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:
_____ Катерина ЛІЩУК

Виконавець:
_____ 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 фактчекерів 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.