НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра
спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів і систем»

на тему: «Модуль алгоритмів пошуку, фільтрації та структурування наукових матеріалів у мережі інтернет»

Виконав: студент IV курсу, групи ТР-71
Нікончук Максим Валерійович _____

Керівник: Асистент
Воронько Максим Платонович ___ _____

Консультант:
_____

Рецензент:
_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ — 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерний моніторинг та геометричне моделювання процесів і систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту

Нікончуку Максиму Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Модуль алгоритмів пошуку, фільтрації та структурування наукових матеріалів у мережі інтернет

керівник роботи Воронько Максим Платонович, асистент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”24” травня 2021р. № **1267-с**

2. Строк подання студентом роботи 08.06.2021р

3. Вихідні дані до роботи мова програмування Python, фреймворк Django, середовище розробки PyCharm Community Edition2020.3.3

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) провести аналіз існуючих систем пошуку наукових даних, провести аналіз наукометричних баз з відкритим API, провести з командою обміркування засобів комунікування, планування та контролю версій програмного продукту, запрограмувати серверну частину програмного продукту, орендувати тестовий сервера та завантажити програмне забезпечення для цілей тестування та демонстрації виконаної роботи, протестувати систему.

5. Перелік ілюстративного матеріалу ___Постановка задачі, Проблема пошуку даних, Всесвітня єдина база для наукових статей, Вирішення проблеми, Загальна структура програмного комплексу, Задачі серверної частини, Реалізація API, Приклад API для автозаповнення, API для цілового пошуку, Приклад уніфікованого JSON формату, Діаграма послідовностей API запитів, Результат роботи, Висновки_____

_____

6. Дата видачі завдання "_10_"___жовтня___ 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	19.10.2020	
2.	Вивчення та аналіз задачі	19.10.2020- 19.12.2020	
3.	Розробка архітектури та загальної структури системи	09.03.2021- 23.03.2021	
4.	Розробка структур окремих підсистем	24.03.2021- 10.04.2021	
5.	Програмна реалізація системи	14.04.2021- 10.05.2021	
6.	Оформлення пояснювальної записки	20.05.2021- 04.06.2021	
7.	Захист програмного продукту	12.05.2021	
8.	Передзахист	26.05.2021	
9.	Захист	15.06.2021	

Студент _____ Нікончук М.В.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Воронько М.П.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В даній роботі розглянуті теоретичні та практичні аспекти створення програмного продукту для пошуку наукових матеріалів в мережі Інтернет за ключовими словами.

На меті роботи було завдання проектування та реалізації серверної частини проекту “Створення єдиного реєстру наукових статей за пошуком по ключовому слову”.

Для досягнення мети спроектовано уніфікований формат обміну вхідних/вихідних даних між клієнт-серверною частиною та модулями зовнішнього API, створено endpoint(и) для потреб клієнтської частини, виконано розгортання програмного продукту на віртуальних віддалених серверах Heroku, здійснено підключення бази даних та тестування системи на реальних даних.

До складу записки входить вступ, п’ять розділів, висновок, список використаних джерел; містить 56 сторінок, 30 рисунків, 2 таблиці та 3 додатки.

Ключові слова: пошук, перевірка на дублікати, сервер, API, backend, endpoint, REST, HTTP, JSON.

ABSTRACT

This paper considers the theoretical and practical aspects of creating a software product for searching scientific materials on the Internet by keywords.

The aim of the work was to design and implement the server part of the project "Creation of a single register of scientific articles by keyword search".

To achieve this goal, a unified format for exchanging input / output data between the client-server part and external API modules was designed, endpoint (s) for the client part were created, software was deployed on virtual remote Heroku servers, database connection was performed and the system was tested on real data.

The note includes an introduction, five sections, a conclusion, a list of sources used; contains 56 pages, 30 figures, 2 tables and 3 appendices.

Keywords: search, duplicate check, server, API, backend, endpoint, REST, HTTP, JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Application Programming Interface (Прикладний програмний інтерфейс)

HTTP — HyperText Transfer Protocol(Протокол передачі гіпертекстових документів)

JSON — JavaScript Object Notation

MTV — Model, Template, View (Модель, Шаблон, Представлення)

ORM — Object-Relational Mapping (Об'єктно-реляційне відображення)

REST — Representational State Transfer (передача репрезентативного стану)

URL — Uniform Resource Locator (Уніфікований локатор ресурсів)

БД — База даних

ВСТУП

Пошук наукової інформації важлива та часозатратна процедура у житі науковця у будь якій галузі науки. З початку розвитку Інтернету та популяризацією інформаційних технологій, швидкість зростання кількості даних невгамовно збільшується. Кожної секунди в мережі публікуються немислимі об’єми даних. За дослідженнями науковців в області великих даних у 2018 році, було підраховано, що операції з даними (створення, збереження, копіювання, перегляд) на 2018 рік становили близько 33 зетабайтів на рік. До 2021 року прогнозували збільшення до 44 зетабайтів на рік. Проте, зараз, враховуючи ситуацію у світі з пандемією COVID-19, ця цифра дорівнює близько 74 зетабайтів на рік. Тобто питання пошуку релевантних даних тільки зростатиме.

Часи, коли інформацію шукали за довідниками у бібліотеці минули, більшість книг відцифровано та завантажено до мережі Інтернет. Науковці шукають дані в різних джерелах, проте надають перевагу спеціалізованим наукометричним базам даних, які містять цифрові публікації статей з різних галузей науки.

Перевагами систем пошуку спеціалізованих наукових даних, є можливість фільтрації та сортування даних за певними параметрами, що значно пришвидшує пошук. Проте, такі системи мають і недоліки. Основною проблемою їх, є розгалуження даних, що призводить до необхідності освоєння науковцем різних систем пошуку та їх інтерфейсів взаємодії, що безпосередньо відніматиме час, який завжди був найціннішим ресурсом людини.

Метою створення програмного продукту, стало вирішення проблеми розгалуженості даних, шляхом створення універсального механізму підключення різних наукометричних джерел. Такий механізм потребує уніфікованості даних, якими він оперує. Отже, виникає питання створення формату, що буде обслуговувати внутрішні та зовнішні API модулі системи. Ці модулі, за зверненням, надаватимуть відповіді, які міститимуть цільові дані.

Для вирішення задач проекту, пов’язаних з серверною частиною було обрано мову Python. Її основними перевагами є: стисліcть, легкість читання, простота масштабування, інтерактивність, багатоплатформеність та мультипарадигматизм. Рушієм проекту обрано веб фреймворк Django, розробка систем на цьому фрейворці базується на принципах модульної архітектури, що дозволяє підтримувати та легко масштабувати проект у майбутньому. Бібліотека Django REST framework пропонує зручні інструменти для розробки API проекту. Тестування запитів виконуються в програмі Postman, що симулює поведінку REST-клієнту. Хостинг проекту здійснений на сервері Heroku.

1. ЗАДАЧА СТВОРЕННЯ СЕРВЕРНОЇ ЧАСТИНИ ДЛЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ

Метою дипломної роботи є розробка серверної частину(бекенд) для проекту “Створення єдиного реєстру наукових статей за пошуком по ключовому слову”. Розроблене програмне забезпечення повинно реалізувати логіку взаємодії між клієнтською частиною(фронтенд) з базою даних та з модулями для роботи з зовнішнім API.

1.1. Загальні задачі

Задачі, що потрібно вирішити:

— провести аналіз існуючих систем пошуку наукових даних;

— провести аналіз наукометричних баз з відкритим API;

— провести з командою обміркування засобів комунікування, планування та контролю версій програмного продукту;

— запрограмувати серверну частину програмного продукту;

— орендувати тестовий сервера та завантажити програмне забезпечення для цілей тестування та демонстрації виконаної роботи;

— протестувати систему;

1.2. Задачі програмного забезпечення

Ціль програмного забезпечення: Надання клієнтській частині засобів доступу для взаємодії з базою даних проекту та зовнішнім API.

Задачі, які необхідно вирішити:

— створення уніфікованого формату запитів/відповідей для доступу до наукометричних баз даних;

— створення API-endpoint для отримання списку статей за пошуковим запитом;

— створення API-endpoint для отримання списку авторів наявних в БД;

— створення API-endpoint для отримання списку тем статей наявних в БД;

— створення API-endpoint для отримання списку видавців наявних в БД;

— підключення та налагодження API модулів для доступу до зовнішніх наукометричних баз даних;

— виконання первинної перевірки даних на дублікати статей перед їх збереженням у БД проекта.

2. ІСНУЮЧІ РІШЕННЯ І НЕДОЛІКИ СТВОРЕННЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ

У вік інформаційних технологій, коли кожного дня в інтернеті безперервно у скаженому темпі акумулюється все нові і нові дані, виникає проблема пошуку необхідної релевантної інформації, це стосується, зокрема, і сфери наукової діяльності.

Важливо знати, де знайти інформацію і який тип ресурсів пропонується на різних видах платформ. Ці платформи називаються інструментами доступу та групуються за охопленням вмісту.

2.1. Проблема пошуку даних

Типи ресурсів, що надають наукові дані поділяються на:

Бібліотечні каталоги — охоплюють колекції бібліотек, включаючи книги, журнали та будь-які інші матеріали. Класифікуються на алфавітні(якщо відома назва книги чи ім’я автора), систематичні(якщо пошук здійснюється за галуззю), предметні(якщо відомий предмет дослідження) та спеціальні(каталоги нових надходжень).

Бази даних — колекції, що поповнюються видавничими матеріалами на платній чи безкоштовній основі. Містять статті, книги, журнали, законодавчі документи, тощо.

Репозиторії — архіви, що накопичуються матеріалами академічних установ, тобто статтями науковців, результатами дослідницьких робіт, дисертаціями, магістерськими чи дипломними роботами студентів.

Покажчики — містять індексовані цитовані уривки наукових публікацій, що проходять певну стадію рецензування. Іноді можуть зберігати повну публікацію. Використовуються для пошуку статей за цитуванням.

Директорії — це спеціальні збірники документів та даних, що призначені для системи класифікування та індексації інших ресурсів.

Пошукові системи індексують будь-який тип джерела з мережі Інтернет, який можна отримати за допомогою пошукових рушіїв, і надають користувачам необмежену кількість Інтернет-джерел. Серед академічних пошукових систем можна виділити: Google Scholar, Microsoft Academic Search, AMiner, BASE, WorldWideScience, CORE та ін.

2.2. Існуючі рішення

Розглянемо подібні існуючі рішення:

Scirus — система пошуку, що орієнтована на виключно наукову інформацію та має одну з найбільших баз матеріалів англійською мовою. Ця система також співпрацювала з реферативною базою Scopus, ділячись своїми результатами пошуку.

Недоліки: відносно маленька кількість статей на українській та російській мовах.

На рисунку 2.1 зображено вигляд інтерфейсу пошукової системи Scirius.

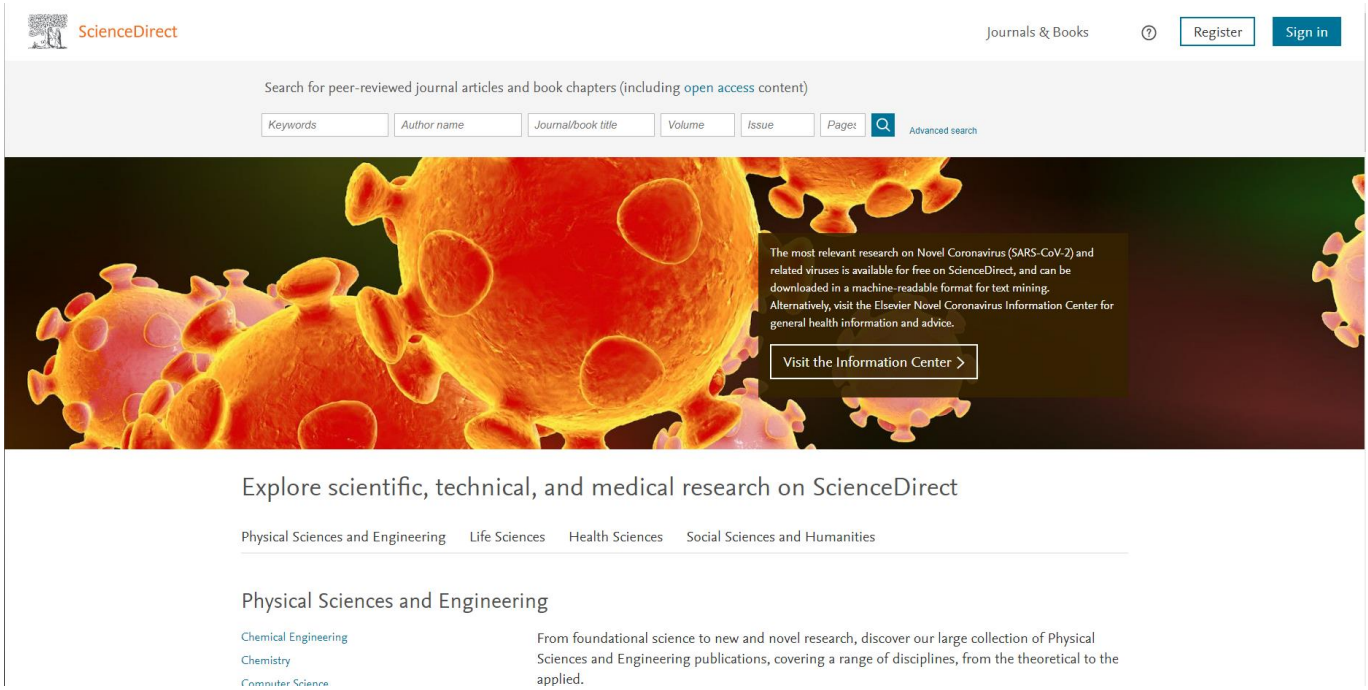

Рисунок 2.1 — Інтерфейс системи Scirius

BASE: Bielefeld Academic Search Engine — один з найбільших наукових інтернет-пошуковиків. Заснована BASE бібліотекою університету Білефельд (Німеччина).

Недоліки: практично відсутні.

Вигляд інтерфейсу системи BASE (рисунок 2.2).

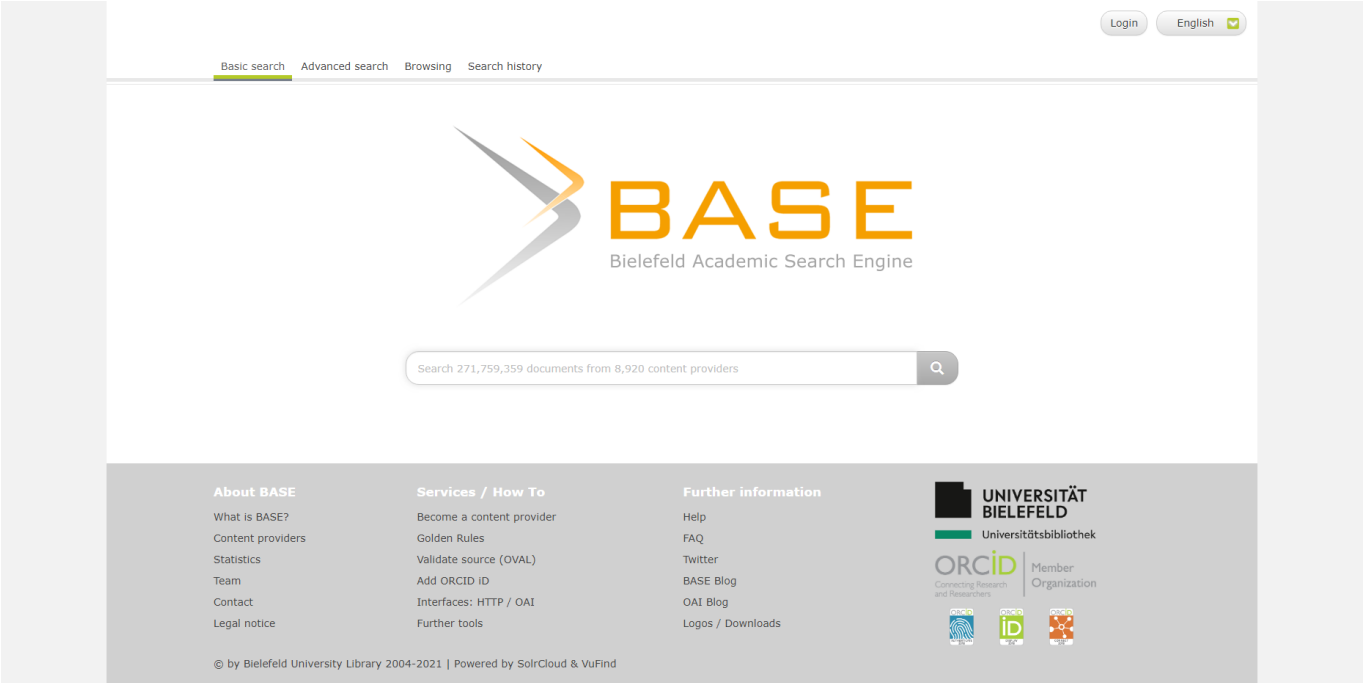

Рисунок 2.2 — Інтерфейс системи BASE.

Google Scholar — Пошукова система для наукової літератури. Містить статті головних наукових видавців, репринтні архіви, публікації на веб-сайтах університетів, наукових товариств та інших наукових організацій. Шукає статті зокрема російською мовою. Використовує поширену технологію PageRank для сортування документів та власний алгоритм індексування для вимірювання впливу кожного матеріалу. База даних наукової пошукової системи включає як загальнодоступні документи, так і матеріали, доступні лише за передплатою.

Недоліки: немає прозорих опцій для звуження результатів пошуку, платна підписка.

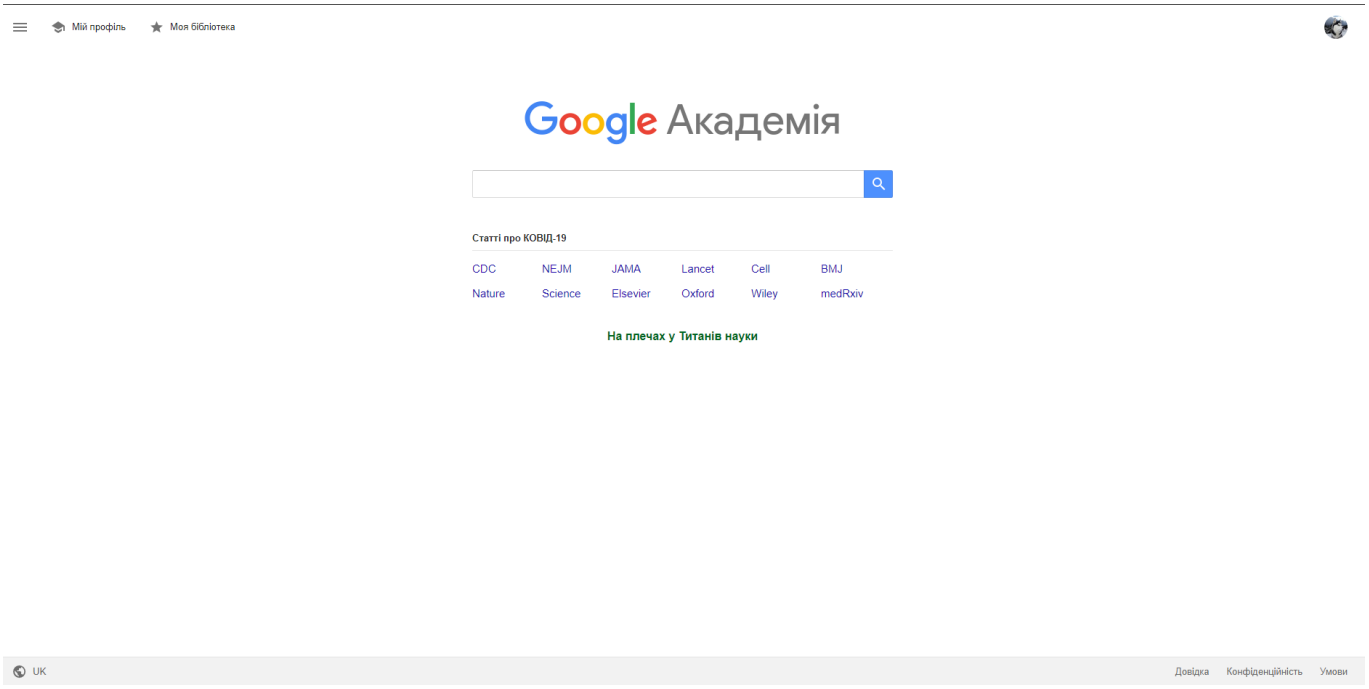

Рисунок 2.3 — Інтерфейс системи Google Scholar.

WorldWideScience.org — пошукова система глобального характеру, що здійснює пошук наукових праць по національних порталах обраних країн.

Недоліки: повертає велику кількість дубльованих даних, що вимагає великих часових затрат на їх обробку.

Рисунок 2.4 — Інтерфейс системи WorldWideScience.

Microsoft Academic Search — статті фокусуються не лише на авторах, а й на організаціях та журналах. Має продвинутый інтерфейс взаємодії з користувачем. Рушій цього сервісу є одним з найбільш інноваційних рішень у сфері наукової інформації, доступної в Інтернеті. Черпає дані з бази даних CrossRef, індексація цитування виконується завдяки алгоритму PopRank**.**

Недоліки: недосконалий механізм пошуку, повільна швидкість оновлення.

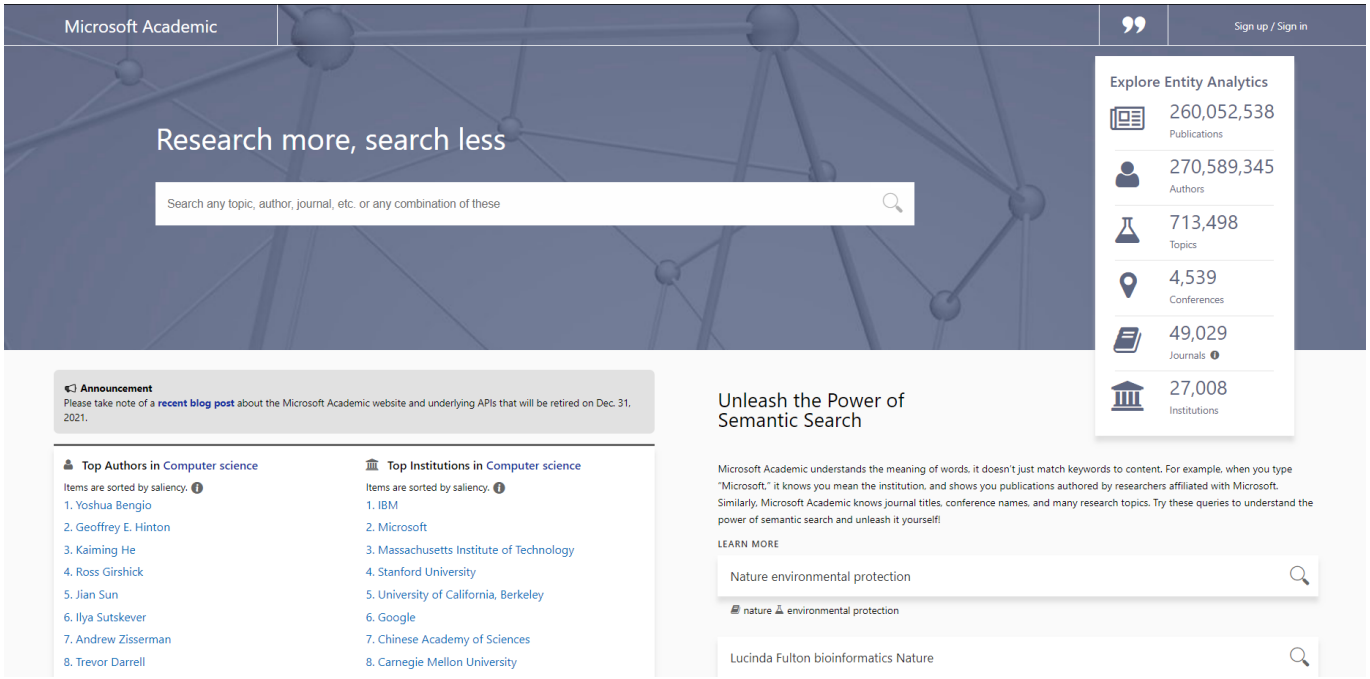

Рисунок 2.5 — Інтерфейс системи Microsoft Academic Search.

AMiner — Університет Цінхуа (Tsinghua) в Китаї вперше представив AMiner (або ArnetMiner) як концепцію особистого профілю науковця, інструмент візуалізації та можливість оцінки авторів та установ. Однак сервіс також містить кілька аномалій та невідповідностей, таких як дефіцитний механізмів пошуку, відсутність прозорості джерел та хаотичний дизайн.

Недоліки: відсутність довіри користувача до цього інструменту, як надійного джерела, не лише для оцінки досліджень, але й для академічної інформації.

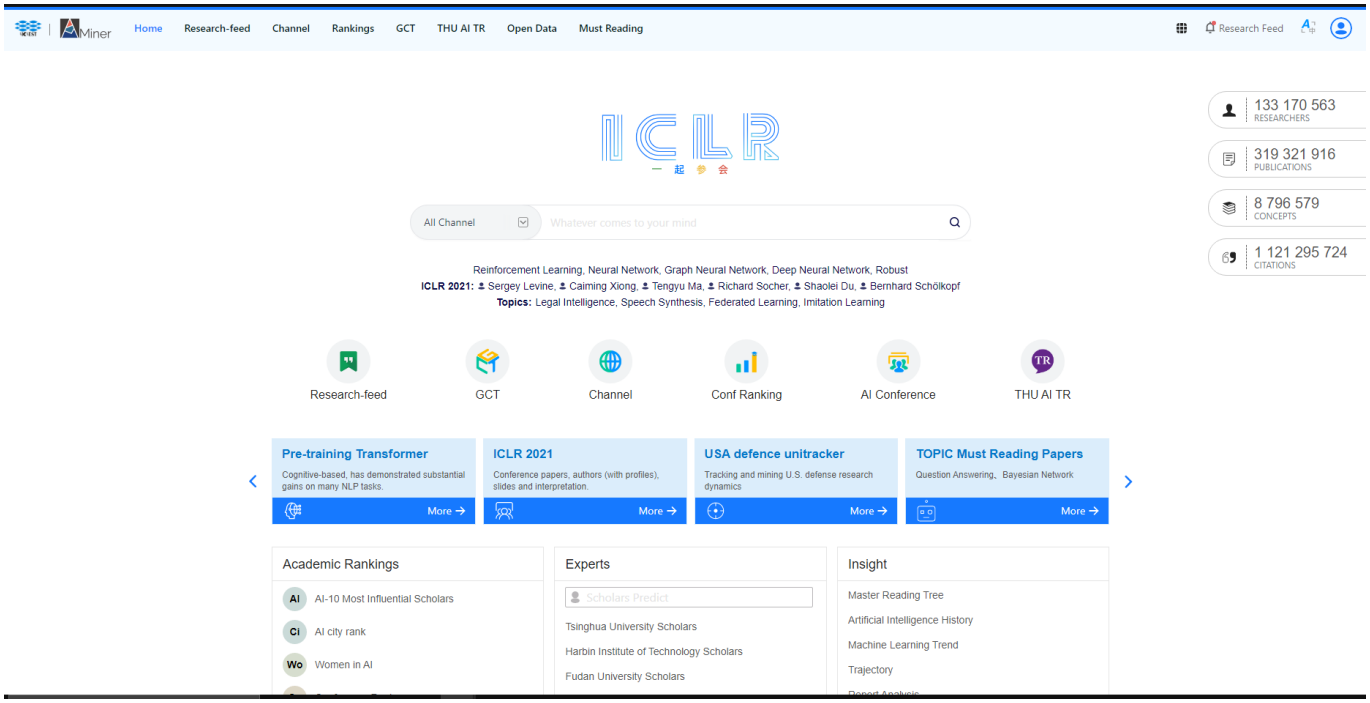

Рисунок 2.6 — Інтерфейс системи AMiner.

Проаналізувавши всі недоліки вищезазначених систем, виникає проблема незручності пошуку інформації. Тобто, користувачеві необхідно знаходити та запам’ятовувати безліч ресурсів, поступово шукати інформацію в кожному з них, витрачаючи час, доки не знайде те, що його задовольнить. Отже, користувач повинен, кропітливо, крок за кроком, виконувати однотипні дії для пошуку статті, чи іншого наукового матеріалу, пристосовуючись до інтерфейсу кожної пошукової системи. Такий підхід не є ефективним, оскільки ціль користувача знайти дані якомога швидше та приступити до їх аналізу, а не сидіти та займатися однотипним пошуком.

2.3. Дослідження та вирішення проблеми пошуку даних

Перше, що спадає на думку, це створити велику організовану базу даних, де будуть зберігатися всі дані, що індексуються як наукові матеріали. Проблема даного підходу тривіальна, це необхідність у великій кількості грошових, трудових та

часових ресурсів для створення такої бази, її підтримки та розвитку, також, створення механізму індексування сторінок, так званого вебкраулера, якому необхідно проіндексувати якомога більше сторінок в Інтернеті. Обміркувавши питання з командою на чолі з дипломним керівником, було вирішено відмовитись від такого підходу.

Після часу відведеного для обговорень, було вирішено: не створювати велику єдину базу даних, а створити механізм підключення безлічі наукометричних баз даних, що надають вільне відкрите API та документацію до нього, для демонстрації роботи були задіяні такі сервіси, як CORE та Scopus.

Слід зазначити, що для реалізації проекту ми не повністю відмовились від створення власної бази даних. Для більшої швидкодії необхідно оперувати власними даними, також, є потреба зберігати історію запитів користувачів для можливого подальшого аналізу та забезпечення актуальності вихідних даних. Тому, було вирішено створити базу даних, яка буде поступово заповнюватися науковими статтями, що повертають зовнішні бази(наразі CORE та Scopus), коли користувач вводитиме унікальний запит. Надалі, користувачам будуть повертатися дані, що збережені в базі проекту, якщо користувача, за якоюсь причини не задовольнять ті статті, що вже знаходяться в локальній базі, він зможе відмітити це у інтерфейсі клієнта, далі виконається запит до сервера про необхідність повернення нових даних, після звернення до зовнішніх баз через зовнішнє API, сервер збереже або оновить дані за поточним запитом та поверне їх клієнту. Клієнт сформує каталог та відобразить його на екрані користувача.

Отже, для потреб багато-джерельної системи, яка буде звертатися до різних зовнішніх сервісів — джерел інформації, необхідно спроектувати та розробити модулі обробки результатів зовнішнього API, для яких виникає потреба створення уніфікованого формату звернення та відповіді.

Зі сторони серверної частини або, як її ще називають, “бекенд”(з англ. “backend”), необхідно створити API-endpoint(и), для здійснення запитів, щодо

отримання масиву статей, авторів, тем та видань з боку локальної бази даних, та з боку зовнішніх баз даних відповідно.

Створені API-endpoint(и), будуть використовуватися програмістом, що відповідає за клієнтську частину проекту або, як її ще називають, “фронтед”(з англ. “frontend”). Клієнтська частина — відповідає за інтерфейс системи та відображення інформації на екрані кінцевого користувача.

2.4 Висновки до розділу

У цьому розділі проведено аналіз існуючих рішень пошукових систем наукометричних даних. Розлянуті інтерфейси та недоліки кожної з систем. Визначений шлях рішень проблеми пошуку даних, а саме не створення чергового модулю індексації та структурування наукових матеріалів, а реалізація системи, що буде об'єднувати дані з різних джерел, шляхом створення механізму для підключення наукометричних баз до цільового продукту.

3. ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ ДЛЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ

3.1. Обґрунтування вибору мови програмування

Для розробки серверної частини вибір мови програмування зосередився між PHP і Python. Для мови Python загалом використовуються фреймворки Django та Flask, а для PHP — Laravel.

Python — це інтерпретована мова програмування високого рівня зі строгою динамічною типізацією, що підтримує основі парадигми програмування такі як ООП та функціональне програмування. Основною перевагою цієї мови програмування є велика кількість написаних бібліотек, в тому числі бібліотеки для роботи з мережевими протоколами, бібліотеки ORM для баз даних. Це одна із найшвидших мов програмування, в плані реалізації ідеї, оскільки вона не потребує великої кількості стрічок коду. Сильною стороною є лего читаємість та простота, код на цій мові читається майже як звичайний англійський текст.

PHP — це серверна мова скриптів, що розшифровується як гіпертекстовий процесор. Використовується для розробки динамічних вебсайтів або вебзастосунків. Може легко інтегруватися з усіма основними вебсервісами в усіх основних операційних системах.

3.1.1 Вимоги до мови програмування

Обрана мова повинна задовольняти наступним вимогам:

— Підтримка бази даних PostgreSQL.

— Підтримка основних операційних систем.

— Бути достатньо актуальною, задля підтримки проекту в майбутньому.

— Надавати можливості імплементації сторонніх модулів.

— Бути зрозумілою та лаконічною в цілях швидкої підтримки та масштабування проекту в перспективі.

3.1.2 Порівняння Python та PHP

Щоб визначитись з мовою розробки, був проведений аналіз тенденцій розвитку мов програмування та фреймворків. Нижче представлена порівняльна таблиця 3.1 .

Таблиця 3.1 — Порівняння обраних мов.

Назва критерію	**Python**	**PHP**
Тип	Мова програмування загального призначення.	Спеціалізується, як мова для веброзробки.
Фреймворки	Має невелику кількість фреймворків.	Має велику кількість фреймворків.
Кращі фреймворки	Django, Flask, FastAPI.	Slim, Laravel, Fast-free framework.
Підтримка баз даних	Підтримка найпопулярніших баз даних(серед яких і PostgreSQL).	Підтримка більшості загальновживаних баз даних(серед яких і PostgreSQL).
Читабельність	Використовує дуже строгі правила відступів	Гарно задукоментований та дотримується класичного С стилю.

	“PEP-8”, що робить його більш читабельним, ніж PHP.	Через можливість вільного стилю написання коду може мати гіршу читабельність.
Синтаксис	Зрозумілий та лаконічний.	Має спектр домовленостей про імена, дещо складніший ніж Python.
Відомі компанії, що використовують технологію	Google, Uber, Dropbox, Facebook.	Viber, Yahoo!
Рейтинг мов програмування TIOBE	2 позиція.	9 позиція.

Провівши порівняння, було вирішено використовувати Python.

Він задовольняє усім вимогам та навіть пропонує модулі, які можуть бути корисними у перспективі, серед яких відкриті бібліотеки для аналізу даних та машинного навчання.

3.2. Додаткові засоби розробки

3.2.1. Вибір фреймворку

Основними вебфреймворками для мови Python є Flask і Django.

Про Django: відкритий високорівневий вебфреймворк написаний мовою Python, призначений для створення веб-систем. Має підтримку на всіх популярних

платформах. Його перевагою є модульний підхід до створення архітектури вебсервісів, що дозволяє створювати нові модулі або додатки не переписуючи логіку усього проекту.

Про Flask: мікрофреймворк для написання вебзастосунків, що поширюється по вільній ліцензії використання. Flask створений, щоб розширюватись. Тому за своїм принципами має тількі основні модулі для представлень та шаблонів, не містить в собі початкових модулів для роботи з базою даних, проте, має великий список підтримуваних розширювань.

Розглянемо головні відмінності у таблиці 3.2.

Таблиця 3.2 — Порівняння фреймворків.

Назва критерію	**Django**	**Flask**
Архітектура	Складається з проекту, що ділиться на застосунки. Проект — коріна папка та файл з глобальними налаштуваннями. Застосунки — логіка, що розподілена між модулями проекту. Кожен проект має свої застосунки. Відноситься до паттерну MTV.	Не має визначеної структури. Структура формується у ході розробки.
Розширення	Для типових проектів вистачає початкових бібліотек фреймворку. Є можливість завантажити сторонні бібліотеки.	З початку ініціалізації проекту є практично порожнім. Для реалізації однієї задачі часто існує декілька розширень, що робить фреймворк гибким
Масштабування	Розширюється завдяки реєстрації нових застосунків, в котрих пишуть додаткову логіку. Швидке масштабування — одна з складових філософії	Надає інтерфейс blueprint через який виконується масштабування.

	фреймворка, тому, структура застосунків створюється з самого початку проекту. Ці застосунки є самостійними, тому, їх можна використовувати у різних частинах проекту	
Підтримка ORM для БД	Має власну ORM.	Власної ORM не має.

Для підтримки та масштабування програмний застосунок повинен мати визначену зрозумілу архітектуру. Провівши аналіз фреймворків, вибір зупинився на Django.

3.2.2. Бібліотека для створення внутрішнього API

Django REST framework — потужна бібліотека, що надає інструменти для написання веб API. Переваги: зручна серіалізація даних, можливість під'єднання модулів OAuth1 та OAuth2 для аутентифікації та авторизації користувача.

3.3. Середовище програмування

PyCharm — професійне IDE(інтегроване середовище розробки) для мови програмування Python. Надає засоби для аналізу коду, графічний зневаджувач, інструмент для запуску юніт-тестів і підтримує веброзробку на Django. Версія Community 2021.1 (рисунок 3.1).

Рисунок 3.1 — Скріншот версії середовища розробки.

3.4. Засоби тестування API

Postman — API REST клієнт, що представляє собою потужний інструмент тестування і розробки API, який при цьому має простий і інтуїтивно зрозумілий інтерфейс (рисунок 3.2). Якщо ви вперше відкриваєте цю програму, знаючи в чому різниця між POST і GET, то ви без зусиль зможете відправити свій перший запит.

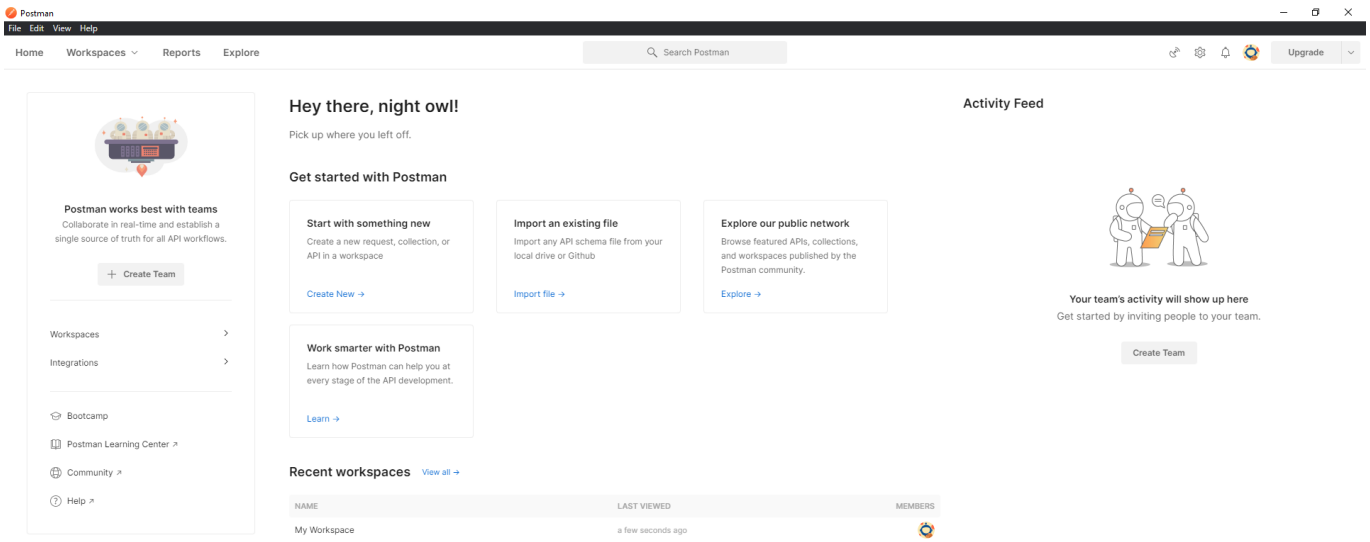

Рисунок 3.2 — Інтерфейс програми Postman.

3.5. Репозиторій та система контролю версій

GitHub — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub.

В цілях дипломного проекту був зареєстрований аккаунт на GitHub. В якому публікуються зміни до поточних версій проекту. Для зберігання серверної частини був створений приватний репозиторій під назвою “affin” (рисунок 3.3).

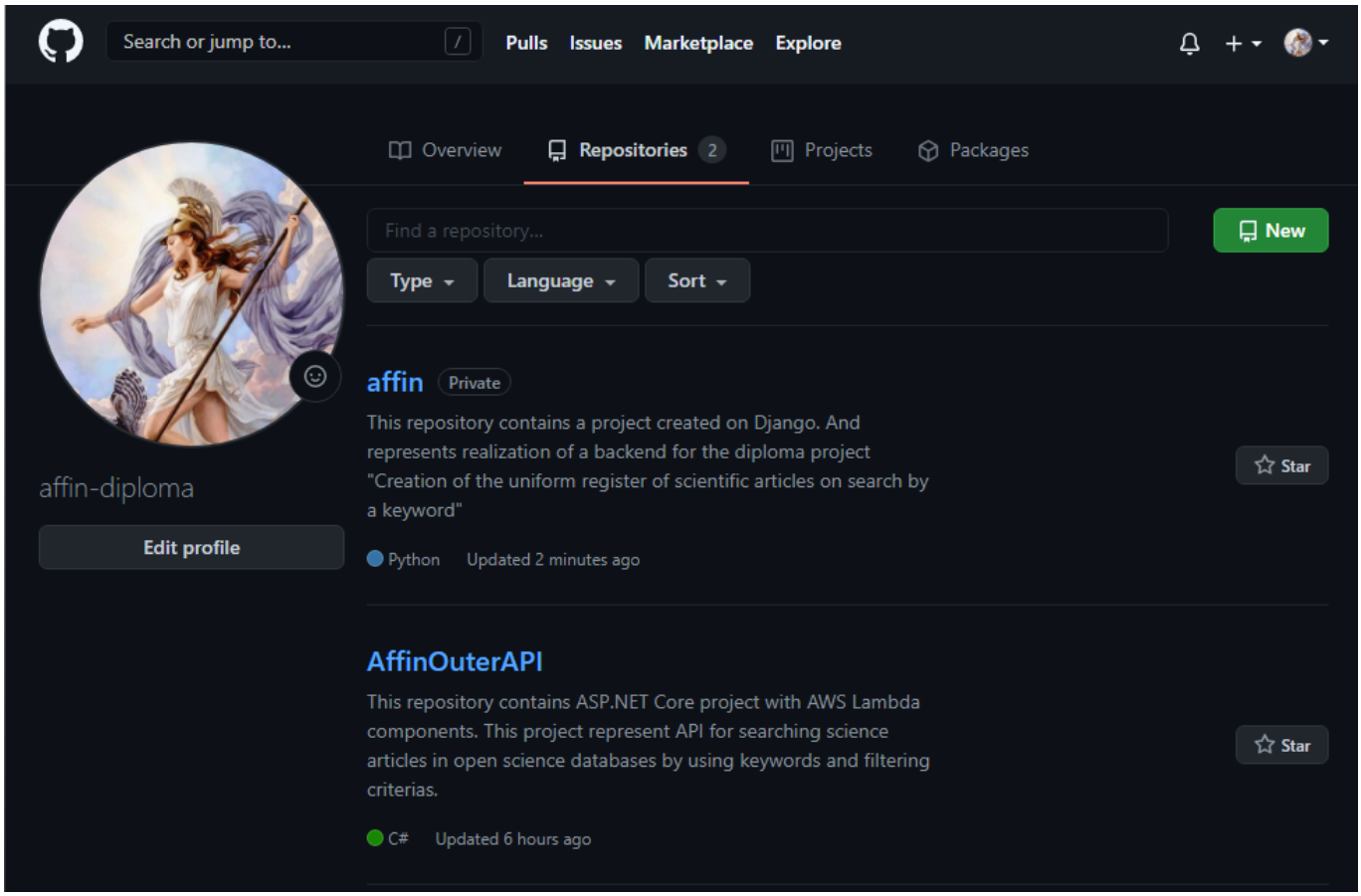

Рисунок 3.3 — Аккаунт на GitHub.

3.6. Система для планування та відстеження прогресу проекту

Clubhouse — соціальна мережа, в якій спілкування здійснюється винятково голосом. Пропонує зручний та гнучкий функціонал для планування та розподілу задач між учасниками клубу. Доступні засоби планування повністю задовольняють організаційні моменти дипломного проекту.

Приклад: Задачі що знаходяться в розробці (рисунок 3.3) та виконані завдання (рисунок 3.4).

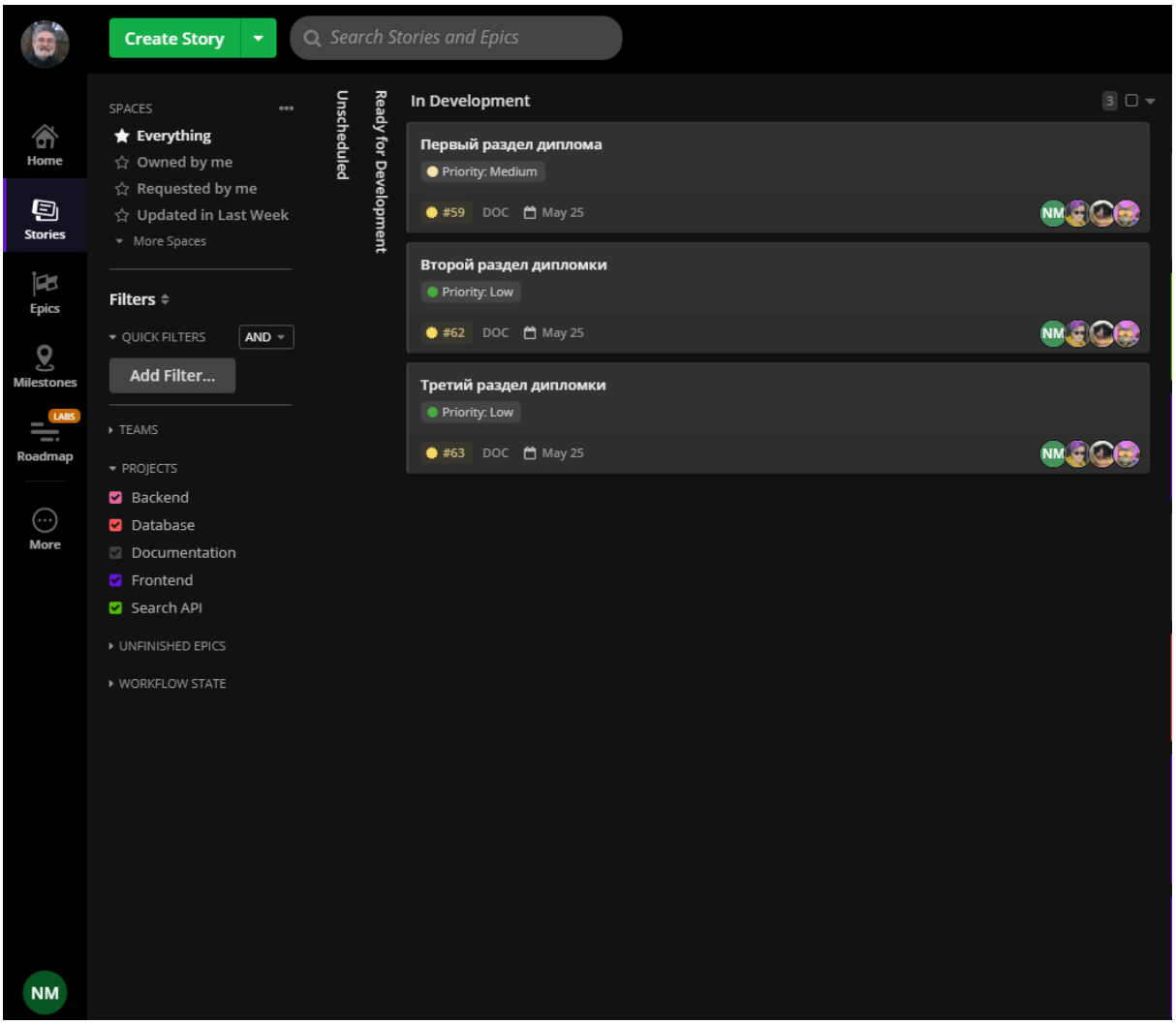

Рисунок 3.4 — Задачі зі статусом в розробці.

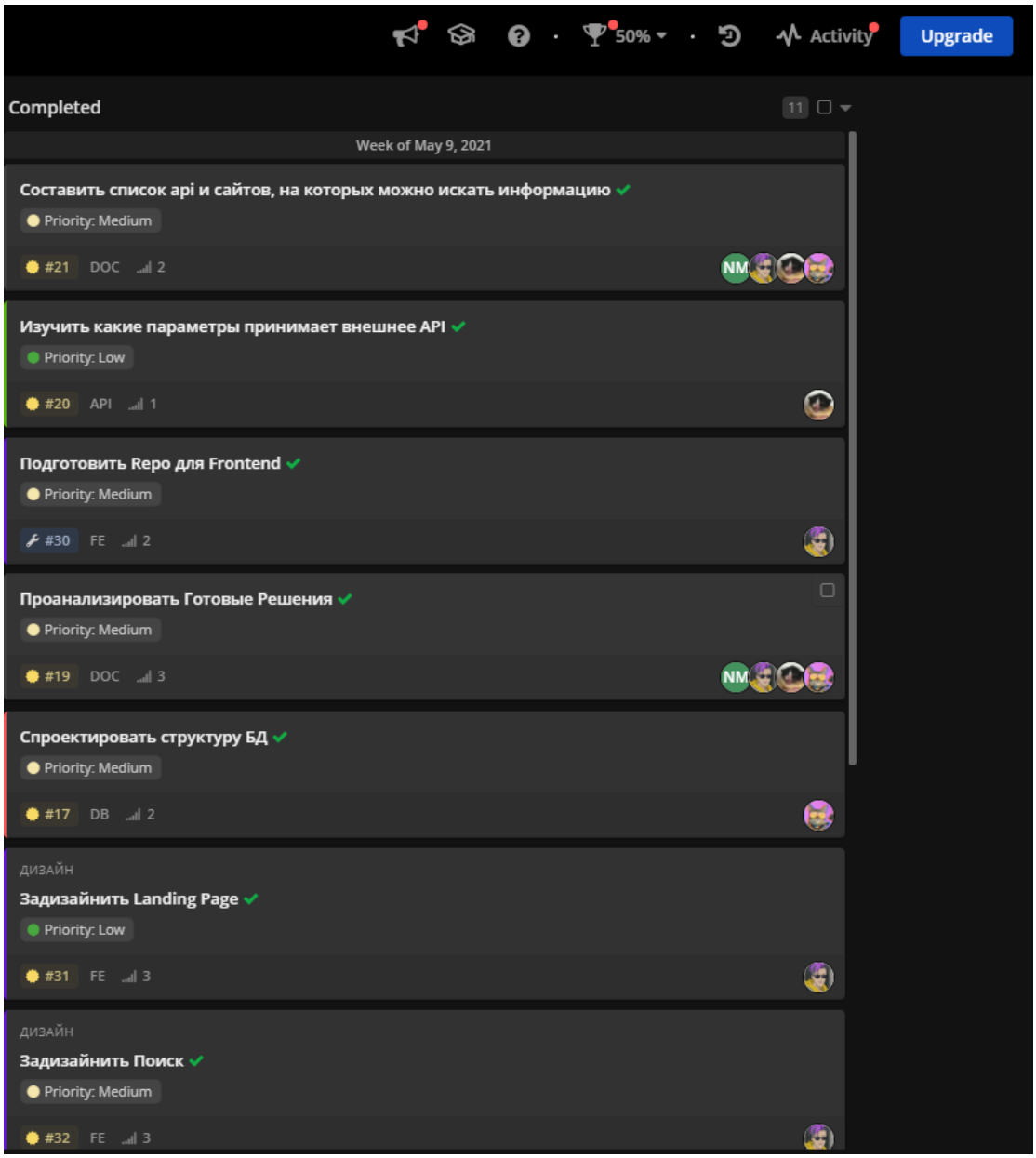

Рисунок 3.5 — Задачі зі статусом виконано.

3.7. Хостинг

Для демонстрації працездатності системи на сервері, було обрано використовувати хмарну платформу Heroku.

Платформа має безліч тарифів серед яких є безкоштовний, його достатньо для демонстраційних потреб працездатності серверної частини.

Також, Heroku надає можливість зв’язувати проект із репозиторієм на GitHub, це означає, що після певної настройки серверу, він буде брати файли зі створеного нами раніше репозиторію “affin” та запускати проект.

3.8. Системні характеристики

Нижче представлені характеристики системи на якій було розроблено ПЗ (рисунок 3.6).

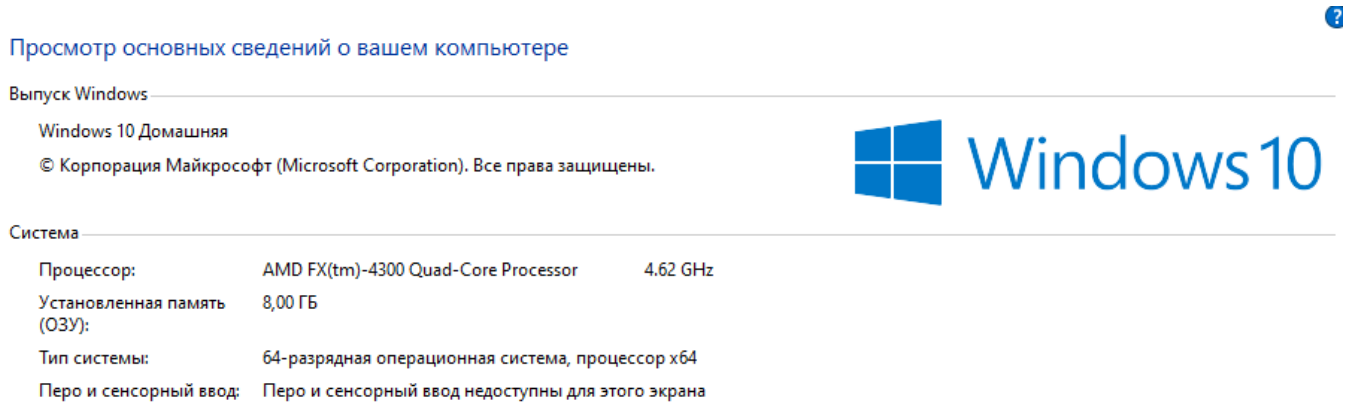

Рисунок 3.6 — Системі характеристики.

3.9 Висновки до розділу

В даному розділі був проведений аналіз та порівняння технологій. Описані мови програмування Python та PHP, що відповідають необхідним вимогам, серед них було обрано Python, як більш розповсюджену та перспективну. Також, описані

фреймворки Flask та Django, останній вибраний як рушій проекту, тому що він має більш чітку архітектуру та необхідні інструменти відразу після ініціалізації. Обрано бібліотеку для пришвидшення розробки API. Як середовище програмування обрано Pycharm. Описані засоби тестування API. Описано систему керування версій та систему планування задач. Обрано хостинг, що підтримує інтеграцію з системою керування версій. Вказані системні характеристики комп'ютера на якому виконувалася розробка серверної частини проекту.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ ДЛЯ ЄДИНОГО РЕЄСТРУ НАУКОВИХ СТАТЕЙ ЗА ПОШУКОМ ПО КЛЮЧОВОМУ СЛОВУ

Роль серверної частини в контексті проекту — обробка запитів та з’єднання усіх компонент системи. Нижче представлена схематична архітектура взаємодії компонентів проекту (рисунок 4.1).

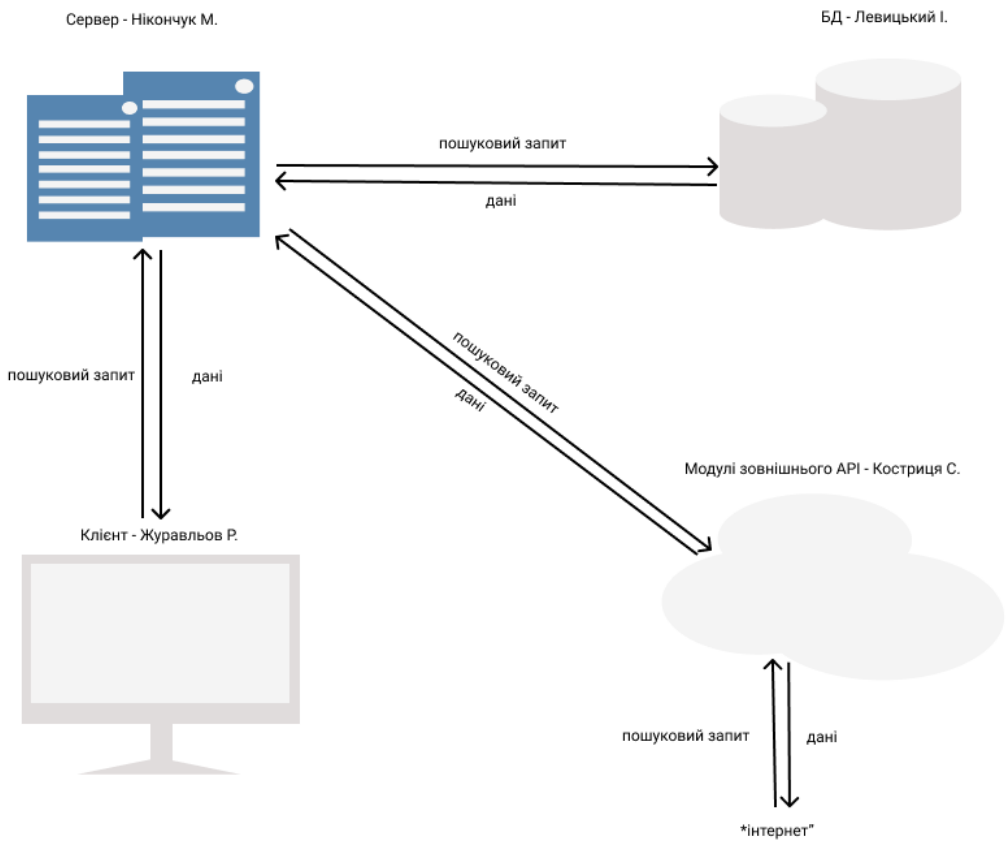

Рисунок 4.1 — Схема взаємодії компонентів проекту.

4.1 Опис реалізації API

Для реалізації API сервера була задіяна бібліотека Django REST framework (DRF), один з її інструментів — механізм серіалізації даних. DRF надає клас серіалізатора, який описує логіку конвертації об’єктів мови Python в json та навпаки.

Логіка запиту прописується у власних класах, що наслідуються від одного з класів представлень(APIView, ViewSet, ModelViewSet - описані в DRF). При потребі вказується класс серіалізатора, який приймає клас для здійснення запитів до бази даних.

JSON (англ. JavaScript Object Notation) — стандартний текстовий формат даних для обміну інформацією між комп’ютерами. Отже, відповідь сервера буде містити json-об’єкт, з якого клієнт буде отримувати цільові дані.

4.1.1 Опис уніфікованого формату даних

JSON структура пошукового запиту(вхідні дані) рисунок 4.2.

```
{
  "searchQuery":"Днк",
  "offset":1,
  "limit":10,
  "filter": {
    "titles": "тема1|тема2",
    "authors":"автор1|автор2",
    "publishers":"публікатор1|публікатор2&публікатор3",
    "languages":"English|Russian|Ukrainian",
    "years": "2010-2018"
  }
}
```

Рисунок 4.2 — Структура вхідних даних у пошуковому запиті.

JSON структура відповіді(вихідні дані) рисунок 4.3.


```
{
    "status": "OK",
    "total": 3042,
    "data": [
        {
            "lang": "Ukrainian",
            "id": "288833370",
            "doi": "10.15587/2312-8372.2012.5503",
            "title": "Застосування гніздної полімеразної ланцюгової реакції для діагностики і
                генотипування хламідій",
            "topics": null,
            "snippet": null,
            "description": "...Литовченко\nВ статье описана разработанная автором методика
                определения \n<strong>ДНК</strong> хламидий, основанная на гнездной ПЦР, и метод
                верификации \nрезультатов с помощью дополнительной амплификации фрагмента
                \n<strong>ДНК</strong> критической плазмиды возбудителя....",
            "authors": [
                "Литовченко, Ольга Олексіївна"
            ],
            "publisher": "'Private Company Technology Center'",
            "downloadUrl": "https://core.ac.uk/download/288833370.pdf",
            "deleted": null,
            "datePublished": null,
            "year": 2012,
            "relations": null,
            "source": "Core"
        },
```


Рисунок 4.3 — Структура вихідних даних пошукового запиту.

4.1.2 Опис функцій API

Кожен запит до серверу має свою функцію, логіку та реалізацію, що описано нижче.

Для отримання масиву публікацій, був розроблений наступний API-endpoint:

https://<domain>/articles (4.2)

де domain - іменований адрес сервера на якому завантажене програмне забезпечення. Звернення за цим запитом обробляє клас DocumentViewSet, цей клас наслідується від DRF класу ModelViewSet. В класі DocumentViewSet переприсвоюються поля. Поле queryset приймає результат запиту до БД. Поле serializer_class приймає серіалізатор(DocumentSerializer), що конвертує дані з формату мови Python в Json та

навпаки. Клас DocumentSerializer наслідується від DRF класу ModelSerializer. Поле model приймає класс Document, що звертається до відповідної таблиці в БД та поле fields в якому задаються необхідні поля з таблиці до якої прив’язаний клас. Нижче представлена діаграма класів (рисунок 4.4), що описує ці зв’язки.

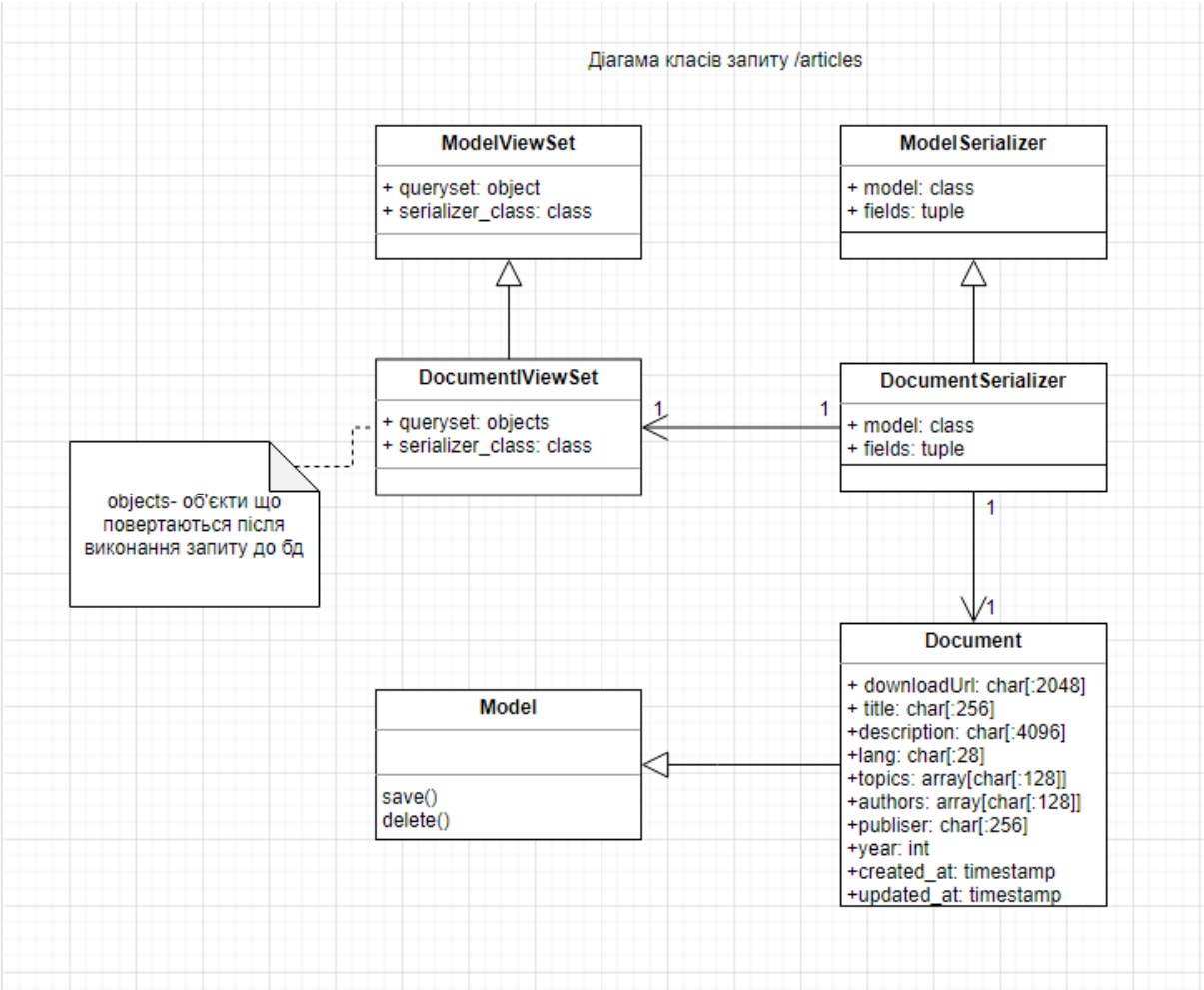

Рисунок 4.4 — Діаграма класів обробника запиту /articles

Для отримання нових публікацій, які надходитимуть із зовнішніх баз, був створений аналогічний API-endpoint:

https://<domain>/pullarticles (4.3)

цей запит повертає розширені оновлені дані, яких ще не було в локальній базі. Оскільки, дані від модулів API надходять у форматі Json, їх не потрібно попередньо конвертувати, тому логіка зв'язків між класам дещо спрощується (рисунок 4.5).

Швидкість виконання обумовлена часом з котрим виконуються запити до зовнішніх баз та перевіркою отриманих даних на дублікати. В тіло запиту необхідно передати headers - заголовки, які містять keys - ключі для службової інформації та Request Payload, що приймає json у якому міститься сам пошуковий запит.

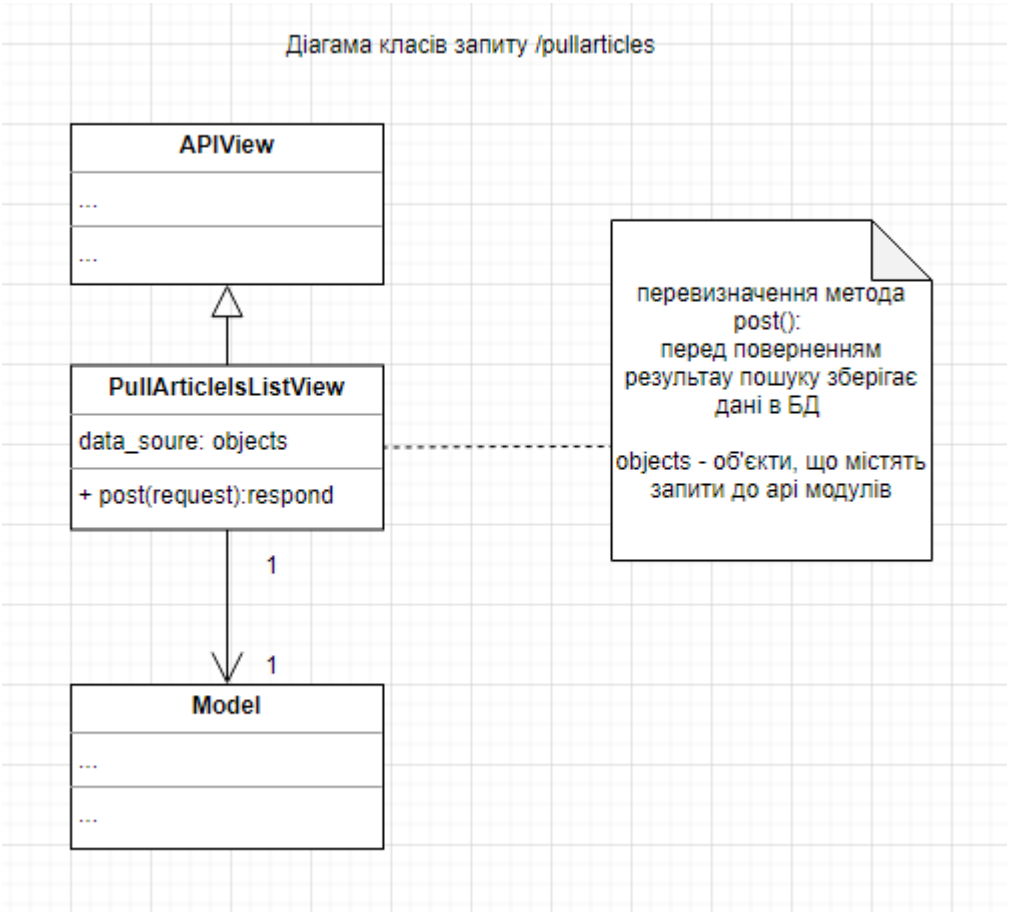

Рисунок 4.5 — Діаграма класів обробника запиту /pullarticles

Запит /pullarticles використовує HTTP метод POST. Запит до API модулів, обробка результату та збереження до БД виконується методом post класу PullArtilcles.

Для зручності користувача, на стороні клієнта, в полях із фільтрами пошуку, було реалізовано автозаповнення. А з боку сервера це представлено запитами до бази даних. Клієнт має можливість звертатися до сервера на отримку даних для автозаповнення наступними запитами:

https://<domain>/authors (4.4)

повертає масив авторів, може приймати ключ “term” - префікс, який ввів користувач, тоді запит буде виглядати наступним чином:

https://<domain>/authors/?term=<префікс> (4.5)

Цей запит використовує HTTP метод GET. Класи, що задіяні для цього запиту зображені на діаграмі (рисунок 4.6).

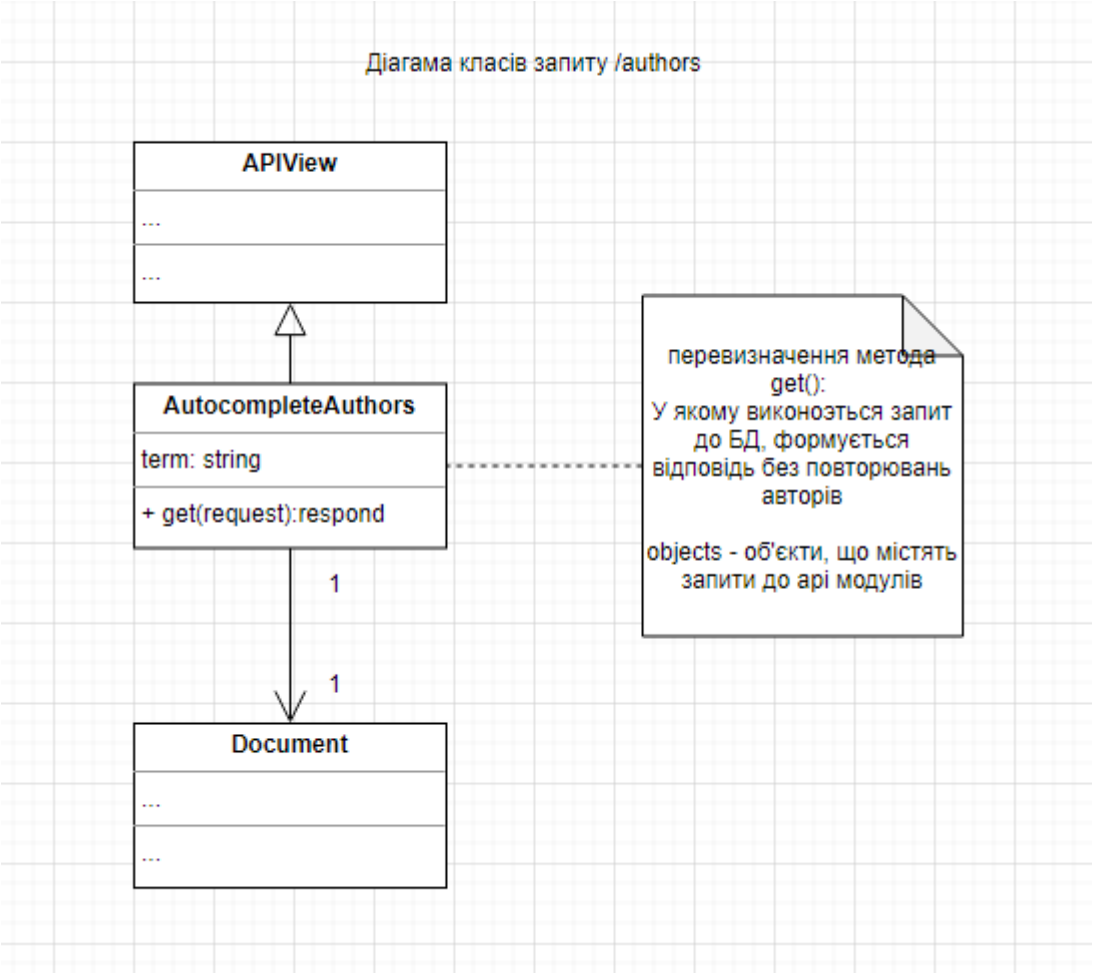

Рисунок 4.6 — Діаграма класів обробника запиту /pullarticles.

Для отримання списку видавництв, було реалізовано запит:

https://<domain>/publsihers (4.6)

може приймати ключ term, як і запит (4.5). Обробник цього запиту має аналогічну реалізацію (рисунок 4.7).

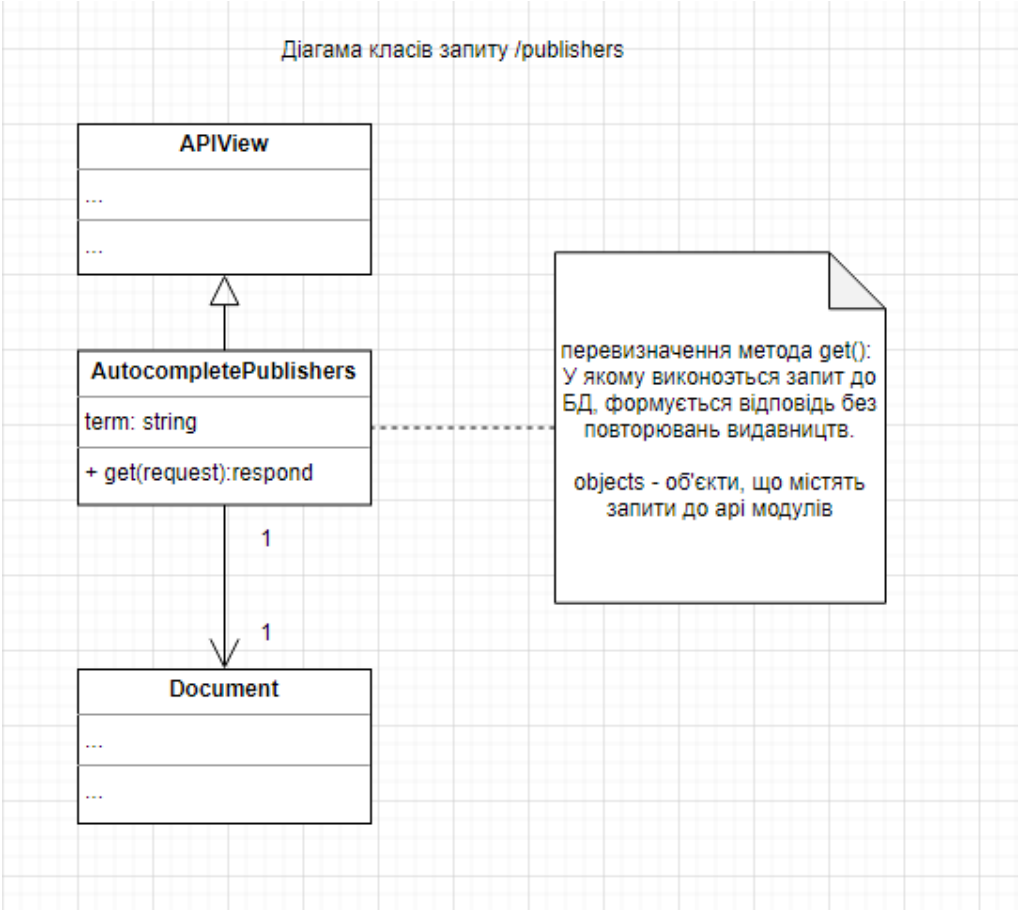

Рисунок 4.7 — Діаграма класів обробника запиту /publishers.

4.1.3 Модель звернення до серверу

Послідовність запитів при здійснені пошуку має наступний характер:

1. Користувач вводить пошуковий запит, при потребі, додає уточнюючі фільтри, та натискає кнопку пошуку.
2. Клієнт формує пошуковий запит згідно до встановленого JSON формату, використовуючи API серверної частини, надсилає запит на отримання статей.
3. Сервер обробляє запит, та надсилає його до БД проекту.
4. а) Якщо БД повернула не порожній список:

5. Сервер повертає клієнту JSON відповідь зі шуканими статтями.

6. Клієнт обробляє відповідь сервера та виводить статті на екран користувачеві.

4. б) Якщо БД повернула порожній список:

5. Сервер надсилає запит до API модулів.

6. Сервер обробляє відповіді, об'єднує їх та надсилає запит на збереження унікальних даних у БД.

7. Сервер надсилає об'єднанні дані клієнту.

8. Клієнт виводить результат пошуку на екран користувача.

Що представлено на діаграмі послідовностей(рисунок 4.8).

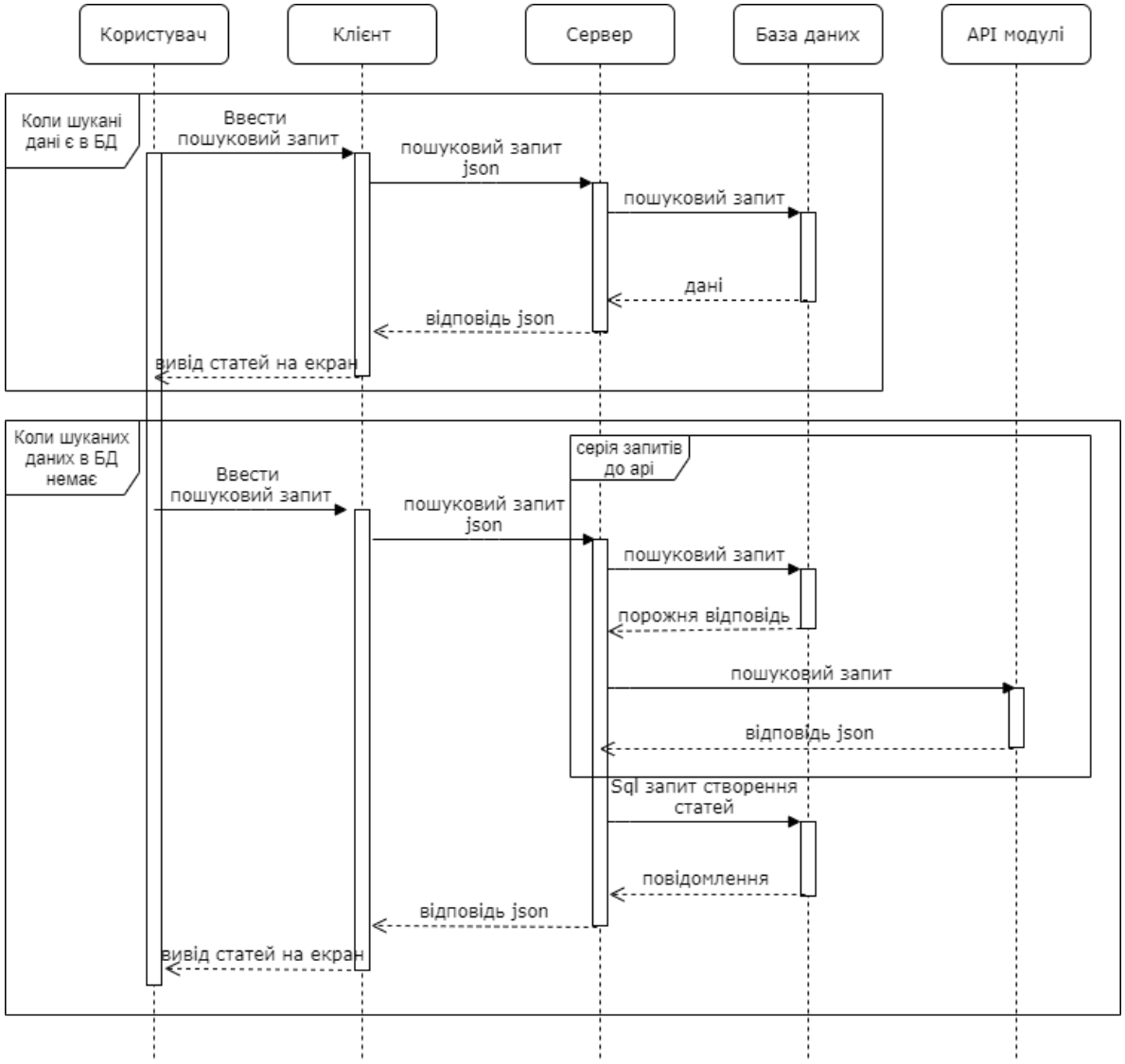

Рисунок 4.8 — Діаграма послідовностей.

4.1.4 Тестування запитів

Для поля автозаповнення необхідно отримувати не більше 20 авторів, результат (рисунок 4.9) виконання запиту /authors з префіксом “Ma”.

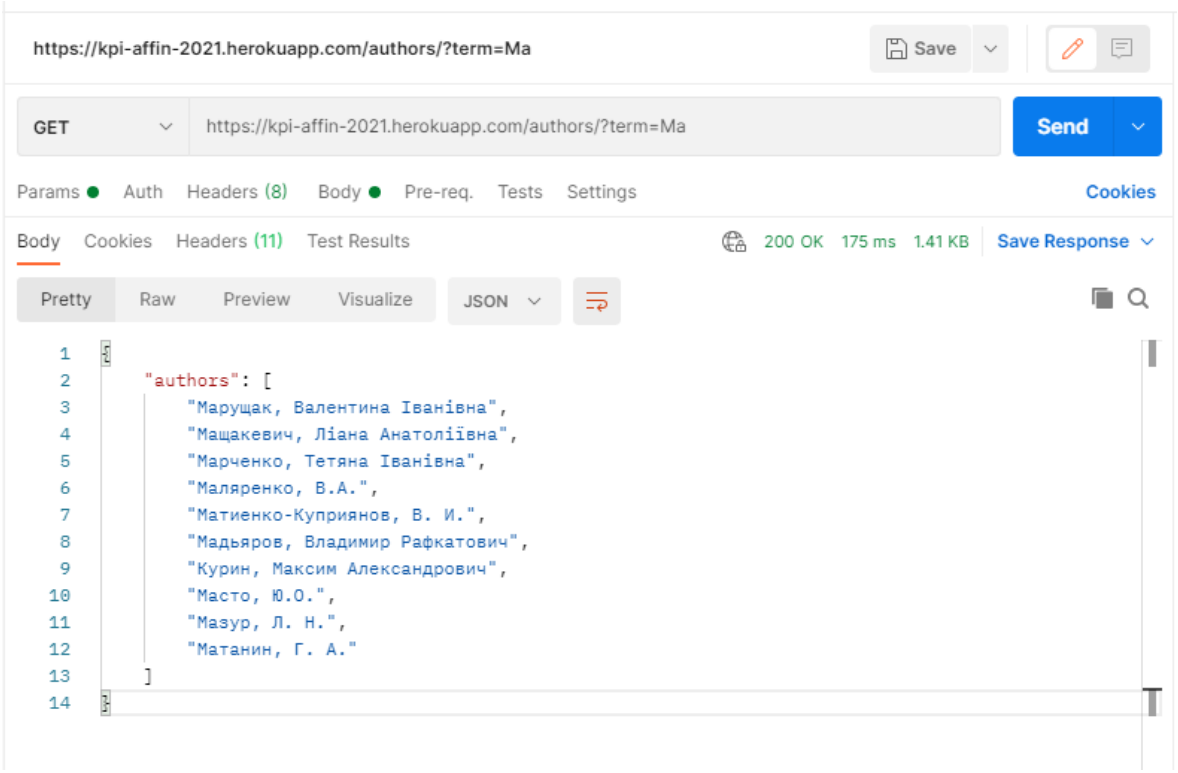

Рисунок 4.9 — Результат виконання запиту /authors.

Запит /publishers працює аналогічно, та повертає до 20 видавництв (рисунок 4.10).

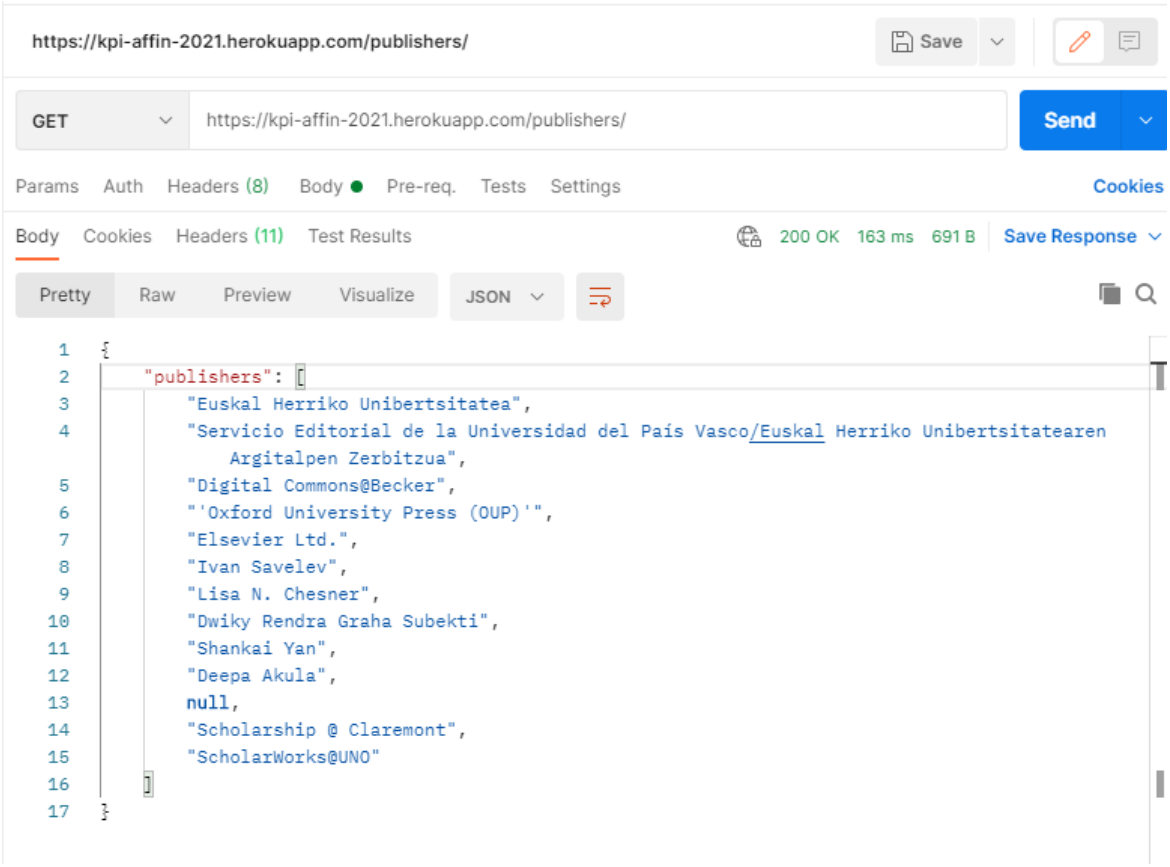

Рисунок 4.10 — Результат виконання запиту /publishers

Запит /pullarticles повертатиме статті відповідно до пошукового запиту та фільтрів (рисунок 4.11).

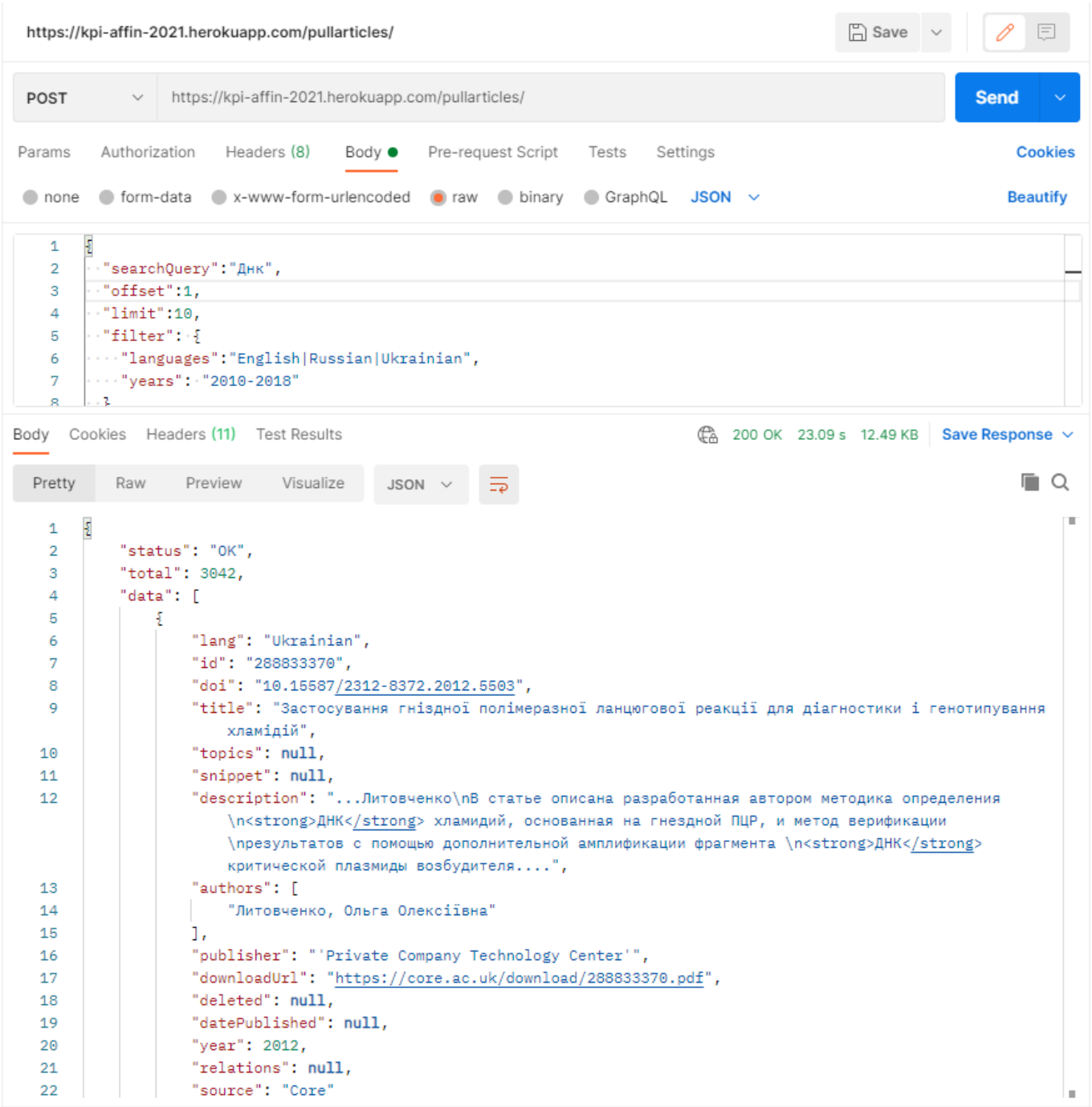

Рисунок 4.11 — Результат виконання запиту /publishers.

4.2 Файлова структура проекту

Специфіка розробки проектів на Django, вимагає модульного підходу до організації коду. Нижче зображено файлову структуру (рисунок 4.12).

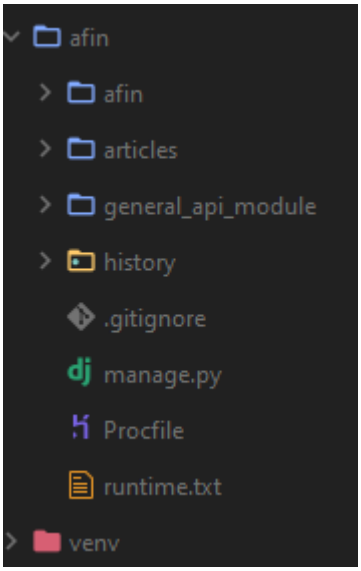

Рисунок 4.12 — Файлова структура проекту.

В модулі “afin” знаходяться налаштування проекту, що відповідають за реєстрацію інших модулів, шаблонів, представлень, бази даних. В модулі “articles” описана логіка API запитів до бази даних. Модуль “general_api_module” описує логіку запитів до зовнішніх модулів API. Модуль “history” неактивный, але в майбутньому планується реалізація перегляду історії пошуку. Файли (.gitignore, manage, Procfile, runtime) виконують службові функції, що не відносяться до архітектури системи, тому їх опис опускається.

Модуль “venv” — створений механізмом мови Python, який реалізує віртуальне оточення, в ньому зберігаються бібліотеки проекту та інтерпретатор мови. Після завантаження необхідних бібліотек прийнято створювати файл “requirements.txt” де описані залежності проекту, тобто бібліотеки та їх версії. Це зручний механізм , що дозволяє легко розгорнути python-проект будь-де.

4.3 Розгортання програмного забезпечення

Для розміщення програмного забезпечення був обраний безкоштовний тариф на хмарному сервісі Heroku. Heroku запускає Django проекти в Dynos, що представляють собою віртуальні Unix-контейнери, в які завантажуються необхідні оточення. Дані, що зберігаються в dynos ізольовані та мають “ефімерну” файлову систему, котра повністю оновлюється та очищається коли dynos перезавантажується. Така архітектура дозволяє Heroku горизонтально масштабувати проект, просто створюючи нові dynos.

Оскільки, файлова система є короткотривалою, тому неможливо напряму підключити власні сервіси(бази даних, системи кешування і.тд.).

Натомість, Heroku надає сервіси, доступні як “незалежні” додатки, також сервіси сторонніх розробників та сервіси від самої Heroku. В момент запуску системи, dynos отримує доступ до сервісів, використовуючи інформацію із конфігураційних файлів проекту. Модель розгортання системи (рисунок 4.13).

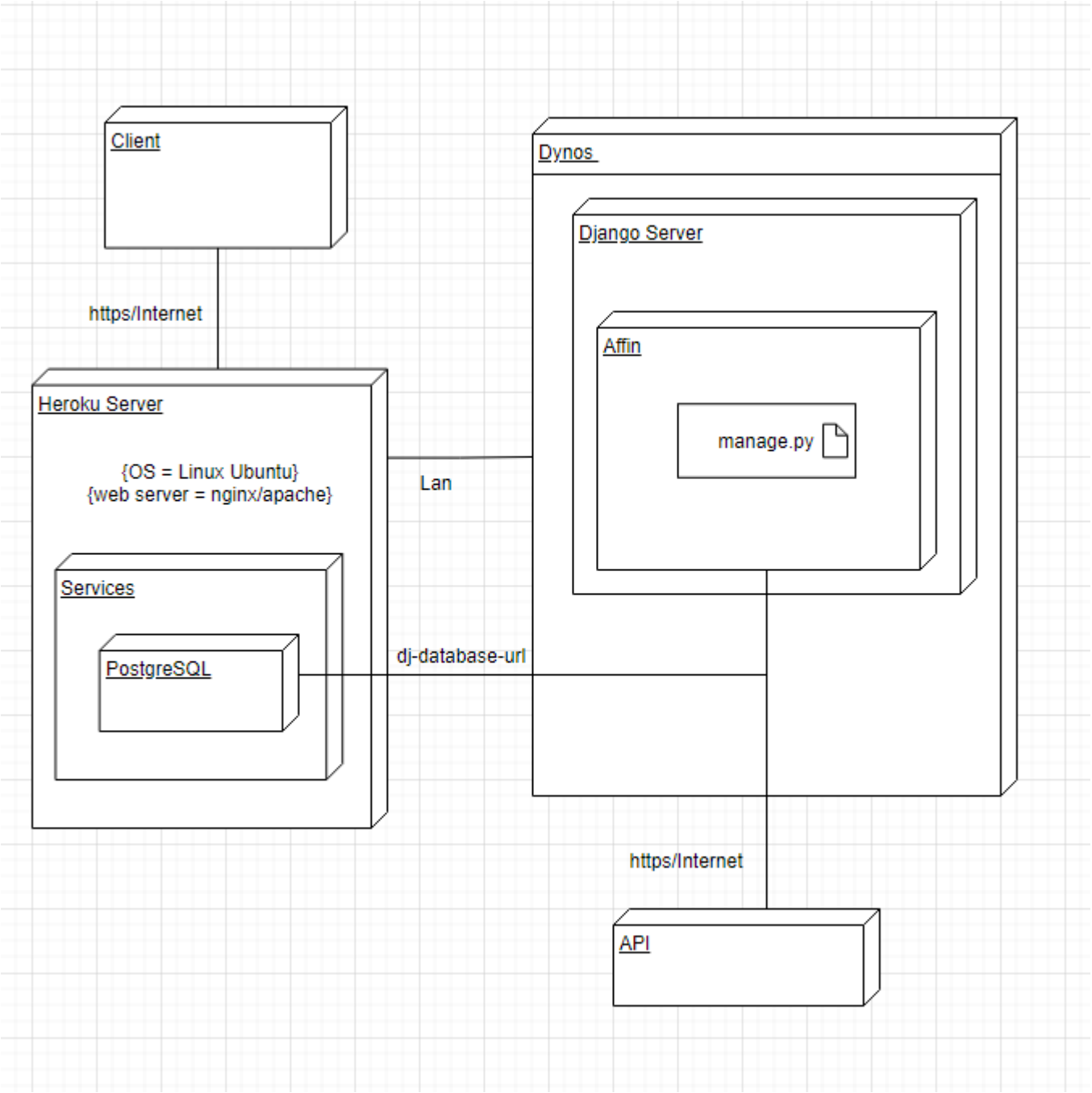

Рисунок 4.13 — Діаграма розгортання

4.3 Злиття компонентів проекту

Нижче представлені компоненти проекту та відповідальні за їх реалізацію:

— Серверна частина(створення логіки обміну/обробки даних між усіма компонентами) (Нікончук Максим);

— База даних(створення архітектури та бд проекту)(Левицький Ілля);

— Модулі зовнішнього API (створення модулів-конекторів для підключення сторонніх джерел даних)(Кострицяя Сергій);

— Клієнт(створення інтерфейсу системи)(Журавльов Роман);

Після підключення, налагодження та проведення тестів усіх частин, створена система була завантажена на тестовий сервер Heroku та доступна для експлуатації.

4.4 Висновки до розділу

У цьому розділі було описано архітектуру системи, створений уніфікований формат обміну даними проекту. Описані функції API для пошукових запитів та автозаповнень, їх послідовність та реалізація, наведені діаграмами класів і послідовностей. Представлена файлова структура проекту та діаграма розміщення. Здійсненні тестові запити до API.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Веб-сервіс має назву “Аффін”, та доступний для користування за посиланням [7]. Інтерфейс інтуїтивний та зрозумілий.

5.1 Вимоги до характеристик системи

Створений програмний продукт не висуває потреб до характеристик системи, а представляє собою вебсервіс, тому не потрібно інсталювати програму, підійде будь який пристрій, що має браузер та доступ в Інтернет.

5.2 Головна сторінка

На головній сторінці(рисунок 5.1) присутні поле для введення пошукового запиту, в ньому кнопка, щоб застосувати фільтри і кнопка пошуку правіше.

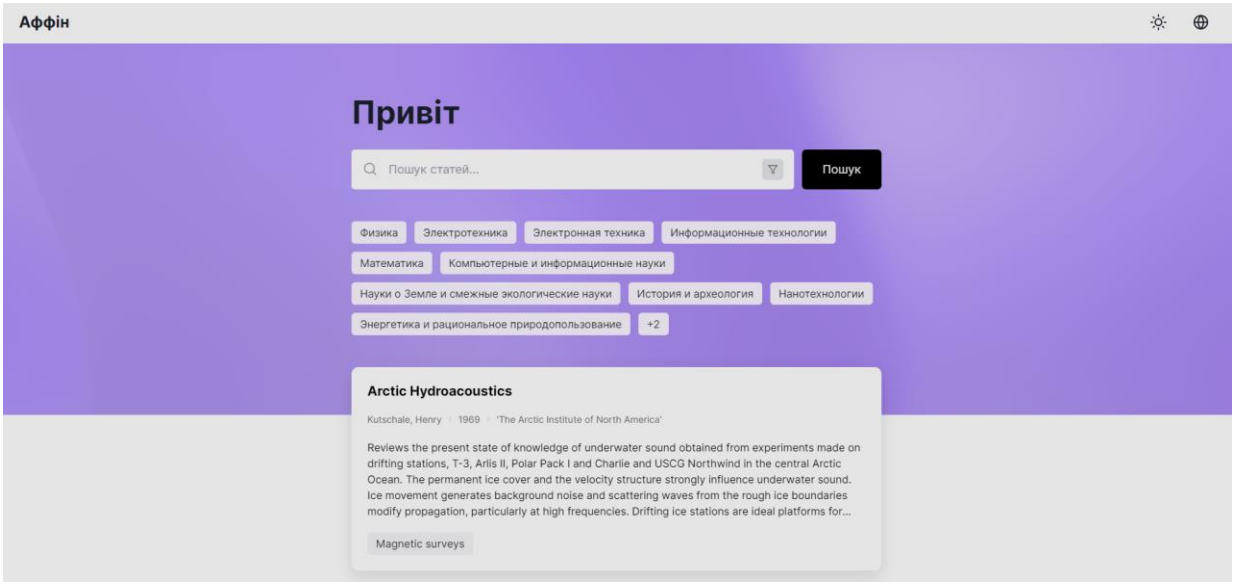

Рисунок 5.1 — Головна сторінка.

Зверху в правому кутку інтерфейсу знаходиться кнопка зміни теми на темну та кнопка вибору мови.

5.3 Створення пошукового запиту

Щоб знайти статтю, спочатку вводиться пошуковий запит в поле “Пошук статей...”. За необхідністю, виставляються пошукові фільтри (рисунок 5.2).

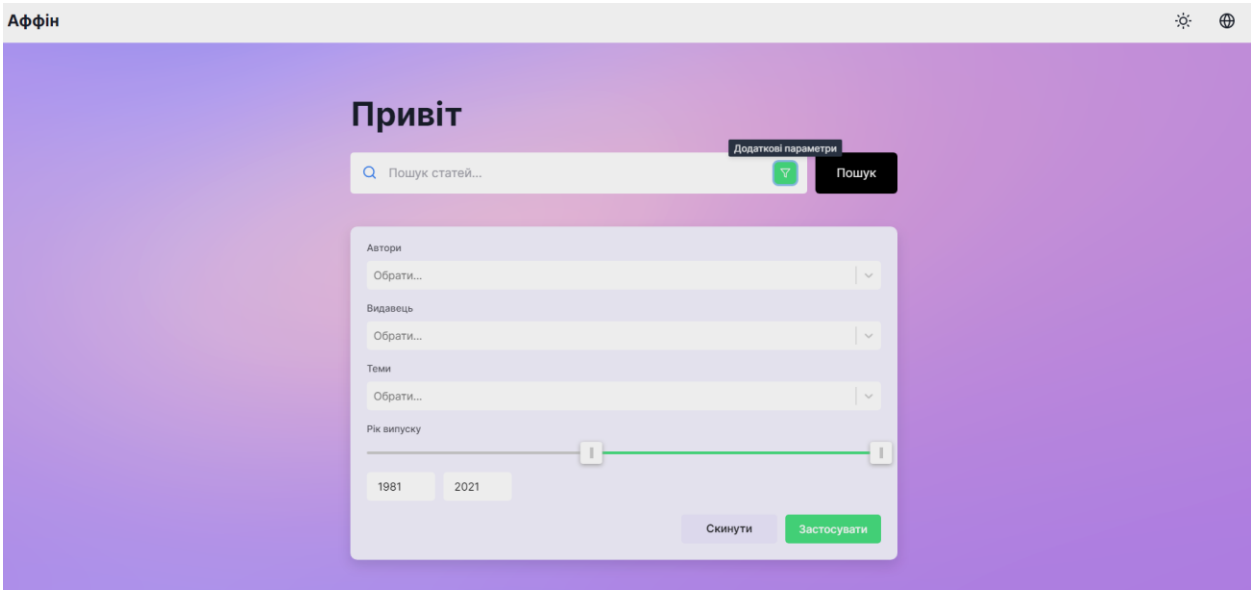

Рисунок 5.2 — Пошукові фільтри

Існує можливість уточнювати запит за такими параметрами, як: імена авторів, назва видавця, тема статті та діапазон років опублікування.

5.4 Результати пошукового запиту

Після виконання пошукового запиту, на екрані з'являються картки (рисунок 5.3), що коротко описують зміст статті, клацнувши на картку відкривається PDF-документ на який посилається стаття.

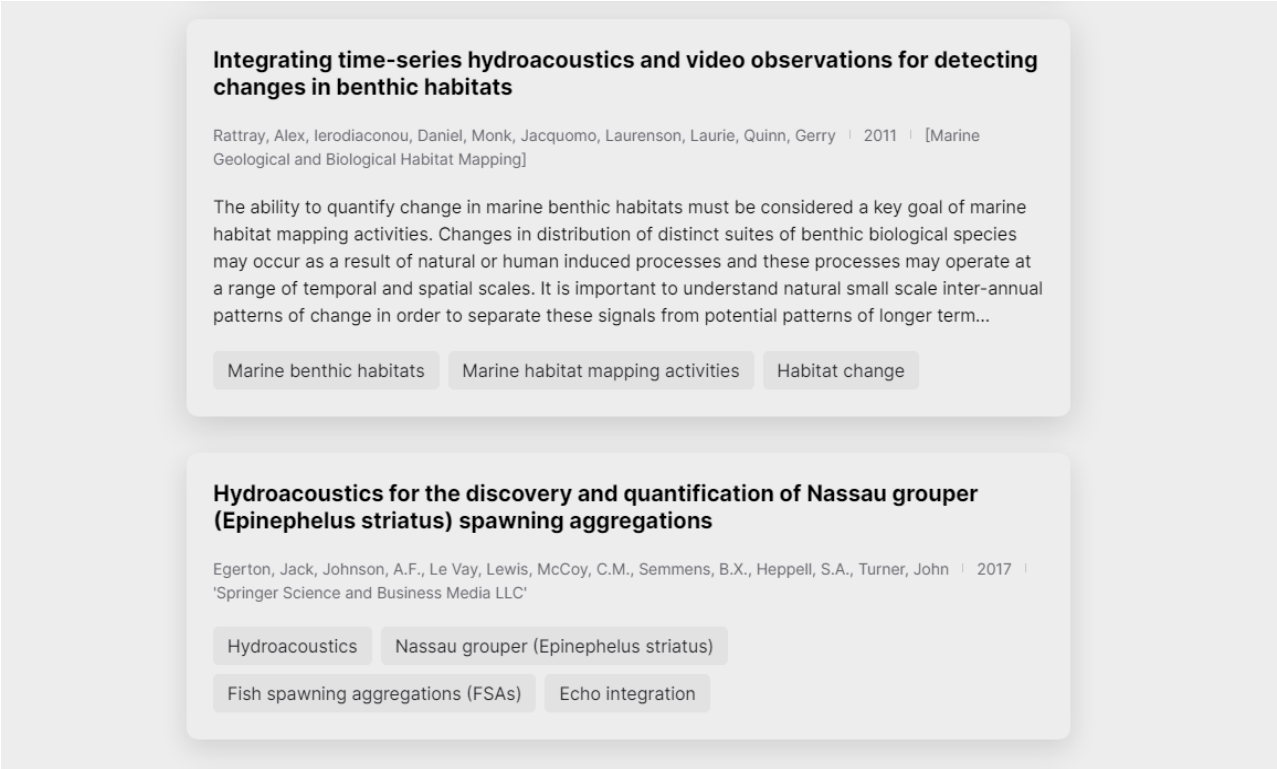

Рисунок 5.3 — Результат пошуку.

Опис статей залежить від сервісу на якому вона опублікована, деякі статті не мають повного опису.

Якщо пошук не дав результатів, на екран виводиться наступне повідомлення (рисунок 5.4).

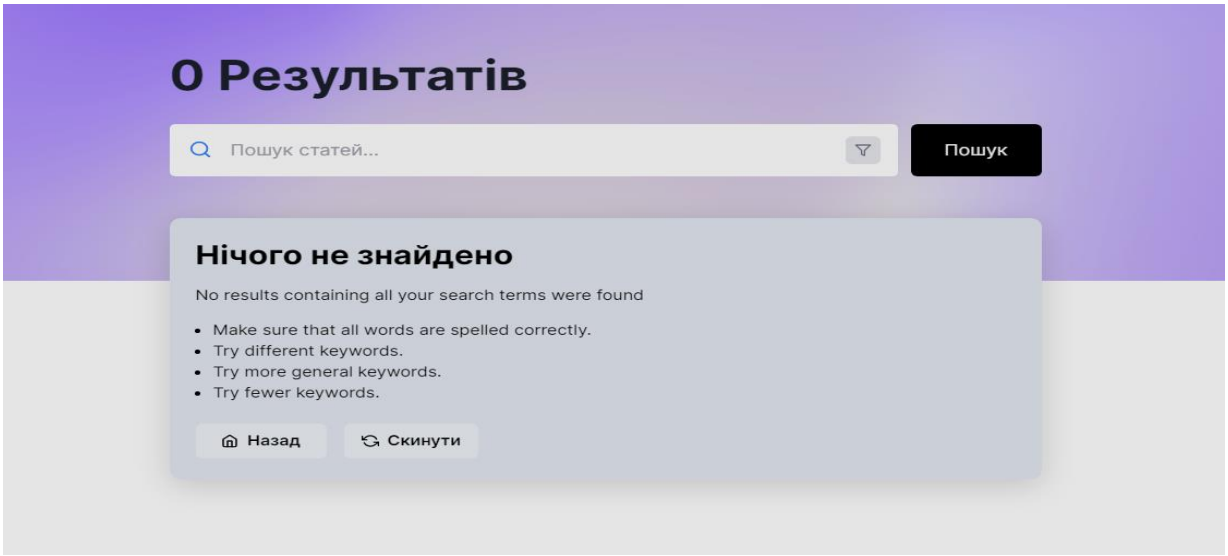

Рисунок 5.4 — Повідомлення, коли пошук не дав результатів.

Також виводяться рекомендації та надається можливість повернутись на головну сторінку або скинути пошукові фільтри.

5.5 Вид та локалізація інтерфейсу

У верхньому правому кутку можна змінювати тему та мову інтерфейсу (рисунок 5.5).

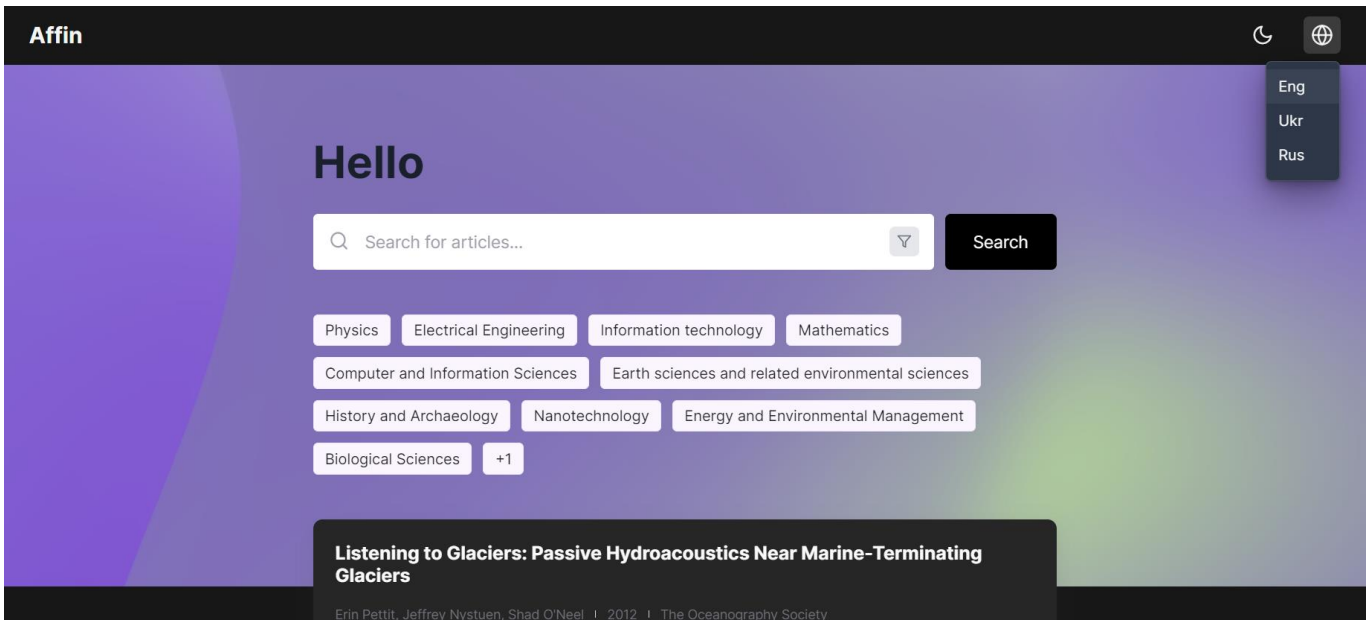

Рисунок 5.5 — Приклад зміни інтерфейсу на темну тему та англійську мову.

5.6 Висновки до розділу

У цьому розділі описані системні характеристики та усі сценарії взаємодії користувача з програмним продуктом.

ВИСНОВКИ

Під час виконання дипломної роботи, я набув навичок проектування та розробки серверної частини вебсервісу, розробленого на сучасному вебфреймворку Django, розробив внутрішні API-endpoint(и) для потреб клієнтської частини, а саме, для пошуку даних у базі даних проекту та зовнішніх джерелах, прийняв участь у створенні стандарту уніфікованого формату даних для запитів та відповідей, протестував запити завдяки REST-клієнту Postman, підключив модулі зовнішнього API для отримання даних з наукометричних баз даних, налаштував механізм перевірки дублікатів статей, налагодив та завантажив проект в хмарний сервіс Heroku.

Оскільки, проект “Створення єдиного реєстру наукових статей за пошуком по ключовому слову” проектувався та розроблявся командними зусиллями, мною були набуті навички роботи в команді, також були освоєні технології планування задач та систем контролю версій.

Створений продукт — це сучасний адаптивний вебсервіс, що підтримується на всіх актуальних платформах з доступом в інтернет. Створена база проекту відкрита для вдосконалень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Samuli Natri — The Easy Way(2end Edition) — Samuli Natri, 2018 - 165 с.
2. Senjeev Jaiswal, Ratar Kumar — Learning Django Web Development — Packt Publishing, 2015 - 336 с.
3. Toby Segaran — Programming Collective Intelligence — O'Reilly Media Inc, 2007 — 388 с.
4. William S. Vincent — Django for Beginners — William S. Vincent, 2018 - 105 с.
5. Django REST framework [Електронний ресурс] — Режим доступу: https://www.django-rest-framework.org
6. Django documentation [Електронний ресурс] — Режим доступу:

   https://djbook.ru

7. Аффін [Електронний ресурс] — Режим доступу:

   https://affin.vercel.app/uk

ДОДАТОК А

Модуль алгоритмів пошуку, фільтрації та структурування наукових матеріалів у мережі інтернет

Специфікація

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б

Аркушів 2

Київ — 2021 року

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 12-1	afin\afin\settings.py	Конфігураційний файл проекту
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 12-2	afin\afin\urls.py	Прив’язка запиту до класів представлень
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 12-3	afin\articles\views.py	Реалізація локальних представлень
УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 12-4	affin\general_api_module\api.py	Реалізація зовнішніх представлень

ДОДАТОК Б

Модуль алгоритмів пошуку, фільтрації та структурування наукових матеріалів у мережі інтернет

Лістинг програми

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 12-1

Аркушів 9

Київ — 2021 року

afin\afin\settings.py

```
"""
Django settings for afin project.

Generated by 'django-admin startproject' using Django 3.2.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""
import os
from pathlib import Path
import django_heroku


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-2g$#_&ag(nl-bs=^qgpnx@-%ad3#ok8jjgg&f%n=lf7^lupfu_'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['*']

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'corsheaders',
```

```
  'articles',
  'general_api_module',
  'history',
]

MIDDLEWARE = [
  'corsheaders.middleware.CorsMiddleware',


  'django.middleware.security.SecurityMiddleware',

  'whitenoise.middleware.WhiteNoiseMiddleware',

  'django.middleware.common.CommonMiddleware',

  'django.contrib.sessions.middleware.SessionMiddleware',
  'django.middleware.common.CommonMiddleware',
  'django.middleware.csrf.CsrfViewMiddleware',
  'django.contrib.auth.middleware.AuthenticationMiddleware',
  'django.contrib.messages.middleware.MessageMiddleware',
  'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'afin.urls'

CORS_ORIGIN_ALLOW_ALL = True
CORS_ALLOW_CREDENTIALS = True

CORS_ORIGIN_REGEX_WHITELIST = [
  'http://localhost:3030',
  'http://localhost:8000',
]

CORS_ALLOW_HEADERS = [
  'accept',
  'accept-encoding',
  'authorization',
  'content-type',
  'dnt',
  'origin',
  'user-agent',
  'x-csrftoken',
  'x-requested-with',
]
```

```
TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
      'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
      ],
    },
  },
]

WSGI_APPLICATION = 'afin.wsgi.application'

DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.postgresql',
    'NAME': 'd32n1t0uovqknh',
    'USER': 'yrwlvdfnthdgcg',
    'PASSWORD': '3104ec7c3364e17ef35ba9881ab8d82e7c418c19e98302bd58563446debe790c',
    'HOST': 'ec2-3-217-219-146.compute-1.amazonaws.com',
    'PORT': 5432,
  }
}

import dj_database_url

db_from_env = dj_database_url.config(conn_max_age=600)
DATABASES['default'].update(db_from_env)

# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
  {
    'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
  },
  {
    'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
```

```
  },
  {
    'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
  },
  {
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
  },
]

# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/


STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)
django_heroku.settings(locals())


# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

REST_FRAMEWORK = {
  'DEFAULT_RENDERER_CLASSES': (
    'rest_framework.renderers.JSONRenderer',
  ),
  'DEFAULT_PARSER_CLASSES': (
    'rest_framework.parsers.JSONParser',
  )
```

```
}
```

afin\afin\urls.py

```
"""afin URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
  https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
  1. Add an import:  from my_app import views
  2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
  1. Add an import:  from other_app.views import Home
  2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
  1. Import the include() function: from django.urls import include, path
  2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from rest_framework.routers import SimpleRouter

from general_api_module.api import PullArticlesListView
from articles.views import DocumentViewSet, AutocompleteAuthors, AutocompletePublishers

router = SimpleRouter()

router.register(r'articles', DocumentViewSet)

urlpatterns = [
  path('admin/', admin.site.urls),
  path('pullarticles/', PullArticlesListView.as_view()),
  path('authors/', AutocompleteAuthors.as_view()),
  path('publishers/', AutocompletePublishers.as_view()),
]

urlpatterns += router.urls
```

afin\articles\views.py

```
from django.http import JsonResponse
from rest_framework.filters import OrderingFilter, SearchFilter
from rest_framework.views import APIView
```

```
from rest_framework.viewsets import ModelViewSet

from articles.models import Document
from articles.serializers import DocumentSerializer, AuthorsListSerializer


class DocumentViewSet(ModelViewSet):
  queryset = Document.objects.all()
  serializer_class = DocumentSerializer


class AuthorsListViewSet(ModelViewSet):
  queryset = Document.objects.all()
  serializer_class = AuthorsListSerializer
  filter_backends = [OrderingFilter, SearchFilter]
  ordering_fields = ['authors']
  search_fields = ['authors']


class AutocompleteAuthors(APIView):
  def get(self, request):
    term = request.GET.get('term')
    authors_list = list()
    if 'term' in request.GET:
      qs = Document.objects.filter(authors__icontains=term)[:20]
      for authors in qs:
        for author in authors.authors:
          if term in author:
            if author not in authors_list:
              authors_list.append(author)

    else:
      qs = Document.objects.all()[:20]
      for authors in qs:
        for author in authors.authors:
          if author not in authors_list:
            authors_list.append(author)

    result = {"authors": authors_list[:20]}
    return JsonResponse(result, safe=False)


class AutocompletePublishers(APIView):
  def get(self, request):
```

```
    term = request.GET.get('term')
    publishers_list = list()
    if 'term' in request.GET:
        qs = Document.objects.filter(publisher__icontains=term)[:20]
        for publisher in qs:
            if publisher.publisher not in publishers_list:
                publishers_list.append(publisher.publisher)

    else:
        qs = Document.objects.all()[:20]
        for publisher in qs:
            if publisher.publisher not in publishers_list:
                publishers_list.append(publisher.publisher)

    result = {"publishers": publishers_list[:20]}
    return JsonResponse(result, safe=False)
```

afin\general_api_module\api.py

```
import pickle
import requests
from rest_framework.views import APIView
from rest_framework.response import Response

from articles.models import Document
from general_api_module.models import DataSource


class PullArticlesListView(APIView):

  def post(self, request):
    data_sources = DataSource.objects.all()
    articles = dict()
    articles["status"] = "ERROR"
    articles["total"] = 0
    articles["data"] = []
    for ds in data_sources:
        req = requests.post(
            url=ds.api,
            json=request.data
        )
```

```
    result = req.json()

    if articles["status"] == "ERROR":
        if result["status"] == "OK":
            articles["status"] = result["status"]

    if result['data']:
        articles["data"] += result["data"]
        articles["total"] += result["total"]

        qs = Document.objects.values_list('downloadUrl')
        local_articles = Document.objects.all()
        local_articles.query = pickle.loads(pickle.dumps(qs.query))
        local_articles = [loc_article['downloadUrl'] for loc_article in local_articles]

        for article in articles.get('data'):
            if article['downloadUrl'] in local_articles:
                pass
            else:
                Document.objects.create(downloadUrl=article['downloadUrl'],
                                        doi=article['doi'],
                                        title=article['title'],
                                        description=article['description'],
                                        lang=article['lang'],
                                        topics=article['topics'],
                                        authors=article['authors'],
                                        publisher=article['publisher'],
                                        year=article['year'],
                                        relations=article['relations']
                                        )

return Response(data=articles)
```

ДОДАТОК В

Модуль алгоритмів пошуку, фільтрації та структурування наукових матеріалів у мережі інтернет

Опис програми

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТР71233_21Б 13-1

Аркушів 10

Київ — 2021 року

АНОТАЦІЯ

Додаток містить опис модуля серверної частини пошуковий системи наукових статей за ключовим словом, що виконує завдання описані в розділі 1, а саме:

— Доступ до інформації накопиченої у базі даних проекту за пошуковим запитом;

— Отримання шуканої інформації від модулів зовнішнього API;

— Автоматична перевірка на дублікати та збереження даних до БД проекту;

— Можливість реєстрування та підключення нових модулів зовнішнього API;

— Перевірка відповідності вхідних даних до єдиного формату.

Серверну частину розроблено мовою програмування Python 3 з використанням фреймворку Django 3 у редакторі коду PyCharm Community Edition 2020.3.3.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку міститься опис основних компонентів серверної частини системи пошуку наукових статей за ключовим словом, що виконує завдання визначені в розділі 1. Додаток Б містить програмний код цих компонентів.

Для комфортної роботи з програмою потрібно мати персональний комп’ютер з обсягом оперативної пам’яті не менше 4 гігабайт та процесор, що має 2, або більше ядер. Також треба мати браузер та підключення до мережі Інтернет. Маніпулятори вводу (клавіатура та миша або сенсорний екран).

Серверну частину розроблено мовою програмування Python 3 з використанням фреймворку Django 3 у редакторі коду PyCharm Community Edition 2020.3.3.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб надає користувачу наступний функціонал:

— Доступ до інформації накопиченої у базі даних проекту за пошуковим запитом;

— Отримання шуканої інформації від модулів зовнішнього API;

— Автоматична перевірка на дублікати та збереження даних до БД проекту;

— Можливість реєстрування та підключення нових модулів зовнішнього API;

— Перевірка відповідності вхідних даних до єдиного формату.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Архітектура програмного забезпечення складається із застосунків: застосунок пошуку даних в БД проекту, застосунок пошуку даних з наукометричних баз.

Застосунки містять представлення та моделі, які разом реалізують обробку пошукових запитів API.

Застосунок — пакет, що містить в собі файли в яких реалізована логіка цього застосунку.

Представлення — клас, що містить функціонал отримання запиту, обробки та повернення результату. Представлення, що реалізують пошук даних в БД проекту, також, містять серіалізатори даних.

Серіалізатор — клас-сутність що конвертує дані з одного формату в інший.

Модель — клас, що описує поля в таблиці БД проекту.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для комфортної роботи з програмою потрібно мати персональний комп’ютер з обсягом оперативної пам’яті не менше 4 гігабайт та процесор, що має 2, або більше ядер. Також треба мати браузер та підключення до мережі Інтернет. Маніпулятори вводу (клавіатура та миша або сенсорний екран).

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблене програмне забезпечення представляє собою вебсервіс, та не потребує завантаження. Для отримання даних від сервера потрібно виконати наступні запити:

1. https://kpi-affin-2021.herokuapp.com/pullarticles/ (виконати HTTP POST запит та передати в тіло запиту JSON, відповідно до формату даних, який описано в розділі 4.1.1)
2. https://kpi-affin-2021.herokuapp.com/articles/ (виконати HTTP GET, детальніше в розділі 4.1.2. та 4.1.4.)
3. https://kpi-affin-2021.herokuapp.com/authors/ (виконати HTTP GET, детальніше в розділі 4.1.2. та 4.1.4.)
4. https://kpi-affin-2021.herokuapp.com/publishers/ (виконати HTTP GET, детальніше в розділі 4.1.2. та 4.1.4.)

ВХІДНІ ДАНІ

Вхідна інформація для виконання пошукового запиту:

1. URL посилання на пошуковий endpoint(HTTP метод POST);
2. Пошукові дані у нотації JSON (відповідно до формату даних, який описано в розділі 4.1.1);
3. URL посилання на endpoint(и)(HTTP метод GET) для отримання авторів і видавництв;

ВИХІДНІ ДАНІ

Вихідна інформація для пошукового запиту API у нотації JSON (див. розділ 4.1):

1) “status” — статусний код виконання запиту;
2) “total” — загальна кількість знайдених статей з усіх доступних джерел;
3) “data” — масив з даними про статті ;
4) “error” — повідомлення в разі помилки.

Вихідна інформація для запиту на отримання списку авторів:

1) ”authors” — масив з авторами.

Вихідна інформація для запиту на отримання списку видавництв:

1) ”publishers” — масив з видавництвами.