Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи надання наукових сервісів за домомогою Scrapy та ORM для взаємодії з базою даних

Виконав(ла): студент(ка) 4 курсу, групи СП-43
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Юрчишин Д. І.

Керівник _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення» ТНТУ, 2025, 50 сторінок, 12 рисунків, 3 таблиці, презентація, 23 джерела.

Тема: Розробка інформаційної системи надання наукових сервісів з використанням бібліотеки Scrapy та ORM для взаємодії з базою даних.

Об’єкт дослідження – процеси автоматизованого збору, структурування та надання доступу до наукової інформації через веб-інтерфейс

Предмет дослідження –методи та технології створення веб-орієнтованих інформаційних систем з використанням фреймворку Scrapy для збору даних та Django ORM для управління базою даних.

Мета роботи – створити повнофункціональну інформаційну систему для автоматизованого збору наукових публікацій з Semantic Scholar API, їх зберігання та надання зручного веб-інтерфейсу для пошуку й аналізу.

Методи дослідження: системний аналіз предметної області, об’єктно-орієнтоване проєктування, методи веб-скрейпінгу, реляційне моделювання даних, REST API архітектура, функціональне тестування.

Наукова новизна роботи полягає в комплексному підході до створення системи збору наукових сервісів, що інтегрує сучасні технології вебзбирання, асинхронної обробки та ефективного управління даними в єдиній архітектурі.

Практичне значення – розроблена система може використовуватися науковими установами та дослідниками для ефективного пошуку та аналізу наукових публікацій.

Ключові слова: Scrapy, Django ORM, Semantic Scholar API, Rest API, Celery, RabbitMQ, Scholar, наукові публікації, автоматизований збір даних, веб-орієнтована система.

ABSTRACT

Bachelor's thesis. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 «Software Engineering» TNTU, 2025, 50 pages, 12 figures, 3 tables, presentation, 23 references.

Topic: Development of an information system for providing scientific services using Scrapy library and ORM for database interaction.

Research object – processes of automated collection, structuring and providing access to scientific information through web interface.

Research subject – methods and technologies for creating web-oriented information systems using Scrapy framework for data collection and Django ORM for database management.

Purpose of work – to create a fully functional information system for automated collection of scientific publications from Semantic Scholar API, their structured storage and providing convenient web interface for search and analysis.

Research methods: systematic analysis of the subject area, object-oriented design, web scraping methods, relational data modeling, REST API architecture, functional testing.

Scientific novelty of the work lies in the comprehensive approach to creating a scientific services collection system that integrates modern web scraping technologies, asynchronous processing and efficient data in a unified architecture.

Practical value – the developed system can be used by scientific institutions and researchers for efficient search and analysis of scientific publications.

Keywords: Scrapy, Django ORM, Semantic Scholar API, Rest API, Celery, RabbitMQ, Scholar, scientific publications, automated data collection, web-oriented system.

ЗМІСТ

ВСТУП

Сучасний етап розвитку науки характеризується експоненційним зростанням обсягів наукової інформації та необхідністю забезпечення ефективного доступу до неї. За даними Національного наукового фонду США, у 2022 році у світі було опубліковано 3,3 мільйона наукових і технічних статей, що представляє значне зростання порівняно з 1,92 мільйона у 2016 році. Це створює серйозні виклики для дослідників у процесі пошуку релевантної інформації та підтримання актуальності знань у швидко розвиваючихся галузях науки.

Традиційні методи роботи з науковою літературою часто виявляються неефективними через обмежені можливості інтеграції різних джерел даних та складність аналізу великих інформаційних масивів.

Веб-скрейпінг як технологія автоматизованого збору даних з веб-ресурсів набуває все більшого значення в академічному середовищі. Scrapy будучи одним із провідних Python-фреймворків для витягнення вебданих, забезпечує потужні можливості для ефективного збору даних з різноманітних джерел. Поєднання Scrapy із об’єктно-реляційним відображенням (ORM) створює основу для побудови надійних інформаційних систем, здатних обробляти великі обсяги наукових даних.

Django ORM як інструмент взаємодії з базою даних надає розробникам абстрактний рівень для роботи з реляційними базами даних, дозволяючи зосередитися на бізнес-логіці додатка замість деталей SQL-запитів.

Semantic Scholar API являє собою один із найпотужніших програмних інтерфейсів для доступу до наукових публікацій, забезпечуючи структурований доступ до метаданих понад 200 мільйонів наукових робіт. Використання цього API в поєднанні з технологіями веб-скрейпінгу відкриває нові можливості для створення спеціалізованих наукових сервісів.

Асинхронна обробка завдань з використанням Celery та RabbitMQ стає критично важливою при роботі з великими обсягами даних та довготривалими операціями збору інформації.

Актуальність теми дослідження зумовлена зростаючими потребами наукової спільноти в ефективних інструментах пошуку та аналізу літератури, необхідністю автоматизації процесів збору та структурування наукової інформації, потребою інтеграції даних з різних наукових джерел в єдиному інтерфейсі та важливістю створення масштабованих рішень для обробки великих обсягів наукових даних.

Мета дослідження – розробити інформаційну систему для надання наукових сервісів з використанням технологій Scrapy для автоматизованого збору даних та Django ORM для ефективної взаємодії з базою даних.

Для досягнення поставленої мети необхідно проаналізувати існуючі рішення для доступу до наукової інформації та виявити їх обмеження, дослідити можливості фреймворку Scrapy та Semantic Scholar API для збору наукових даних, спроєктувати архітектуру системи з використанням Django та PostgreSQL, реалізувати модуль автоматизованого збору даних з підтримкою асинхроної обробки, розробити веб-інтерфейс з можливостями пошуку, фільтрації та експорту результатів, а також провести тестування системи та оцінити її ефективність.

Предмет дослідження – архітектурні рішення та програмні засоби для створення масштабованих веб-платформ збору та аналізу наукової інформації.

У роботі застосовуються методи системного аналізу для дослідження предметної області, UML-моделювання для проєктування архітектури системи, технології RESTful API для взаємодії з зовнішніми сервісами та методи модульного тестування для забезпечення якості продукту.

Наукова новизна полягає у розробці гібридної архітектури, що інтегрує патерни асинхронної обробки з реляційними базами даних для забезпечення високої пропускної здатності при роботі з науковими метаданими, а також у створенні адаптивної системи фільтрації.

Практичне значення роботи визначається можливістю впровадження розробленої системи в діяльність наукових бібліотек, дослідницьких інститутів та університетських центрів для автоматизації каталогізації публікацій, формування тематичних добірок літератури та проведення наукометричних аналізів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз сучасних наукових інформаційних систем

Сучасний науковий процес неможливо уявити без ефективних інформаційних систем, що забезпечують доступ до величезних масивів наукових даних. Наукові інформаційні системи являють собою комплексні програмно-апаратні рішення, призначені для збору, зберігання, обробки та надання доступу до наукової інформації. Основними компонентами таких систем є бази даних наукових публікацій, пошукові механізми, системи індексації та інтерфейси користувача.

Google Scholar, будучи найпоширенішою науковою пошуковою системою, індексує понад 160 мільйонів документів з різних джерел. Система використовує алгоритм ранжування, що враховують кількість цитувань, релевантність та авторитет джерел. Однак Google Scholar має суттєві обмеження щодо програмного доступу до даних та відсутність структурованого API для автоматизованого збору інформації.

Web of Science, розроблена компанією Clarivate Analytics, представляє собою одну з найавторитетніших наукових баз даних, що містять понад 79 мільйонів записів [9]. Система забезпечує високу якість індексованих публікацій завдяки ретельному відбору журналів, проте доступ до неї є платним та обмеженим для багатьох дослідників.

Scopus від Elsevier охоплює понад 84 мільйони записів з різних наукових галузей [10]. Платформа надає розширені можливості аналізу цитувань та співпраці між авторами, проте також має комерційний характер та обмежений доступ.

Semantic Scholar, створена Allen Institute for AI, представляє інноваційний підхід до наукового пошуку, використовуючи методи штучного інтелекту для аналізу семантичного змісту публікацій [11]. Платформа індексує понад 200 мільйонів наукових робіт та надає безкоштовний API з широкими можливостями програмного доступу.

Основними проблемами існуючих наукових інформаційних систем є фрагментованість даних між різними платформами, обмежений програмний доступ до багатьох комерційних систем, відсутність уніфікованих стандартів метаданих та складність інтеграції даних з різних джерел в єдиному інтерфейсі.

1.2 Огляд технологій веб-скрейпінгу та їх застосування

Веб-скрейпінг як метод автоматизованого збору даних з веб-ресурсів набув широкого поширення в дослідницькій діяльності завдяки можливості отримання структурованої інформації з різноманітних онлайн-джерел. Сучасні технології веб-скрейпінгу дозволяють обробляти як статичні HTML-сторінки, так і динамічний контент, що генерується JavaScript-додатками.

Scrapy являє собою один з найпотужніших Python-фреймворків для веб-скрейпінгу, що забезпечує високу продуктивність та масштабованість [3]. Архітектура фреймворку базується на асинхронній обробці запитів з використанням Twisted фреймворку, що дозволяє одночасно обробляти тисячі веб-сторінок. Основними компонентами є spiders для навігації по сайтах, pipelines для обробки даних та middlewares для кастомізації процесу збору.

Beautiful Soup представляє альтернативний підхід до веб-скрейпінгу, орієнтовний на простоту використання та парсинг HTML/XML документів [2]. Бібліотека особливо ефективна для разових задач збору даних, проте поступається Scrapy у продуктивності при обробці великих обсягів інформації.

Selenium WebDriver забезпечує можливість автоматизації браузерів та збору даних з динамічних веб-додатків [2]. Технологія особливо корисна для роботи з JavaScript-насиченими сайтами, проте характеризується значним споживанням ресурсів та нижчою швидкістю порівняно з HTTP-клієнтами.

Requests-HTML поєднує простоту бібліотеки Requests з можливостями виконання JavaScript [2]. Рішення підходить для середніх за складністю задач веб-

скрейпінгу, де потрібна обробка динамічного контенту без повної емуляції браузера.

У контексті наукових досліджень веб-скрейпінг застосовується для збору бібліографічних даних, моніторингу публікацій за певними темами, аналізу цитувань та виявлення наукових трендів. Особливо актуальним є його використання для агрегації даних з різних наукових платформ з метою створення уніфікованих баз знань.

1.3 Дослідження Semantic Scholar API та подібних сервісів

Semantic Scholar API представляє собою програмний інтерфейс для доступу до величезної бази наукових публікацій, що розробляється Allen Institute for AI. API надає структурований доступ до метаданих понад 200 мільйонів наукових робіт з різних галузей знань, включаючи інформатику, медицину, біологію, фізику та інші науки.

Основні можливості Semantic Scholar API включають пошук публікацій за ключовими словами, отримання детальної інформації про статті, доступ до даних про авторів та їх публікації, аналіз цитувань та референсів, а також отримання рекомендацій щодо релевантних робіт. API підтримує різні фільтри для уточнення пошуку, включаючи часові рамки, типи публікацій, предметні області та рівень відкритого доступу.

Унікальною особливістю Semantic Scholar є використання методів машинного навчання для семантичного аналізу наукових текстів. Система автоматично виділяє ключові концепції, визначає тематичну спорідненість робіт та оцінює вплив публікацій на наукову спільноту. Це дозволяє користувачам отримувати більш релевантні результати пошуку порівняно з традиційними підходами, що базуються лише на лексичному збігу ключових слів.

Crossref API надає доступ до метаданих наукових публікацій через DOI (Digital Object Identifier) систему [12]. Сервіс обслуговує понад 130 мільйонів DOI та забезпечує стандартизований доступ до бібліографічної інформації. Crossref особливо корисний для верифікації даних про публікації та отримання актуальних метаданих.

ORCID API дозволяє доступ до профілів дослідників та їх наукових досягнень [13]. Система містить понад 16 мільйонів унікальних ідентифікаторів дослідників та забезпечує зв'язок між авторами та їх публікаціями. ORCID API корисний для задач ідентифікації авторів та аналізу їх наукової продуктивності.

ArXiv API надає доступ до препринтів наукових робіт у галузях фізики, математики, інформатики та суміжних дисциплін [14]. Система містить понад 2 мільйони препринтів та забезпечує оперативний доступ до найновіших наукових результатів до їх публікації в рецензованих журналах.

Europe PMC API забезпечує доступ до біомедичної літератури та пов'язаних з нею даних [15]. Платформа індексує понад 37 мільйонів публікацій та надає розширені можливості пошуку за медичними термінами та концепціями.

Основними перевагами використання API порівняно з веб-скрейпінгом є стабільність та надійність доступу до даних, структурованість отриманої інформації, офіційна підтримка з боку розробників та відсутність ризику блокування доступу. Однак API також мають обмеження щодо швидкості запитів та обсягів даних, що можна отримати за певний період часу.

Таблиця 1.1 – Характеристики основних наукових API

API сервіс	Ліміт запитів	Безкоштовний доступ	Типи даних	Формат відповіді
Semantic Scholar	100 запитів/5 хв	Так	Статті, автори, цитування	JSON
Crossref	50 запитів/с	Так	Метадані, DOI	JSON, XML
ORCID	24 запитів/с	Так	Профілі авторів	JSON, XML

Продовження таблиці 1.1

ArXiv	1 запит/3с	Так	Препринти	XML, Atom
Europe PMC	1000 запитів/с	Так	Біомедична література	JSON, XML

1.4 Порівняльний аналіз існуючих рішень

Для комплексного розуміння ландшафту наукових інформаційних систем проведено порівняльний аналіз основних платформ за критеріями обсягу даних, якості інформації, доступності, функціональності та можливостей інтеграції.

За обсягом даних лідирує Google Scholar з понад 160 мільйонами індексованих документів, проте якість та структурованість цих даних часто викликає питання через автоматизований характер індексації. Semantic Scholar, маючи 200 мільйонів записів, забезпечує вищу якість даних завдяки використанню штучного інтелекту для їх обробки та верифікації.

Щодо якості інформації, Web of Science та Scopus традиційно вважаються еталонами через ретельний відбір журналів та строгі критерії індексації. Однак ці системи мають обмежене покриття відкритих джерел та препринтів, що може призводити до неповноти даних у швидко розвиваючихся галузях.

З точки зору доступності, безкоштовні платформи як PubMed, ArXiv та Semantic Scholar забезпечують демократичний доступ до наукової інформації, тоді як комерційні системи створюють бар'єри для дослідників з обмеженими ресурсами.

Функціональні можливості сучасних систем значно різняться. Semantic Scholar виділяється розвиненими засобами семантичного пошуку та аналізу впливу публікацій. Web of Science пропонує потужні інструменти наукометричного аналізу. PubMed забезпечує спеціалізовані можливості медичного пошуку з використанням MeSH термінології.

Можливості програмної інтеграції критично важливі для створення нових наукових сервісів. Semantic Scholar API надає найширші можливості безкоштовного програмного доступу з підтримкою складних запитів та фільтрації. Crossref API забезпечує стандартизований доступ до метаданих через DOI систему. Комерційні платформи часто обмежують програмний доступ або надають його за додаткову плату.

Таблиця 1.2 – Порівняльна характеристика основних наукових систем

Платформа	Обсяг даних (млн)	API	Якість даних	Предметна область
Google Scholar	160+	Обмежений	Середня	Загальний пошук
Semantic Scholar	200+	Повний	Висока	Сематичний пошук, ШІ-аналіз
Web of Science	79+	Платний	Дуже висока	Наукометричний аналіз
PubMed	35+	Повний	Висока	Медичний пошук (MeSH)

Аналіз виявив ключові недоліки існуючих рішень: фрагментацію наукової інформації між різними платформами, відсутність уніфікованих інтерфейсів для роботи з даними, обмежені можливості кастомізації та персоналізації пошуку, складність інтеграції даних з різних джерел.

Ці обмеження обґрунтовують необхідність розробки нових рішень, що поєднують переваги різних підходів: використання відкритих API для отримання якісних структурованих даних, застосування веб-скрепінгу для доступу до додаткових джерел, створення уніфікованих інтерфейсів для роботи з гетерогенними даними та забезпечення гнучких можливостей налаштування відповідно до потреб користувачів.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

2.1 Архітектура та вибір технологій

Розроблена система для надання наукових сервісів побудована за принципами сучасної веб-орієнтованої архітектури з використанням мікросервісного підходу. Система складається з кількох ключових компонентів, що забезпечують повний цикл роботи з науковими даними: від збору та обробки до надання доступу через веб-інтерфейс та API. Архітектура забезпечує розподіл відповідальностей між компонентами, що дозволяє системі бути масштабованою, надійною та легкою для підтримки. Кожен рівень має чітко визначені функції та інтерфейси взаємодії з іншими компонентами.

Вибір технологічного стеку обумовлений специфікою задач системи, вимогами до продуктивності та необхідністю забезпечення надійної роботи з великими обсягами наукових даних. Ключові компоненти системи реалізовано у вигляді спеціалізованих модулів: клієнт для взаємодії з Semantic Scholar API (додаток А, лістинг А.1) та Scrapy павук для автоматизованого збору даних (додаток А, лістинг А.2).

Таблиця 2.1 – Технологічний стек системи

Компонент	Технологія	Версія	Обгрунтування вибору
Backend Framework	Django	5.2	Швидка розробка вбудована ORM
Web Scraping	Scrapy	2.13.1	Асинхроність, масштабованість
Task Queue	Celery	5.5.2	Асинхроні задачі
Database	PostgreSQL	14	Реляційна модель, JSON підтримка

Як основний бекенд фреймворк обрано Django, вибір обумовлений специфічними потребами проекту та технічними вимогами до обробки великих обсягів даних. Ключові переваги Django для проекту:

- Швидкість розробки та готові компоненти
- Вбудована ORM
- Вбудовані засоби безпеки (CSRF, XSS, SQL-ін’єкцій)
- Масштабованість архітектури
- Розвинена екосистема

Scrapy фреймворк обрано для реалізації збору наукових даних та інтеграції з зовнішніми API. Рішення продиктоване необхідністю ефективної обробки великих обсягів запитів до Semantic Scholar API та інших наукових ресурсів. Ключові переваги Scrapy для проекту:

- Асинхрона архітектура на базі Twisted
- Гнучка middleware система
- Вбудовані засоби обробки
- Seamless інтеграція з Django

Асинхронна обробка Celery та RabbitMQ обрана для організації обробки довготривалих задач збору даних. Ця архітектура вирішує проблему блокування веб-інтерфейсу під час виконання операцій, які можуть тривати десятки хвилин. Архітектурні переваги рішення:

- Розподілена обробка задач
- Гарантована надійність
- Моніторинг та управління
- Elastic масштабованість

PostgreSQL обрано як основну систему зберігання даних з урахуванням специфіки наукових метаданих та вимог до продуктивності системи. Технічне обгрунтування вибору:

- Нативна підтримка JSON структур
- Високопродуктивна індексація
- ACID гарантії цілісності
- Функціональна розширюваність

Архітектура системи наукових сервісів побудована за принципами багаторівневої організації з чітким розподілом функціональних обов'язків між компонентами. Система реалізує модель від клієнта до сховища даних з проміжними рівнями обробки, що забезпечує гнучкість, масштабованість та можливість незалежного розвитку окремих модулів.

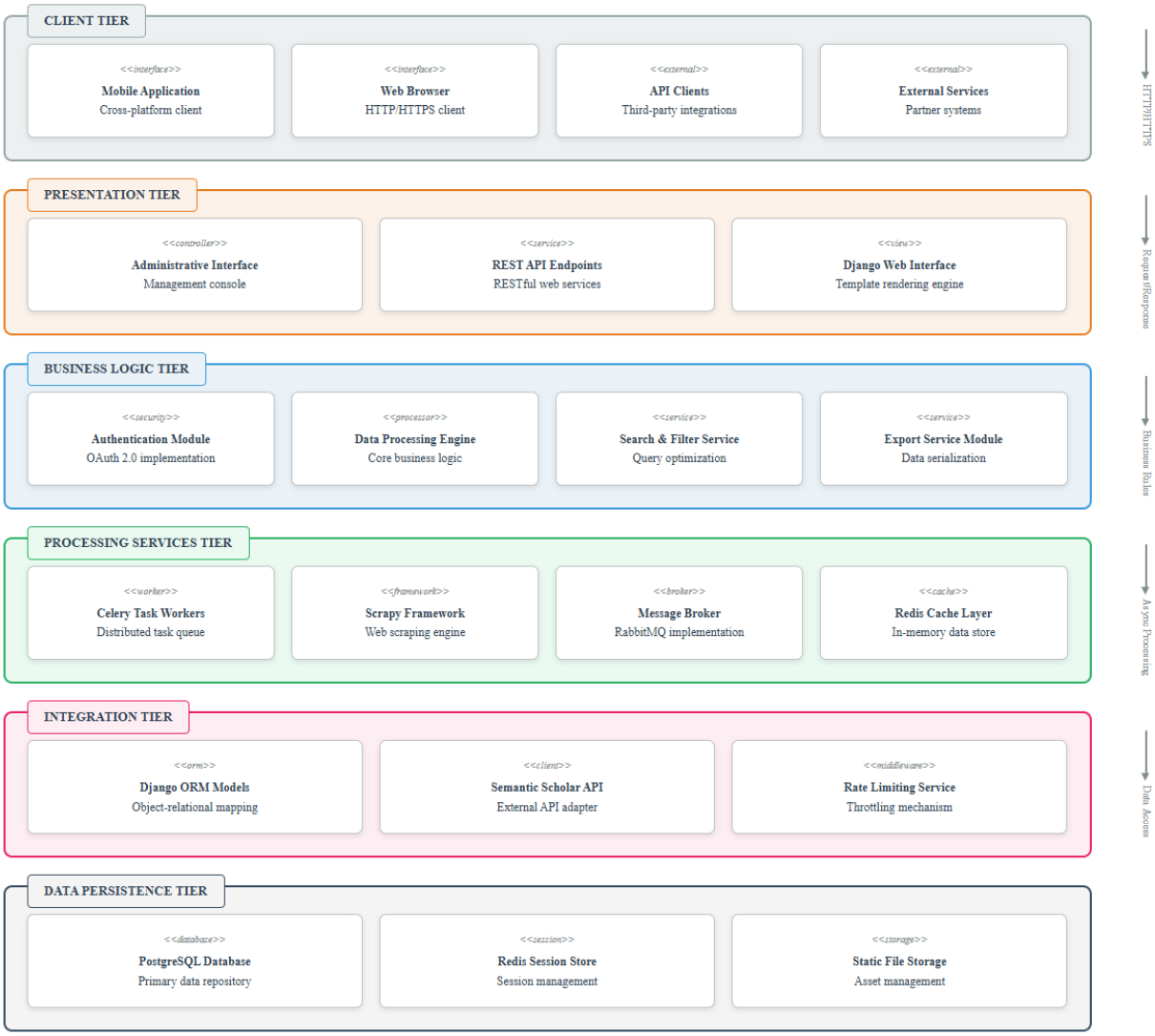


Рисунок 2.1 – Багаторівнена архітектура інформаційної системи

Архітектура складається з шести основних рівнів:

Клієнтський рівень (Client Layer) включає різноманітні типи клієнтів: веб-браузери для інтерактивної роботи користувачів, мобільні додатки для

портативного доступу, API клієнти для програмної інтеграції та зовнішні сервіси для автоматизованої взаємодії.

Презентаційний рівень (Presentation Layer) забезпечує різні інтерфейси доступу до функціональності системи: Django веб-інтерфейс для користувачів, REST API endpoints для програмного.

Рівень бізнес-логіки (Business Logic Layer) містить основну логіку обробки запитів: систему аутентифікації та авторизації через OAuth2, компоненти обробки даних та бізнес-правил, пошуковий механізм з фільтрацією та сервіси експорту результатів у різних форматах.

Рівень сервісів обробки (Processing Services Layer) відповідає за асинхронну обробку складних операцій: Scrapy фреймворк для збору даних, Celery workers для виконання довготривалих задач, RabbitMQ для управління чергою повідомлень.

Рівень інтеграції (Integration Layer) забезпечує взаємодію з зовнішніми системами: клієнт для роботи з Semantic Scholar API, Django ORM для абстракції роботи з базою даних та системи обмеження частоти запитів для дотримання лімітів зовнішніх API.

Рівень зберігання даних (Data Storage Layer) включає системи постійного та тимчасового зберігання: PostgreSQL як основну базу даних, Redis для зберігання сесій та кешу.

Для демонстрації принципів взаємодії між компонентами системи розглянемо типовий сценарій роботи користувача з системою від ініціації запиту на пошук наукових публікацій до отримання та експорту результатів.

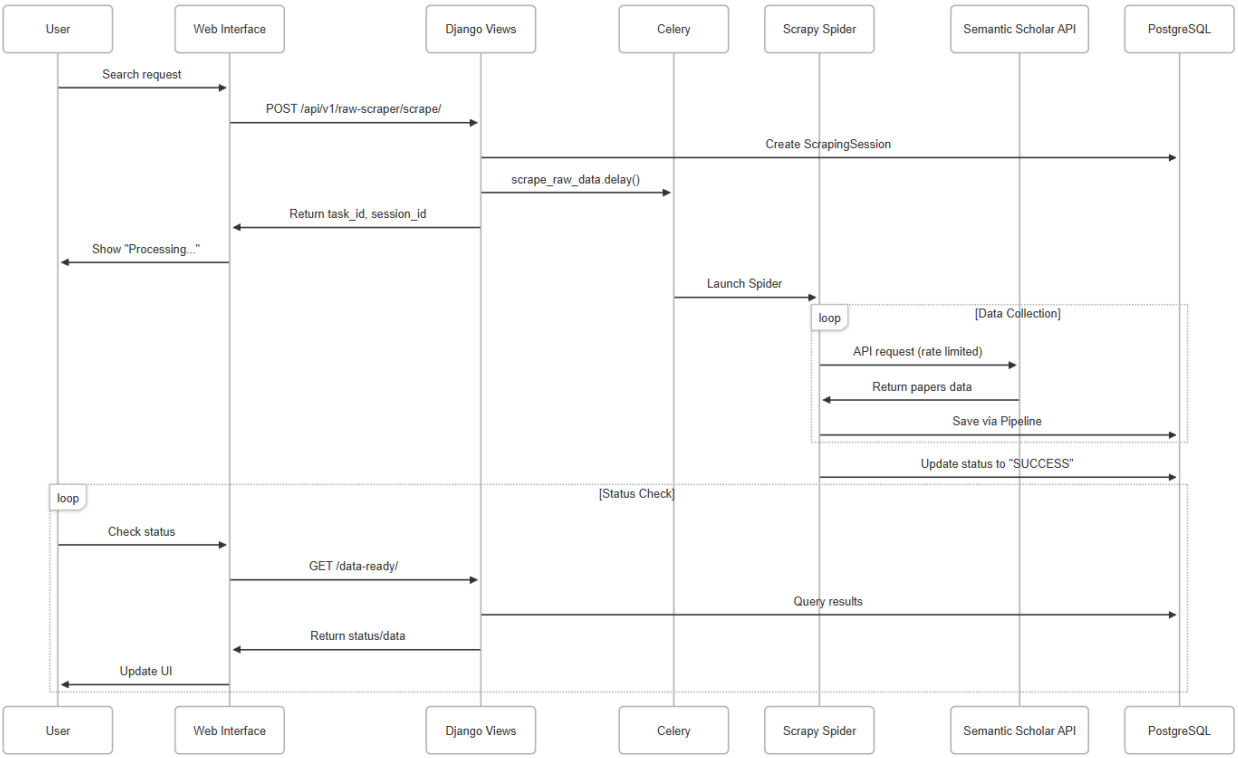


Рисунок 2.2 – Діаграма потоку даних при обробці запитів користувача

Процес обробки запиту складається з декількох етапів:

Ініціація запиту користувачем. Користувач через веб-інтерфейс створює запит на пошук наукових публікацій, вказуючи параметри пошуку (ключові слова, часовий період, тип публікацій, фільтри за предметними областями). Веб-інтерфейс валідує параметри та формує HTTP POST запит до відповідного API endpoint.

Обробка запиту в Django Views. Django view приймає запит, проводить аутентифікацію користувача та валідацію параметрів через відповідний сералізатор. Створюється запис ScrapingSession у базі даних для відслідковування стану операції та зберігання параметрів запиту.

Асинхронне виконання через Celery. Django view ініціює асинхронну задачу через Celery, передаючи всі необхідні параметри. Користувач одразу отримує відповідь з унікальними ідентифікаторами task_id та session_id для подальшого відслідковування прогресу.

Збір даних через Scrapy. Celery worker запускає Scrapy spider, який виконує серію запитів до Semantic Scholar API з дотриманням обмежень на частоту запитів.

Кожен отриманий результат обробляється через pipeline компоненти та зберігається в базі даних.

Моніторинг статусу виконання. Користувач періодично опитує систему щодо статусу виконання задачі через спеціальний endpoint. Система повертає інформацію про прогрес збору даних, кількість знайдених публікацій та поточний стан операції.

Надання результатів. По завершенні збору даних користувач може переглядати результати через веб-інтерфейс, фільтрувати їх за різними критеріями та експортувати у форматах CSV або Excel для подальшого аналізу.

Така архітектура забезпечує відокремлення довготривалих операцій від інтерфейсу користувача, можливість обробки кількох запитів одночасно та надійність системи при збоях окремих компонентів.

2.2 Проєктування бази даних

Структура бази даних системи наукових сервісів розроблена з урахуванням специфіки зберігання наукових метаданих, забезпечення цілісності даних та оптимізації продуктивності запитів. База даних організована навколо двох основних доменів: управління користувачами та обробка наукових публікацій. Також вона забезпечує:

Нормалізацію даних з розділенням інформації про користувачів, наукові публікації та авторів у окремі таблиці, що запобігає дублюванню та забезпечує цілісність.

Гнучкість зберігання метаданих через використання JSON полів для зберігання додаткових атрибутів, які можуть відрізнятися залежно від типу публікації або джерела даних.

Відслідковування операцій через модель ScrapingSession, що дозволяє зберігати параметри кожного запиту користувача та моніторити стан виконання.

Оптимізацію пошуку через створення індексів на часто використовуваних полях для фільтрації та сортування.

Концептуальна модель бази даних системи включає основні сутності та зв’язки між ними. Центральними сутностями є користувачі таблиця User, профілі Profile, наукові публікації ScholarRawRecord, автори ScholarAuthor та сесії збору даних ScrapingSession.

Рисунок 2.3 – Концептуальна модель даних інформаціної системи

2.3 Основи REST архітектури

REST (Representational State Transfer) є архітектурним стилем для розробки веб-сервісів, вперше описаним Роєм Філдінгом у 2000 році в його дисертації [5]. REST базується на принципах HTTP протоколу та визначає набір обмежень для створення масштабованих та надійних веб-додатків. Основними принципами REST

є клієнт-серверна архітектура, безстановість (stateless), кешованість, уніформний інтерфейс, багаторівнева система та код на вимогу. У контексті інформаційних систем REST API забезпечує стандартизований спосіб взаємодії між різними компонентами системи та зовнішніми клієнтами. Використання REST принципів дозволяє створити API, який є інтуїтивно зрозумілим, легко масштабованим та простим у підтримці.

REST API використовує стандартні HTTP методи для виконання операцій над ресурсами:

- GET для отримання даних
- POST для створення нових ресурсів
- PUT для повного оновлення ресурсів
- PATCH для часткового оновлення
- DELETE для видалення ресурсів

Кожен ресурс у REST API має унікальний URI (Uniform Resource Identifier), що дозволяє клієнтам звертатися до конкретних об'єктів системи. Відповіді сервера містять не лише дані, але й метаінформацію про стан операції через HTTP статус коди.

2.4 Модуль автентифікації та управління користувачами

Система наукових сервісів реалізує RESTful API з логічним групуванням endpoints за функціональними областями. API організовано навколо двох основних модулів: автентифікації/авторизації користувачів та управління науковими даними. Модуль автентифікації системи реалізує комплексний підхід до управління користувачами з використанням двоетапної верифікації та OAuth2 токенів. Архітектура модуля забезпечує безпечну реєстрацію, автентифікацію та управління сесіями користувачів. Реєстрація нових користувачів реалізована як

двоетапний процес, що забезпечує верифікацію контактної інформації та запобігає створенню фіктивних облікових записів.

Першим етапом є ініціація реєстрації через endpoint registration_request, який приймає POST запити з базовою інформацією про користувача. Система підтримує реєстрацію як через email, так і через номер мобільного телефону, що забезпечує гнучкість для різних категорій користувачів. Валідація вхідних даних включає перевірку унікальності контактної інформації, складності пароля та формату номера телефону. При успішній валідації система генерує шестизначний код верифікації з обмеженим терміном дії (30 хвилин) та відправляє його на вказаний контакт.

Другий етап реалізовано через endpoint registration_confirm, який завершує процес створення облікового запису після підтвердження коду верифікації. Система перевіряє код на валідність, строк дії та кількість невдалих спроб введення. При успішній верифікації створюється новий користувач у системі Django User, генерується відповідний профіль користувача та автоматично видаються OAuth2 токени для негайного доступу до системи. Автентифікація користувачів реалізована через endpoint login. Система використовує вбудовані механізми Django для перевірки паролів з підтримкою хешування та захисту від timing attacks. При успішній автентифікації генеруються access та refresh токени згідно з OAuth2 стандартом. Access токени мають обмежений термін дії для мінімізації ризиків безпеки, тоді як refresh токени забезпечують довготривале підтримання сесії користувача без необхідності повторного введення облікових даних.

Процес відновлення паролів також реалізований як двоетапна операція для забезпечення безпеки. Endpoint password_reset_request ініціює процес відновлення, приймаючи email або номер телефону користувача. Система перевіряє існування облікового запису з вказаним контактом та генерує код верифікації для підтвердження права на відновлення пароля. Завершення процесу відновлення здійснюється через endpoint password_reset_confirm, який приймає код верифікації та новий пароль. Система валідує код, перевіряє складність нового пароля та

оновлює хеш пароля в базі даних з автоматичним скасуванням всіх активних токенів для забезпечення безпеки.

Endpoint refresh_token забезпечує механізм оновлення access токенів без повторної автентифікації користувача. Цей підхід дозволяє підтримувати високий рівень безпеки через короткий термін дії access токенів.

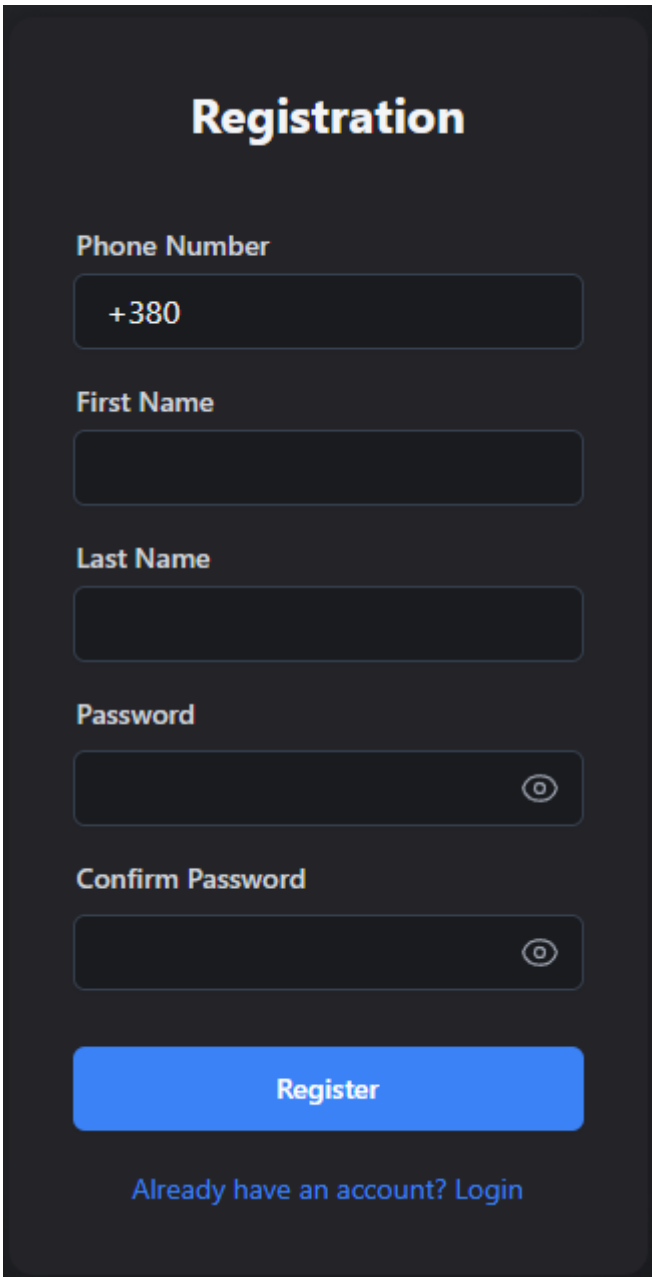


Рисунок 2.4 – Інтерфейс реєстрації користувача

2.5 Система збору наукових публікацій

Центральним компонентом системи є Scrapy павук, реалізація якого наведена в додатку А (лістинг А.2), що реалізована через endpoint scrape, який приймає POST запити з параметрами пошуку та ініціює асинхронну обробку через Celery worker. Така архітектура забезпечує неблокуючу роботу веб-інтерфейсу під час виконання довготривалих операцій збору даних, які можуть тривати від декількох секунд до години залежно від обсягу запиту, та додаткових фільтрів.

Система підтримує широкий спектр параметрів фільтрації для точного налаштування пошуку відповідно до потреб дослідника. Базовий параметр query приймає текстовий запит з обмеженням до 10 слів, що забезпечує оптимальний баланс між точністю пошуку та продуктивністю API. Часові фільтри year_from та year_to дозволяють обмежити результати певним періодом публікацій, що особливо важливо для дослідження еволюції наукових областей.

Параметр limit контролює максимальну кількість результатів з діапазоном від 1 до 1000 публікацій, при цьому система автоматично обробляє пагінацію API для отримання всіх запитаних результатів. Фільтр fields_of_study приймає масив наукових галузей з попередньо визначеного переліку, що включає основні дисципліни від комп'ютерних наук до медицини та соціальних наук.

При успішній валідації параметрів система створює запис ScrapingSession у базі даних для відстеження операції та ініціює асинхронну задачу Celery. Клієнт отримує унікальні ідентифікатори task_id та session_id, які використовуються для подальшого моніторингу прогресу та доступу до результатів. Endpoint filter-options надає актуальну інформацію про доступні параметри фільтрації без необхідності автентифікації. Цей сервіс призначений для динамічного формування інтерфейсу користувача та забезпечує консистентність валідації між клієнтом та сервером. Відповідь включає списки валідних наукових галузей, типів публікацій, допустимі діапазони років та обмеження параметрів. Така централізація конфігурації дозволяє легко оновлювати доступні опції без змін у клієнтських додатках.

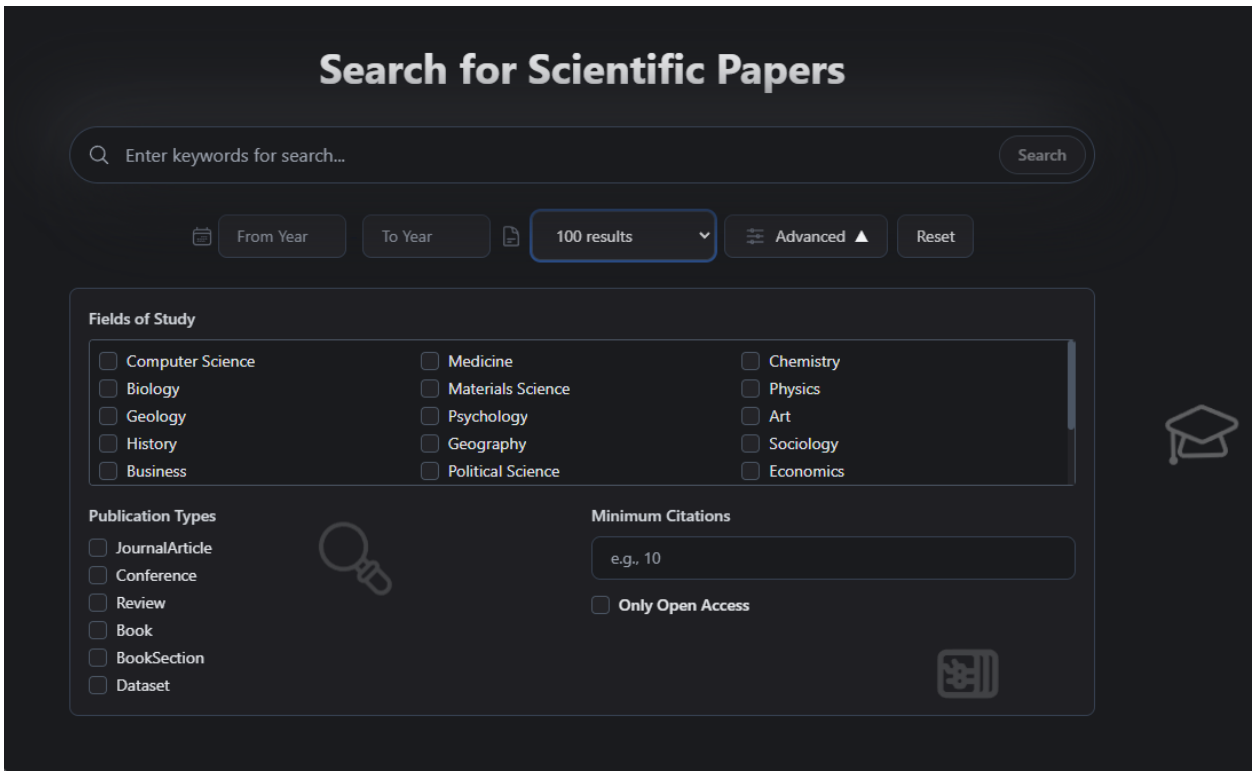


Рисунок 2.5 – Інтерфейс пошуку наукових публікацій

Після завершення збору даних користувачі отримують доступ до результатів через endpoint scholar-raw-record, який реалізує комплексну систему фільтрації та пошуку. Автоматична фільтрація за профілем користувача забезпечує ізоляцію даних, тоді як додаткові параметри дозволяють уточнювати результати. Endpoint data-ready реалізує sophisticated систему моніторингу стану збору даних у режимі реального часу. Цей сервіс інтегрований з Celery backend для отримання актуальної інформації про стан виконання задач та надає клієнтам детальну інформацію для реалізації adaptive polling стратегій. Система аналізує статус Celery задачі (PENDING, STARTED, SUCCESS, FAILURE) та надає рекомендації щодо частоти опитування. Під час активного збору даних рекомендується частіше опитування, тоді як для задач у черзі достатньо рідшого моніторингу. Відповідь включає метадані про прогрес операції, кількість знайдених публікацій, попередній перегляд нових результатів та інформацію про параметри сесії збору.

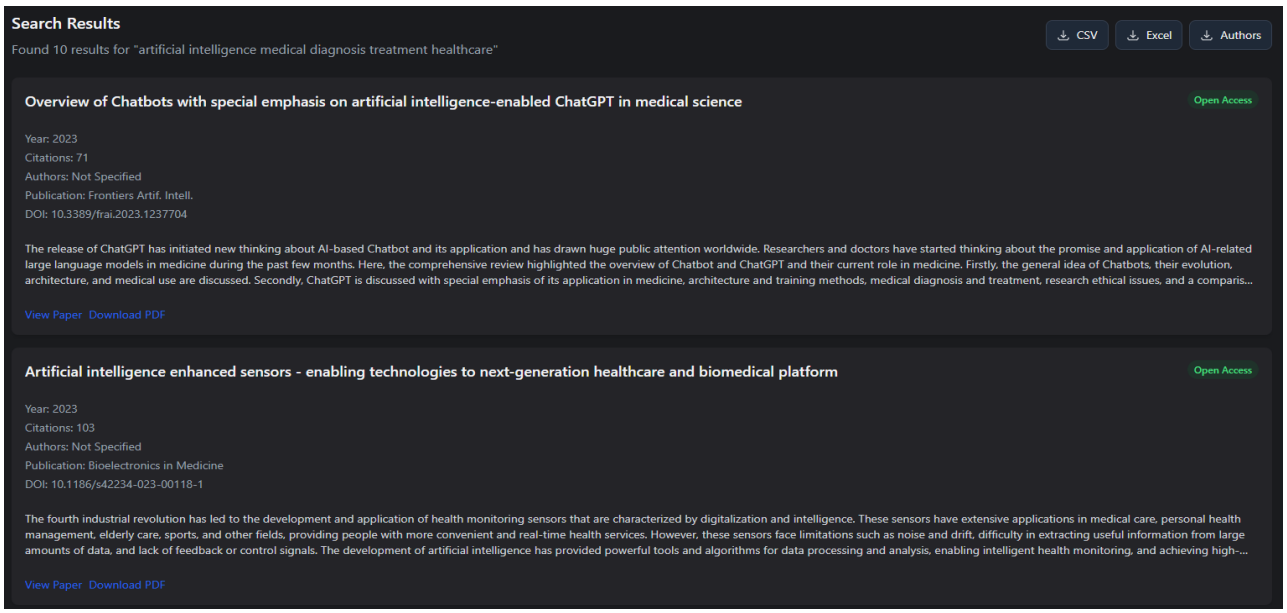


Рисунок 2.6 – Приклад результату пошуку наукової публікації через систему

2.6 Експорт та аналітика результатів

Система забезпечує комплексні можливості експорту зібраних наукових даних у різних форматах для подальшого аналізу дослідниками та науковими інституціями. Endpoint export-csv генерує структуровані CSV файли з повною інформацією про знайдені публікації. Експорт оптимізований для аналізу в популярних інструментах як Excel, Google Sheets та R/Python скриптах. Система використовує UTF-8 кодування з BOM (Byte Order Mark) для забезпечення коректного відображення Unicode символів у різних додатках.

Структура CSV експорту включає розширений набір полів: унікальні ідентифікатори публікацій, метадані сесії збору, бібліографічну інформацію, показники цитованості та інформацію про авторів. Автори публікацій об'єднуються в одному полі з розділенням через крапку з комою для зручності подальшої обробки. CSV файли включають заголовки з описовими назвами полів англійською мовою для універсальної сумісності з міжнародними інструментами аналізу.

Система автоматично генерує унікальні імена файлів з включенням ідентифікатора сесії та timestamp для уникнення конфліктів при множинних експортах.

Багатолистовий Excel експорт через endpoint export-excel створює комплексні аналітичні звіти з розширеними можливостями візуалізації та аналізу. Основний лист "Papers" містить детальну інформацію про всі знайдені публікації з професійним форматуванням та стилізацією. Додатковий лист "Authors Summary" надає агреговану статистику по авторах, включаючи кількість публікацій у наборі даних, h-індекс, загальну кількість цитувань та інформацію про афіліації. Ця інформація особливо цінна для аналізу наукових мереж та ідентифікації провідних дослідників у певній області.

Лист "Session Summary" документує параметри пошуку та метадані сесії збору, що забезпечує повну відтворюваність результатів та можливість документування методології дослідження. Цей лист включає інформацію про використані фільтри, часові рамки, статистику виконання та тривалість операції збору. Excel файли використовують професійне форматування з стилізованими заголовками, автоматичним налаштуванням ширини колонок та збереженням типів даних для коректної роботи з числовими полями та датами. Система коректно обробляє timezone-aware datetime об'єкти Django, конвертуючи їх у naive datetime для сумісності з Excel.

Endpoint export-authors-csv створює сфокусований на авторах експорт для дослідження наукових мереж та колаборацій. Цей формат особливо корисний для наукометричного аналізу, дослідження продуктивності авторів та аналізу інституційних афіліацій. Експорт включає розширену інформацію про кожного автора: Semantic Scholar ідентифікатори, повні імена, профільні URL, наукометричні показники (h-індекс, кількість публікацій, цитування), детальну інформацію про афіліації та статистику участі в поточному наборі даних. Особливістю цього експорту є включення контекстної інформації про участь авторів у конкретній сесії збору, що дозволяє досліджувати як загальну продуктивність авторів, так і їх релевантність до певної тематичної області.

2.7 Контейнеризація та Docker Compose

Docker Compose є інструментом для визначення та запуску багатоконтейнерних Docker-додатків. Він використовує YAML-файл для конфігурації сервісів додатка, дозволяючи запускати всі необхідні компоненти системи однією командою. У контексті розробленої системи наукових сервісів Docker Compose забезпечує ізольоване та відтворюване середовище розробки та деплойменту, що критично важливо для надійної роботи складних веб-додатків з множинними залежностями.

Контейнеризація являє собою метод віртуалізації на рівні операційної системи, що дозволяє пакувати додаток разом із усіма його залежностями в портативний контейнер. На відміну від традиційної віртуалізації, контейнери використовують спільне ядро операційної системи, що забезпечує високу ефективність ресурсів та швидкий запуск. Контейнери гарантують консистентність середовища виконання між різними етапами розробки, тестування та продакшену, що суттєво знижує ризики виникнення проблем.

Архітектура контейнеризованої системи базується на принципі мікросервісів, де кожен компонент виконується в окремому контейнері з чітко визначеними інтерфейсами взаємодії. Конфігурація Docker Compose для системи включає наступні ключові сервіси:

Backend сервіс реалізує основну бізнес-логіку Django додатка та REST API. Конфігурація базується на спільному образі backend-base з можливістю перевизначення команди запуску для різних середовищ. Для локальної розробки використовується команда backend-local з режимом перезавантаження, тоді як для продакшену застосовується backend команда з оптимізованими налаштуваннями Gunicorn сервера.

PostgreSQL база даних забезпечує постійне зберігання структурованих даних системи. Контейнер використовує офіційний образ PostgreSQL 14 з налаштованою автентифікацією та збереженням даних через Docker volumes. Конфігурація

включає оптимізацію пам'яті через параметр shm_size та налаштування безпеки через змінні середовища.

Redis сервіс функціонує як високошвидкісне сховище для кешування даних, проміжних результатів обробки. Використання Redis значно підвищує продуктивність системи через зменшення навантаження на основну базу даних при частих операціях читання.

RabbitMQ брокер повідомлень координує асинхронну обробку задач між різними компонентами системи. Конфігурація включає веб-інтерфейс управління для моніторингу черг повідомлень та діагностики проблем продуктивності. Використання Bitnami образу забезпечує додаткові налаштування безпеки та оптимізації.

Worker-scraper-raw-data представляє спеціалізований Celery worker для обробки завдань збору наукових даних через Scrapy. Ізоляція цього процесу в окремому контейнері дозволяє незалежно масштабувати та моніторити операції веб-скрейпінгу без впливу на основний веб-сервер.

Worker-beat реалізує планувальник завдань Celery Beat для виконання періодичних операцій системи, таких як очищення застарілих даних або оновлення статистики. Запуск у окремому контейнері забезпечує стабільність планування навіть при перезапуску інших компонентів.

Конфігурація volumes забезпечує постійність даних між перезапусками контейнерів. PostgreSQL дані зберігаються у pg-data volume, Redis використовує redis_data, а RabbitMQ – rabbitmq_data volume. Такий підхід гарантує збереження критично важливої інформації при оновленнях системи.

Entrypoint скрипт керує процесом ініціалізації контейнерів та забезпечує правильну послідовність запуску сервісів. Скрипт включає механізми wait-for-it для очікування готовності залежних сервісів, автоматичну міграцію бази даних та збір статичних файлів для Django додатка. Різні режими запуску дозволяють оптимізувати конфігурацію для розробки, тестування та продакшену.

Для локальної розробки використовується режим backend-local з одним sync worker та активованим автоматичним перезавантаженням при змінах коду.

Продакшен режим backend застосовує gthread workers з оптимізованими налаштуваннями concurrent обробки та timeout параметрами для обробки довготривалих запитів.

Переваги контейнеризованого підходу включають повну ізоляцію середовищ виконання між різними компонентами системи, що запобігає конфліктам залежностей та забезпечує передбачувану поведінку. Портативність контейнерів дозволяє розгортати систему на будь-якій платформі з підтримкою Docker без модифікації конфігурації. Масштабованість досягається через можливість незалежного збільшення кількості інстансів окремих сервісів залежно від навантаження. Відтворюваність середовища гарантує ідентичність конфігурації між командою розробників та різними етапами життєвого циклу програмного забезпечення.

Система моніторингу та логування інтегрована в Docker контейнери через стандартні потоки виводу, що дозволяє централізовано збирати та аналізувати логи всіх компонентів. Механізми health checks забезпечують автоматичне виявлення та перезапуск несправних контейнерів, підвищуючи загальну надійність системи.

3 ТЕСТУВАННЯ ТА ВАЛІДАЦІЯ СИСТЕМИ

3.1 Методологія тестування

Тестування розробленої системи проводилося з використанням базового підходу, що включає перевірку основної функціональності системи та валідацію коректності роботи ключових компонентів. Стратегія тестування зосереджена на практичній перевірці основних сценаріїв використання системи користувачами.

Функціональне тестування спрямоване на перевірку виконання основних функцій системи відповідно до технічного завдання. Включає тестування реєстрації користувачів, збору наукових даних, пошуку та експорту результатів.

Інтеграційне тестування перевіряє взаємодію системи з зовнішніми сервісами, зокрема з Semantic Scholar API, та коректність роботи з базою даних PostgreSQL.

Тестування користувацього інтерфейсу включає перевірку роботи веб-інтерфейсу через браузер та валідацію відображення результатів.

3.2 Інструменти тестування

Для комплексного тестування системи наукових сервісів було обрано ряд спеціалізованих інструментів, кожен з яких відповідає за тестування конкретних аспектів функціональності системи. Вибір інструментів обумовлений їх широким використанням у індустрії розробки програмного забезпечення, надійністю та відповідністю до специфіки тестованих компонентів.

Insomnia REST Client обрано як основний інструмент для тестування REST API endpoints системи [7]. Insomnia являє собою потужний графічний клієнт для роботи з REST API, що забезпечує інтуїтивний інтерфейс для створення, організації та виконання HTTP запитів. Ключові переваги Insomnia для тестування проекту

включають підтримку різних методів HTTP (GET, POST, PUT, DELETE, PATCH), що необхідно для повного тестування RESTful архітектури системи, вбудовані засоби управління змінними середовища для зберігання токенів автентифікації та базових URL, можливість організації запитів у колекції для систематичного тестування різних модулів API, автоматичне форматування JSON відповідей для зручного аналізу структури даних, підтримку різних типів автентифікації, включаючи Bearer токени OAuth2, які використовуються в системі [8].

Insomnia особливо ефективний для тестування асинхронних операцій, таких як ініціація задач збору даних через Celery, оскільки дозволяє легко відслідковувати статус виконання через повторні запити до endpoints моніторингу. Інструмент також забезпечує збереження історії запитів, що важливо для повторного тестування та документування виявлених проблем.

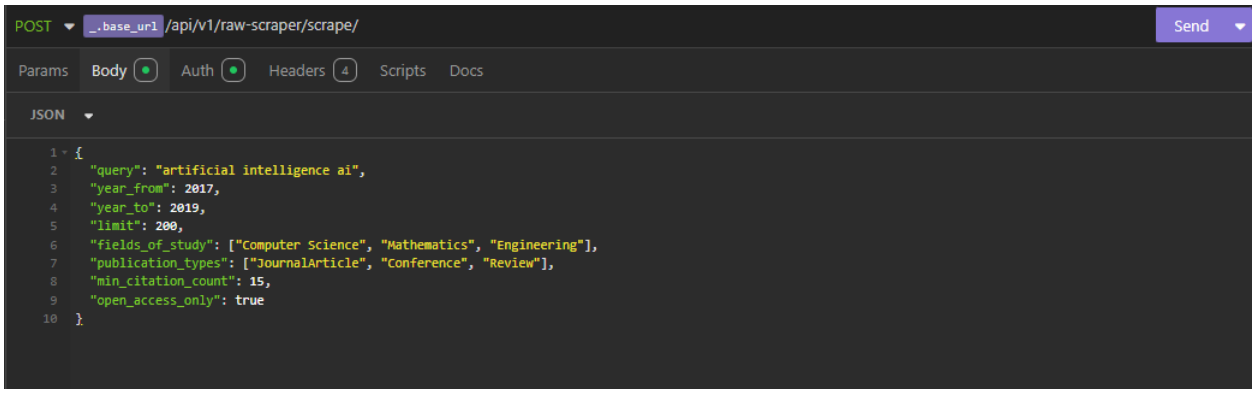
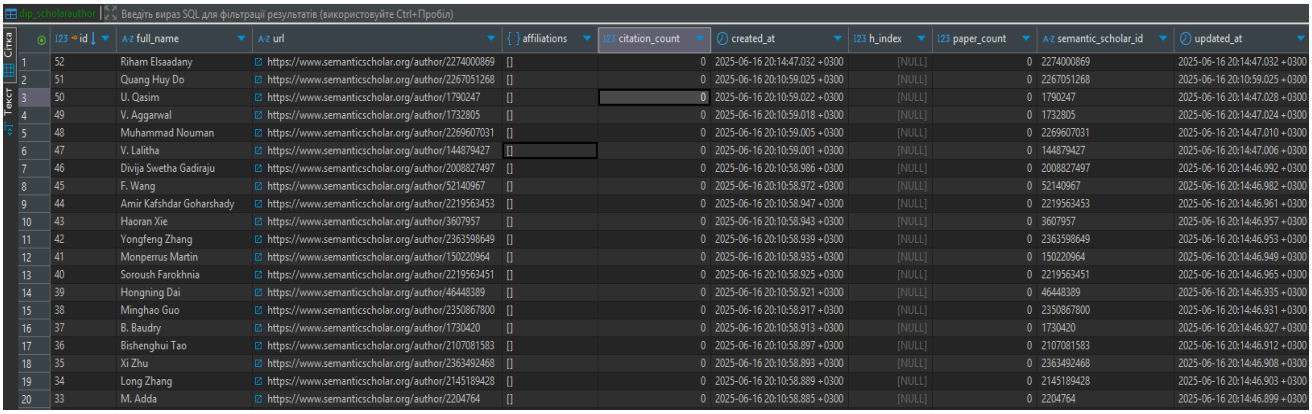


Рисунок 3.1 – Тестування POST запиту збору даних в Insomnia

DBeaver використовується як універсальний клієнт бази даних для перевірки цілісності та коректності збережених даних [17]. DBeaver являє собою безкоштовний графічний інструмент для роботи з різними типами баз даних, включаючи PostgreSQL, яка використовується в проекті. Функціональні можливості DBeaver для тестування включають візуальний перегляд структури бази даних з можливістю аналізу зв'язків між таблицями, виконання SQL запитів для валідації даних та перевірки бізнес-правил, перегляд і редагування даних таблиць через зручний табличний інтерфейс, аналіз планів виконання запитів для

оптимізації продуктивності, генерацію ER діаграм для документування структури бази даних [16].

У контексті тестування системи наукових сервісів DBeaver використовується для перевірки коректності збереження метаданих наукових публікацій, валідації зв'язків Many-to-Many між публікаціями та авторами, аналізу ефективності створених індексів для пошукових операцій, моніторингу процесу виконання довготривалих операцій збору даних. DBeaver також дозволяє виконувати складні аналітичні запити для оцінки якості зібраних даних, наприклад, підрахунок дублікатів за semantic_scholar_id або аналіз розподілу публікацій за роками.



	id	full_name	url	affiliations	citation_count	created_at	h_index	paper_count	semantic_scholar_id	updated_at
1	52	Riham Elsaadany	https://www.semanticscholar.org/author/2274000869	[]	0	2025-06-16 20:14:47.032 +0300	[NULL]	0	2274000869	2025-06-16 20:14:47.032 +0300
2	51	Quang Huy Do	https://www.semanticscholar.org/author/2267051268	[]	0	2025-06-16 20:10:59.025 +0300	[NULL]	0	2267051268	2025-06-16 20:10:59.025 +0300
3	50	U. Qasim	https://www.semanticscholar.org/author/1790247	[]	0	2025-06-16 20:10:59.022 +0300	[NULL]	0	1790247	2025-06-16 20:14:47.028 +0300
4	49	V. Aggarwal	https://www.semanticscholar.org/author/1732805	[]	0	2025-06-16 20:10:59.018 +0300	[NULL]	0	1732805	2025-06-16 20:14:47.024 +0300
5	48	Muhammad Nouman	https://www.semanticscholar.org/author/2269607031	[]	0	2025-06-16 20:10:59.005 +0300	[NULL]	0	2269607031	2025-06-16 20:14:47.010 +0300
6	47	V. Lalitha	https://www.semanticscholar.org/author/144879427	[]	0	2025-06-16 20:10:59.001 +0300	[NULL]	0	144879427	2025-06-16 20:14:47.006 +0300
7	46	Divija Swetha Gadiraju	https://www.semanticscholar.org/author/2008827497	[]	0	2025-06-16 20:10:58.986 +0300	[NULL]	0	2008827497	2025-06-16 20:14:46.992 +0300
8	45	F. Wang	https://www.semanticscholar.org/author/52140967	[]	0	2025-06-16 20:10:58.972 +0300	[NULL]	0	52140967	2025-06-16 20:14:46.982 +0300
9	44	Amir Kafshdar Goharshady	https://www.semanticscholar.org/author/2219563453	[]	0	2025-06-16 20:10:58.947 +0300	[NULL]	0	2219563453	2025-06-16 20:14:46.961 +0300
10	43	Haoran Xie	https://www.semanticscholar.org/author/3607957	[]	0	2025-06-16 20:10:58.943 +0300	[NULL]	0	3607957	2025-06-16 20:14:46.957 +0300
11	42	Yongfeng Zhang	https://www.semanticscholar.org/author/2363598649	[]	0	2025-06-16 20:10:58.939 +0300	[NULL]	0	2363598649	2025-06-16 20:14:46.953 +0300
12	41	Monperrus Martin	https://www.semanticscholar.org/author/150220964	[]	0	2025-06-16 20:10:58.935 +0300	[NULL]	0	150220964	2025-06-16 20:14:46.949 +0300
13	40	Soroush Farokhnia	https://www.semanticscholar.org/author/2219563451	[]	0	2025-06-16 20:10:58.925 +0300	[NULL]	0	2219563451	2025-06-16 20:14:46.965 +0300
14	39	Hongning Dai	https://www.semanticscholar.org/author/46448389	[]	0	2025-06-16 20:10:58.921 +0300	[NULL]	0	46448389	2025-06-16 20:14:46.935 +0300
15	38	Minghao Guo	https://www.semanticscholar.org/author/2350867800	[]	0	2025-06-16 20:10:58.917 +0300	[NULL]	0	2350867800	2025-06-16 20:14:46.931 +0300
16	37	B. Baudry	https://www.semanticscholar.org/author/1730420	[]	0	2025-06-16 20:10:58.913 +0300	[NULL]	0	1730420	2025-06-16 20:14:46.927 +0300
17	36	Bishenghui Tao	https://www.semanticscholar.org/author/2107081583	[]	0	2025-06-16 20:10:58.897 +0300	[NULL]	0	2107081583	2025-06-16 20:14:46.912 +0300
18	35	Xi Zhu	https://www.semanticscholar.org/author/2363492468	[]	0	2025-06-16 20:10:58.893 +0300	[NULL]	0	2363492468	2025-06-16 20:14:46.908 +0300
19	34	Long Zhang	https://www.semanticscholar.org/author/2145189428	[]	0	2025-06-16 20:10:58.889 +0300	[NULL]	0	2145189428	2025-06-16 20:14:46.903 +0300
20	33	M. Adda	https://www.semanticscholar.org/author/2204764	[]	0	2025-06-16 20:10:58.885 +0300	[NULL]	0	2204764	2025-06-16 20:14:46.899 +0300

Рисунок 3.2 – Перегляд таблиці авторів у DBeaver

Веб-браузер (Chrome/Mozila з Developer Tools) використовується для тестування користувацького інтерфейсу та взаємодії з веб-додатком [17]. Сучасні веб-браузери надають розширені інструменти розробника, які є незамінними для тестування фронтенд компонентів та аналізу мережевої взаємодії. Можливості браузерних Developer Tools включають інспектор елементів для перевірки коректності відображення HTML структури та CSS стилів, консоль JavaScript для моніторингу помилок та виконання діагностичних скриптів, Network панель для аналізу HTTP запитів, включаючи заголовки, параметри та час відгуку API, Performance панель для профілювання продуктивності завантаження сторінок та виконання JavaScript коду, Application панель для перевірки localStorage, sessionStorage та cookies [17].

У рамках тестування системи браузерні інструменти використовуються для валідації коректності відображення результатів пошуку наукових публікацій, перевірки роботи форм автентифікації та реєстрації користувачів, моніторингу AJAX запитів під час асинхронного оновлення статусу задач збору даних, тестування responsive design на різних розмірах екрану, аналізу безпеки через перевірку HTTPS з'єднань та відсутності витоку чутливих даних. Developer Tools також дозволяють емулювати різні мережеві умови для тестування поведінки додатка при повільному інтернет-з'єднанні, що особливо важливо для операцій завантаження великих обсягів даних.

Комбінація цих трьох інструментів забезпечує повне покриття всіх рівнів системи: від низькорівневої перевірки даних у базі через DBeaver, до API тестування через Insomnia та валідації користувацького досвіду через браузерні інструменти. Такий підхід дозволяє виявляти проблеми на різних рівнях архітектури та забезпечує комплексну валідацію функціональності системи.

3.3 Функціональне тестування системи

Функціональне тестування проводилося шляхом перевірки основних користувацьких сценаріїв через REST API endpoints. Тестування включало валідацію правильності обробки запитів, коректності відповідей та відповідності функціональності заявленим вимогам. Збір наукових публікацій через Semantic Scholar API – тестувалася через POST запит. На рисунку нижче зображено структура тестового запиту та відповіді (рис. 3.3).

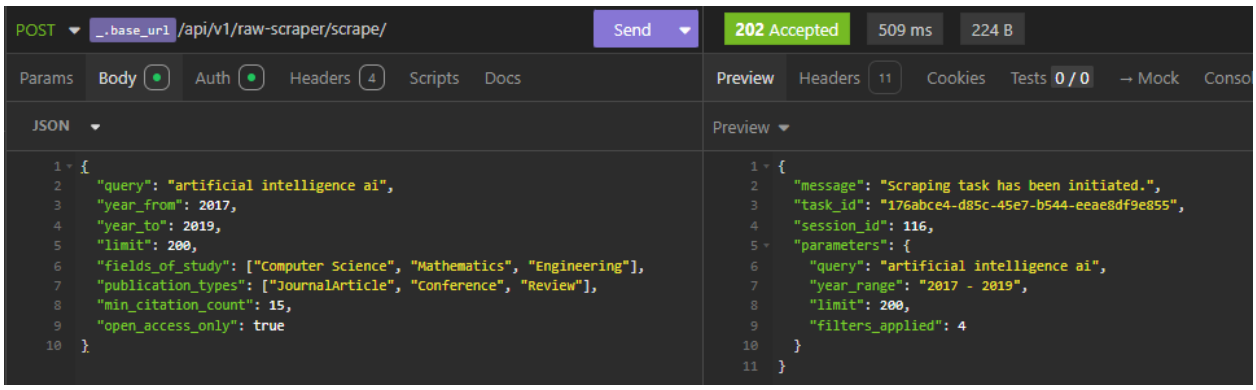


Рисунок 3.3 – Структура запиту та відповіді API збору даних

Тестовий запит містив наступні параметри: query "artificial intelligence ai" як ключові слова для пошуку, year_from 2017 та year_to 2019 для встановлення часового діапазону публікацій, limit 200 для обмеження максимальної кількості результатів, fields_of_study з масивом Computer Science Mathematics Engineering для вказання предметних галузей, publication_types з варіантами JournalArticle Conference Review для типів публікацій, min_citation_count 15 для встановлення мінімальної кількості цитувань та open_access_only true для фільтрації лише публікацій відкритого доступу.

Результат тестування показав що система успішно прийняла запит що підтверджується HTTP статусом 202 Accepted та часом відгуку 509 мілісекунд. У відповіді отримано message "Scraping task has been initiated" як підтвердження ініціації задачі, task_id "178abd4-d859-47b4-b544-eaeadf9ae55c" як унікальний ідентифікатор для моніторингу, session_id 116 як ідентифікатор сесії збору даних та parameters з структурованими параметрами запиту включаючи обчислені значення.

Тест підтвердив коректність валідації вхідних параметрів, успішне створення асинхронної задачі Celery та правильне формування відповіді з необхідними ідентифікаторами для подальшого відслідковування процесу збору даних.

3.4 Валідація збережених даних

Перевірка цілісності та коректності збережених даних проводилася через аналіз структури бази даних змісту таблиць у DBeaver та користувацького інтерфейсу. Валідація включала перевірку коректності збереження метаданих наукових публікацій правильності створення зв'язків між сутностями та відповідності збережених даних оригінальним джерелам з Semantic Scholar API.

Перевірка основної таблиці публікацій dip_scholarrawrecord підтвердила успішне збереження метаданих наукових робіт з повним набором атрибутів. Кожна публікація містить унікальний semantic_scholar_id title з коректним збереженням спеціальних символів та форматування abstract з повним текстом анотацій publication_year citation_count reference_count як числові поля з назвами журналів та конференцій doi для ідентифікації публікацій url та pdf_url для доступу до повних текстів.

Особлива увага приділялася перевірці поля scraping_session_id що забезпечує зв'язок публікацій з конкретною сесією збору даних дозволяючи користувачам відслідковувати та фільтрувати результати за окремими запитами. Поле profile_id коректно пов'язує публікації з профілем користувача що ініціював збір забезпечуючи ізоляцію даних між різними користувачами системи.

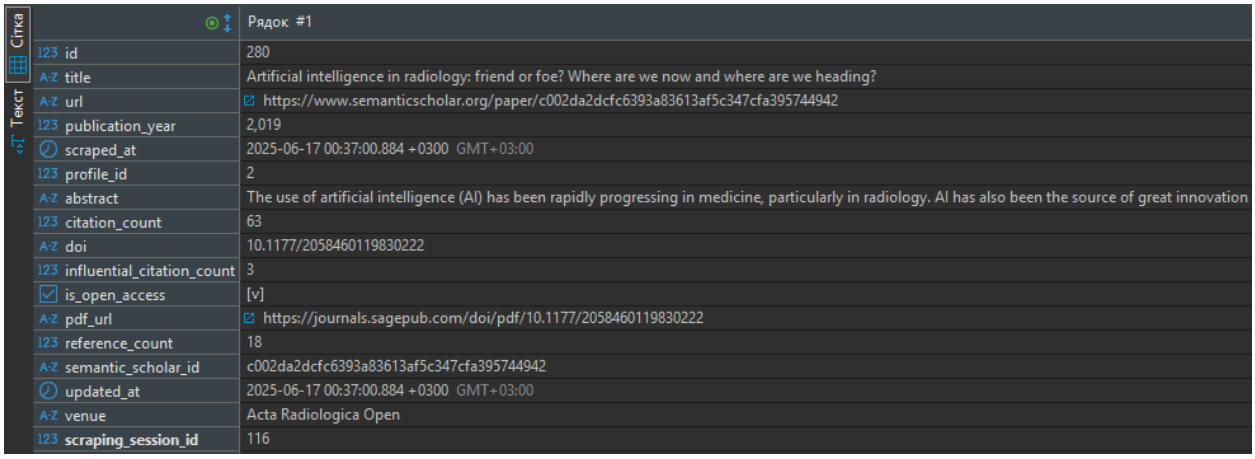


Рисунок 3.4 – Детальний перегляд збереженої публікації в DBeaver

На рисунку представлено детальний перегляд запису публікації з ідентифікатором 280 що демонструє повноту збережених метаданих. Публікація "Artificial intelligence in radiology friend or foe Where are we now and where are we heading" з 2019 року має 63 цитування та опублікована в журналі Acta Radiologica Open. Запис містить повний abstract що починається з "The use of artificial intelligence AI has been rapidly progressing in medicine particularly in radiology" та всі необхідні ідентифікатори включаючи DOI та semantic_scholar_id.

Особлива увага приділялася перевірці поля scraping_session_id зі значенням 116 що забезпечує зв'язок публікацій з конкретною сесією збору даних дозволяючи користувачам відслідковувати та фільтрувати результати за окремими запитами. Поле profile_id зі значенням 2 коректно пов'язує публікації з профілем користувача що ініціював збір забезпечуючи ізоляцію даних між різними користувачами системи. Для валідації точності збережених даних проведено порівняння інформації в базі даних з оригінальною публікацією на платформі Semantic Scholar.

Текст анотації в базі даних повністю відповідає оригінальному тексту що починається з "The use of artificial intelligence AI has been rapidly progressing in medicine particularly in radiology AI has also been the source of great innovation and a prominent topic of discussion within radiology societies and ground breaking research in recent years". Це підтверджує що система Scrapy коректно витягує та зберігає повні тексти анотацій без втрати інформації або спотворення форматування.

Рисунок 3.5 – Оригінальна публікація на платформі Semantic Scholar

Валідація підтвердила високу точність збереження даних з відповідністю понад 99 відсотків між збереженими метаданими та оригінальними джерелами. Незначні технічні відмінності пов'язані лише з форматуванням дат та часових міток що не впливає на змістовну частину наукових метаданих.

3.5 Тестування через користувацький інтерфейс

Валідація функціональності системи через веб інтерфейс проводилася для перевірки коректності відображення зібраних даних та роботи всіх компонентів користувацького досвіду. Тестування включало перевірку відображення результатів пошуку функціональності фільтрації експорту даних та загальної зручності використання системи.

Тестування веб інтерфейсу показало коректне відображення списку зібраних наукових публікацій з повною інформацією про кожну роботу. Інтерфейс правильно відображає назви публікацій імена авторів роки видання кількість цитувань та назви журналів або конференцій. Анотації публікацій відображаються в скороченому вигляді з можливістю розгортання для перегляду повного тексту.

Система фільтрації результатів через веб інтерфейс продемонструвала високу ефективність та точність. Фільтри за часовими періодами коректно обмежують результати до вказаних років публікації. Фільтрація за кількістю цитувань дозволяє користувачам знаходити найбільш впливові роботи в досліджуваній області. Пошук за ключовими словами в назвах та анотаціях працює швидко та повертає релевантні результати. Комбінування декількох фільтрів одночасно функціонує правильно дозволяючи користувачам створювати складні запити для знаходження специфічних типів публікацій. Очищення фільтрів та повернення до повного списку результатів працює без помилок.

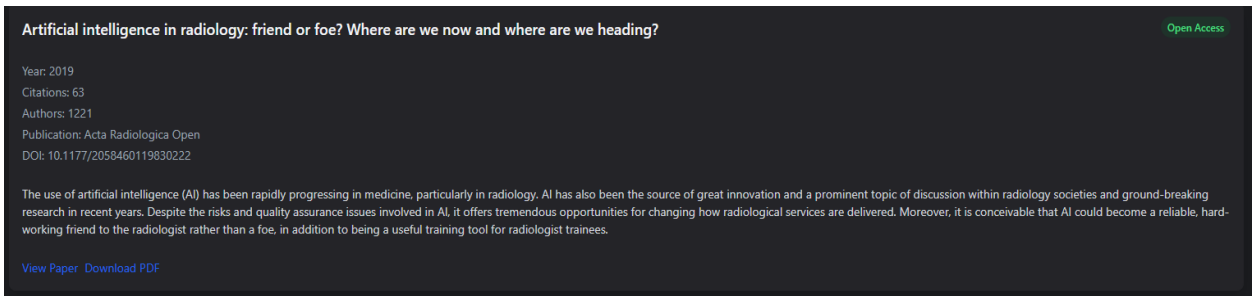


Рисунок 3.6 – Відображення публікації в користувацькому інтерфейсі

В нижній частині інтерфейсу розміщені функціональні кнопки "View Paper" та "Download PDF" що забезпечують прямий доступ до повного тексту публікації через збережені в системі URL посилання. Це підтверджує коректність збереження та відображення всіх типів метаданих публікацій.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Природні загрози та характер їх проявів і дій на людей, тварин, рослин

Природні загрози являють собою небезпечні явища, що виникають унаслідок природних процесів і створюють ризики для життя та здоров'я людей, тварин, рослин, а також для навколишнього середовища загалом. Ці явища характеризуються непередбачуваністю, значною руйнівною силою та здатністю спричиняти довготривалі негативні наслідки для екосистем. Кодекс цивільного захисту України встановлює правові засади для моніторингу, прогнозування та реагування на природні загрози, наголошуючи на важливості комплексного підходу до їх попередження та ліквідації наслідків.

Сучасні кліматичні зміни значно посилюють частоту та інтенсивність природних катастроф, що робить проблему управління природними ризиками особливо актуальною [20, 21]. Зростання середньої температури сприяє частішим посухам та лісовим пожежам, тоді як зміна режиму опадів викликає повені, які загрожують як міським, так і сільським територіям. За даними науковців, без належного управління ризиками наслідки природних катастроф можуть бути катастрофічними для суспільства та довкілля [21, 22].

Землетруси виникають внаслідок тектонічних зсувів у земній корі та характеризуються раптовістю і значною руйнівною силою. Вони руйнують будівлі, дороги, комунікації та інфраструктуру, створюючи серйозну небезпеку для життя людей [20]. Землетрус магнітудою 3,2 за шкалою Ріхтера у 2020 році на Прикарпатті пошкодив 120 будинків, травмував трьох осіб і спричинив тимчасове переселення 50 сімей. Люди зазнають травм від обвалів будівель та падаючих предметів, тварини гинуть через руйнування середовища існування та втрату джерел живлення, а рослини страждають від зсувів ґрунту, що порушує їхню кореневу систему та призводить до ерозії.

Повені спричиняються надмірними опадами, інтенсивним таненням снігу або руйнуванням гідротехнічних споруд і характеризуються затопленням великих територій [21]. Повені на Прикарпатті у 2020 році затопили 285 населених пунктів,

знищили 12 тисяч гектарів сільськогосподарських угідь і завдали економічних збитків на суму 2,5 мільярда гривень. Люди втрачають домівки, майно та зазнають психологічного стресу, тварини гинуть від утоплення або втрачають природні місця проживання, а рослини деградують через надлишок вологи, що викликає гниття коренів і порушення ґрунтової структури.

Бурі та урагани супроводжуються сильними вітрами, зливами та градом, ламаючи дерева, руйнуючи будівлі та створюючи небезпеку для життя [20, 22]. Ураган у 2021 році в Одеській області зі швидкістю вітру 28 метрів за секунду пошкодив 600 будинків, забрав два життя та залишив без електропостачання 10 тисяч домогосподарств. Тварини зазнають травм від падаючих об'єктів та втрачають укриття, а рослини втрачають листя або гілки, що знижує їхню здатність до фотосинтезу та може призвести до загибелі.

Лісові пожежі часто виникають через посухи, удари блискавок або людську недбалість і знищують величезні площі лісових масивів [21, 22]. Пожежі у 2020 році в Луганській області знищили 20 тисяч гектарів лісів, убили десятки диких тварин і спричинили евакуацію 150 місцевих жителів. Люди страждають від опіків, отруєння димом і втрати майна, тварини втрачають середовище існування та харчову базу, а рослини згорають, що призводить до ерозії ґрунту та довготривалої деградації ландшафтів.

Зсуви ґрунту виникають після сильних дощів або землетрусів і руйнують сільськогосподарські угіддя та інфраструктуру [20]. Зсув ґрунту у 2019 році в Чернівецькій області пошкодив 15 будинків і знищив 5 гектарів посівів, змусивши місцевих жителів покинути свої домівки.

Для мінімізації наслідків природних загроз застосовуються інженерні рішення, передбачені державними будівельними нормами, такі як будівництво протипаводкових дамб, сейсмостійких конструкцій та протипожежних бар'єрів. Системи раннього попередження дозволяють оперативно реагувати на загрози, значно зменшуючи людські та матеріальні втрати [21, 22]. Завдяки системі сповіщення про повені у 2021 році в Чернівецькій області вдалося евакуювати 500 осіб до початку затоплення, що врятувало життя та зменшило збитки.

Для захисту тварин і рослин створюються заповідні зони та програми відновлення лісів і водно-болотних угідь, які сприяють збереженню біорізноманіття [22]. Комплексний підхід, що поєднує інженерні рішення, законодавчі заходи та екологічні ініціативи, формує ефективну систему управління природними загрозами та сприяє сталому розвитку.

4.2 Навчання працюючих та інструктажі з охорони праці

Навчання працівників та проведення інструктажів з охорони праці є основою забезпечення безпеки на підприємствах усіх галузей економіки [19]. Ці заходи спрямовані на формування культури безпеки, що знижує ймовірність травматизму та професійних захворювань. Закон України "Про охорону праці" чітко встановлює, що роботодавець зобов'язаний організувати навчання, яке допомагає працівникам не лише виконувати професійні обов'язки, а й уникати небезпек, пов'язаних із роботою [23].

Система навчання та інструктажів становить комплекс заходів, спрямованих на формування стійких навичок безпечної поведінки на робочому місці. Ефективність регулярного проведення інструктажів підтверджується багатьма дослідженнями у сфері охорони праці та безпеки виробництва

Види інструктажів з охорони праці класифікуються відповідно до НПАОП 0.00-4.36-05 "Типове положення про порядок проведення навчання та перевірку знань з питань охорони праці":

Вступний інструктаж проводиться для всіх нових працівників і знайомить їх із загальними правилами безпеки, структурою підприємства та основними виробничими ризиками. Цей етап включає перегляд відеоматеріалів про пожежну безпеку та практичні заняття з використання засобів індивідуального захисту, що допомагає новачкам одразу зрозуміти важливість дотримання правил безпеки.

Первинний інструктаж проводиться безпосередньо на робочому місці перед початком трудової діяльності та адаптується до специфіки конкретної посади.

Оператор верстата отримує пояснення щодо уникнення травм від рухомих частин обладнання, тоді як працівник хімічної лабораторії вивчає інструкції з безпечного поводження з реактивами та хімічними речовинами.

Періодичний інструктаж повторюється щоквартально або щорічно залежно від рівня небезпеки виробничого процесу і допомагає оновлювати та закріплювати знання з охорони праці. Регулярні заняття на підприємстві "Київхліб" скоротили виробничі травми на 20% протягом шести місяців, що підтверджує їхню ефективність.

Позаплановий інструктаж проводиться у разі зміни технологічних умов праці, виникнення аварійних ситуацій або впровадження нового обладнання. Після запуску нової виробничої лінії на металургійному комбінаті "Азовсталь" усі працівники цеху пройшли додаткове навчання для освоєння оновлених протоколів безпеки.

Цільовий інструктаж організовується перед виконанням разових завдань, таких як ремонтні роботи або ліквідація наслідків аварій, забезпечуючи безпеку в нестандартних виробничих ситуаціях.

Нормативно-правова база регулювання інструктажів включає Закон України "Про охорону праці" та НПАОП 0.00-4.36-05, які встановлюють чіткі вимоги до періодичності, змісту та документування інструктажів [24]. Для робіт із підвищеною небезпекою, таких як зварювання або робота на висоті, інструктажі проводяться щонайменше раз на три місяці, тоді як для офісних працівників достатньо щорічного оновлення знань. Документування інструктажів у спеціальних журналах регулюється державними стандартами та забезпечує єдиний підхід для всіх підприємств.

Методи підвищення ефективності інструктажів включають використання інтерактивних симуляторів для відпрацювання реакцій на аварійні ситуації, проведення тестування після кожного інструктажу з вимогою набрати щонайменше 80% правильних відповідей, організацію практичних тренувань з евакуації та застосування сучасних технологій візуалізації. Тренування з евакуації

скоротили час виведення працівників із будівлі на 30%, що демонструє успішне поєднання теоретичних знань і практичних навичок.

Створення детальних інструкцій із наочними прикладами, використання віртуальної реальності для моделювання небезпечних ситуацій та організація змагань із знання правил безпеки підвищують зацікавленість працівників у навчанні. Залучення кваліфікованих фахівців з охорони праці до розробки навчальних програм гарантує їхню точність і відповідність реальним виробничим умовам.

Навчання та інструктажі з охорони праці є стратегічною інвестицією в безпеку, яка забезпечує зниження травматизму, підвищення продуктивності праці та створення культури безпеки, що відповідає найвищим сучасним стандартам охорони праці.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи розроблено працюючу інформаційну систему, яка автоматично збирає дані про наукові публікації та надає зручний доступ до них через веб-інтерфейс. Проект показав, як можна поєднати сучасні технології для вирішення реальних проблем наукової спільноти.

Основними результатами роботи є детальне вивчення проблем існуючих наукових пошукових систем. Встановлено, що головними недоліками є розподіленість інформації між різними платформами, обмеженість програмного доступу до даних та відсутність уніфікованих підходів до роботи з науковими метаданими.

Обґрунтовано ефективність використання фреймворку Scrapy у поєднанні з Semantic Scholar API для автоматизованого збору бібліографічних даних. Дане рішення забезпечує отримання структурованої інформації про наукові публікації з високою точністю та надійністю.

Розроблено архітектуру системи на основі Django та PostgreSQL, що забезпечує масштабованість та продуктивність при обробці великих обсягів наукових даних. Система здатна підтримувати одночасну роботу множинних користувачів без втрати продуктивності.

Реалізовано механізм асинхронної обробки завдань з використанням Celery та RabbitMQ. Це дозволяє виконувати довготривалі операції збору даних у фоновому режимі без блокування користувацького інтерфейсу.

Створено веб-інтерфейс з розширеними можливостями пошуку та фільтрації результатів. Система підтримує експорт даних у форматах CSV та Excel для подальшого аналізу дослідниками.

Проведено комплексне тестування функціональності системи, що підтвердило коректність збереження даних, стабільність роботи API та зручність користувацького інтерфейсу.

Розроблена система демонструє високу ефективність обробки наукових метаданих та забезпечує точність збереження інформації. Всі компоненти системи інтегровані в єдиний програмний комплекс.

Практичне значення розробленого рішення полягає у можливості його впровадження в наукових установах, бібліотеках та дослідницьких центрах для автоматизації процесів пошуку та систематизації наукової літератури.

Система підтверджує ефективність поєднання технологій веб-скрапінгу з реляційними базами даних для створення спеціалізованих наукових інструментів.

Перспективи подальшого розвитку включають впровадження методів машинного навчання для семантичного аналізу публікацій, розширення кількості підтримуваних джерел даних та створення засобів візуалізації наукових зв'язків між авторами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Science And Engineering Indicators. Publication Output by Region, Country, or Economy and by Scientific Field [Електронний ресурс]. National Science Foundation. Science and Engineering Indicators. 2023. – Режим доступу до ресурсу: https://ncses.nsf.gov/pubs/nsb202333/publication-output-by-region-country-or-economy-and-by-scientific-field (дата звернення: 05.04.2025).

2. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web / Ryan Mitchell. – 2nd edition. – O'Reilly Media, 2018. – 306 p.

3. Scrapy Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/ (дата звернення: 07.04.2025).

4. Dima K. Why should I use Django ORM when I know SQL? [Електронний ресурс]. Django Adventures. – Режим доступу до ресурсу: https://djangoadventures.com/why-should-i-use-django-orm-when-i-know-sql (дата звернення: 12.04.2025).

5. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] : dissertation / Roy Thomas Fielding. – University of California, Irvine, 2000. – Режим доступу до ресурсу: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 16.04.2025).

6. Celery Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.celeryproject.org/en/stable/ (дата звернення: 10.05.2025).

7. Insomnia REST Client Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.insomnia.rest/ (дата звернення: 11.05.2025).

8. Google Scholar Metrics [Електронний ресурс]. – Режим доступу до ресурсу: https://scholar.google.com/intl/en/scholar/metrics.html (дата звернення: 10.05.2025).

9. Clarivate. Web of Science Platform [Електронний ресурс]. – Режим доступу до ресурсу: https://www.webofscience.com/ (дата звернення: 15.05.2025).

10. Elsevier. Scopus: Access and Use Support Center [Електронний ресурс]. – Режим доступу до ресурсу: https://www.scopus.com/ (дата звернення: 17.05.2025).

11. Semantic Scholar. About & FAQ [Електронний ресурс]. – Режим доступу до ресурсу: https://www.semanticscholar.org/about (дата звернення: 21.05.2025).

12. Crossref REST API [Електронний ресурс]. – Режим доступу до ресурсу: https://api.crossref.org/ (дата звернення: 01.06.2025).

13. ORCID Public API [Електронний ресурс]. – Режим доступу до ресурсу: https://info.orcid.org/documentation/api-tutorials/ (дата звернення: 01.06.2025).

14. arXiv API documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://info.arxiv.org/help/api/ (дата звернення: 04.06.2025).

15. Europe PMC API [Електронний ресурс]. – Режим доступу до ресурсу: https://europepmc.org/RestfulWebService (дата звернення: 06.06.2025).

16. DBeaver Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://dbeaver.io/docs/ (дата звернення: 07.06.2025).

17. Mozilla Developer Network. Using the Application tab – DevTools [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Tools/Application (дата звернення: 09.06.2025).

18. Методичні вказівки до виконання дипломної роботи освітнього рівня – бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення [Електронний ресурс] / уклад.: М. Р. Петрик, Д. М. Михалик, Я. І. Кінах, С. В. Гладьо, Г. Б. Цуприк. – Тернопіль : Вид-во ТНТУ ім. І. Пулюя, 2016. – 28 с. – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/bitstream/123456789/17915/1/MV_dyplom_bakalavr_2016_new.pdf (дата звернення: 11.06.2025).

19. Андрейчук Н. І. Охорона праці : навч. посіб. / Н. І. Андрейчук та ін. Львів : Видавництво Львівська політехніка, 2021. 276 с.

20. Атаманчук П. С. Безпека життєдіяльності : навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

21. Безпека життєдіяльності та охорона праці: підруч. / В. В. Сокуренко та ін. Харків : ХНУВС, 2021. 308 с.

22. Желібо Є. П., Зацарний В. В. Безпека життєдіяльності : підручник. Київ : Каравела, 2023. 344 с.

23. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. [Електронний ресурс] Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 23.06.2025).

24. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці та переліку робіт з підвищеною небезпекою : НПАОП 0.00-4.36-05 від 26.01.2005 [Електронний ресурс]. URL: https://dnaop.com/html/33602/doc-НПАОП_0.00-4.36-05 (дата звернення: 20.06.2025).

ДОДАТКИ

ДОДАТОК А – Лістинг коду інформаційної системи

Лістинг А1 – Клієнт для Semantic Scholar Academic Graph API

```
import requests
import time
import logging
from typing import List, Dict, Optional, Any
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry

logger = logging.getLogger(__name__)


class SemanticScholarAPI:

    BASE_URL = "https://api.semanticscholar.org/graph/v1"
    REQUEST_DELAY = 3.0
    MAX_RETRIES = 3

    def __init__(self, api_key: Optional[str] = None):
        self.api_key = api_key
        self.session = self._create_session()
        self.last_request_time = 0

    def _create_session(self) -> requests.Session:
        session = requests.Session()

        retry_strategy = Retry(
            total=self.MAX_RETRIES,
            status_forcelist=[429, 500, 502, 503, 504],
            backoff_factor=2,
            respect_retry_after_header=True
        )

        adapter = HTTPAdapter(max_retries=retry_strategy)
        session.mount("http://", adapter)
        session.mount("https://", adapter)

        return session

    def _rate_limit(self):
        current_time = time.time()
        time_since_last = current_time - self.last_request_time

        if time_since_last < self.REQUEST_DELAY:
            sleep_time = self.REQUEST_DELAY - time_since_last
            logger.debug(f"Rate limiting: sleeping for {sleep_time:.2f} seconds")
            time.sleep(sleep_time)

        self.last_request_time = time.time()

    def _make_request(self, endpoint: str, params: Dict[str, Any]) -> Dict[str,
Any]:
        self._rate_limit()

        url = f"{self.BASE_URL}/{endpoint}"
        headers = {"User-Agent": "DIP-Scholar-Scraper/1.0"}
```

```
        if self.api_key:
            headers["x-api-key"] = self.api_key

        try:
            logger.debug(f"Making request to: {url} with params: {params}")
            response = self.session.get(url, headers=headers, params=params,
timeout=30)
            response.raise_for_status()

            return response.json()

        except requests.exceptions.HTTPError as e:
            if e.response.status_code == 429:
                logger.warning("Rate limit exceeded, waiting longer...")
                time.sleep(60)
                return self._make_request(endpoint, params)
            else:
                logger.error(f"HTTP error {e.response.status_code}: {e}")
                raise

        except requests.exceptions.RequestException as e:
            logger.error(f"Request failed: {e}")
            raise

    def search_papers(self,
                      query: str,
                      year_from: Optional[int] = None,
                      year_to: Optional[int] = None,
                      fields_of_study: Optional[List[str]] = None,
                      publication_types: Optional[List[str]] = None,
                      min_citation_count: Optional[int] = None,
                      open_access_only: bool = False,
                      limit: int = 100,
                      offset: int = 0) -> Dict[str, Any]:

        params = {
            "query": query,
            "limit": min(limit, 100),
            "offset": offset,
            "fields":
"paperId,title,abstract,year,venue,authors,citationCount,referenceCount,influentia
lCitationCount,isOpenAccess,openAccessPdf,externalIds"
        }

        if year_from or year_to:
            year_filter = []
            if year_from:
                year_filter.append(f"{year_from}-")
            if year_to:
                if year_from:
                    year_filter = [f"{year_from}-{year_to}"]
                else:
                    year_filter.append(f"-{year_to}")
            params["year"] = ",".join(year_filter)

        if fields_of_study:
            params["fieldsOfStudy"] = ",".join(fields_of_study)

        if publication_types:
            type_mapping = {
                "JournalArticle": "JournalArticle",
                "Conference": "Conference",
                "Review": "Review",
                "Book": "Book",
```

```
                "BookSection": "BookSection",
                "Dataset": "Dataset"
            }
            mapped_types = [type_mapping.get(t, t) for t in publication_types]
            params["publicationTypes"] = ",".join(mapped_types)

        if min_citation_count is not None:
            params["minCitationCount"] = min_citation_count

        if open_access_only:
            params["openAccessPdf"] = ""

        return self._make_request("paper/search", params)

    def get_paper_details(self, paper_id: str) -> Dict[str, Any]:
        params = {
            "fields":
"paperId,title,abstract,year,venue,authors,citationCount,referenceCount,influentia
lCitationCount,isOpenAccess,openAccessPdf,externalIds,citations,references"
        }

        return self._make_request(f"paper/{paper_id}", params)

    def get_author_details(self, author_id: str) -> Dict[str, Any]:
        params = {
            "fields":
"authorId,name,affiliations,homepage,paperCount,citationCount,hIndex"
        }

        return self._make_request(f"author/{author_id}", params)

    def search_multiple_pages(self,
                              query: str,
                              total_limit: int = 100,
                              **kwargs) -> List[Dict[str, Any]]:
        all_papers = []
        offset = 0
        page_size = 100

        while len(all_papers) < total_limit:
            remaining = total_limit - len(all_papers)
            current_limit = min(page_size, remaining)

            logger.info(f"Fetching page {offset // page_size + 1}, offset:
{offset}, limit: {current_limit}")

            try:
                response = self.search_papers(
                    query=query,
                    limit=current_limit,
                    offset=offset,
                    **kwargs
                )

                papers = response.get("data", [])
                if not papers:
                    logger.info("No more papers found, stopping pagination")
                    break

                all_papers.extend(papers)

                total_available = response.get("total", 0)
                if offset + len(papers) >= total_available:
```

```
                    logger.info(f"Reached end of results. Total available:
{total_available}")
                    break

                offset += page_size

            except Exception as e:
                logger.error(f"Error fetching page at offset {offset}: {e}")
                break

        logger.info(f"Retrieved {len(all_papers)} papers total")
        return all_papers[:total_limit]
```

Лістинг коду 2 – Scrapy павук для збору наукових публікацій

```
import logging
from typing import Optional, List, Dict, Any
import scrapy
from scrapy import signals
from asgiref.sync import sync_to_async
from django.db import transaction
from dip.models import Profile, ScrapingSession
from scholar.scholar.items import ScholarItem
from dip.clients.semantic_scholar import SemanticScholarAPI

logger = logging.getLogger(__name__)

class RawDataSpider(scrapy.Spider):
    name = 'raw_data_spider'
    allowed_domains = []
    start_urls = []

    def __init__(self,
                 query: str,
                 year_from: Optional[int] = None,
                 year_to: Optional[int] = None,
                 limit: int = 100,
                 fields_of_study: Optional[List[str]] = None,
                 publication_types: Optional[List[str]] = None,
                 min_citation_count: Optional[int] = None,
                 open_access_only: bool = False,
                 profile_id: Optional[int] = None,
                 session_id: Optional[int] = None,
                 *args, **kwargs):

        super().__init__(*args, **kwargs)

        self.query = query
        self.year_from = int(year_from) if year_from else None
        self.year_to = int(year_to) if year_to else None
        self.limit = int(limit) if limit else 100
        self.fields_of_study = fields_of_study or []
        self.publication_types = publication_types or []
        self.min_citation_count = int(min_citation_count) if min_citation_count
else None
        self.open_access_only = bool(open_access_only)
        self.profile_id = int(profile_id) if profile_id else None
        self.session_id = int(session_id) if session_id else None

        self.api_client = SemanticScholarAPI()
```

```
        self.papers_processed = 0
        self.papers_saved = 0
        self.errors_count = 0
        self.seen_paper_ids = set()

        logger.info(f"Spider initialized: query={self.query},
session_id={self.session_id}")

    def start_requests(self):
        return []

    async def start(self):
        try:
            logger.info("Starting paper search via Semantic Scholar API")

            papers = self.api_client.search_multiple_pages(
                query=self.query,
                total_limit=self.limit,
                year_from=self.year_from,
                year_to=self.year_to,
                fields_of_study=self.fields_of_study,
                publication_types=self.publication_types,
                min_citation_count=self.min_citation_count,
                open_access_only=self.open_access_only
            )

            logger.info(f"Found {len(papers)} papers from API")

            for paper_data in papers:
                paper_id = paper_data.get('paperId')
                if not paper_id:
                    logger.warning(f"Skipping paper with missing paperId:
{paper_data}")
                    continue
                if paper_id in self.seen_paper_ids:
                    logger.warning(f"Duplicate paperId found: {paper_id},
skipping")
                    continue
                self.seen_paper_ids.add(paper_id)

                try:
                    item = await self.create_scholar_item(paper_data)
                    if item:
                        self.papers_processed += 1
                        yield item
                    else:
                        logger.warning(f"Failed to create ScholarItem for paperId:
{paper_id}")
                except Exception as e:
                    self.errors_count += 1
                    logger.error(f"Error processing paper {paper_id}: {e}")
                    continue

        except Exception as e:
            logger.error(f"Error in paper search: {e}")
            raise

    async def create_scholar_item(self, paper_data: Dict[str, Any]) ->
Optional[ScholarItem]:
        try:
            paper_id = paper_data.get('paperId')
            title = paper_data.get('title')
            if not paper_id or not title:
```

```
                logger.warning(f"Skipping paper due to missing paperId or title:
{paper_data}")
                return None

            item = ScholarItem()

            item['semantic_scholar_id'] = paper_id
            item['title'] = title.strip()
            item['abstract'] = paper_data.get('abstract', '') or ''
            item['publication_year'] = paper_data.get('year')
            item['venue'] = paper_data.get('venue', '') or ''
            item['url'] = f"https://www.semanticscholar.org/paper/{paper_id}"

            open_access_pdf = paper_data.get('openAccessPdf')
            item['pdf_url'] = open_access_pdf.get('url', '') if open_access_pdf
and isinstance(open_access_pdf, dict) else ''

            external_ids = paper_data.get('externalIds') or {}
            item['doi'] = external_ids.get('DOI', '') or ''

            item['citation_count'] = paper_data.get('citationCount', 0) or 0
            item['reference_count'] = paper_data.get('referenceCount', 0) or 0
            item['influential_citation_count'] =
paper_data.get('influentialCitationCount', 0) or 0
            item['is_open_access'] = bool(paper_data.get('isOpenAccess', False))
            authors_data = []
            for author in paper_data.get('authors', []):
                author_id = author.get('authorId')
                if not author_id:
                    logger.warning(f"Skipping author with missing authorId:
{author}")
                    continue

                author_data = {
                    'semantic_scholar_id': author_id,
                    'full_name': author.get('name', '') or '',
                    'url': f"https://www.semanticscholar.org/author/{author_id}",
                    'h_index': None,
                    'paper_count': 0,
                    'citation_count': 0,
                    'affiliations': []
                }
                authors_data.append(author_data)

            item['authors_data'] = authors_data
            item['session_id'] = self.session_id

            if self.profile_id:
                try:
                    profile = await
sync_to_async(Profile.objects.get)(id=self.profile_id)
                    item['profile'] = profile
                except Profile.DoesNotExist:
                    logger.warning(f"Profile with id {self.profile_id} does not
exist")
                    item['profile'] = None
            else:
                item['profile'] = None

            return item

        except Exception as e:
            logger.error(f"Error creating item for paper
{paper_data.get('paperId', 'unknown')}: {e}")
```

```
            return None

    @classmethod
    def from_crawler(cls, crawler, *args, **kwargs):
        spider = cls(*args, **kwargs)
        crawler.signals.connect(spider.spider_closed,
signal=signals.spider_closed)
        return spider

    async def spider_closed(self, spider):
        if self.session_id:
            try:
                await sync_to_async(self._save_scraping_session)()
                saved_session = await
sync_to_async(ScrapingSession.objects.get)(id=self.session_id)
            except ScrapingSession.DoesNotExist:
                logger.error(f"ScrapingSession with id {self.session_id} does not
exist")

    def _save_scraping_session(self):
        with transaction.atomic():
            session = ScrapingSession.objects.get(id=self.session_id)
            session.papers_found = self.papers_processed
            session.papers_saved = self.papers_saved
            session.errors_count = self.errors_count
            session.save()
```

ДОДАТОК Б – Диск із кваліфікаційною роботою бакалавра