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
(підпис)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 122 Комп’ютерні науки та інформаційні технології

на тему Автоматизація оцінювання продуктивності наукової діяльності

Виконав: студент 4 курсу, групи ТР-62

Саухін Владислав Віталійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц., к.т.н., доц. Стативка Ю.І. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент мол.інж.прогр., к.б.н., ст.н.співр. Потягайло А.Л. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Київ – 2020 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
Напрям підготовки	122 Комп’ютерні науки та інформаційні технології
Спеціалізація	Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Саухіну Владиславу Віталійович
(прізвище, ім’я, по батькові)

1. Тема роботи Автоматизація оцінювання продуктивності наукової діяльності
керівник роботи доц., к.т.н., доц._Стативка Юрій Іванович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ”__”___ 2020 р. № __
2. Строк подання студентом роботи
3. Вихідні дані до роботи Програмний продукт для роботи з бібліографічними даними та можливістю проведення оцінки продуктивності наукової діяльності на основі системи індикаторів
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідити можливі джерела бібліографічних даних. Реалізувати програмну взаємодію з бібліографічними даними. Проаналізувати отримані дані та виконати розрахунок показників продуктивості та значимості наукової діяльності.
5. Перелік ілюстративного матеріалу Показники науково-технічної діяльності. Діаграма класів та діаграма даних. Приклади використання розробленого пакету

для кожної з функцій. Графіки для відображення динаміки розрахованих показників на основі отриманих даних.

6. Дата видачі завдання “11” жовтня 2019р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	01.10.2019 р.	
2.	Вивчення та аналіз задачі	15.11.2019 р.	
3.	Розробка архітектури та загальної структури системи	15.01.2020 р.	
4.	Розробка структур окремих підсистем	15.02.2020 р.	
5.	Програмна реалізація системи	15.03.2020 р.	
6.	Оформлення пояснювальної записки	25.05.2020 р.	
7.	Захист програмного продукту	11.05.2020 р.	
8.	Передзахист		
9.	Захист		

Студент _____ Саухін В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стативка Ю.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота бакалавра містить 50 сторінок, 26 рисунків, 5 формул, 3 додатки та 6 посилань.

Мета роботи – створення зручного та легко масштабованого пакету для роботи з бібліографічними даними та проведення оцінки науково-технічної продуктивності на основі індикаторів.

Робота складається з чотирьох розділів.

У першому розділі сформовано постановку задачі та частини, які необхідно реалізувати.

У другому розділі розглянуто поширеність проблеми аналізу науково-технічної діяльності, джерела бібліографічних даних, існуючі рішення для взаємодії з ними, а також доведення необхідності у створенні програмного пакету.

У третьому розділі вказані деталі реалізації та необхідні залежності для роботи з програмним пакетом.

У четвертому розділі описаний покроковий алгоритм взаємодії користувача з системою та приклади програм з використанням основних методів взаємодії та можливі результати.

ABSTRACT

The note contains 50 pages, 26 pictures, 5 formulas, 3 appendices and 6 links.

The purpose of the work is to create a convenient and easily scalable package for working with bibliographic data and assessing scientific and technical performance based on indicators.

The work consists of four sections.

In the first section, the statement of the problem and the parts that need to be implemented are formed.

The second section considers the prevalence of the problem of analysis of scientific and technical activities, sources of bibliographic data, existing solutions for interaction with them, as well as proving the need to create the software package.

The third section provides details of the implementation and the necessary dependencies for working with the software package.

The fourth section describes the step-by-step algorithm of user interaction with the system and examples of programs using basic methods and possible results.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

ScopusID – унікальний ідентифікатор в бібліографічній базі Scopus

EID – Electronic Identifier

DOI – Document Object Identifier

ORCID – Open Researcher and Contributor ID

PII – Publication Item Identifier

ВСТУП

Управління науковою діяльністю потребує формальної та загальноприйнятої процедури оцінки продуктивності праці дослідників або дослідницьких колективів за для визначення взаємозв'язків, прогнозування та побудови стратегічних рішень та можливості їх співставлення до фактичних результатів. Одним із інструментів вирішення цієї задачі є застосування індикаторів, або їх комплексу за для відображення користі, стабільності та інтенсивності наукової діяльності суб'єкту. Також індикатори можуть сприяти позиціюванню суб'єкта наукової діяльності у рамках наукової галузі та оцінити власні нематеріальні активи.

Побудова систем індикаторів – це комплексна задача, яка включає багато аспектів та зовнішніх факторів, такі як збереження, однозначного ідентифікування та розробки універсальних або спеціалізованих методів аналізу бібліографічних даних. Сьогодні існують багато сервісів, які виконують той чи інший етап цієї глобальної задачі, наприклад Scimago Institution Rankings, ResearchGate, ORCID. Джерелом бібліографічних даних слугують спеціальні друковані каталоги, проте основна світова тенденція – це оцифровування даних та розміщення їх на спеціалізованих серверах. До них відносять такі гіганти Web of Science, Scopus та інші.

Проте той чи інший сервіс має свої обмеження у використанні. Одне з найголовніших обмежень – це використання рейтингування лише за індексом цитування, або своїми розробленими алгоритмами з наданням обмеженої кількість інформації за запитами користувача, без можливості інтеграції своїх наукометричих показників, що обмежує процес оцінки науково-технічної діяльності. Також існує проблема у створенні та тестуванні нових методів аналізу при використанні побічних джерел даних, а отже існує необхідність робота з першоджерелами напряму. Ці фактори є причиною до створення зручного та легко масштабованого пакету для роботи з бібліографічними даними.

Розроблений програмний пакет дозволить користувачу взаємодіяти з великими масивами даних за допомогою зрозумілого пошукового інтерфейсу.

1. ОЦІНКА ПРОДУКТИВНОСТІ НАУКВОЇ ДІЯЛЬНОСТІ

Першочерговим завданням роботи було дослідження різних джерел бібліографічних даних та аналіз можливості використання цих даних за допомогою стороннього програмного забезпечення. Результатом цього дослідження стало те, що у якості такої бази став Scopus, який надає доступ до своїх даних через безкоштовний API. Таким чином, формулювання наступних задач (пункт 1.1) відбувається з урахуванням Scopus Api.

1.1. Загальне формулювання завдання

Для створення програмного пакету необхідно вирішити наступні задачі:

1. Розробити модуль обробки HTTP запитів.
2. Розробити модуль обробки пошукових запитів до Search Scopus API.
3. Розробити модуль обробки пошукових запитів до Retrieval Scopus API.
4. Створити модуль доступу до системи для управління потоком даних між модулями та користувачем.
5. Реалізувати цільові функції описані в пункті 1.2.
6. Реалізувати додатковий функціонал описаний в пункті 1.3.

1.2. Цільові функції

У якості цільових функцій було обрано найпопулярніші індекси для оцінки науково-технічної продуктивності авторів, а саме: індекс цитування, коефіцієнт впливовості, індекс Хірша (h-index), g-index, а також індикатор оцінки продуктивності групи авторів – I-index.

Індекс цитування та індекс Хірша у базі даних Scopus можна отримати станом на сьогоднішній день, проте для аналізу за попередні роки необхідно перераховувати

значення, тому для підрахування значень цих індексів скористаємось наступними формулами.

1.2.1. Індекс Хірша (Гірша)

Одним із найпопулярніших показників науково-технічної діяльності суб'єкту дослідження є індекс Хірша (h-index), заснований на кількості публікацій та їх цитуваннях. Розрахунок відбувається за формулою (1.1)

$$h\_index(f) = \max_i \min(f(i), i), \tag{1.1}$$

де f - впорядкований за спаданням список кількості цитувань усіх публікацій автора;

i - позиція публікації у списку.

Ефективність даного показника зростає з роками, оскільки враховується загальна кількість цитувань. Малоефективний показник для авторів-початківців.

1.2.2. Коефіцієнт впливовості

Коефіцієнт впливовості, першочергово створений для оцінки впливовості наукових журналів (відношення числа процитованих статей за певний період відносно всіх опублікованих статей за цей же період у цьому журналі), може бути розрахований для оцінки наукової діяльності автора. На відміну від коефіцієнта впливу журналу, коли враховуються всі публікації, опубліковані в журналі за два роки, яких, як правило, багато, для авторів ці цифри можуть бути низькими, оскільки цей параметр дуже залежить від сфери та області діяльності. Тому, для підрахування цього індикатору для авторів рекомендовано використовувати щонайменше останні 5 років [1]. Розраховується за формулою (1.2)

$$КВ_{рік} = \frac{\sum_{j=1}^{3} Ц_{рік-j}}{\sum_{j=1}^{3} П_{рік-j}}, \tag{1.2}$$

де $Ц_i$ – кількість цитувань публікацій за і-тий рік;

$П_i$ - кількість публікацій за і-тий рік.

1.2.3. Індекс цитувань

Інструмент, що дозволяє оцінити загальну кількість цитувань публікацій автора та розраховується за формулою 1.3

$$SCI = \sum_{j=1}^{n} Ц_j , \tag{1.3}$$

де n - кількість публікацій автора;

$Ц_i$ – кількість цитувань і-тої публікацій;

1.2.4. g-index

Індекс для вимірювання наукової продуктивності, що розраховується на основі розподілу цитувань робіт автора за допомогою формули 1.4

$$g\_index(f) = \max_i (\sum_{n=1}^{i} f(n) >= i\text{^}2), \tag{1.4}$$

де f - впорядкований за спаданням список кількості цитувань усіх публікацій автора;

i - позиція публікації у списку.

1.2.5. I-index

Індикатор для вимірювання активності наукової групи авторів. Розраховується на основі індексу Хірша кожного з авторів за допомогою формули 1.5

$$I\_index(f) = \max_i \min(f(i), i), \tag{1.5}$$

де f - впорядкований за спаданням список індексів Хірша кожного із групи авторів;

i - позиція автора у списку.

1.3. Додатковий функціонал

Для комфортної роботи з пакетом необхідно реалізувати додатковий функціонал:

1. Для зменшення кількості запиті до Scopus API та вирішення проблеми повторного збору даних з метою конкретизувати запит, або розширити його необхідно реалізувати збереження отриманих даних.

2. Необхідно забезпечити достатнє логування етапів виконання на шляху розрахунку цільових функцій

- Етап отримання даних
- Етап збереження даних
- Результати

2. ДАНІ ТА ДЖЕРЕЛА НАУКОМЕТРИЧНИХ ДАНИХ

Оцінка рівня науково-технічної діяльності дає великий потенціал у розумінні потенціалу та продуктивності суб’єкта наукової діяльності у тій чи іншій сфері. Рейтинги допомагають виділити фаворитів серед більшості за потрібними показниками, дають можливість поглянути на роботу установи/підрозділу під різними кутами, виявити цінність нематеріальних активів.

Сукупність показників, даних про факти науково-технічної та інноваційної діяльності, дає можливість будувати індикатори, котрі визначають вплив суб’єкта оцінювання на локальну, регіональну чи світову науки.

2.1. Індикатори

На сьогоднішній день існує багато різних індикаторів, що виконують свою функцію в тому чи іншому степені – дають відповідь на питання про діяльність у межах певної моделі уявлень про сутність. Одним із найголовніших показників продуктивності науковців є індекс цитування [2]. За допомогою нього можна оцінити вклад автора у світову науку та дає можливість зрозуміти важливість та корисність тієї чи іншої публікації. На основі кількості цитувань публікацій або статей – можна отримати різні моделі росту та стабільності наукового потенціалу за певний проміжок часу. Але для створення власних показників необхідно враховувати представлення та вимоги, котрі висувають до індикаторів:

- Наявність посилань, або явного опису поняття, що характеризується індикатором.
- Наявність переліку видів діяльності, на які впливає стратегія, орієнтована на індикатор.
- Доцільність індикатора з огляду на якість та доступність даних.

- Зрозумілість для користувачів сенсу та обмежень індикатора.

2.2. Бібліографічні бази даних, їх представники та особливості.

Бібліографічна інформація – це інформація про джерело інформації, що дає можливість проводити ефективні та однозначні операції з нею. Це означає що повинна міститись достатня інформація про автора чи авторів публікації, інформація про установи, які безпосередньо приймали участь у створенні роботи, або співпрацювали з ними, інформацію про лабораторії де були проведені дослідження, ким робота була профінансована робота та з якою ціллю, про області в рамках яких було створено роботу, а також найголовніше – опис роботи, ключові слова та висновки. Бібліографічні бази даних витісняють традиційні друковані каталоги. Оцифровування друкованих матеріалів привносить більш широкий погляд на ту чи іншу область знань. Легкий доступ до робіт дає змогу молодим вченим швидше буди замічеными, що впливає на темпи розвитку світової науки в цілому. Проте, одне джерело не може всесторонньо увібрати в собі всі напрацювання вчених зі всього світу, саме тому при створенні систем бібліографічної інформації керуються перш за все потребами у такій інформації. Цей факт відповідає на питання чому джерел може бути багато і чому вони такі різні. Кожен з них відповідає за свою область у світовій науці, розглянемо деякі із них.

Web of Science – науко метрична платформа, на якій розміщена інформація про публікації, патенти книг та матеріалів наукових конференцій в таких областях, як: природничі, технічні, біологічні, суспільні, гуманітарні науки та мистецтво. Свої дані Web of Science індексую на основі журналів з 1900 року, а з 2015 року активно розширяє свої баз даних новими електронними виданнями.

Scopus – позиціонується корпорацією Elsevier як найбільша в світі універсальна реферативна база даних [3]. Дані збираються з таких областей, як: фізичні, медичні, соціогуманітарні та науки про життя. Цією бібліографічною базою даних індексуються наукові журнали, матеріали конференцій та серійні книжкові видання. База даних Scopus для багатьох країн, особливо європейських, є одним із

найвпливовіших інструментів для внутрішньої оцінки дослідницької діяльності. Архівні данні, включені до бібліографічних баз даних, трактуються з 1869 року.

Відносним новачком серед вище описаних гігантів виступає Google Scholar – система пошуку, що містить у собі повний текст наукових публікацій, дисертацій, технічних звітів, книг та веб сторінок, які мають науковий статус [4]. Користувач може отримати доступ до повного тексту таких джерел, або анотації до них, а також важливої інформації про індекс цитування роботи. Збір даних не обмежується якимись областями чи певними журналами, проте і не всі доступні джерела готові надавати свої бази чи архіви для поповнення контенту Google Scholar.

2.3. Існуючі сервіси по роботі з бібліографічною інформацією

Для взаємодії з бібліографічною інформацією, текстом, інформацією про автора та цитуванням робіт існує багато сервісів, як платних так і безкоштовних. Такі сервіси можна умовно поділити на внутрішні – які надаються як інтерактивний доступ до бібліографічних баз описаних в пункті 2.2 та зовнішні, які використовують ці бази з ціллю вирішення проблем приналежності роботи тому чи іншому автору, застосування омогліфів відносно імені автора при цитуванні його роботи, або звичайної індексації задля швидкості доступу до певних областей даних. Розглянемо декілька із них, почавши з першої групи - веб інтерфейсу Scopus, який надає можливість пошуку інформації про авторів, установ та документів та відображає її у стислому описовому вигляді. На рисунку 2.1 зображено інтерфейс при роботі з пошуком авторів, а на рисунку 2.2 – результат пошуку.

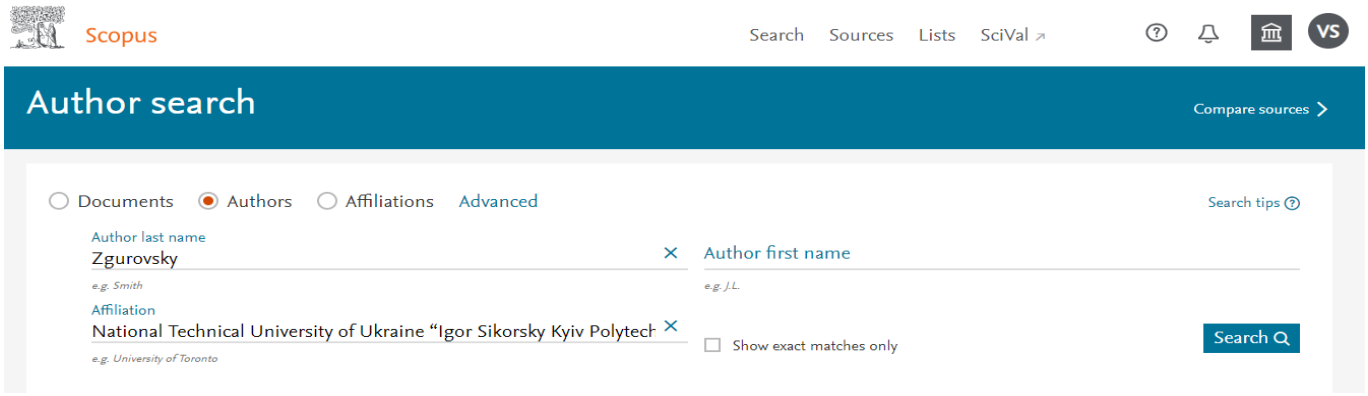

Рисунок 2.1 – Інтерфейс пошуку Scopus

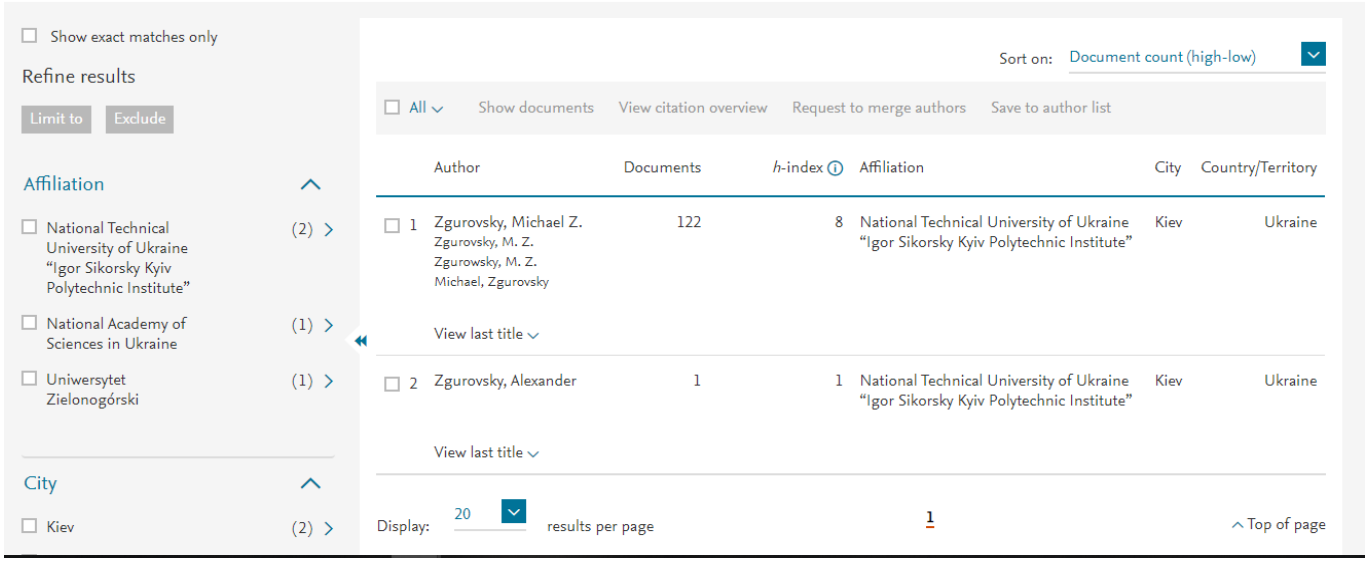

Рисунок 2.2 – Результат пошуку при запиті на рисунку 2.1

Сервіс надається безкоштовно та має API доступ з певними обмеженнями, які можна зняти за допомогою платної підписки.

Web of Science надає схожий перелік інформації про публікації автора, але працює за методом пошуку інформації за темами, а саме комплексних пошук по ключових словах, анотаціях, інформації про автора або видавця. На рисунку 2.3 можна побачити результат пошуку за темою “Kyiv Polytechnic Institute”, а на рисунку 2.4 можна якість і кількість наданої інформації.

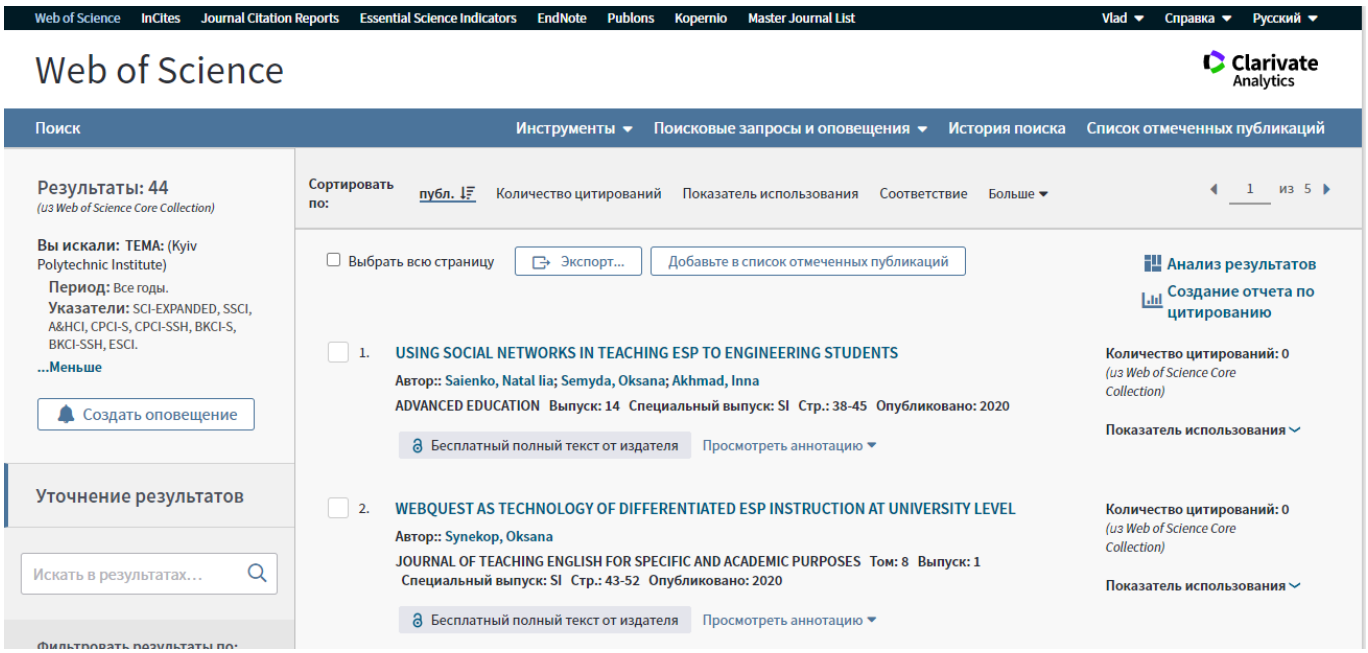

Рисунок 2.3 - Результат пошуку за темою “Kyiv Polytechnic Institute”

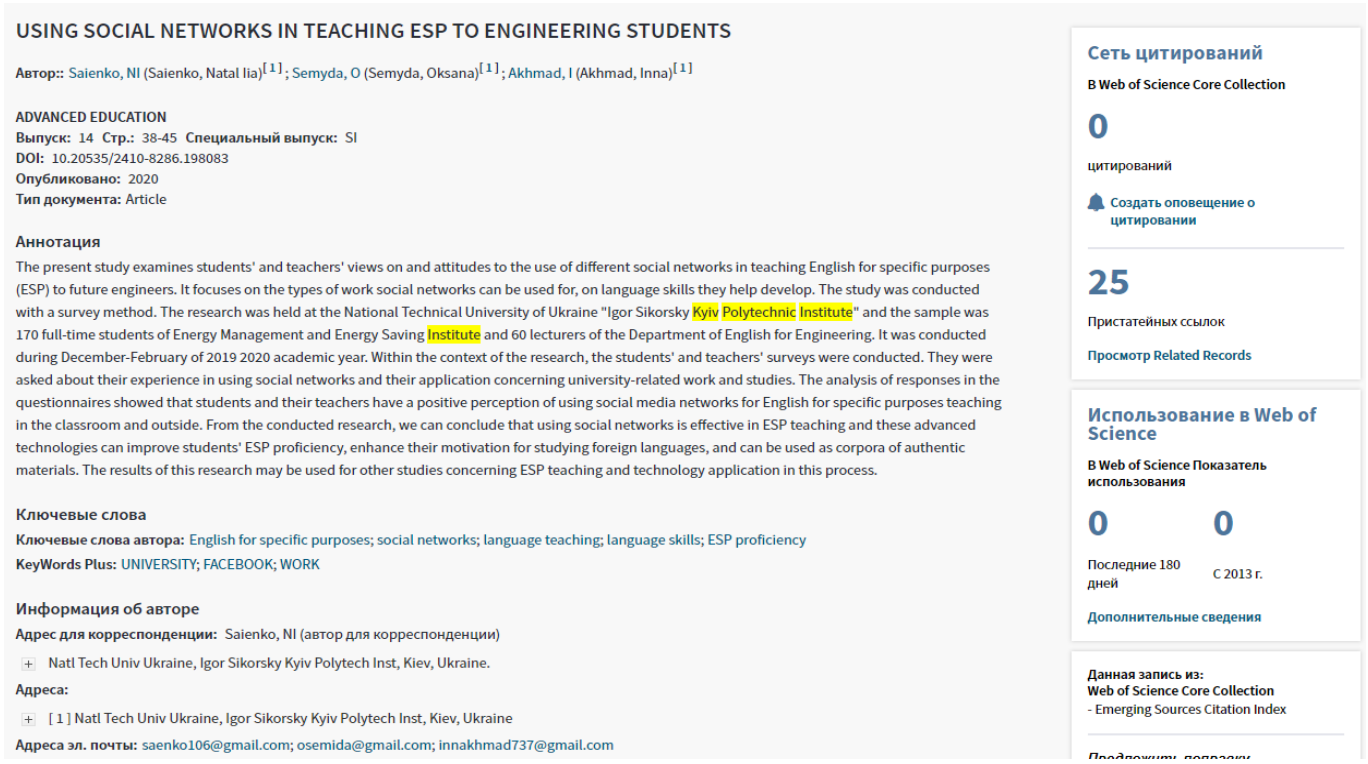

Рисунок 2.4 – Частина інформації про перший результат пошуку на рисунку 2.3

До другої групи належать сервіси сторонніх компаній. Усі вони створені за для розширення базових можливостей сервісів, що постачаються разом із базами бібліографічних даних.

Scimago – це сервіс, що працює з базою даних Scopus. Його методологія включає вирішення проблеми приналежності авторів до певних установ та проблеми з різними варіантами імен установ. Сам сервіс надає різні рейтинги установ за такими параметрами, як: сектор діяльності, країна або регіон, рік та тип рейтингу. Існують чотири типи рейтингу: загальний, дослідницький, інновацій та суспільний. На рисунку 2.5 зображено інтерфейс пошуку та результат у вигляді списку, який відсортований по обраному рейтингу.

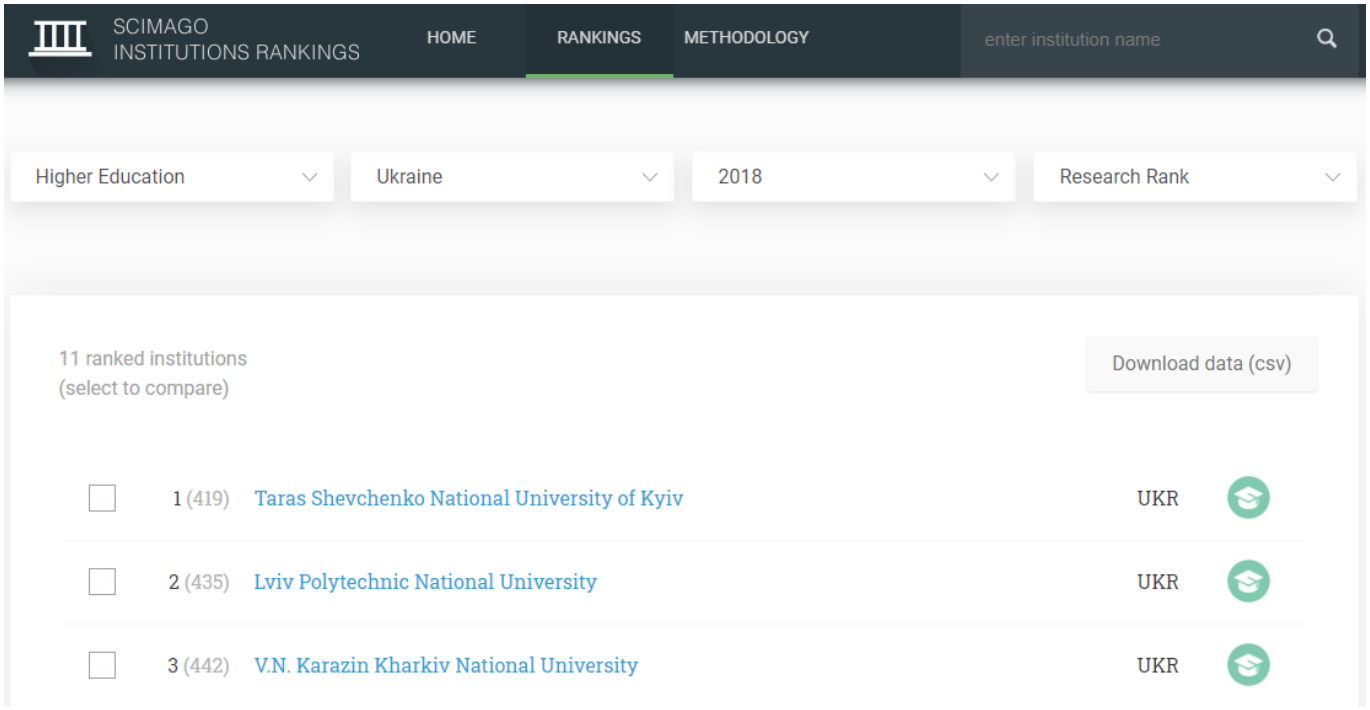

Рисунок 2.5 – Інтерфейс та результат рейтингу в Scimago

Також існує сервіс ORCID, який обрав за мету створення єдиного міжнародного реєстру вчених, яким надаються унікальні ідентифікатори, за якими можна слідкувати за автором через роки. Цей сервіс має тісну інтеграцію з Google Scholar, ResearcherID – сервіс, який взаємодіє з Web of Science, Scopus та дозволяє розв'язати проблему з нечіткістю в ідентифікуванні імені автора. На рисунку 2.6 можна інтерфейс та результат пошуку за прізвищем вченого.

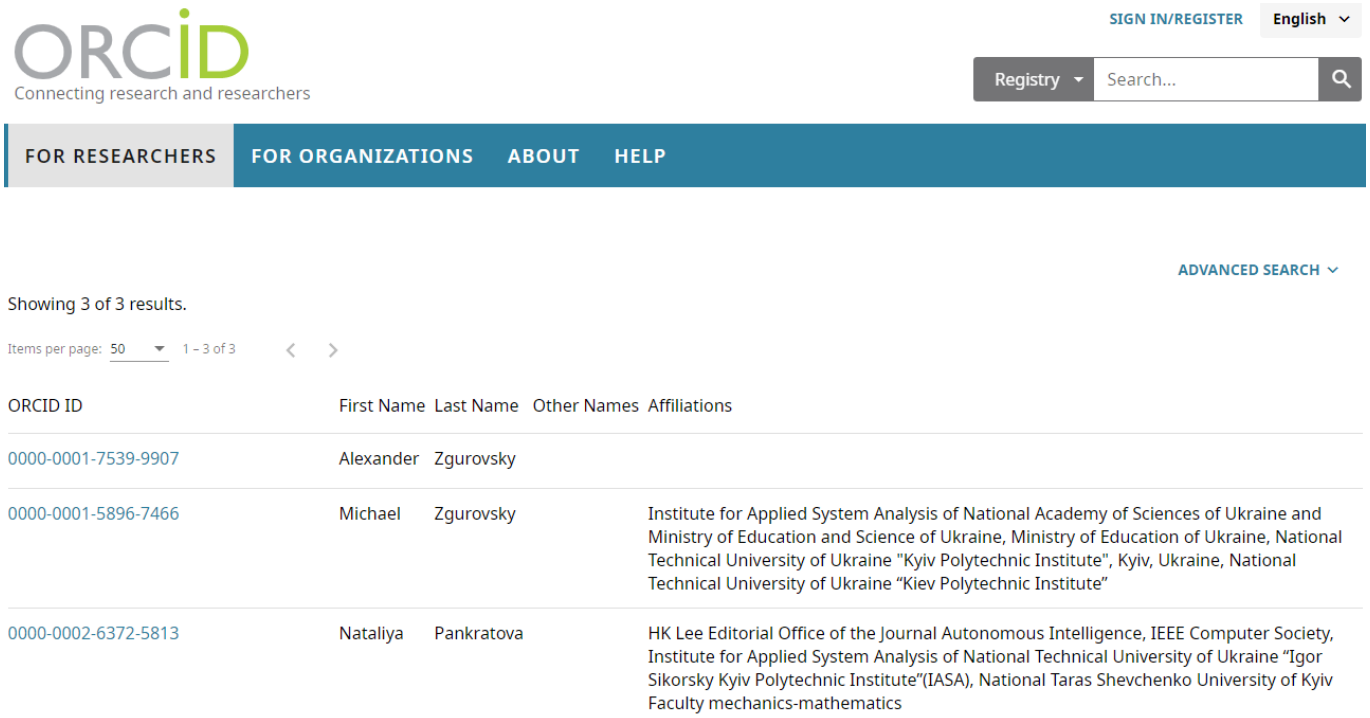

Рисунок 2.6 – Інтерфейс та результат пошуку на сервісі ORCID

ORCID розповсюджується як безкоштовний продукт з вашим особистим кабінетом автора. ORCID ID – один з небагатьох способів не втрачати важливі знання про цитування робіт автора за допомогою однозначно та багато джерельного ідентифікування.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПАКЕТА CORE

Для реалізації поставлених задач у розділі 1 було створено проект з структурою каталогів зображених на рисунку 3.1.

```
Gateway/                    < Кореневий каталог проекту
|
+---Core/                   < Каталог з розробленим пакетом
|
+---Data/                   < Каталог для збереження проміжних результатів на диск
|   |
|   +---RetrievalCache/             < Результати роботи з Retrieval API. Файли з повною інформацією
|   |    |
|   |    +---Author/                        < Інформація про авторів
|   |    |
|   |    +---AuthorMetrics/                 < Інформація метрики авторів
|   |    |
|   |    +---Affiliation/                   < Інформація про установи
|   |    |
|   |    +---ABS/                           < Інформація про документи
|   |
|   +---SearchCache/                < Результати загальних та великих пошукових запитів до Search API. Файли
|                                         з необхідною інформацією для ідентифікації об'єктів пошуку
|
+---Logs/                   < Лог файли про виконання userScript.py
|
+---Test/                   < Каталог для тестування пакету Core
|   |
|   +---TestData/                   < Каталог з тестовими даними
|   |
|   +---tests.py                    < Файл, що містить функції тестування модулів пакету Core
|
+---confin.json             < Файл конфігурації
|
+---userScript.py           < Будь-який файл користувача з кодом неохідних пошуково-аналітичних запитів
```

Рисунок 3.1 – Структура каталогів та файлів проекту з поясненнями

Обрана структура проекту надає легку орієнтацію та швидку взаємодію користувача із системою. Коренева група каталогів відповідає основним задачам, а саме програмний код пакету, місце збереження даних та місце збереження інформації про етапи виконання запитів користувача в *.py файлах (наприклад userScript.py).

3.1. Інструменти розробки та залежності

3.1.1. Мова програмування та необхідні інструменти

Для написання програми було використано мову програмування Python3.6, тому для використання створеного пакету Core (рис. 3.1) необхідно інсталювати на робочу машину Python3.6 та додаткові пакети, такі як:

- requests
- json
- urllib
- pathlib (pathlib2)
- logging
- threading
- queue
- fuzzywuzzy
- python-Levenshtein
- pytest

3.1.2. Сховище даних

Для вирішення проблеми повторного збору даних було використано тимчасове збереження до файлової системи. Для зручності навігації по збереженим даним, відповідно до рисунку 3.1, файли було розділено на дві зони, пошукові та файли з повною інформацією про конкретні об’єкти.

До перших належать файли для збереження результату параметризованого пошукового запиту до Search API з метою отримання ідентифікаторів для подальшого пошуку та/або іншої корисної інформації. Такі файли знаходяться у директорії ./Data/SearchCache/, мають розширення .json та є само-ідентифікованими, тобто їх ім’я відповідає повній структурі пошукового запиту, заданого користувачем. Проте не всі пошукові запити зберігаються; збереження таких результатів доцільно тільки для пошуку великих груп інформації, що займає тривалий час.

До других належать файли, отримані за допомогою Retrieval API. Ці файли лежать в директорії ./Data/RetrievalCache/, містять розширену інформацію про автора, його метрики, інституцію чи публікацію. Ця інформація зберігається у відповідні директорії ./Data/RetrievalCache/Author/, ./Data/RetrievalCache/AuthorMetrics/, ./Data/RetrievalCache/Affiliation/, ./Data/RetrievalCache/ABS до файлів з структурою імені author_{ScopusID}.json, aff_{ScopusID}.json та abs_{ScopusID або DOI}.json.

Отримані кеш файли з часом втрачають свою актуальність у зв'язку з оновленням бази даних Scopus, тому рекомендується періодично видаляти їх. Також рекомендується не зловживати збереженням коротких пошукових запитів, це може привести до колізії в даних під час виконання інших пошукових запитів. Найкращим рішенням є створення безпечного формату рішення залежно від використання для збереження проміжних пошукових результатів.

3.1.3. Обліковий запис Scopus API

Для використання розробленого пакету необхідно мати доступ до Scopus API. Для цього потрібно перейти на офіційний сайт для використання API та створити обліковий запис, при тому необхідною умовою для роботи з API є те, що пошукові запити користувача повинні відправлятись з тієї ж самої IP адреси, з якої була здійснена реєстрація. Наступний крок – створення ключів доступу. Для цього на головній сторінці потрібно перейти за вказаними посиланнями на рисунку 3.2 та 3.3.

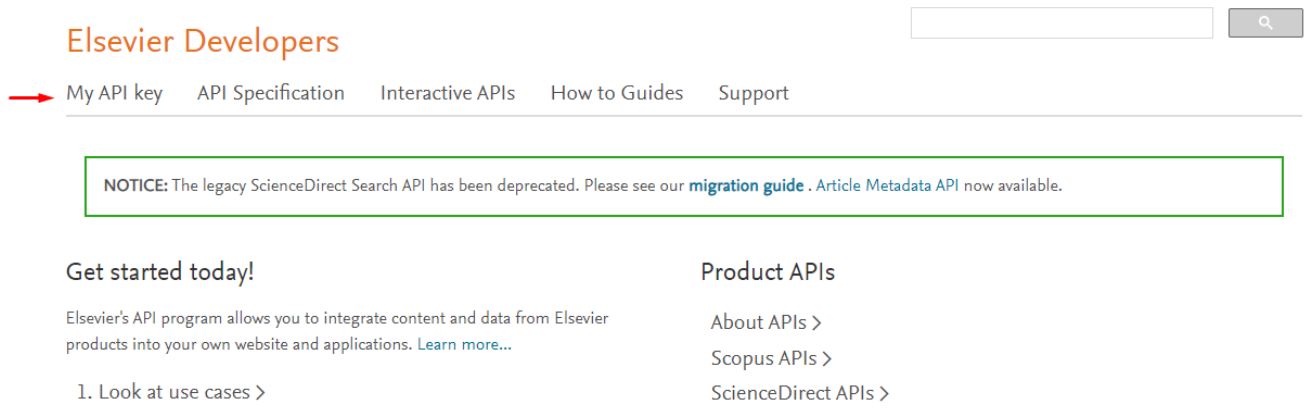

Рисунок 3.2 – Навігація на веб сторінці для створення ключів доступу, крок 1

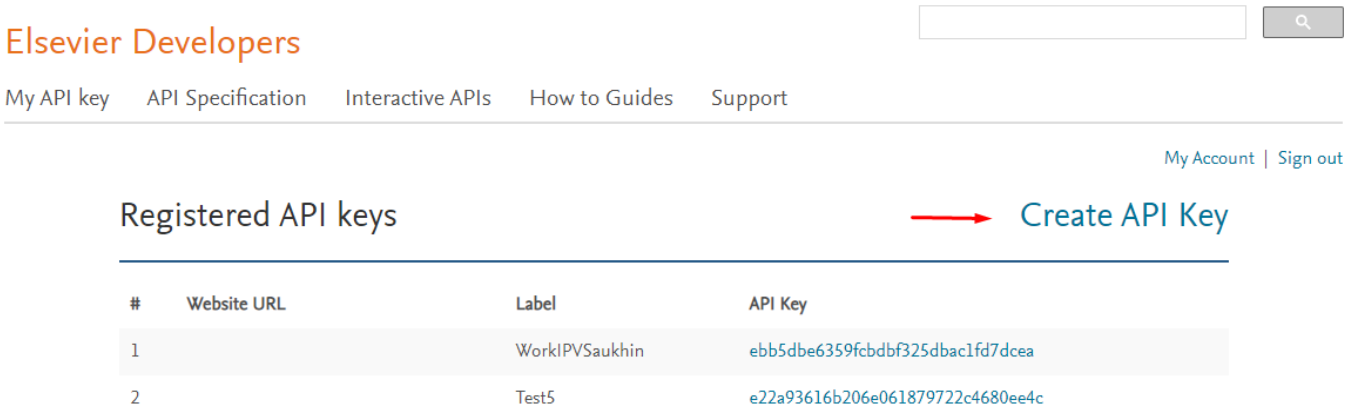

Рисунок 3.3 – Навігація на веб сторінці для створення ключів доступу, крок 2

Далі вказуємо ім’я ключа, будь-яке значення із латинських букв та цифр 0-9, погоджуємося з угодою користування та отримуємо ключ для базового користувача. Можна створити довільну кількість ключів, проте рекомендується створити не менше ніж десять ключів доступу, якщо ви збираєтесь виконувати аналіз для великої групи об’єктів, тому що від кількості ключів доступу напряму залежить кількість потоків виконання завантаження даних. Базовий пакет користувача має обмеження в 20000 запитів на тиждень, проте ви можете придбати передплату на офіційному сайті та зняти це обмеження.

3.2. Опис інтерфейсів доступу Scopus API

За для кращого розуміння створених класів та їх функцій необхідно розібрати інтерфейс доступу до даних. Він представляє собою реалізацію методу Get, представлення вказаного ресурсу, для HTTP запита з параметрами, які залежать від типу використаного інтерфейс [5].

Розглянемо типи інтерфейсів, їх параметри та дані, які можна отримати. Для базового користувача доступні Scopus Retrieval API Scopus Search API, перший з яких дає розгорнуту інформацію про авторів, установ та публікацій за такими ключами, як ScopusID, EID, DOI, ORCID, PII. До Retrieval API відносяться наступні посилання на ресурс:

1. Інтерфейс для пошуку інформації про установи має два варіанти:

1.1. https://api.elsevier.com/content/affiliation/affiliation_id/{affiliation_id}

1.2. EID - https://api.elsevier.com/content/affiliation/eid/{eid}

Інформація про установу включає в себе загальну інформацію про назву, місцезнаходження, посилання на веб сторінку установи, кількість публікацій та авторів закріплених за установою. Також важливим є список назв, за якими ідентифікують установу, допустимі скорочення та абревіатури, назви до змін на поточну назву.

2. Інтерфейс для пошуку інформації про авторів має три варіанта:

2.1. https://api.elsevier.com/content/author/author_id/{author_id}

2.2. EID - https://api.elsevier.com/content/author/eid/{eid}

2.3. ORCID - https://api.elsevier.com/content/author/orcid/{orcid}

Отримані дані містять повну інформацію про профіль автора, загальні характеристики, такі як: кількість документів, загальна кількість цитувань усіх робіт автор, які розміщені та доступні на Scopus, індекс Хірша розрахований на сьогоднішній день. Також міститься інформація про установи з якими була співпраця та спільні роботи, а також області наукової діяльності. Аналогічно до установи є список імен за якими неоднозначно ідентифікують автора, наприклад тільки прізвище та ініціали, або тільки прізвище. З цими іменами можливо хтось цитував роботи автора.

3. Інтерфейс для пошуку інформації про публікації має чотири варіанта:

3.1. https://api.elsevier.com/content/abstract /scopus_id/{scopus_id}

3.2. EID - https://api.elsevier.com/content/abstract/eid/{eid}

3.3. DOI - https://api.elsevier.com/content/abstract/doi/{doi}

3.4. PII - https://api.elsevier.com/content/abstract/pii/{pii}

Дані про публікацію містять в собі такі ключові пункти, як: назва, опис, рік публікації, область, ким було профінансовано та автор(и), що приймав(ли) участь у створенні. Також інформація включає в собі список усіх посилань цього документа, яка містить текст посилання, рік видання документу на який посилається(ються)

автор(и) цієї роботи. Для отримання списку публікацій, які цитують публікацію необхідно мати платну підписку та скористатись одним із заголовків для пошуку

- https://api.elsevier.com/content/abstract/citation-count?scopus_id={scopus_id}
- https://api.elsevier.com/content/abstract/citation-count?doi={doi}

Друга категорія відповідає за інтерфейс пошуку за заданими параметрами. Для отримання інформації потрібно створити булевий пошуковий запит (query), який слід виконати на кластері даних однієї із категорій, описаних вище. Під булевим запитом мається на увазі використання таких елементів як ‘and’, 'or' та дужок ‘(‘, ‘)’. Посилання для отримання даних:

1. https://api.elsevier.com/content/search/affiliation?query=’’
2. https://api.elsevier.com/content/search/author?query=’’
3. https://api.elsevier.com/content/search/scopus?query=’’

Також існує можливість додаткового уточнення пошукового запиту за допомогою наступних значень:

- field – уточнює, які поля слід повернути; список полів специфічний для кожного із пошукових інтерфейсів;
- start – дає можливість отримання даних зі зміщенням від першого результату; необхідно для продовження отримання даних відносно останнього результату до втрати сесії;
- count – кількість результатів, які повернуться за один запит; значення за замовчуванням було обрано 200, оскільки саме при такому навантаженні ми не отримуємо помилок від серверу;
- sort – надає можливість отримувати результати уже в відсортованому порядку; має обмеження на сортування за трьома полями максимум; для задання порядку сортування слід додати ‘+’ – за збільшенням чи ‘-‘ – за зменшенням для обраних полів, за замовченням сортування за збільшенням.

Для детальнішої інформації можна звернутись до документації на офіційному сайті.

3.3. Опис класів, об’єктів та функцій пакету Core

Пакет Core створений за для обробки даних, отриманих за допомогою інтерфейсів, описаних в пункті 3.2. Весь процес взаємодії із системою можна описати за допомогою діаграми прецедентів на рисунку 3.4. А також розглянемо детальніше реалізацію Core за допомогою діаграми класів на рисунку 3.6.

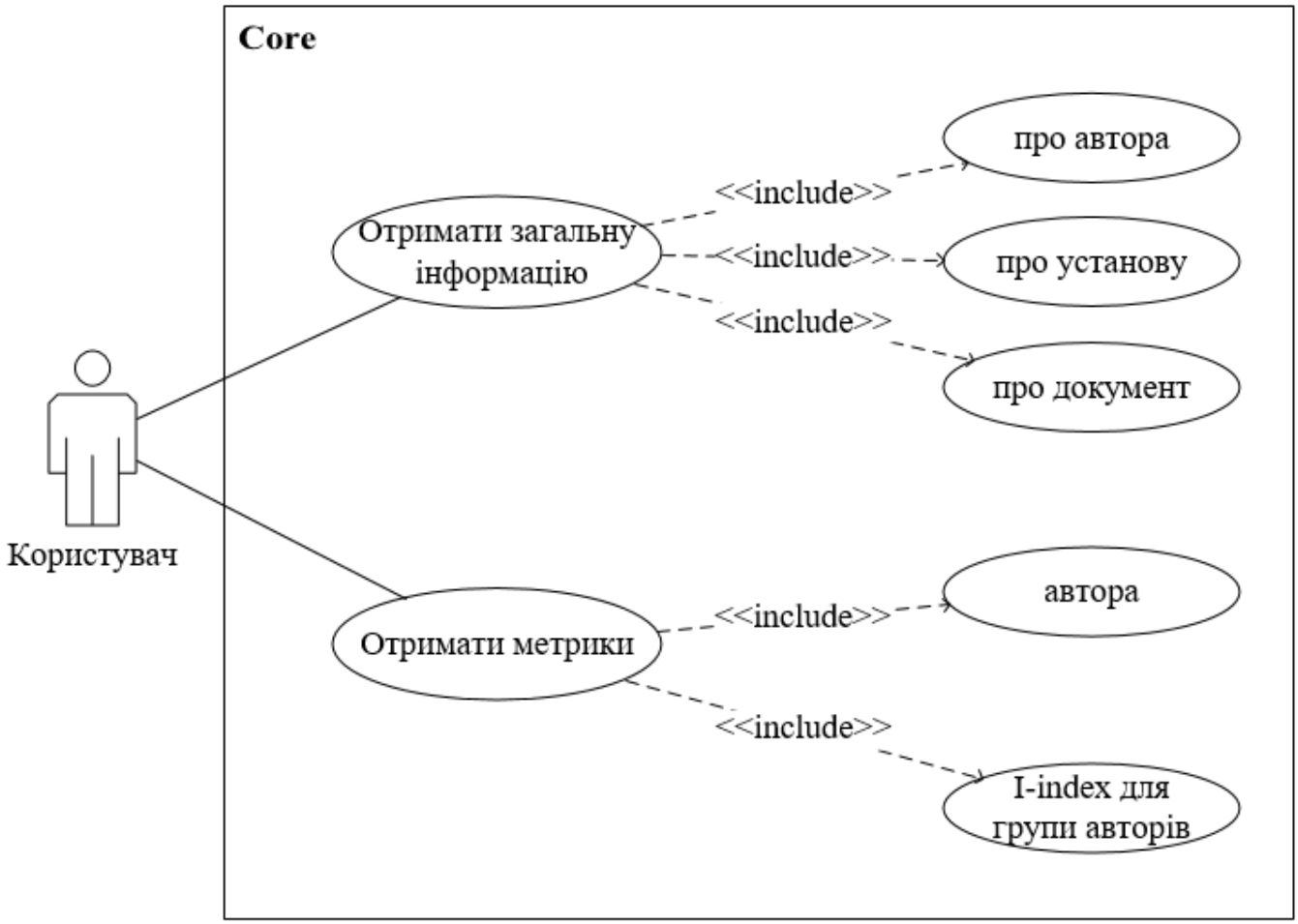

Рисунок 3.4 – Діаграма прецедентів

З діаграми на рисунку 3.4 видно, що для користувача виділено дві основні можливості, отримання загальної інформації та метричних даних. Взаємодія користувача із системою відбувається за допомогою модулю Gateway. Для роботи з цим класом необхідно створити конфігурацію, яку ініціалізувати одним із двох способів:

1. Створити або змінити файл config.py.

2. Створити змінну у вигляді словника (dictionary), ініціалізувати її відповідно до рисунку 3.5 та передати її до конструктора класу Gateway. Має пріоритет перед конфігурацією з файлу.

Файл конфігурації повинен мати синтаксис JSON та структуру описану на рисунку 3.5:

```
{
  "ScopusApi": {
    "apikeys": [
      "",
      ...
    ],
    "insttoken": ""
  },

  "Gateway": {
    "logLevel": "",
    "keysPerThread": ""
  }
}
```

Рисунок 3.5 – Структура файлу конфігурації

Конфігурація складається з двох саб-модулів. Саб-конфігурація ScopusAPI включає наступні значення:

- apiKeys – список ключів доступу, створених у пункті 3.1.3;
- insttoken – ідентифікатор, який ви можете отримати, якщо матимете платну передплату до даних Scopus. На основі цього значення можуть бути зняті обмеження доступу до безкоштовних інтерфейсів, а також доступ до нових, описаних в пункті 3.2.

Саб-конфігурація Gateway включає такі параметри, як:

- logLevel – вказує, які типи повідомлень необхідно логувати; приймає декілька значень:

1. logLevel = 1, якщо користувач хоче бачити тільки повідомлення про помилки та критичні помилки;

2. logLevel = 2, якщо користувач бажає бачити всі етапи виконання;

- keysPerThread – цей параметр вказує, скільки ключів доступу apiKeys необхідно виділити на один потік завантаження; це необхідно для забезпечення стабільності доступу до інтерфейсів Scopus API у випадку досягнення ліміту завантаження, або інших серверних помилок - оскільки відбудеться автоматичне переключення на доступ за іншим ключем.

Розглянемо основні методи класу Gateway, які інкапсулюють реалізацію взаємодії користувача з інтерфейсом, описаної на рисунку 3.4.

1. GetAffiliationInfo – метод для отримання інформації про установу(и). Пошук відбувається за допомогою наступних параметрів:

- ScopusAffilID – унікальний ідентифікатор інституції у бібліографічній базі Scopus; має найвищий пріоритет у пошуку;
- AffilName – варіант назви установи; за допомогою цього параметру можна передати повну, або коротку назву установи чи абревіатуру;
- City – місто, де знаходиться установа;
- Country - країна, де знаходиться установа;
- TopNMatch – при неточному порівнянні за заданими параметрами, может бути повернено максимум TopNMatch знайдених значень відсортованих за кількістю документів, які належать установі.

Як результат функція повертає одне, або список значень типу AffiliationInfo в залежності від пошукових параметрів. З описом AffiliationInfo можна ознайомитись на рисунку 3.6. Це значення можна відобразити у консольному варіанті за допомогою функції модулю ViewHelper – ViewAffiliationInfo. Вона приймає одне, або список значень AffiliationInfo, а також можна передати ідентифікатор indent, що дозволить виводити значення з відступом від лівого краю.

Для методу GetAffiliationInfo виникає проблема в ідентифікації інституцій, оскільки Scopus у свої базі може визначити факультет як незалежну одиницю, що заважає в опрацюванні даних. Також існує проблема у фантомних даних – інституціях, що мають ту саму назву, проте інше місцезнаходження та повну відсутність яких небудь робіт пов’язаних з нею. Наприклад, інститути мають на своїй базі факультети або інші підпорядковані інститути. Для вирішення цих проблем було

використано бібліотеку fuzzywuzzy для нечіткого порівняння назв. За основу береться інституція з найбільшою кількістю документів – як найймовірніший результат пошуку, проте після цього використовується нечітке порівняння назв закладів, щоб відсікти незаіндексовані дублікати, або несправжні дані. Для проходження бар'єру необхідно розходження імен більше чим на 20 відсотків.

2. GetAuthorInfo – метод для отримання інформації про автора(ів) та установу, на основі якої він працював, або працює. У якості аргументі функція може приймати параметри, описані для функції GetAffiliationInfo, а також:

- ScopusAuthorID – унікальний ідентифікатор автора у бібліографічній базі Scopus; має найвищий пріоритет у пошуку;
- AuthorName – ім'я або ініціали автора;
- AuthorLastname – прізвище автора.

Як результат функція повертає одне, або список значень типу AuthorInfo в залежності від пошукових параметрів. З описом AuthorInfo можна ознайомитись на рисунку 3.6. Це значення можна відобразити у консольному варіанті за допомогою функції модулю ViewHelper – ViewAuthorInfo. Вона приймає одне, або список значень AuthorInfo, а також можна передати ідентифікатор indent, що дозволить виводити значення з відступом від лівого краю.

3. GetDocumentInfo – метод для отримання інформації про документ(и). У якості аргументі функція може примати параметри, описані для функцій GetAffiliationInfo та GetAuthorInfo, а також:

- DOI – Document Object Identifier; має найвищий пріоритет;
- DocScopusID – унікальний ідентифікатор документа у бібліографічній базі Scopus; має найвищий пріоритет;
- Title – повна або часткова назва документа ;
- Summary – фраза, або список слів що можуть описати анотацію до роботи;
- Keywords – ключові слова, що описують роботу, її зміст та тематику;
- Year – чіткий рік публікації документа; має пріоритет над YearBefore та YearAfter

- YearBefore – під час пошуку необхідно орієнтуватись на роки до заданого;
- YearAfter – під час пошуку необхідно орієнтуватись на роки після заданого.

Як результат функція повертає одне, або список значень типу DocumentInfo в залежності від пошукових параметрів. З описом DocumentInfo можна ознайомитись на рисунку 3.6. Це значення можна відобразити у консольному варіанті за допомогою функції модулю ViewHelper – ViewDocumentInfo. Вона приймає одне, або список значень DocumentInfo та відображає результат у вигляді посилань на документи. Має додатковий параметр ViewAdditionalInformtion типу Bool зі значенням за замовченням – False. Якщо значення дорівнює True функція додатково відображає інформацію про того хто опублікував, опис, що було прикріплено до документу, інформацію про інститути, які співпрацювали для створення роботи, та повну інформацію про автора, який вважається контактним обличчям для журналістів.

4. GetAuthorMetrics – метод для отримання метричних даних, що включають у собі значення цільових функцій, описаних в пункті 1.2. У якості аргументі функція приймає ті ж параметри, що описані для функції GetAuthorInfo, але також приймає два обов’язкових параметра:

- Year – рік, для якого розраховуються цільові функції з пункту 1.2; цей параметр можна використовувати як спосіб побудови прогресу автора по рокам;
- ImpactFactorDelta – кількість років, які враховуються для розрахунку індикатора, описаного в пункті 1.2.2.

Як результат функція повертає одне, або список значень типу AuthorMetrics в залежності від пошукових параметрів. З описом AuthorMetrics можна ознайомитись на рисунку 3.6. Це значення можна відобразити у консольному варіанті за допомогою функції модулю ViewHelper – ViewAuthorMetrics. Вона приймає одне, або список значень AuthorMetrics.

5. I_indexForGroup – метод, який використовує список AuthorMetrics для підрахунку I-index, описаного в пункті 1.2.5. Функція повертає числове значення.

Параметри для функцій GetDocumentInfo, GetAffiliationInfo, GetAuthorMetrics та GetAuthorInfo користувач може задавати в будь-якому порядку та в будь-якій комбінації, проте для виконання пошуку необхідно задати хоча б одне з полів. Існує пряма залежність між точністю та кількістю параметрів пошуку та кількістю та якістю результату.

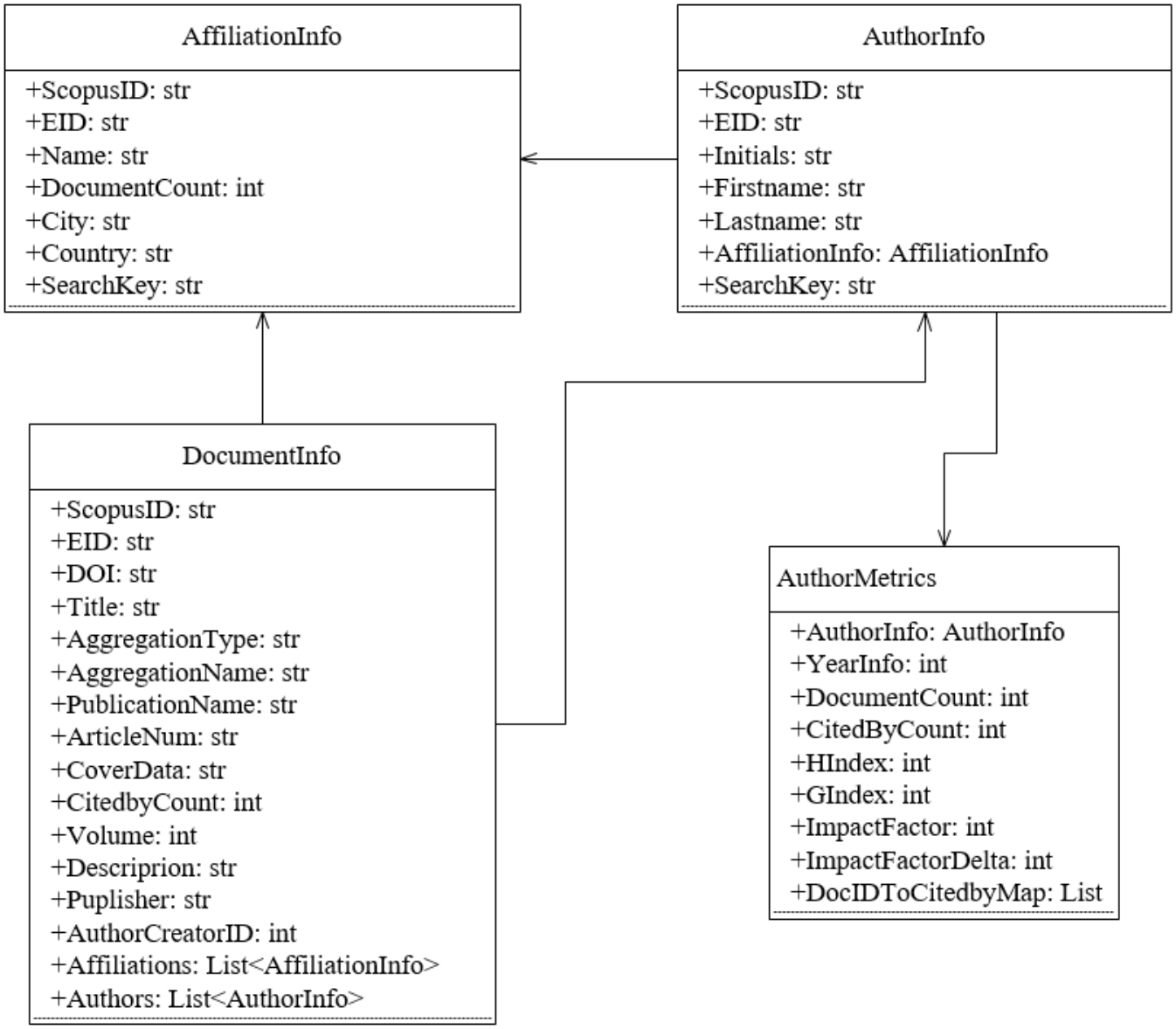

Рисунок 3.6 – Діаграма опису структур даних, що повертаються користувачу як результат виконання основних методів та зв’язки між ними

Такі структури, як AffiliationInfo, AuthorInfo, DocumentInfo, AuthorMetrics представляють собою набір даних, який повною мірою відображаються доступну

інформацію для відповідного об’єкту та виступають зручним застосунком як для подальшого використання користувачем пакету так і для внутрішнього обміну даних.

Для забезпечення роботи класу Gateway було створено класи, описані на діаграмі класів на рисунку 3.7, що забезпечують доступ до даних в Scopus та файлової системи. Класи ScopusRetrievalHandler та ScopusSearchHandler надають доступ до відповідних інтерфейсів Scopus API описаних у розділі 3.2.

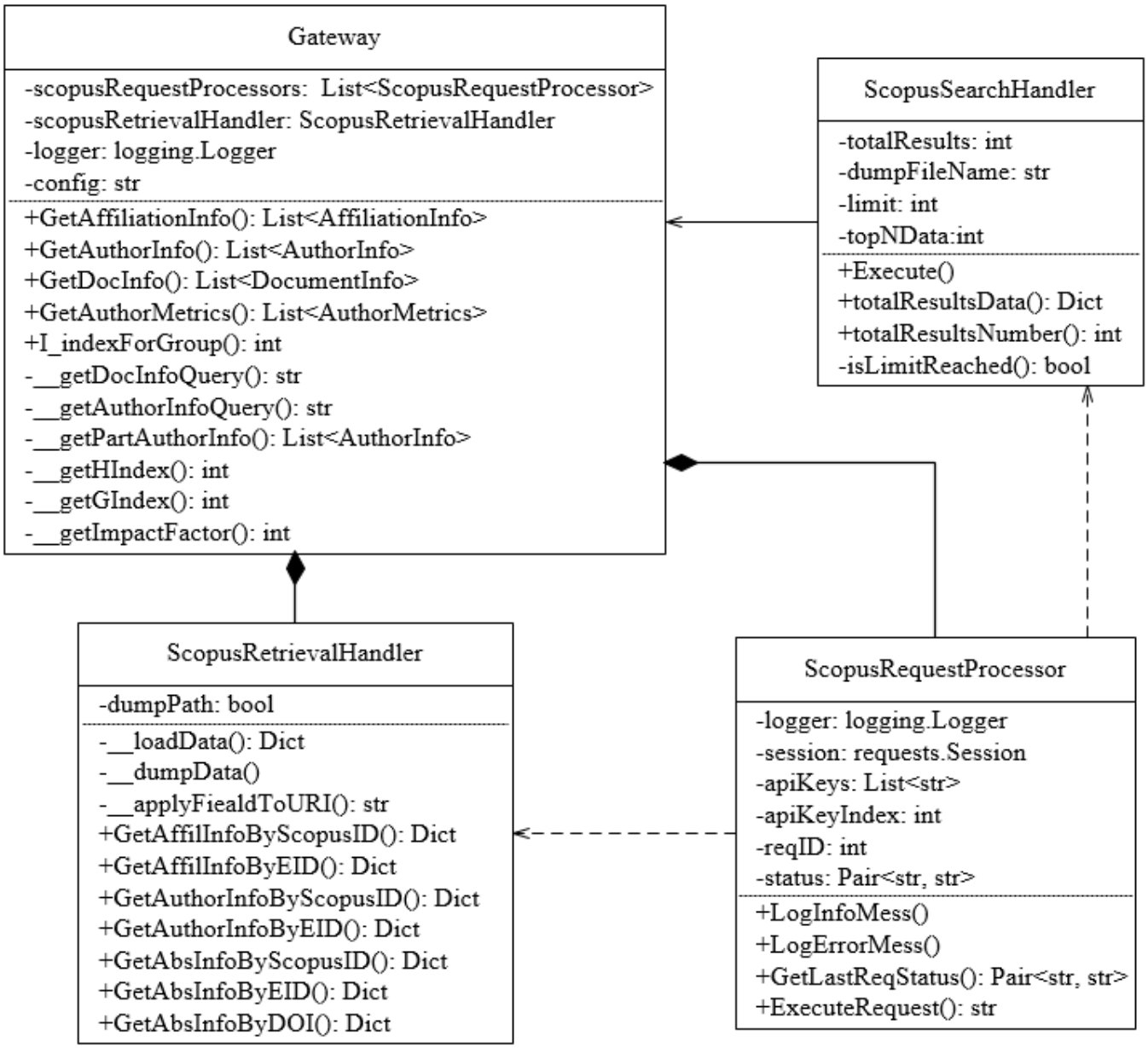

Рис. 3.7. Діаграма класів

Основна їх задача – сформулювати правильний запит та передати його на виконання до класу ScopusRequestProcessor. Побічна задача цих класів – збереження отриманих даних до файлової системи згідно з рисунком 3.1 зі спеціальним іменем файлу, за яким, при повторному пошук цих самих даних, інформація буде повернена. Запити до Scopus API можуть бути сформульовані та відправлені не частіше чим один раз на секунду – це обмеження не можна обійти. Також, не можна отримати повну інформацію за допомогою ScopusRetrievalHandler для багатьох ідентифікаторів за раз, оскільки інтерфейс розрахований на одиночне отримання інформації. Усі ці обмеження наштовхнули на використання додаткового класу, що дозволяє завантажувати дані в багато поточному режимі. Кількість потоків залежить від кількості наданих ключів та конфігуційного параметру keysPerThread. Цей клас вбудовано до методів GetAuthorMetrics та GetDocumentInfo, а в якості об’єкту синхронізації було використано чергу, куди передаються значення ідентифікаторів, що необхідно завантажити. Коли значення в черзі закінчяться – це буде означати, що весь запит виконано.

На основі отриманих даних, користувач може однозначно ідентифікувати об’єкти та проводити аналіз з використанням свої цільових функцій.

3.4. Тестування

Для перевірки роботи всіх модулів пакету Core було створено набір тестів з використанням фреймворку pytest. Тести необхідно виконати перед початком роботи з пакетом з метою перевірки правильності виконання усіх необхідних умов зазначених в пункті 3.1 та як гарантія того, що очікуваний процес виконання збережений після внесення змін до коду пакета.

Для гарантування коректної роботи тест повинні перевірити наступні сценарії:

1. Перевірити структуру файлу конфігурації, якщо такий існує.
2. Перевірити можливість доступу до всіх інтерфейсів Scopus API з використанням наданих ключі від користувача для роботи ScopusSearchHandler та ScopusRetrialHandler.

3. Перевірка роботи ScopusSearchHandler для пошуку інформації по трьом категоріям: “author”, “affiliation”, “scopus”.

4. Перевірка роботи ScopusSearchHandler з файловою системою.

5. Перевірка роботи всіх методів класу ScopusRetrialHandler.

6. Перевірка роботи збереження ти вичитки даних з файлової системи при взаємодії з методами ScopusRetrialHandler.

7. Перевірка компонування результату виконання методу GetAffiliationInfo, використовуючи дані з директорії ./Test/TestData/.

8. Перевірка роботи методу ViewAffiliationInfo з можливими результатами.

9. Перевірка компонування результату виконання методу GetAuthorInfo, використовуючи дані з директорії ./Test/TestData/.

10. Перевірка роботи методу ViewAuthorInfo з можливими результатами.

11. Перевірка компонування результату виконання методу GetAuthorMetrics, використовуючи дані з директорії ./Test/TestData/.

12. Перевірка роботи методу ViewAuthorMetrics з можливими результатами.

13. Перевірка компонування результату виконання методу GetDocumentInfo, використовуючи дані з директорії ./Test/TestData/.

14. Перевірка роботи методу ViewDocumentInfo з можливими результатами.

15. Перевірка розрахунку I-index для різних груп авторів, використовуючи дані з директорії ./Test/TestData/.

4. ПРИКЛАДИ ВИКОРИСТАННЯ ПАКЕТУ CORE

4.1. Алгоритм використання пакету Core

Для початку роботи з програмою необхідно виконати наступні дії:

1. Інсталювати необхідні інструменти описані в пункті 3.1.1.
2. Створити обліковий запис та ключі доступу для роботи з Scopus API за вказівками в пункті 3.1.3.
3. Створити або змінити, якщо потрібко, файл конфігурації з назвою config.json. Файл конфігурації повинен лежати поруч з файлом користувача, згідно з рисунку 3.1.
4. Створити новий файл з розширенням .py та імпортувати туди Gateway та за бажанням ViewHelper модулі, що дасть доступ до використання основних функцій пакету, описаних на рисунку 3.4.
5. Створити запит до системи та запустити програму на виконання. Виконання деяких запитів може займати тривалий час, особливо якщо немає можливості перевикористати дані зі старих запитів. Саме тому у якості індикатора прогресу виконання було створено відображення до консолі символу “.” (крапка) як результат виконання взаємодії з функціями отримання даних по мережі або вичитки з файлової системи.

4.2. Приклади використання методів пошуку та аналізу даних

Розглянемо детальніше всі основні методи класу Gateway на прикладі опису можливих варіантів програм та їх результату. На рисунку 4.2 зображено пошуковий запит для отримання інформації про інституцію за допомогою варіанту параметрів імені та міста з вказанням конфігурації програмно, замість використання файлу

конфігурації. Усі наступні приклади будуть використовувати конфігурацію з файлу config.py.

```
1   from Core import Gateway
2   from Core import ViewHelper
3
4   config = {
5     "ScopusApi": {
6       "apikeys": [
7         "ebb5dbe6359fcbdbf325dbac1fd7dcea", "9db4c7920cbf44ee705125a1962dcd50",
8         "10be8094129e9f35bb6b1d21d835cd53", "b04fabb2b511cc5afedadbebb1ab519b"],
9       "insttoken": ""},
10    "Gateway": {
11      "logLevel": "2",
12      "keysPerThread": "1"}
13  }
14
15  core = Gateway.Gateway(config = config)
16
17  affInfo = core.GetAffiliationInfo(City='Kyiv', Country='Ukraine', TopNMatch=3)
18  ViewHelper.ViewAffiliationInfo(affInfo)
```

Рисунок 4.1 – Пошук інформації за допомогою методу GetAffiliationInfo

На рисунку 4.1 на рядку 4 створюється змінна конфігурації згідно з описом на рисунку 3.5, яка передається як параметр до конструктору класу Gateway на рядку 15.

```
1. AffiliationName: Taras Shevchenko National University of Kyiv
 SearchKey: Kyiv / Ukraine
    ScopusID: 60023137
    EID: 10-s2.0-60023137
    DocumentCount: 19162
    Country: Ukraine
    City: Kiev
2. AffiliationName: National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
 SearchKey: Kyiv / Ukraine
    ScopusID: 60003172
    EID: 10-s2.0-60003172
    DocumentCount: 8729
    Country: Ukraine
    City: Kiev
3. AffiliationName: National University of Kyiv-Mohyla Academy
 SearchKey: Kyiv / Ukraine
    ScopusID: 60024416
    EID: 10-s2.0-60024416
    DocumentCount: 608
    Country: Ukraine
    City: Kiev Region
```

Рисунок 4.2 – Результат виконання програми на рисунку 4.1

Метод ViewAffiliationInfo виведе приблизний до рисунка 4.2 результат, де відображено топ 3 закладів у місті Києві за чисельністю наукових робіт, проіндексованих в базі Scopus на 2020 рік. Дані можуть змінюватись.

На рисунку 4.3 наведено код програми, як приклад запиту для пошуку за допомогою метода GetAuthorInfo з використанням таких параметрів як: прізвище авторів, назвою інституту та назвою міста, де знаходиться заклад. Параметр AuthorLastname на рядку 5 було ініціалізовано декількома прізвищами за допомогою створення логічного виразу з використанням “or”.

```
1  from Core import Gateway
2  from Core import ViewHelper
3
4  core = Gateway.Gateway(config = config)
5
6  authorInfo = core.GetAuthorInfo(
7     AuthorLastname = 'Fokin or Poplavko or Loktev',
8     AffilName = 'KPI', City = 'Kyiv', TopNMatch = 10)
9
10 ViewHelper.ViewAuthorInfo(authorInfo)
11
12 for au in authorInfo:
13  ViewHelper.ViewAuthorMetrics(
14     core.GetAuthorMetrics(2020, 7, ScopusAuthorID = au.scopusID)
15  )
```

Рисунок 4.3 – Приклад програми з використанням GetAuthorInfo та GetAuthorMetrics

Як результат ми отримаємо групу із трьох авторів, інформацію про яких можна побачити на рисунку 4.4.

1. PreferredName: Vadim M. Loktev (V.M.)
   ScopusID: 14009023200
   EID: 9-s2.0-14009023200
   DocumentCount: 278
   SearchKey: KPI / Kyiv
   Affiliation:
   1. AffiliationName: National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
      ScopusID: 60003172
      Country: Ukraine
      City: Kiev
2. PreferredName: Yu. M. Poplavko (Y.M.)
   ScopusID: 7004301579
   EID: 9-s2.0-7004301579
   DocumentCount: 153
   SearchKey: KPI / Kyiv
   Affiliation:
   1. AffiliationName: National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
      ScopusID: 60003172
      Country: Ukraine
      City: Kiev
3. PreferredName: Andrey A. Fokin (A.A.)
   ScopusID: 7102931335
   EID: 9-s2.0-7102931335
   DocumentCount: 145
   SearchKey: KPI / Kyiv
   Affiliation:
   1. AffiliationName: National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
      ScopusID: 60003172
      Country: Ukraine
      City: Kiev

Рисунок 4.4 – Результат виконання методу ViewAuthorInfo
на рядку 10 рисунка 4.3

Метод GetAuthorInfo повернув список значень, які найймовірніше відповідають до запиту користувача. Користувач може вибрати необхідний набір даних проаналізувавши відображений результат, або уточнити пошуковий запит на основі отриманої інформації та продовжити роботу з масивом даних.

Іншою частиною виконання коду на рисунку 4.3 було розрахунком показників продуктивності наукової діяльності. Результат можна побачити на рисунку 4.5.

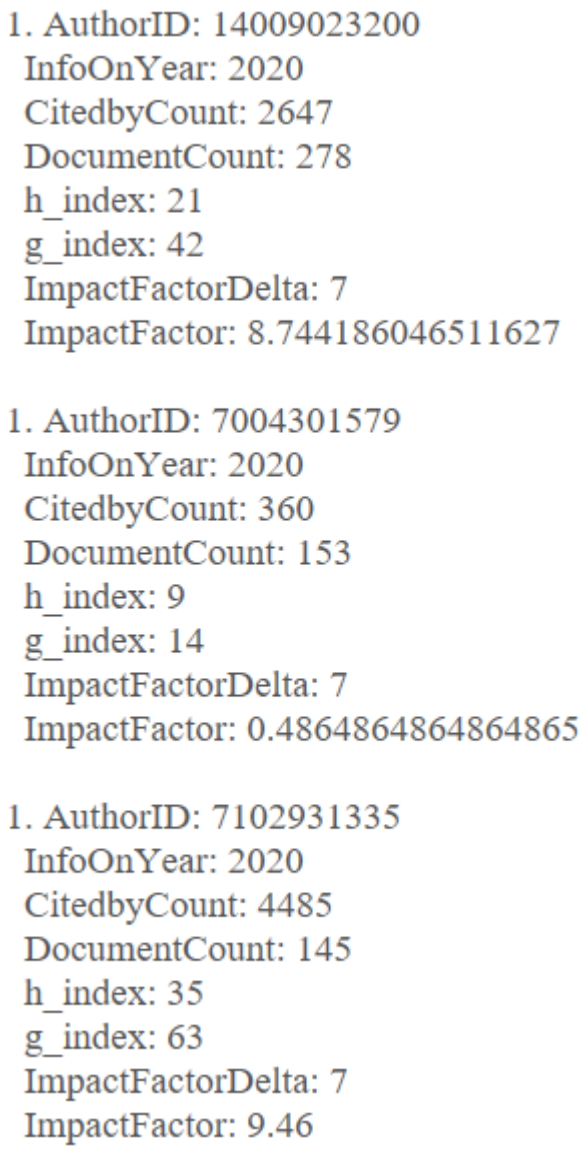

Рисунок 4.5 – Результат виконання методів ViewAuthorMetrics на рядку 13 рисунка 4.3

Необхідно пам’ятати, що метод ViewAuthorMetrics не відображає такого поля структури AuthorMetrics як docIDToCitedbyMap через його можливий великий розмір. Проте користувачу надана можливість його використання, тому для точного розуміння як сформоване це поле, розглянемо приклад на рисунку 4.6. Розглянемо кожне з елементів окремо:

- citedby-count – загальна кількість цитувань документа на сьогоднішній день;
- coverDate – день публікації документа; може бути використаний у рамках аналізу на проміжках часу;

- scopusABSID – унікальний ідентифікатор документа в базі Scopus;
- scopusAFIDs – список унікальних ідентифікаторів установ, у підпорядкуванні яких було створено документ; за допомогою цього поля можна розширювати методи аналізу діяльності автора, або групи авторів у межах однієї або декількох обраних установ; це поле може мати будь-яку кількість ідентифікаторів, включаючи жодного.


```
[
 {
  "citedby-count": "134",
  "coverDate": "2000-01-01",
  "scopusABSID": "0033667142",
  "scopusAFIDs": [
   "60068500"
  ]
 },
 {
  "citedby-count": "109",
  "coverDate": "2003-01-01",
  "scopusABSID": "0037299383",
  "scopusAFIDs": [
   "60068500",
   "60002414"
  ]
 },
 {
  "citedby-count": "102",
  "coverDate": "2007-09-01",
  "scopusABSID": "34547665107",
  "scopusAFIDs": []
 },
...
]
```


Рисунок 4.6 – Приклад заповнення поля docIDToCitedbyMap у структурі AuthorMetrics

На основі будь-якої групи вчених можна побудувати її I-index описаний у розділі 1.2.5. На рисунку 4.7 можна побачити модифіковану програму з рисунка 4.3,

де було додано підрахування індексу для тієї ж групи вчених. З можливим результатом виконання прикладу на рисунку 4.7 можна ознайомитись на рисунку 4.8 де було відображено повідомлення та числове значення, що відповідає індексу групи з топ 50 авторів пов’язаних з Київським Політехнічним Інститутом за кількістю робіт проіндексованих у базі Scopus. Дані можуть відрізнятися.

```
1   from Core import Gateway
2   from Core import ViewHelper
3
4   core = Gateway.Gateway()
5
6   result = core.GetAuthorMetrics(
7       2020, 7,
8       AffilName = 'Kyiv Polytechnic Institute',
9       Country='Ukraine', TopNMatch = 50)
10
11  indexRes = core.I_indexForGroup(result)
12  print('The I-index for the group  is {}'.format(indexRes))
```

Рисунок 4.7 – Приклад програми для отримання I-index

```
The I-index for the group  is 16
```

Рисунок 4.8 – Результат програми на рисунку 4.7

Розглянемо приклад програми з використанням GetDocumentInfo та ViewDocumentInfo. На рисунку 4.9 представлено код програми для пошуку найбільш процитованих робіт у “Київського Політехнічного Інституту” між 2012 та 2014 роками. Параметри YearAfter та YearBefore (рядок 8) задають значення не включно.

```
1   from Core import Gateway
2   from Core import ViewHelper
3
4   core = Gateway.Gateway()
5
6   res = core.GetDocumentInfo(
7       AffilName='Kyiv Polytechnic Institute', TopNMatch=5,
8       YearAfter=2011, YearBefore=2015)
9
10  ViewHelper.ViewDocumentInfo(res)
```

Рисунок 4.9 – Приклад програми з використанням GetDocumentInfo

Результатом виконання пошукового запиту стало п’ять значень, які було відображено за допомогою ViewDocumentInfo у вигляді цитування на відповідний документ, що можна використати для своїх наукових робіт. Кожен запис включає кожен з наступних параметрів:

1. список авторів, кількість яких залежить від параметру FirstAuthorCount, що має значення за замовченням – три;

2. назву роботи;

3. ім’я джерела, де опубліковано роботу;

4. ідентифікатор серійної публікації;

5. ідентифікатор для документа, який використовуються кількома видавцями замість або на додаток до номерів сторінок; номери статей можуть бути призначені під час електронного опублікування, тому документи можна цитувати та шукати раніше в процесі публікації;

6. дата публікації;

7. DOI;

8. кількість цитувань.

Прикладом можливого результату виконання 10 рядка на рисунку 4.9 є наступний список документів:

1. Denafas G., Ruzgas T., Martuzevičius D., Seasonal variation of municipal solid waste generation and composition in four East European cities, Resources, Conservation

and Recycling, 89, 22-30, (2014-01-01). doi:10.1016/j.resconrec.2014.06.001 (cited 48 times)

2. Gallego F.J., Kussul N., Skakun S., Efficiency assessment of using satellite data for crop area estimation in Ukraine, International Journal of Applied Earth Observation and Geoinformation, 29, 22-30, (2014-01-01). doi:10.1016/j.jag.2013.12.013 (cited 63 times)

3. Kogan F., Kussul N., Adamenko T., Winter wheat yield forecasting in Ukraine based on Earth observation, meteorologicaldata and biophysical models, International Journal of Applied Earth Observation and Geoinformation, 23, 192-203, (2013-01-01). doi:10.1016/j.jag.2013.01.002 (cited 84 times)

4. Feinberg E.A., Kasyanov P.O., Zadoianchuk N.V., Average cost Markov decision processes with weakly continuous transition probabilities, Mathematics of Operations Research, 37, 591-607, (2012-11-01). doi:10.1287/moor.1120.0555 (cited 41 times)

5. Skakun S., Kussul N., Shelestov A., Flood Hazard and Flood Risk Assessment Using a Time Series of Satellite Images: A Case Study in Namibia, Risk Analysis, 34, 1521-1537, (2014-01-01). doi:10.1111/risa.12156 (cited 47 times)

Проте це не вся інформація, яку можна отримати при використанні ViewDocumentInfo. За допомогою не булевого параметру ViewAdditionalInformtion, який має значення за замовчуванням – False, можна отримати додаткову інформацію по кожному з документів: опис, повний список інформації про авторів (або автора, що вважається представником в залежності від параметру FirstAuthorCount та кількості авторів, включених до сформованого посилання на документ) та інституції, що були задіяні в роботі. Розглянемо детальну інформацію для роботи під назвою “Average cost Markov decision processes with weakly continuous transition probabilities” що має таку додаткову інформацію:

Publisher: No information

Description: This paper presents sufficient conditions for the existence of stationary optimal policies for average cost Markov decision processes with Borel state and action sets and weakly continuous transition probabilities. The one-step cost functions may be unbounded, and the action sets may be noncompact. The main contributions of this paper are: (i) general sufficient conditions for the existence of stationary discount optimal and average cost

optimal policies and descriptions of properties of value functions and sets of optimal actions, (ii) a sufficient condition for the average cost optimality of a stationary policy in the form of optimality inequalities, and (iii) approximations of average cost optimal actions by discount optimal actions. © 2012 INFORMS.

Representative author:

1. PreferredName: Feinberg E.A. Feinberg (E.A.)
   ScopusID: 7006435357
   Affiliation:
   1. AffiliationName: Stony Brook University
      ScopusID: 60026415
      Country: United States
      City: Stony Brook

Affiliation involved in the creation:

1. AffiliationName: Stony Brook University
   ScopusID: 60026415
   Country: United States
   City: Stony Brook
2. AffiliationName: National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”
   ScopusID: 60003172
   Country: Ukraine
   City: Kiev

4.3. Приклади розрахунку індикаторів на основі функцій користувача

Методи GetAffiliationInfo, GetAuthorInfo та GetDocumentInfo націлені на пошук списку необхідних ScopusID, для того, щоб забезпечити однозначо-ідентифіковані вхідні данні для GetAuthorMetrics. На основі описаної структури на рисунку 4.6 користувач пакету Core може будувати свої індикатори та/або динаміку

протягом років. Розглянемо приклади можливого використання для проведення аналізу за допомогою власних функцій користувача на основі алгоритму модифікованого алгоритму по підрахунку індексу Хірша. На рисунку 4.10 наведений приклад програми для проведення аналізу метрик автора у період між 1990 та 2020 роками. На рядках 5 та 6 можна бачити ScopusID для об'єктів дослідження, які були отримані за допомогою методів пошуку інформації. Для розрахунку нових індексів розроблено дві функції GetNewIndex та GetNewIndex2. GetNewIndex використовує алгоритм індексу Хірша, проте враховую лише ті документи, де кількість цитувань більша за 10. GetNewIndex2 використовує алгоритм індексу Хірша, проте враховую кількість цитувань робіт, написаних за останні 10 років відносно року для якого розраховується індикатор.

```
1  from Core import Gateway
2
3  core = Gateway.Gateway()
4
5  KPIID = '60003172'
6  LoktevVadimID = '14009023200'   #current year: doc: 279; h-index: 21
7
8  def GetNewIndex(docIDToCitedbyMap):
9    if len(docIDToCitedbyMap) != 0:
10     citedList = [int(i.get('citedby-count')) for i in docIDToCitedbyMap if int(i.get('citedby-count') or 0) > 10]
11     citedList.sort(reverse=True)
12     return max([min(i + 1, citedList[i]) for i in range(len(citedList))])
13   return None
14
15 def GetNewIndex2(docIDToCitedbyMap, year):
16   if len(docIDToCitedbyMap) != 0:
17     lastNYear = 10
18     citedList = [int(i.get('citedby-count')) for i in docIDToCitedbyMap if int(i.get('coverDate')[:4]) in range(year - lastNYear, year + 1)]
19     citedList.sort(reverse=True)
20     return max([min(i + 1, citedList[i]) for i in range(len(citedList))])
21   return None
22
23 result = {}
24 newHIndex = {}
25 newHIndex2 = {}
26 for year in range(1990, 2021):   # year range [1990;2020]
27   result[year] = core.GetAuthorMetrics(
28       year, 7,
29       ScopusAuthorID = LoktevVadimID,
30       ScopusAffilID = KPIID)
31   newHIndex[year] = GetNewIndex(result[year][0].docIDToCitedbyMap)
32   newHIndex2[year] = GetNewIndex2(result[year][0].docIDToCitedbyMap, year)
```

Рисунок 4.10 – Приклад програми для побудови нових індикаторів наукової діяльності автора

Результат виконання програми можна представити у вигляді графіків на

рисунках 4.11 – 4.13.

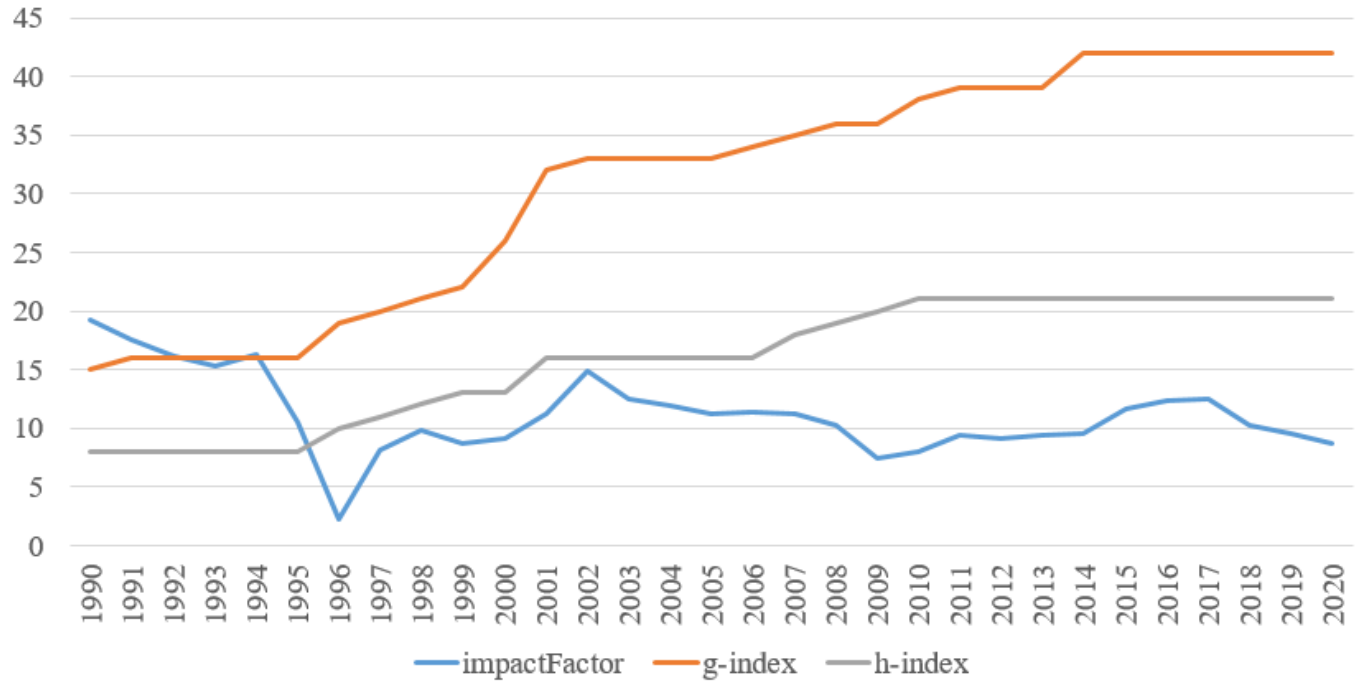

Рисунок 4.11 – Графік зміни індикаторів (індекс Хірша, g-index, фактор впливовості), що розраховуються автоматично за допомогою методу GetAuthorMetrics для автора на проміжку від 1990 до 2020 років

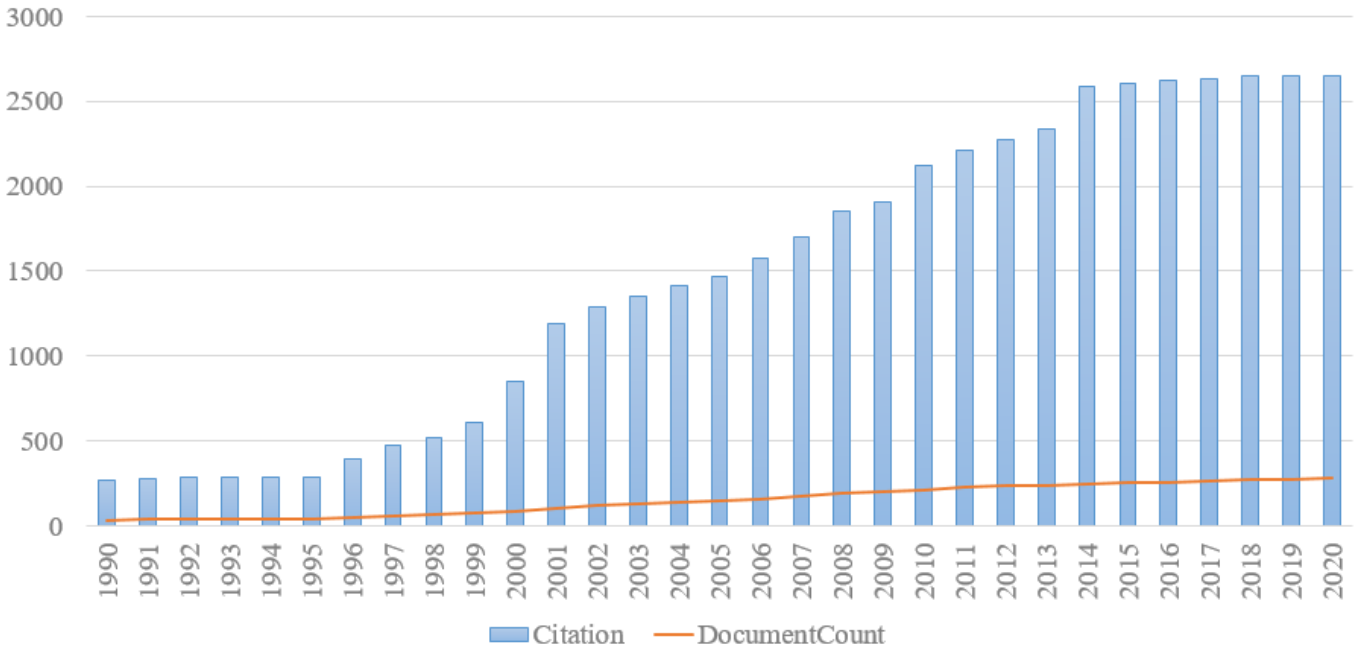

Рисунок 4.12 – Графік співставлення зміни кількості документів до загальної кількості цитувань автора на проміжку від 1990 до 2020 років

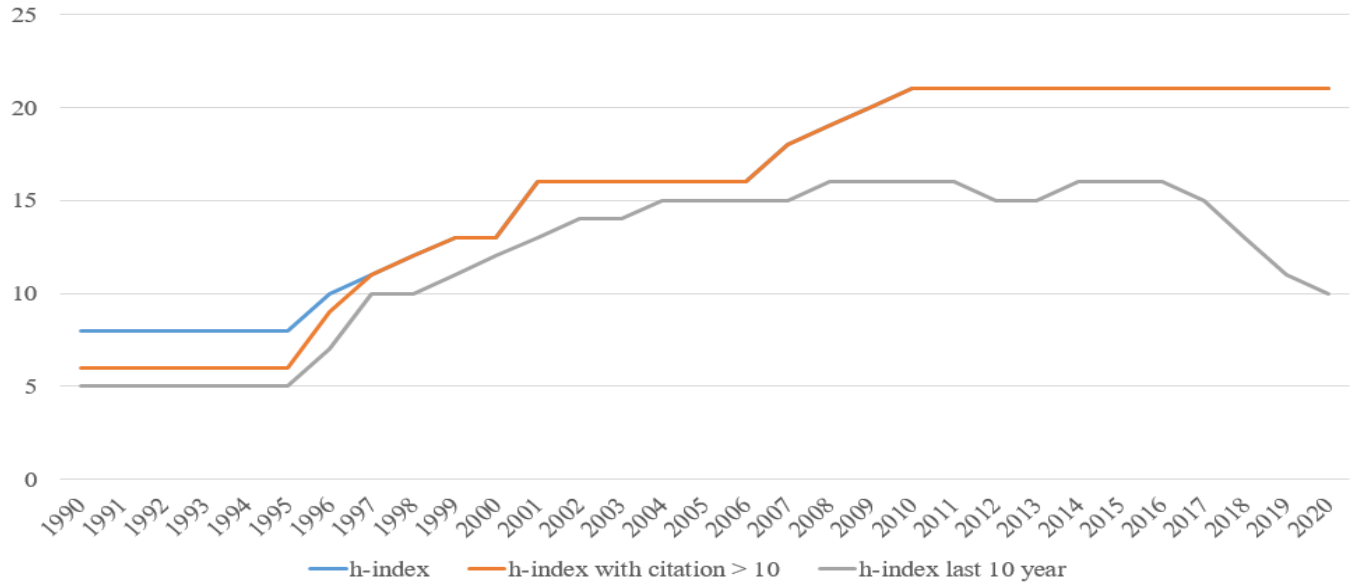

Рисунок 4.13 – Графік співставлення зміни оригінального індексу Хірша та двох запропонованих (рис. 4.10, рядок 8-21) його модифікацій на проміжку від 1990 до 2020 років

Також, на рисунку 4.14 приведений наглядний приклад як змінюється розрахунок фактору впливовості відносно кількості років, що враховується при розрахунку.

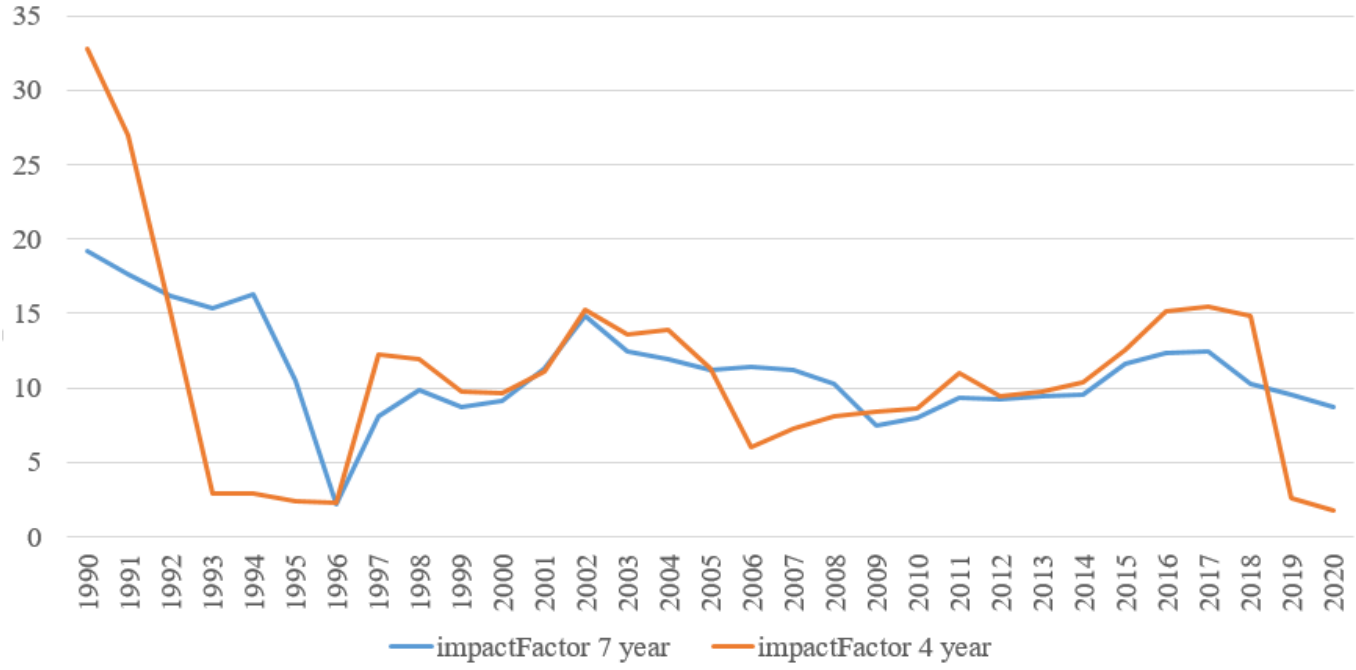

Рисунок 4.14 – Графік співставлення зміни фактору впливовості при врахуванні 7 або 4 останніх років роботи автора на проміжку від 1990 до 2020 років

ВИСНОВКИ

1. Проведено аналіз систем для роботи з бібліографічними даними та основними індикаторами продуктивності наукової діяльності (індекс цитування, індекс Хірша) у пункті 2.3.

2. Проведено пошук доступу до бібліографічними баз даних, описаних в пункті 2.2. Обрано Scopus у якості основного джерела даних через безкоштовний доступ до API інтерфейсів.

3. Розроблено модуль обробки HTTP запитів до Scopus API.

4. Розроблено модуль управління запитами до Search Scopus API.

5. Розроблено модуль управління запитами до Retrieval Scopus API.

6. Створено методи для отримання інформації про авторів, установ та докуметів для ознайомлення, або підготовки вхідних даних для проведення аналізу.

7. Створено метод для підрахунку цільових функцій, описаних в пункті 1.2, за для надання інформації про продуктивність науково технічної діяльності авторів.

8. Реалізовано класифіковане локальне збереження даних для забезпечення швидкості виконання повторних чи суміжних запитів користувача.

9. Додано достатнє логування всіх етапів виконання запитів користувача з метою кращого розуміння можливих помилок розробленого пакету чи інтерфейсів доступу до даних.

10. Наведено приклади використання пакету Core для підготовки вхідних даних та можливих варіантів використання даних користувачем з метою аналізу продуктивності наукової діяльності за допомогою власних експерементальних індикаторів.

Реалізований програмний пакет Core для роботи з бібліографічними даними має великий потенціал до розширення при використанні платних інтерфейсів доступу Scopus та/або Web of Science та використанні алгоритмів об’єднання схожих даних на основі неточного порівняння та кластеризації даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pan, R., Fortunato, S. Author Impact Factor: tracking the dynamics of individual scientific impact. Sci Rep **4,** 4880 (2015). https://doi.org/10.1038/srep04880.

2. Kumar A. D. Research evaluation metrics / Anup Das Kumar., 2015. – 122 с.

3. Scopus Content Coverage Guide. Scopus Info. Elsevier. October 2007. https://www.elsevier.com/solutions/scopus.

4. Vine, Rita (January 2006). Google Scholar. Journal of the Medical Library Association 94 (1): 97–9. PMC 1324783.

5. Chan J. Python API Development Fundamentals / J. Chan, R. Chung, J. Huang., 2019. – 372 с.

6. Idris I. Python Data Analysis / Ivan Idris.. 2014. – 398 с.

ДОДАТОК А

Система автоматизації оцінки наукової діяльності

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сукорського”_ТЕФ_ФПЕПС_ТР62147_20Б

Таблиця 1. Специфікація

Позначення	Найменування	Примітки
Документація		
ПЗ	Пояснювальна записка	Містить обґрунтування методів вирішення проблеми аналізу наукових показників. Містить поради та приклади експлуатації створеного пакету.
Компоненти		
	ScopusSearchHandler	Створення запитів на основі вхідних параметрів до Scopus Search API та збереження отриманих результатів.
	ScopusRetrievalHandler	Створення запитів на основі вхідних параметрів до Scopus Retrieval API та збереження отриманих результатів.
	ScopusRequestProcessor	Виконує HTTP запити до Scopus API.
	Gateway	Надає доступ користувачу до системи через публічні функції.

ДОДАТОК Б

Система автоматизації оцінки наукової діяльності

Лістинг програми

УКР.НТУУ”КПІ імені Ігоря Сукорського”_ТЕФ_ФПЕПС_ТР62147_20Б

Лістинг основного модуля пакету Core – Gateway.

```
# include external dependency
import json
import sys
import os
import threading
import copy
from queue import Queue
from fuzzywuzzy import fuzz
from fuzzywuzzy import process

try:
    from pathlib import Path
except ImportError:
    from pathlib2 import Path

# include internal dependency
from . import LogUtil
from . import Entities
from . import ScopusClient
from . import ScopusSearchHandler
from . import ScopusRetrievalHandler

# Main class for comunication between user and Core package
# Provides 5 main features throuth public methods:
#   GetAffiliationInfo
#   GetAuthorInfo
#   GetAuthorMetrics
#   GetDocumentInfo
#   I_indexForGroup
class Gateway():

  config_file_path = Path('./config.json')

  # config = {
  #   "ScopusApi": {"apikeys": [], "insttoken": ""},
  #   "Gateway": {"logLevel": "", "keysPerThread": ""}
  # }
  def __init__(self, config = None):
    self.logger = LogUtil.GetLogger(__name__)
    self.logger.info('Starting Gateway...')

    if config is None:
      with self.config_file_path.open(mode='r') as config_file:
        self.config = json.load(config_file)
    else:
      self.config = config

    # copy logLevel config param into ScopusApi sub-config
    self.config['ScopusApi']['logLevel'] = self.config['Gateway']['logLevel']
    # create client for no multithreading operation usage
```

```
    self.scopusAPIHandler = ScopusClient.ScopusAPIHandler(self.config['ScopusApi'], self.logger)

    # separate apiKeys to between clients for multithreading usage
    newScopusApiConfig = []
    keyPerThread = int(self.config['Gateway']['keysPerThread'])
    oldKeyNum = len(self.config['ScopusApi']['apikeys'])
    newKeyNumber = int(oldKeyNum / keyPerThread) if oldKeyNum % keyPerThread == 0 else (int(oldKeyNum / keyPerThread) + 1)
    lastUsedID = 0
    for i in range(newKeyNumber):
      newConf = copy.deepcopy(self.config['ScopusApi'])
      newConf['apikeys'] = []
      for j in range(lastUsedID, lastUsedID + min(keyPerThread, oldKeyNum - lastUsedID)):
        newConf['apikeys'].append(self.config['ScopusApi']['apikeys'][j])
        lastUsedID += 1
      newScopusApiConfig.append(newConf)
    self.scopusRetAPIHandler = [ScopusClient.ScopusAPIHandler(newScopusApiConfig[i], self.logger) for i in range(len(newScopusApiConfig))]

    self.scopusRetrievalHandler = ScopusRetrievalHandler.ScopusRetrievalHandler()

    self.logger.info('Gateway is started')

  def __del__(self):
    self.logger.info('Gateway is closed')

  # Build query by given params using API keywords
  # https://dev.elsevier.com/tips/AuthorSearchTips.htm
  # https://dev.elsevier.com/tips/AuthorSearchTips.htm
  def __getAuthorInfoQuery(self, ScopusAuthorID:str = None, AuthorName:str = None, AuthorLastname:str = None,
    ScopusAffilID:str = None, AffilName:str = None, City:str = None, Country:str = None):
    query = ''
    if ScopusAuthorID is not None:
      query = 'AU-ID({})'.format(ScopusAuthorID)
    else:
      if AuthorName is not None:
        query = 'AUTHFIRST({})'.format(AuthorName)
      if AuthorLastname is not None:
        if len(query) != 0:
          query += ' and '
        query += 'AUTHLASTNAME({})'.format(AuthorLastname)
      if ScopusAffilID is not None:
        if len(query) != 0:
          query += ' and '
        query += 'AF-ID({})'.format(ScopusAffilID)
      elif AffilName is not None or City is not None or Country is not None:
        if len(query) != 0:
          query += ' and '
        query += 'AFFIL({})'.format(' and '.join([value for value in [AffilName, City, Country] if value]))
    return query
```

```
  # Get info about authors using query and Scopus Search API
  # Can hide matched by Scopus data if affiliation paramether was added
  #   and discrepancy in inaccurate comparison more than 30%
  # return {'total': int, 'best': []} where int = total number of search result, [] - list of AuthorInfo obj
  def __getAuthorInfoPartData(self, query:str, searchKey:str, TopNMatch:bool = None, start:int = None,
    AffilName:str = None, City:str = None, Country:str = None):
    #get data
    searchByParams = ScopusSearchHandler.ScopusSearchHandler('author', query, topNData = TopNMatch, sort = 'document-count', start = start)
    searchByParams.Execute(self.scopusAPIHandler, isGetAll = True, isDumpData=False, tryToLoadFromCache=False)
    #prepare result
    result = {'total': searchByParams.totalResultsNumber, 'best': []}
    for item in searchByParams.totalResultsData:
      if item["affiliation-current"].get("affiliation-name") is None:
        continue
      affiliationNameMatch = fuzz.WRatio(item["affiliation-current"].get("affiliation-name"), AffilName) if AffilName is not None else 100
      if affiliationNameMatch < 70:
        continue
      affiliationCityMatch = fuzz.WRatio(item["affiliation-current"].get("affiliation-city"), City) if City is not None else 100
      if affiliationCityMatch < 70:
        continue
      affiliationCountryMatch = fuzz.WRatio(item["affiliation-current"].get("affiliation-country"), Country) if Country is not None else 100
      if affiliationCountryMatch < 70:
        continue
      authorInfo = Entities.AuthorInfo()
      authorInfo.scopusID = item["dc:identifier"][10:]
      authorInfo.eID = item.get("eid")
      authorInfo.initials = item["preferred-name"].get("initials")
      authorInfo.firstname = item["preferred-name"].get("indexed-name") or item["preferred-name"].get("given-name")
      authorInfo.lastname = item["preferred-name"].get("surname")
      authorInfo.documentCount = item.get("document-count")
      authorInfo.affilation = Entities.AffiliationInfo()
      authorInfo.affilation.scopusID = item["affiliation-current"].get("affiliation-id")
      authorInfo.affilation.name = item["affiliation-current"].get("affiliation-name")
      authorInfo.affilation.city = item["affiliation-current"].get("affiliation-city")
      authorInfo.affilation.country = item["affiliation-current"].get("affiliation-country")

      authorInfo.searchKey = searchKey
      result['best'].append(authorInfo)
    #print('result best = {} was = {} total = {} start = {}'.format(len(result['best']), len(searchByParams.totalResultsData), searchByParams.totalResultsNumber, start))
    return result

  # Get info about authors using parameters
  # ScopusAuthorID - unique Scopus author ID
  # AuthorName - name of author: firast name or initials; takes a line with one or move value separated by 'or'
```

```
  # AuthorLastname - surname of author; takes a line with one or move value separated by 'or'
  # ScopusAffilID - unique Scopus affiliation ID
  # AffilName - affilation name: full name, part name, abbreviation; takes a line with one
  # City - city of the affiliations
  # Country - country of the affiliations
  # TopNMatch - get top N match data sorted by document count
  def GetAuthorInfo(self, ScopusAuthorID:str = None, AuthorName:str = None, AuthorLastname:str = None,
    ScopusAffilID:str = None, AffilName:str = None, City:str = None, Country:str = None, TopNMatch:int = None):

    searchKey = ' / '.join([value for value in [ScopusAuthorID, AuthorName, AuthorLastname, ScopusAffilID, AffilName, City, Country] if value])
    self.logger.info('Get author info for given key = \'{}\".format(searchKey))

    query = self.__getAuthorInfoQuery(ScopusAuthorID = ScopusAuthorID, AuthorName = AuthorName, AuthorLastname = AuthorLastname,
      ScopusAffilID = ScopusAffilID, AffilName = AffilName, City = City, Country = Country)
    if len(query) == 0:
      self.logger.error('Cannot get author info without search parameters.')
      return None

    getAuthorInfoResult = []
    partData = self.__getAuthorInfoPartData(query, searchKey, TopNMatch, AffilName = AffilName, City = City, Country = Country)
    if TopNMatch is not None and partData['total'] > TopNMatch:#< try to find more data if partData < TopNMatch
      newStart = TopNMatch
      while newStart < partData['total'] or len(partData['best']) != 0:
        #put data to final result list
        getAuthorInfoResult.extend(partData['best'])

        if len(getAuthorInfoResult) < TopNMatch:
          partData = self.__getAuthorInfoPartData(query, searchKey, TopNMatch, start=newStart, AffilName = AffilName, City = City, Country = Country)
          newStart += TopNMatch
        else:
          #return no more than TopNMatch
          getAuthorInfoResult = getAuthorInfoResult[:TopNMatch]
          break
    else:
      getAuthorInfoResult = partData['best']

    sys.stdout.write('\n')
    return getAuthorInfoResult

  # Calculate h-index based on cited list of docs of the author
  def __getHIndex(self, citedbyList):
    if len(citedbyList) != 0:
      citedbyList.sort(reverse=True)
      return max([min(i + 1, citedbyList[i]) for i in range(len(citedbyList))])
    return None
```

```
  # Calculate impact factor based on cited list of docs of the author
  def __getImpactFactor(self, year, impactFactorDelta, docs):
    citedbyCount = 0
    docCount = 0
    for doc in docs:
      if int(doc.get('coverDate')[:4]) in range(year - impactFactorDelta, year):
        citedbyCount += int(doc.get('citedby-count'))
        docCount += 1

    if docCount != 0:
      return citedbyCount / docCount
    return None

  # Calculate g-index based on cited list of docs of the author
  def __getGIndex(self, citedbyList):
    if len(citedbyList) != 0:
      citedbyList.sort(reverse=True)
      return max([(i + 1) for i in range(len(citedbyList)) if sum(citedbyList[:(i + 1)]) >= (i + 1)**2])
    return None

  # Get metrics of authors using parameters
  # Year - consider all information about the author until this year
  # ImpactFactorDelta - how many year consider for impact factor calculation
  # ScopusAuthorID - unique Scopus author ID
  # AuthorName - name of author: first name or initials; line with one or move value separated by 'or'
  # AuthorLastname - surname of author; takes a line with one or move value separated by 'or'
  # ScopusAffilID - unique Scopus affiliation ID
  # AffilName - affilation name: full name, part name, abbreviation; takes a line with one
  # City - city of the affiliations
  # Country - country of the affiliations
  # TopNMatch - get top N match data sorted by document count
  def GetAuthorMetrics(self, Year:int, ImpactFactorDelta:int, ScopusAuthorID:str = None,
AuthorName:str = None, AuthorLastname:str = None,
    ScopusAffilID:str = None, AffilName:str = None, City:str = None, Country:str = None, TopNMatch:int
= None):

    self.logger.info('Get author metrics for given key = \'{}\".format(
      ' / '.join([value for value in [ScopusAuthorID, AuthorName, AuthorLastname, ScopusAffilID,
AffilName, City, Country] if value])))

    # firstly get authors info for given param
    authors = self.GetAuthorInfo(ScopusAuthorID = ScopusAuthorID, AuthorName = AuthorName,
AuthorLastname = AuthorLastname,
      ScopusAffilID = ScopusAffilID, AffilName = AffilName, City = City, Country = Country, TopNMatch
= TopNMatch)

    if authors is None:
      self.logger.error('Cannot get author metrics without author info.')
      return None
```

```
    # download data about documents for each in {authors}
    searchDocsResult = {}
    queue = Queue()
    class Downloader(threading.Thread):
      def __init__(self, queue, client):
        threading.Thread.__init__(self)
        self.queue = queue
        self.client = client

      def run(self):
        while True:
          scopusID = self.queue.get()
          query = 'AU-ID({}) and PUBYEAR < {}'.format(scopusID, Year + 1)
          searchDocs = ScopusSearchHandler.ScopusSearchHandler('scopus', query, sort = 'citedby-count',
fields = ['dc:identifier', 'citedby-count', 'prism:coverDate', 'afid'])
          searchDocs.Execute(self.client, isGetAll = True)
          searchDocsResult[scopusID] = searchDocs.totalResultsData

          self.queue.task_done()
    # init threads
    for i in range(len(self.scopusRetAPIHandler)):
      t = Downloader(queue, self.scopusRetAPIHandler[i])
      t.setDaemon(True)
      t.start()

    # init queue
    for au in authors:
      queue.put(au.scopusID)

    # wait for complition
    queue.join()

    # prepare metrics
    result = []
    for key, value in searchDocsResult.items():
      metric = Entities.AuthorMetrics()
      for au in authors:
        if au.scopusID == key:
          metric.authorInfo = au
          break
      metric.yearInfo = Year
      metric.impactFactorDelta = ImpactFactorDelta
      metric.citedbyCount = 0
      for doc in value:
        del doc['@_fa']
        del doc['prism:url']
        doc['coverDate'] = doc.get('prism:coverDate')
        doc['scopusABSID'] = doc.get('dc:identifier')[10:]
        doc['scopusAFIDs'] = [aff.get('afid') for aff in doc.get('affiliation')] if doc.get('affiliation') else []
        del doc['prism:coverDate']
        del doc['dc:identifier']
        if doc.get('affiliation'):
```

```
          del doc['affiliation']
        metric.citedbyCount += int(doc.get('citedby-count'))
      metric.docIDToCitedbyMap = value
      metric.documentCount = len(metric.docIDToCitedbyMap)
      print(value)

      citedbyList = [int(val.get('citedby-count')) for val in metric.docIDToCitedbyMap]
      metric.h_index = self.__getHIndex(citedbyList)
      metric.g_index = self.__getGIndex(citedbyList)
      metric.impactFactor = self.__getImpactFactor(Year, ImpactFactorDelta, metric.docIDToCitedbyMap)

      result.append(metric)

    sys.stdout.write('\n')
    return result

  # Get info about affiliations using parameters
  # ScopusAffilID - unique Scopus affiliation ID
  # AffilName - affilation name: full name, part name, abbreviation; takes a line with one
  # City - city of the affiliations
  # Country - country of the affiliations
  # TopNMatch - get top N match data sorted by document count
  # https://dev.elsevier.com/tips/AffiliationSearchTips.htm
  # https://dev.elsevier.com/guides/AffiliationSearchViews.htm
  def GetAffiliationInfo(self, ScopusAffilID:str = None, AffilName:str = None, City:str = None, Country:str
= None,
    TopNMatch:int = None):

    self.logger.info('Get affiliation info for given key = \'{}\'".format(
      ' / '.join([value for value in [ScopusAffilID, AffilName, City, Country] if value])))

    if ScopusAffilID is not None:
      query = 'AF-ID({})'.format(ScopusAffilID)
    elif AffilName is not None or City is not None or Country is not None:
      query = 'AFFIL({})'.format(' and '.join([value for value in [AffilName, City, Country] if value]))
    else:
      self.logger.error('Cannot get affiliation info without search parameters.')

    searchByParams = ScopusSearchHandler.ScopusSearchHandler('affiliation', query, topNData =
TopNMatch, sort = 'document-count')
    searchByParams.Execute(self.scopusAPIHandler, isGetAll = True)

    result = []
    searchKey = ' / '.join([value for value in [ScopusAffilID, AffilName, City, Country] if value])
    for item in searchByParams.totalResultsData:
      if item["parent-affiliation-id"] == "0":
        affInfo = Entities.AffiliationInfo()
        affInfo.scopusID = item["dc:identifier"][15:]
        affInfo.eID = item.get("eid")
        affInfo.name = item.get("affiliation-name")
        affInfo.documentCount = item.get("document-count")
        affInfo.city = item.get("city")
```

```
        affInfo.country = item.get("country")
        affInfo.searchKey = searchKey
        result.append(affInfo)

    if AffilName is not None and len(result) != 1:
      newResult = [result[i] for i in range(1, len(result)) if fuzz.WRatio(AffilName, result[i].name) < 80]
      newResult.append(result[0])
      result = newResult
    sys.stdout.write('\n')
    return result

  # Build query by given params using API keywords
  # https://dev.elsevier.com/tips/ScopusSearchTips.htm
  # https://dev.elsevier.com/guides/ScopusSearchViews.htm
  def __getDocInfoQuery(self, Title:str = None, Summary:str = None, Keywords:str = None, Year:int =
None,
    YearBefore:int = None, YearAfter:int = None, ScopusAuthorID:str = None, AuthorName:str = None,
    AuthorLastname:str = None, ScopusAffilID:str = None, AffilName:str = None, City:str = None,
Country:str = None):
    query = ''
    if Title is not None:
      query = 'TITLE-ABS-KEY("{}")'.format(Title)
    if Summary is not None:
      if len(query) != 0:
        query += ' and '
      query += 'TITLE-ABS-KEY({})'.format(Summary)
    if Keywords is not None:
      if len(query) != 0:
        query += ' and '
      query += 'TITLE-ABS-KEY({})'.format(Keywords)

    if Year is not None:
      if len(query) != 0:
        query += ' and '
      query += 'PUBYEAR = {}'.format(Year)
    else:
      if YearBefore is not None:
        if len(query) != 0:
          query += ' and '
        query += 'PUBYEAR < {}'.format(YearBefore)
      if YearAfter is not None:
        if len(query) != 0:
          query += ' and '
        query += 'PUBYEAR > {}'.format(YearAfter)

    if ScopusAuthorID is not None:
      if len(query) != 0:
        query += ' and '
      query += 'AU-ID({})'.format(ScopusAuthorID)
    else:
      if AuthorName is not None:
        if len(query) != 0:
```

```
        query += ' and '
      query += 'AUTHFIRST({})'.format(AuthorName)
    if AuthorLastname is not None:
      if len(query) != 0:
        query += ' and '
      query += 'AUTHLASTNAME({})'.format(AuthorLastname)

  if ScopusAffilID is not None:
    if len(query) != 0:
      query += ' and '
    query += 'AF-ID({})'.format(ScopusAffilID)
  elif AffilName is not None or City is not None or Country is not None:
    if len(query) != 0:
      query += ' and '
    query += 'AFFIL({})'.format(' and '.join([value for value in [AffilName, City, Country] if value]))
  return query

 # Get info about documents using parameters
 # DOI - Document Object Identifier
 # Title - title of a doc
 # Summary - summary of a doc
 # Keywords - keywords of a doc
 # Year - consider all information about docs for this year
 # YearBefore - considef all information in range (inf; YearBefore)
 # YearAfter - considef all information in range (YearAfter; inf)
 # ScopusAuthorID - unique Scopus author ID
 # AuthorName - name of author: firast name or initials; takes a line with one or move value separated by 'or'
 # AuthorLastname - surname of author; takes a line with one or move value separated by 'or'
 # ScopusAffilID - unique Scopus affiliation ID
 # AffilName - affilation name: full name, part name, abbreviation; takes a line with one
 # City - city of the affiliations
 # Country - country of the affiliations
 # TopNMatch - get top N match data sorted by document count
 def GetDocumentInfo(self, DOI:str = None, Title:str = None, Summary:str = None, Keywords:str = None,
   Year:int = None, YearBefore:int = None, YearAfter:int = None, ScopusAuthorID:str = None, AuthorName:str = None,
   AuthorLastname:str = None, ScopusAffilID:str = None, AffilName:str = None, City:str = None, Country:str = None, TopNMatch:int = None):

   self.logger.info('Get document info for given key = \'{}\'".format(
     ' / '.join([str(value) for value in [DOI, Title, Summary, Keywords, Year, YearBefore, YearAfter, ScopusAuthorID,
       AuthorName, AuthorLastname, ScopusAffilID, AffilName, City, Country] if value is not None])))

   if DOI is not None:
     searchResult = [self.scopusRetrievalHandler.GetAbsInfoByDOI(self.scopusAPIHandler, DOI)]
   else:
     query = self.__getDocInfoQuery(Title = Title, Summary = Summary, Keywords = Keywords, Year = Year, YearBefore = YearBefore,
       YearAfter = YearAfter, ScopusAuthorID = ScopusAuthorID, AuthorName = AuthorName, AuthorLastname = AuthorLastname,
```

```
      ScopusAffilID = ScopusAffilID, AffilName = AffilName, City = City, Country = Country)
    if len(query) == 0:
      self.logger.error('Cannot get document info without search parameters.')
      return
    # find scopus ID for all docs with best match by Scopus Search API
    searchByParams = ScopusSearchHandler.ScopusSearchHandler('scopus', query, topNData = TopNMatch, sort = 'citedby-count', fields = ['dc:identifier'])
    searchByParams.Execute(self.scopusAPIHandler, isGetAll = True, isDumpData = False, tryToLoadFromCache = False)
    searchResult = []

    for resData in searchByParams.totalResultsData:
      if resData.get('error') is not None:
        self.logger.error('Cannot get any data with this search key. Please, try again with more detailed key.')
        return []

    # download data by given scopusIDs in searchByParams.totalResultsData using Scopus Retrial API
    queue = Queue()
    class Downloader(threading.Thread):
      def __init__(self, queue, scopusRetrievalHandler, client):
        threading.Thread.__init__(self)
        self.queue = queue
        self.scopusRetrievalHandler = scopusRetrievalHandler
        self.client = client

      def run(self):
        while True:
          scopusID = self.queue.get()
          searchResult.append(self.scopusRetrievalHandler.GetAbsInfoByScopusID(self.client, scopusID))
          self.queue.task_done()

    for i in range(len(self.scopusRetAPIHandler)):
      t = Downloader(queue, self.scopusRetrievalHandler, self.scopusRetAPIHandler[i])
      t.setDaemon(True)
      t.start()

    for item in searchByParams.totalResultsData:
      queue.put(item['dc:identifier'][10:])
    # wait for download
    queue.join()

  result = []
  for item in searchResult:
    docInfo = Entities.DocumentInfo()
    docInfo.scopusID = item['coredata']['dc:identifier'][10:]
    docInfo.eid = item['coredata'].get('eid')
    docInfo.doi = item['coredata'].get('prism:doi')
    docInfo.title = item['coredata'].get('dc:title')
    docInfo.aggregationType = item['coredata'].get('prism:aggregationType')
    docInfo.aggregationName = item['coredata'].get('prism:publicationName')
    docInfo.articleNum = item['coredata'].get('prism:pageRange') or item['coredata'].get('article-number')
    docInfo.coverDate = item['coredata'].get('prism:coverDate')
```

```
    docInfo.citedbyCount = item['coredata'].get('citedby-count')
    docInfo.volume = item['coredata'].get('prism:volume')
    docInfo.description = item['coredata'].get('dc:description')
    docInfo.publisher = item['coredata'].get('dc:publisher')
    docInfo.creatorAuthorID = [au.get('@auid') for au in item['coredata']['dc:creator']['author']]

    docInfo.affiliations = []
    tempAffilList = item['affiliation']
    if not isinstance(tempAffilList, list):
      tempAffilList = [tempAffilList]
    for aff in tempAffilList:
      affInfo = Entities.AffiliationInfo()
      affInfo.scopusID = aff.get('@id')
      affInfo.name = aff.get('affilname')
      affInfo.city = aff.get('affiliation-city')
      affInfo.country = aff.get('affiliation-country')
      docInfo.affiliations.append(affInfo)

    docInfo.authors = []
    tempAuthorList = item['authors']['author']
    if not isinstance(tempAuthorList, list):
      tempAuthorList = [tempAuthorList]
    for au in tempAuthorList:
      auInfo = Entities.AuthorInfo()
      auInfo.scopusID = au.get('@auid')
      auInfo.initials = au.get('ce:initials')
      auInfo.firstname = au.get('ce:indexed-name') or au.get('ce:given-name')
      auInfo.lastname = au.get('ce:surname')
      if au.get('affiliation'):
        auAffilTempList = au['affiliation']
        if not isinstance(auAffilTempList, list):
          auAffilTempList = [auAffilTempList]
        auAffilTempList = [i.get('@id') for i in auAffilTempList]
        auInfo.affilation = [aff for aff in docInfo.affiliations if aff.scopusID in auAffilTempList]
      docInfo.authors.append(auInfo)

    result.append(docInfo)
  sys.stdout.write('\n\n')
  return result

# Calculate I-index based on AuthorMetrics list, need h-index and scopusID for log
def I_indexForGroup(self, authorsMetricsList):
  if not isinstance(authorsMetricsList, list):
    authorsMetricsList = [authorsMetricsList]

  self.logger.info('Get I-index for authors = \'{}\".format(
    ' / '.join([au.authorInfo.scopusID for au in authorsMetricsList])))

  for au in authorsMetricsList:
    if au.h_index is None:
      self.logger.error('Cannot get I-index, because author = \'{}\' hasnt h-index'.format(au.scopusID))
      return None
```

```
    hIndexAuthorsList = [au.h_index for au in authorsMetricsList if au.h_index]
    if len(hIndexAuthorsList) != 0:
      hIndexAuthorsList.sort(reverse=True)
      return max([min(i + 1, hIndexAuthorsList[i]) for i in range(len(hIndexAuthorsList))])
    return None

  # You can try to do domething with Scopus Cited API, need premium account keys
  def GetDocumentCitationInfo(self, ScopusABSID:str):
    searchResult = self.scopusRetrievalHandler.GetAbsCitedInfoByScopusID(self.scopusAPIHandler,
ScopusABSID)
    print(searchResult)
```

ДОДАТОК В

Система автоматизації оцінки наукової діяльності

Опис програмного коду

УКР.НТУУ”КПІ імені Ігоря Сукорського”_ТЕФ_ФПЕПС_ТР62147_20Б

АНОТАЦІЯ

Основний модуль програми складається з 5 публічних та 6 приватних методів. Основна задача – обробка отриманих з Scopus API даних, побудова пошукових запитів на основі переваг, маніпуляція з отриманими даними – підрахунок цільових функцій, які фактично є індикаторами науково-технічної діяльності.

Перевагами цієї системи є прямий доступ до бібліографічних даних, простий інтерфейс, який інкапсулює складну побудову запитів та обробку результатів, легка інтеграція нових цільових функцій.

Подальша робота із зібраними даними спрощує аналітичну роботу при побудові графіків, гістаграм, або створенні та тестуванні своїх цільових функцій.

1. АЛГОРИТМИ ВИКОНАННЯ

Розглянемо алгоритми виконання основних публічних методів по збору та обробці інформації для взаємодії користувача з розробленим пакетом Core.

Алгоритм роботи методу GetAffiliationInfo:

1. Залогувати запис про початок виконання з інформацією вхідних параметрів функції.
2. Створити рядок запиту на основі вхідних параметрів.
3. Створити об’єкт класу ScopusSearchHandler для пошуку інфорації за створеним в пункті 2 запитом.
4. Виконати запит до Scopus Search API.
5. Створити список об’єктів типу AffiliationInfo на основі отриманих у пункті 4 результатів, відкидаючи результати, що не сходяться по непрямим причинам (повернуті дані не актуальні, не відповідають дійсності, фантомні дані).
6. Повернути користувачу створений у пункті 5 список.

Алгоритм роботи методу GetAuthorInfo:

1. Залогувати запис про початок виконання з інформацією вхідних параметрів функції.
2. Створити рядок запиту на основі вхідних параметрів.
3. Створити об’єкт класу ScopusSearchHandler для пошуку інформації за створеним в пункті 2 запитом.
4. Виконати запит до Scopus Search API.
5. Створити список об’єктів типу AuthorInfo на основі отриманих у пункті 4 результатів, відкидаючи результати, що не сходяться по непрямим причинам (повернуті дані не актуальні, не відповідають дійсності, фантомні дані).
6. Якщо кількість елементів у списку відповідає запиту користувача перейти до пункту 7, інакше – до пункту 3.
7. Повернути користувачу створений у пункті 5 список.

Алгоритм роботи методу GetAuthorMetrics:

1. Залогувати запис про початок виконання з інформацією вхідних параметрів функції.
2. Виконати пошук авторів на основі алгоритму GetAuthorInfo.
3. Створити рядок запиту для пошуку документів, на основі ScopusID авторів.
4. Створити об’єкт класу ScopusSearchHandler для пошуку інформації за створеним в пункті 3 запитом.
5. Виконати запит до Scopus Search API.
6. Створити список об’єктів типу AuthorMetricsна основі отриманих у пункті 5 результатів; підрахувати h-index, g-index, загальну кількість цитувань, кількість документів.
7. Повернути користувачу створений у пункті 6 список.

Алгоритм роботи методу GetDocumentInfo:

1. Залогувати запис про початок виконання з інформацією вхідних параметрів функції.
2. Створити рядок запиту на основі вхідних параметрів.
3. Створити об’єкт класу ScopusSearchHandler для пошуку інформації за створеним в пункті 2 запитом.
4. Виконати запит до Scopus Search API.
5. На основі отриманих у пункті 4 ідентифікаторів ScopusID виконати запит до Scopus Retrieval API.
6. Створити список об’єктів типу DocumentInfo на основі отриманих у пункті 5 результатів.
7. Повернути користувачу створений у пункті 6 список.

2. ВХІДНІ ДАННІ

Якість роботи алгоритмів напряму залежить від якості та точності вхідних даних. Найкращим варіантом є використання ScopusID параметрів, тому що так можна однозначно ідентифікувати цілі дослідження.

Перевага розробленого пакету Core полягає у тому, що стартовими можуть бути будь-які данні, оскільки використання методів GetAffiliationInfo, GetAuthorInfo та GetDocumentInfo разом з допоміжними функціями ViewAffiliationInfo, ViewAuthorInfo, ViewDocumentInfo може допомогти конкретизувати вхідні параметри до списку необхідних ідентифікаторів (ScopusID) в цілях побудови індикаторів наукової діяльності.

3. ВИХІДНІ ДАННІ

Результатом роботи GetAffiliationInfo, GetAuthorInfo, GetAuthorMetrics та GetDocumentInfo є список об’єктів відповідного типу даних. Зі структурою AffiliationInfo, AuthorInfo, DocumentInfo та AuthorMetrics можна ознайомитись на рисунку 3.6 та за допомогою результатів виконання прикладів, описаних в пункті 4.