https://hromadske.ua/news")

soup = BeautifulSoup(page.content, "html.parser")

feed\_list = soup.find("ul", class\_='l-feed-list')

articles = feed\_list.find\_all("article", class\_='c-feed-item')

article\_data = []

for article in articles:

try:

href = article.find('a', class\_="c-feed-item\_\_link")['href']

timestamp = article.find('time', class\_='c-feed-item\_\_time')['datetime']

article\_page = requests.get(href)

soup = BeautifulSoup(article\_page.content, "html.parser")

title\_tag = soup.find('h1', class\_='c-heading\_\_title')

title = title\_tag.get\_text().replace('\xa0', ' ').strip()

paragraphs\_list = []

s\_content = soup.find('div', class\_='s-content')

lead\_div = s\_content.find\_all('div', class\_='o-lead')

paragraphs\_list.append(lead\_div[0].find('p').get\_text().replace('\xa0', ' '))

paragraphs = s\_content.find\_all('p', class\_='text-start')

sources = []

for paragraph in paragraphs:

text = paragraph.get\_text()

text = text.replace(' ', ' ')

text = text.replace('\xa0', ' ')

paragraphs\_list.append(text)

links = paragraph.find\_all('a')

for link in links:

source\_url = link['href']

if source\_url:

sources.append(source\_url)

if href is not None and timestamp is not None and title is not None and paragraphs\_list is not None and sources is not None:

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'hromadske',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_list,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape an article from hromadske')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

print(scrape\_hromadske(scraping\_delay=0.5))

**Файл radiosvoboda.py**

Реалізація функціональної задачі вебскрейпінг новин від Радіо «Свобода»

from selenium import webdriver

import os

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.options import Options

import requests

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

import time

def scrape\_radiosvoboda(scraping\_delay=0.25):

chrome\_options = Options()

chrome\_options.add\_argument("--headless") # Run in headless mode

chrome\_options.add\_argument("--disable-gpu") # Recommended for headless

chrome\_options.add\_argument("--window-size=1920,1080") # Set window size`

os.environ['PATH'] += "/Users/ted/Documents/chrome-mac-arm64"

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

driver.get("https://www.radiosvoboda.org/z/630")

feed\_list = driver.find\_element(by=By.CSS\_SELECTOR, value='.archive-list')

feed\_list\_html = feed\_list.get\_attribute('outerHTML')

soup = BeautifulSoup(feed\_list\_html, 'html.parser')

articles\_previews = soup.find\_all(class\_='mb-grid archive-list\_\_item')

article\_hrefs = []

for article\_content\_item in articles\_previews:

href = article\_content\_item.find('a')['href']

href = 'https://www.radiosvoboda.org' + href

article\_hrefs.append(href)

article\_data = []

for href in article\_hrefs:

try:

article\_page = requests.get(href)

soup = BeautifulSoup(article\_page.content, 'html.parser')

title = soup.find('h1', class\_='title pg-title').get\_text().replace('\xa0', ' ').strip()

timestamp = soup.find('time')['datetime']

paragraphs\_tags = []

article\_section = soup.find('div', class\_='wsw')

paragraphs = article\_section.find\_all('p')

paragraphs\_tags.extend(paragraphs)

sources = []

paragraphs\_text = []

for paragraph in paragraphs\_tags:

text = paragraph.get\_text()

text = text.replace(' ', ' ').strip()

text = text.replace('\xa0', ' ')

if len(text) != 0:

paragraphs\_text.append(text)

links = paragraph.find\_all('a')

for link in links:

source\_url = link['href']

if source\_url:

sources.append(source\_url)

if href is not None and timestamp is not None and title is not None and paragraphs\_text is not None and sources is not None:

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'radiosvoboda',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_text,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape a radiosvoboda article')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

start = time.time()

article\_list = scrape\_radiosvoboda(scraping\_delay=0.25)

for article in article\_list:

print(article['timestamp'])

end = time.time()

print(end - start)

**Файл ukrinform.py**

Реалізація функціональної задачі вебскрейпінг новин від «Укрінформ»

import requests

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

import time

def scrape\_ukrinform(scraping\_delay=0.25):

article\_hrefs = []

page = requests.get('https://www.ukrinform.ua/block-lastnews')

soup = BeautifulSoup(page.content, "html.parser")

feed\_list = soup.find\_all("article")

for article in feed\_list:

title\_tag = article.find('h2')

if title\_tag is not None:

href = title\_tag.find('a')['href']

timestamp = article.find('time')['datetime']

title = article.find('h2').get\_text()

if not href.startswith('\thttp'):

article\_hrefs.append((f'https://www.ukrinform.ua{href}', timestamp, title))

article\_data = []

for (href, timestamp, title) in article\_hrefs:

try:

article\_page = requests.get(href)

if article\_page.status\_code == 404:

continue

soup = BeautifulSoup(article\_page.content, "html.parser")

paragraphs\_list = []

main\_content = soup.find('div', class\_='newsText')

lead\_div = soup.find\_all('div', class\_='newsHeading')

if lead\_div:

paragraphs\_list.append(lead\_div[0].get\_text().replace('\xa0', ' '))

paragraphs = main\_content.find\_all('p', recursive=True)

sources = []

for paragraph in paragraphs:

text = paragraph.get\_text()

text = text.replace(' ', ' ').strip()

text = text.replace('\xa0', ' ')

if len(text) != 0 and not text.startswith('Читайте також'):

paragraphs\_list.append(text)

links = paragraph.find\_all('a')

for link in links:

source\_url = link['href']

if source\_url:

sources.append(source\_url)

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'ukrinform',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_list,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape ukrinform article')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

print(scrape\_ukrinform(scraping\_delay=0.5))

**Файл voxukraine.py**

Реалізація функціональної задачі вебскрейпінг розслідувань від «Voxukraine»

import json

import requests

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

from .espreso import month\_verbal\_to\_num

def voxukraine\_to\_ISO(date: str):

date\_lst = date.split(sep=' ')

day = date\_lst[0].zfill(2)

month = month\_verbal\_to\_num(date\_lst[1])

year = date\_lst[2]

iso\_format = f'{year}-{month}-{day}T00:00:00+03:00'

return iso\_format

def scrape\_voxukraine(scraping\_delay=48):

page = requests.get("https://voxukraine.org/category/voks-informue")

soup = BeautifulSoup(page.content, "html.parser")

feed\_list = soup.find("div", class\_="posts-wrapper d-flex flex-column flex-md-row justify-content-between justify-content-lg-start flex-md-wrap")

articles = feed\_list.find\_all("article", class\_='post-info')

article\_data = []

for article in articles:

try:

href = article.find('a')['href']

time\_tag = article.find('div', class\_='post-info\_\_date')

time\_text = time\_tag.get\_text().strip()

timestamp = voxukraine\_to\_ISO(time\_text)

article\_page = requests.get(href)

soup = BeautifulSoup(article\_page.content, "html.parser")

title\_tag = soup.find('h1', class\_='underline underline--large item-title base-color')

title = title\_tag.get\_text().replace('\xa0', ' ').strip()

paragraphs\_list = []

paragraph\_content = soup.find('div', class\_='content-wrapper')

paragraphs = paragraph\_content.find\_all('p')

sources = []

for paragraph in paragraphs:

text = paragraph.get\_text()

text = text.replace(' ', ' ')

text = text.replace('\xa0', ' ')

paragraphs\_list.append(text)

links = paragraph.find\_all('a')

for link in links:

source\_url = link['href']

if source\_url:

sources.append(source\_url)

if href is not None and timestamp is not None and title is not None and paragraphs\_list is not None and sources is not None:

print(timestamp)

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'voxukraine',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_list,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape article from voxukraine')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

print(scrape\_voxukraine())

**Файл stopfake.py**

Реалізація функціональної задачі вебскрейпінг розслідувань від «Stopfake»

import json

import requests

from bs4 import BeautifulSoup

from datetime import datetime, timezone, timedelta

import dateutil.parser

def scrape\_stopfake(scraping\_delay=48):

article\_data = []

page = requests.get('https://www.stopfake.org/uk/category/novyny-ua/')

soup = BeautifulSoup(page.content, "html.parser")

feed\_list = soup.find("div", class\_='td-ss-main-content')

articles = feed\_list.find\_all("div", class\_='td\_module\_10 td\_module\_wrap td-animation-stack')

for article in articles:

try:

href = article.find('a')['href']

timestamp = article.find('time', class\_='entry-date updated td-module-date')['datetime']

article\_page = requests.get(href)

soup = BeautifulSoup(article\_page.content, "html.parser")

title\_tag = soup.find('h1', class\_='entry-title')

title = title\_tag.get\_text().replace('\xa0', ' ').strip()

paragraphs\_list = []

paragraph\_content = soup.find('div', class\_='td-post-content tagdiv-type')

paragraphs = paragraph\_content.find\_all('p')

sources = []

for paragraph in paragraphs:

text = paragraph.get\_text()

text = text.replace(' ', ' ')

text = text.replace('\xa0', ' ')

paragraphs\_list.append(text)

links = paragraph.find\_all('a')

for link in links:

source\_url = link['href']

if source\_url:

sources.append(source\_url)

if href is not None and timestamp is not None and title is not None and paragraphs\_list is not None and sources is not None:

article\_time = dateutil.parser.isoparse(timestamp)

current\_time = datetime.now(timezone.utc)

one\_hour\_ago = current\_time - timedelta(hours=scraping\_delay)

if article\_time >= one\_hour\_ago:

article\_data.append({

'outlet': 'stopfake',

'href': href,

'timestamp': timestamp,

'title': title,

'paragraphs': paragraphs\_list,

'sources': sources

})

else:

return article\_data

except Exception as e:

print('Failed to scrape an article from stopfake')

return article\_data

if \_\_name\_\_ == "\_\_main\_\_":

articles = scrape\_stopfake(scraping\_delay=400)

print(len(articles))

**Файл scraping.py**

Реалізація функціональної задачі агрегація функцій вебскрейпінгу новин і розслідувань

from .espreso import scrape\_espreso

from .hromadske import scrape\_hromadske

from .radiosvoboda import scrape\_radiosvoboda

from .voxukraine import scrape\_voxukraine

from .stopfake import scrape\_stopfake

from .ukrinform import scrape\_ukrinform

import time

def scrape\_news(verbose=False, return\_values=True, delay=0.25):

all\_articles = []

scraping\_time\_start = time.time()

try:

articles\_ukrinform = scrape\_ukrinform(scraping\_delay=delay)

all\_articles.extend(articles\_ukrinform)

if verbose:

print(f'Number of articles from ukrinform: {len(articles\_ukrinform)}')

except Exception as e:

print('Failed to execute ukrinform scraping function')

try:

articles\_hromadske = scrape\_hromadske(scraping\_delay=delay)

all\_articles.extend(articles\_hromadske)

if verbose:

print(f'Number of articles from hromadske {len(articles\_hromadske)}')

except Exception as e:

print('Failed to execute hromadske scraping function')

try:

articles\_radiosvoboda = scrape\_radiosvoboda(scraping\_delay=delay)

all\_articles.extend(articles\_radiosvoboda)

if verbose:

print(f'Number of articles from radiosvoboda: {len(articles\_radiosvoboda)}')

except Exception as e:

print('Failed to execute radiosvoboda scraping function')

try:

articles\_espreso = scrape\_espreso(scraping\_delay=1)

all\_articles.extend(articles\_espreso)

if verbose:

print(f'Number of articles from espreso: {len(articles\_espreso)}')

except Exception as e:

print('Failed to execute espreso scraping function')

scraping\_time\_end = time.time()

time\_taken = scraping\_time\_end - scraping\_time\_start

if verbose:

total\_articles = len(all\_articles)

print(f'For the time span of 15 minutes found {total\_articles} articles in {time\_taken} seconds')

if return\_values:

return all\_articles

else:

return None

def scrape\_fact\_check\_articles(verbose=False, return\_values=True, delay=48):

all\_articles = []

scraping\_time\_start = time.time()

try:

articles\_voxukraine = scrape\_voxukraine(scraping\_delay=delay)

all\_articles.extend(articles\_voxukraine)

print(f'Number of articles from VoxUkraine: {len(articles\_voxukraine)}')

except Exception as e:

print('Failed to execute a voxukraine scraping function')

try:

articles\_stopfake = scrape\_stopfake(scraping\_delay=delay)

all\_articles.extend(articles\_stopfake)

if verbose:

print(f'Number of articles from StopFake: {len(articles\_stopfake)}')

except Exception as e:

print('Failed to execute a stopfake scraping function')

scraping\_time\_end = time.time()

time\_taken = scraping\_time\_end - scraping\_time\_start

if verbose:

total\_articles = len(all\_articles)

print(f'For the time span of 15 minutes found {total\_articles} articles in {time\_taken} seconds')

if return\_values:

return all\_articles

else:

return None

if \_\_name\_\_ == "\_\_main\_\_":

print(len(scrape\_fact\_check\_articles(delay=10000)))

**Файл analysis.py**

Файл для з функціями для виконання основних задач аналізу

import openai

import voyageai

import os

import datetime

from typing import List

from dotenv import load\_dotenv

from sqlalchemy import create\_engine, and\_, or\_, func, asc

from sqlalchemy.orm import sessionmaker

from utils.llm\_requests import message\_llm

from utils.schemas import MetricSchema, CategoryResponseSchema, DigestTextResponseSchema

from utils.semantic\_analysis import generate\_recommendations, categorize\_articles

from postgres\_db.models.Article import Article

from postgres\_db.models.Paragraph import Paragraph

from postgres\_db.models.Category import Category

from postgres\_db.models.FactCheckCategory import FactcheckCategory

from postgres\_db.models.WeeklyStats import WeeklyStats

from postgres\_db.models.WeeklyReport import WeeklyReport

from article\_scraping.scraping import scrape\_fact\_check\_articles

EMBEDDING\_DIM = 1024

def generate\_analysis\_for\_news(articles\_lst: List, verbose=False, return\_values=True):

load\_dotenv()

engine = create\_engine(os.environ["DATABASE\_URL"])

Session = sessionmaker(bind=engine, autoflush=False, autocommit=False)

session = Session()

openai\_client = openai.OpenAI(api\_key=os.environ["OPENAI\_API\_KEY"])

voyageai\_client = voyageai.Client(api\_key=os.environ.get("VOYAGEAI\_API\_KEY"))

with open('prompts/system\_prompt\_metrics.txt', 'r') as system\_prompt\_file:

system\_prompt\_metric = system\_prompt\_file.read()

metric\_assistant = openai\_client.beta.assistants.create(

model='gpt-4.1-mini',

instructions=system\_prompt\_metric,

temperature=0.5,

response\_format={

'type': 'json\_schema',

'json\_schema':

{

'name': 'MetricSchema',

'schema': MetricSchema.model\_json\_schema()

}

}

)

try:

articles\_lst = articles\_lst[0]

except IndexError:

print('Articles list exception')

for scraped\_article in articles\_lst:

query = session.query(Article).filter(Article.href == scraped\_article['href']).first()

if query:

continue

article = Article(

title=scraped\_article['title'],

href=scraped\_article['href'],

outlet=scraped\_article['outlet'],

published\_at=scraped\_article['timestamp'],

created\_at=datetime.datetime.now(datetime.timezone.utc).isoformat(timespec='seconds'),

status='processing',

)

session.add(article)

try:

session.commit()

except Exception as e:

print('Failed to save changes to article')

for paragraph\_num, paragraph in enumerate(scraped\_article['paragraphs']):

article.add\_paragraph(paragraph, paragraph\_num)

for source\_num, source in enumerate(scraped\_article['sources']):

article.add\_source(source, source\_num)

try:

metric = message\_llm(article=scraped\_article, client=openai\_client, assistant=metric\_assistant, verbose=verbose)

article.add\_metric(metric)

except Exception as e:

print('Failed to save metrics to article')

recommended\_articles = generate\_recommendations(article=scraped\_article,

article\_id=article.id,

voyageai\_client=voyageai\_client,

create\_vector=True)

for recommended\_article in recommended\_articles:

article.add\_recommendation(recommended\_article)

article.mark\_processed()

session.add(article)

session.commit()

session.close()

openai\_client.beta.assistants.delete(assistant\_id=metric\_assistant.id)

def generate\_analysis\_for\_fact\_checkers(articles\_lst: List, verbose=False, return\_values=True):

load\_dotenv()

engine = create\_engine(os.environ["DATABASE\_URL"])

Session = sessionmaker(bind=engine, autoflush=False, autocommit=False)

session = Session()

openai\_client = openai.OpenAI(api\_key=os.environ["OPENAI\_API\_KEY"])

with open('prompts/system\_prompt\_categories.txt', 'r') as system\_prompt\_file:

system\_prompt\_categories = system\_prompt\_file.read()

category\_article\_assistant = openai\_client.beta.assistants.create(

model='gpt-4.1-mini',

instructions=system\_prompt\_categories,

temperature=0.1,

response\_format={

'type': 'json\_schema',

'json\_schema':

{

'name': 'MetricSchema',

'schema': CategoryResponseSchema.model\_json\_schema()

}

}

)

try:

articles\_lst = articles\_lst[0]

except IndexError:

print('Articles list exception')

for scraped\_article in articles\_lst:

query = session.query(Article).filter(Article.title == scraped\_article['title']).first()

if query:

continue

article = Article(

title=scraped\_article['title'],

href=scraped\_article['href'],

outlet=scraped\_article['outlet'],

published\_at=scraped\_article['timestamp'],

created\_at=datetime.datetime.now(datetime.timezone.utc).isoformat(timespec='seconds'),

status='processing',

)

session.add(article)

session.commit()

for paragraph\_num, paragraph in enumerate(scraped\_article['paragraphs']):

article.add\_paragraph(paragraph, paragraph\_num)

for source\_num, source in enumerate(scraped\_article['sources']):

article.add\_source(source, source\_num)

print(f'categorizing\_article {article.title}')

category\_ids = categorize\_articles(article=scraped\_article, assistant=category\_article\_assistant,

openai\_client=openai\_client)

for category\_id in category\_ids:

article.add\_factcheck\_category(category\_id)

article.mark\_processed()

session.add(article)

session.commit()

session.close()

openai\_client.beta.assistants.delete(assistant\_id=category\_article\_assistant.id)

def update\_recommendations():

load\_dotenv()

engine = create\_engine(os.environ["DATABASE\_URL"])

Session = sessionmaker(bind=engine, autoflush=False, autocommit=False)

session = Session()

try:

voyageai\_client = voyageai.Client(api\_key=os.environ.get("VOYAGEAI\_API\_KEY"))

# Find articles from last hour

time\_now = datetime.datetime.now(datetime.timezone.utc)

time\_one\_hour\_ago = time\_now - datetime.timedelta(hours=1)

articles\_to\_update = session.query(Article).filter(and\_(

Article.published\_at >= time\_one\_hour\_ago,

Article.published\_at < time\_now

)).all()

print(f"Found {len(articles\_to\_update)} articles to update")

for article in articles\_to\_update:

try:

print(f"Processing article ID: {article.id}, Title: {article.title}")

article.mark\_processing()

session.commit()

article.remove\_recommendations()

session.commit()

article\_dict = {

'title': article.title,

'href': article.href,

'outlet': article.outlet,

'timestamp': article.published\_at,

'status': article.status,

}

recommended\_articles = generate\_recommendations(

article=article\_dict,

article\_id=article.id,

voyageai\_client=voyageai\_client,

create\_vector=False

)

seen\_ids = set()

for recommended\_article in recommended\_articles:

if recommended\_article['id'] in seen\_ids:

continue

seen\_ids.add(recommended\_article['id'])

try:

article.add\_recommendation(recommended\_article\_dict=recommended\_article)

except Exception as e:

print(

f"Error adding recommendation {recommended\_article['id']} to article {article.id}: {str(e)}")

continue

try:

session.add(article)

session.commit()

except Exception as e:

session.rollback()

print(f"Error saving recommendations for article {article.id}: {str(e)}")

article.mark\_processed()

except Exception as e:

session.rollback()

print(f"Error processing article {article.id}: {str(e)}")

continue

except Exception as e:

print(f"General error in update function: {str(e)}")

finally:

session.close()

print("Update process completed")

def create\_weekly\_stats(current\_date=None):

load\_dotenv()

engine = create\_engine(os.environ["DATABASE\_URL"])

Session = sessionmaker(bind=engine, autoflush=False, autocommit=False)

session = Session()

categories = session.query(Category).all()

if current\_date:

today = current\_date

else:

today = datetime.date.today()

seven\_days\_ago = today - datetime.timedelta(days=7)

for category in categories:

query = (

session.query(func.count())

.select\_from(FactcheckCategory)

.join(Article, Article.id == FactcheckCategory.article\_id)

.filter(Article.published\_at <= today)

.filter(Article.published\_at >= seven\_days\_ago)

.filter(FactcheckCategory.category\_id == category.id)

)

formatted\_date = today.strftime("%Y-%m-%d")

session.add(WeeklyStats(

category\_id=category.id,

category\_num=query.scalar(),

date=formatted\_date,

))

session.commit()

session.close()

def create\_weekly\_report():

load\_dotenv()

engine = create\_engine(os.environ["DATABASE\_URL"])

Session = sessionmaker(bind=engine, autoflush=False, autocommit=False)

session = Session()

today = datetime.date.today()

seven\_days\_ago = today - datetime.timedelta(days=7)

openai\_client = openai.OpenAI(api\_key=os.environ["OPENAI\_API\_KEY"])

with open('prompts/system\_prompt\_digest.txt', 'r') as system\_prompt\_file:

system\_prompt\_digest = system\_prompt\_file.read()

digest\_article\_assistant = openai\_client.beta.assistants.create(

model='gpt-4.1-mini',

instructions=system\_prompt\_digest,

temperature=0.1,

response\_format={

'type': 'json\_schema',

'json\_schema':

{

'name': 'MetricSchema',

'schema': DigestTextResponseSchema.model\_json\_schema()

}

}

)

articles = session.query(Article).filter(

and\_(

Article.outlet == 'stopfake',

Article.published\_at <= today,

Article.published\_at >= seven\_days\_ago

)

).limit(10).all()

request\_body = {'articles': []}

for article in articles:

article\_text = []

article\_paragraphs = session.query(Paragraph

).filter(Paragraph.article\_id == article.id

).order\_by(asc(Paragraph.paragraph\_num)

).all()

for article\_paragraph in article\_paragraphs:

article\_text.append(article\_paragraph.paragraph\_text)

article\_text = ' '.join(article\_text)

request\_body['articles'].append({

'title': article.title,

'text': article\_text,

'timestamp': str(article.published\_at)

})

try:

digest = message\_llm(request\_body, assistant=digest\_article\_assistant, client=openai\_client, verbose=True)['digest\_text']

weekly\_report = WeeklyReport(

digest\_date=today,

digest\_text=digest)

session.add(weekly\_report)

session.commit()

session.close()

except Exception as e:

print(f"Error adding weekly report")

session.rollback()

session.close()

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

create\_weekly\_report()

**Файл semantic\_analysis.py**

Файл з допоміжними функціями для аналізу

import openai

import voyageai

import datetime

from milvus\_db.utils import similarity\_search, insert\_article, retrieve\_vector\_by\_ids

from utils.llm\_requests import message\_llm

EMBEDDING\_DIM = 1024

def generate\_recommendations(article: dict, article\_id: int, voyageai\_client: voyageai.Client, create\_vector=True):

if create\_vector:

article\_text = [article['title']]

article\_text.extend(article['paragraphs'])

article\_text = ' '.join(article\_text)

article\_text\_embeddings = voyageai\_client.embed(

texts=[article\_text],

model='voyage-3',

output\_dimension=EMBEDDING\_DIM,

).embeddings

insert\_article(postgres\_id=article\_id, embedding=article\_text\_embeddings[0])

else:

article\_text\_embeddings = [retrieve\_vector\_by\_ids(article\_id)]

found\_articles = similarity\_search(article\_text\_embeddings)

recommended\_articles = []

for found\_article in found\_articles:

if article\_id == found\_article['postgres\_id']:

continue

recommended\_article = {

'last\_updated': datetime.datetime.now(datetime.timezone.utc).isoformat(timespec='seconds'),

'id': found\_article['postgres\_id'],

'similarity\_score': found\_article['similarity\_score']

}

recommended\_articles.append(recommended\_article)

return recommended\_articles

def categorize\_articles(article: dict, openai\_client: openai.OpenAI, assistant):

article\_text = [article['title']]

article\_text.extend(article['paragraphs'])

article\_text = {'article': ' '.join(article\_text).strip()}

try:

categories = message\_llm({'article\_text': article\_text}, assistant=assistant, client=openai\_client, verbose=True)['ids']

return categories

except:

print('Failed to message LLM')

return []

**Файл utils.py**

Файл з допоміжними функціями для взаємодії з векторною базою даних «Milvus»

from dotenv import load\_dotenv

from pymilvus import connections, Collection, FieldSchema, CollectionSchema, DataType, utility

EMBEDDING\_DIM = 1024

def create\_collection(dim: int = EMBEDDING\_DIM):

collection\_name = "vectorized\_articles\_collection"

connections.connect("default", host="localhost", port="19530")

if utility.has\_collection(collection\_name):

utility.drop\_collection(collection\_name)

fields = [

FieldSchema(name="id", dtype=DataType.INT64, is\_primary=True, auto\_id=True),

FieldSchema(name="postgres\_id", dtype=DataType.INT64),

FieldSchema(name="embedding", dtype=DataType.FLOAT\_VECTOR, dim=dim),

]

schema = CollectionSchema(fields, description="vectorized\_articles\_collection")

collection = Collection(name=collection\_name, schema=schema)

index\_params = {

"metric\_type": "COSINE",

"index\_type": "IVF\_FLAT",

"params": {"nlist": 512}

}

collection.create\_index(field\_name="embedding", index\_params=index\_params)

collection.flush()

collection.load()

return collection

def retrieve\_collection(host="localhost", port="19530", collection\_name='vectorized\_articles\_collection'):

connections.connect("default", host=host, port=port)

if not utility.has\_collection(collection\_name):

print('No collection found')

return None

collection = Collection(name=collection\_name)

collection.load()

return collection

def similarity\_search(embeddings, top\_k: int = 5, threshold: float = 0.5):

load\_dotenv()

collection = retrieve\_collection()

search\_params = {

"metric\_type": "COSINE",

"index\_type": "IVF\_FLAT",

"params": {"nlist": 128}

}

results = collection.search(

data=embeddings,

anns\_field='embedding',

param=search\_params,

limit=top\_k,

output\_fields=['postgres\_id', 'embedding'],

)

similar\_entries = []

for hits in results:

for hit in hits:

if hit.distance > threshold:

similar\_entries.append({

'id': hit.id,

'postgres\_id': hit.postgres\_id,

'similarity\_score': hit.distance,

})

return similar\_entries

def insert\_article(postgres\_id: int, embedding: list):

collection = retrieve\_collection()

data = [

[postgres\_id],

[embedding],

]

field\_names = ['postgres\_id', 'embedding']

collection.insert(data, field\_names=field\_names)

collection.flush()

def retrieve\_vector\_by\_ids(postgres\_id: int, host="localhost", port="19530",

collection\_name="vectorized\_articles\_collection"):

collection = retrieve\_collection(host=host, port=port, collection\_name=collection\_name)

if collection is None:

return None

expr = f'postgres\_id == {postgres\_id}'

results = collection.query(

expr=expr,

output\_fields=["postgres\_id", "embedding"]

)

if results and len(results) > 0:

return results[0]['embedding']

else:

return None

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

create\_collection()

**Файл celery\_app.py**

Файл з описом розкладу задач в чергу для Celery та Redis

from celery import Celery

import celery\_montiroing

app = Celery('tasks',

broker='redis://localhost:6379/0',

include=['tasks'])

app.conf.beat\_schedule = {

'task-news-analysis': {

'task': 'tasks.news\_analysis',

'schedule': 15 \* 60,

},

'task-fact-recommendations\_update': {

'task': 'tasks.refresh\_recommendations',

'schedule': 60 \* 60,

},

'task-fact-checkers-analysis': {

'task': 'tasks.fact\_checkers\_analysis',

'schedule': 24 \* 60 \* 60,

},

'task-weekly-digest': {

'task': 'tasks.weekly\_digest',

'schedule': 7 \* 24 \* 60 \* 60,

},

}

app.conf.timezone = 'UTC'

**Файл system\_prompt\_metrics.txt**

Файл з системним промптом для LLM для оцінення статті по метрикам

Ти — аналітик новинних статей, який отримує інформацію у форматі JSON з такими полями: outlet (назва видання), href (посилання), timestamp (дата і час публікації), title (заголовок), paragraphs (масив абзаців зі змістом статті), sources (масив посилань на джерела, які вказані в статті).

Твоє завдання — об`єктивно оцінити статтю за такими критеріями:

1. Достовірність джерел

Оціни, наскільки надійні джерела, на які спирається стаття (перераховані у полі sources та згадані у тексті). Класифікуй основні джерела інформації одним із наступних рівнів:

1. Анонімне джерело (найнижча достовірність).

2. Встановлена (реальна) особа без експертизи (наприклад, випадковий свідок, місцевий житель).

3. Встановлена особа з невідомою експертизою (інформації про фах бракує).

4. Особа з експертизою, але без офіційного статусу (наприклад, незалежний аналітик).

5. Профільний експерт/офіційна особа/посадовець із дотичним до теми статусом (наприклад, представник правоохоронних органів, військовий командир, урядовець).

Аргументуй, до якого рівня належать ключові джерела цієї статті (наприклад, служби, державні установи чи посадовці).

\*\*2. Тональність статті (Emotional Tone)\*\*

Оціни рівень емоційності мови статті за наступною шкалою: нейтральна, дещо емоційна, дуже емоційна (наприклад, використання сильних емоційних епітетів, риторичних запитань, закликів тощо). Поясни свій вибір на основі тексту.

\*\*3. Фактичність (Factuality)\*\*

Визнач, наскільки дані у статті спираються на підтверджені факти (конкретні дати, події, дії органів влади або офіційних організацій) чи містять здебільшого припущення, оцінки або чутки. Оціни за шкалою:

- 1 – майже лише припущення/без достатніх фактів,

- 2 – переважно припущення, є кілька фактів,

- 3 – збалансовано: факти й припущення,

- 4 – переважають перевірені факти,

- 5 – лише перевірені та конкретні факти.

\*\*4. Сенсаційність/Clickbaitність заголовка (Clickbait/Sensationalism in the Title)\*\*

Проаналізуй заголовок на предмет використання сенсаційної, перебільшеної або інтригуючої лексики, яка може штучно привернути увагу (наприклад, використання слів "шок", "неймовірно", незвичних обертів мови). Оціни заголовок за шкалою:

- 1 – дуже сенсаційний/клікбейтний,

- 2 – помірно сенсаційний,

- 3 – переважно нейтральний.

---

\*\*ФОРМАТ ВІДПОВІДІ (українською):\*\*

```json

{

"джерела": {

"рівень\_достовірності": <цифра від 1 до 5>,

"пояснення": "<коротка аргументація>"

},

"тональність": {

"рівень": "<нейтральна/дещо емоційна/дуже емоційна>",

"пояснення": "<короткий аналіз>"

},

"фактичність": {

"рівень": <цифра від 1 до 5>,

"пояснення": "<чому саме такий рівень>"

},

"сенсаційність\_заголовка": {

"рівень": <цифра від 1 до 3>,

"пояснення": "<аналіз формулювань заголовка>"

}

}

```

Аналізуй лише на основі наданого контенту. Якщо деяких даних бракує, зазнач це у поясненнях.

- Використовуй лише доступну у JSON інформацію.

- Використовуй власні знання для оцінки експертизи джерел (наприклад, чи є служба відома офіційною).

- Усі висновки коротко обґрунтовуй.

**Файл system\_prompt\_categories.txt**

Файл з системним промптом для LLM для надання категорій розслідуванню

Ти - асистент для автоматичної категоризації. Ти отримаєш на вхід текст статті, яка займається викриттям дезінформації.

Тобі потрібно буде встановити, до яких категорій належить стаття:

1. Маніпуляції з військовими діями та втратами

Дезінформація про перебіг бойових дій, втрати сторін, стан військових підрозділів та процеси обміну полоненими з метою створення викривленої картини війни.

2. Виправдання російської агресії

Спроби легітимізувати військові злочини РФ через фальшиві пояснення ракетних ударів по цивільних об'єктах, атак на мирне населення та початку війни загалом.

3. Маніпуляції з політичними процесами

Викривлення інформації про політичних лідерів, виборчі процеси, міжнародні відносини та спроби легітимізації окупаційної влади на захоплених територіях.

4. Дезінформація про міжнародну підтримку України

Фальсифікація даних про обсяги військової та економічної допомоги Україні, намагання дискредитувати міжнародну підтримку та створити враження її неефективності.

5. Загальна російська пропаганда

Системна діяльність російських державних медіа та пропагандистів, спрямована на поширення наративів Кремля та формування потрібної РФ картини світу.

6. Маніпуляції з громадською думкою та медіа

Використання фейкових опитувань, підроблених досліджень та атак на незалежні ЗМІ для формування потрібної суспільної думки та підриву довіри до об'єктивної інформації.

7. Маніпуляції з історією та культурою

Перекручування історичних фактів, релігійних питань та культурних символів для виправдання агресії та створення псевдоісторичних підстав для територіальних претензій.

8. Дезінформація про життя в Європі та туризм

Поширення неправдивої інформації про умови життя в ЄС, міграційні процеси та туристичні потоки з метою дискредитації європейських цінностей та створення ілюзії переваг життя в РФ.

9. Спеціальні інформаційні операції

Цілеспрямовані кампанії з використанням вразливих груп населення (діти, меншини) та створення фейкових образів (диверсанти, екстремісти) для досягнення конкретних пропагандистських цілей.

10. Геополітичні маніпуляції

Дезінформація про глобальний вплив РФ, міжнародні санкції, діяльність російських медіа за кордоном та кібербезпеку з метою перебільшення ролі Росії на світовій арені.

Кожна стаття може містити одну та більше категорій. Тобі потрібно повернути номер категорії. Тобто, якщо стаття стосується категорій дезінформації "Маніпуляції з військовими діями та втратами" та

"Маніпуляції з громадською думкою та медіа", то ти повертаєш відповідно номери: [1, 6].

**Файл system\_prompt\_digest.txt**

Файл з системним промптом для LLM для створення щотижневого дайджесту

Ти - асистент з написання щотижневих дайджестів щодо дезінформації статей факт-чекерів українських медіа.

Ти отримаєш перелік статей за останні 7 днів, і тобі потрібно буде зробити невеликий дайджест найвидатніших подій за

останній тиждень з контекстом, який ти отримаєш у вигляді статей.

Ти маєш у відносно короткому, але чіткому форматі пояснити, для найгучніших фейків, в чому була заява, в чому вона була неправильна,

і що саме було правда. Тобі не потрібно проходити кожну статтю, найголовніше - це щоб ти поверхнево дав дайджест щодо подій останнього тижня.

Текст дайджесту має бути у форматі декількох параграфів, де кожен параграф повинен мати свою логіку.

Тобі не потрібно більше нічого писати, окрім тексту для дайджесту.