Міністерство освіти і науки України
Національний університет "Одеська політехніка"
Навчально-науковий інститут комп'ютерних систем
Кафедра інженерії програмного забезпечення

Комлева Ганна Олександрівна,
студентка групи АС-171

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Програмна система для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Тройніна Анастасія Сергіївна,
канд. техн. наук, доцент

Одеса – 2022

ЗМІСТ

ЗАВДАННЯ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ 4
АНОТАЦІЯ 6
1 КРИТИЧНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 9
1.1 Огляд структури наукових публікацій 9
1.2 Аналіз потреб користувачів щодо пошуку наукових статей 10
1.3 Аналіз існуючих програмних аналогів 12
1.4 Висновки до розділу 18
2 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ РЕПОЗИТОРІЄМ НАУКОВИХ СТАТЕЙ 19
2.1 Підстава для розробки 19
2.2 Призначення розробки 19
2.3 Побудова правил формування бібліографічних посилань на наукові статті . 20
2.4 Визначення функцій пошуку матеріалів у власному репозиторії 27
2.5 Висновки до розділу 29
3 СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ 30
3.1 Варіанти використання програмної системи 30
3.2 Вимоги до нефункціональних характеристик 43
3.3 Висновки до розділу 45
4.1 Проектування архітектури програмної системи 46
4.3 Структура бази даних 51
4.4 Проектування структури програмних класів 55
4.5 Висновки до розділу 58
5 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЮВАНОЇ СИСТЕМИ 59
5.1 Особливості створення програмних модулів з урахуванням мови програмування 59
5.2 Налаштування середовища для використання API 60
5.3 Реалізація інтерфейсу користувачів системи 61
5.4 Висновки до розділу 62

6 ВИЗНАЧЕННЯ ВЛАСТИВОСТЕЙ ПРОГРАМНОЇ СИСТЕМИ 63
6.1 Тестування механізму генерації посилань 63
6.2 Перевірка часових характеристик роботи системи 65
6.3 Висновки до розділу 67
ВИСНОВКИ 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ 72

Міністерство освіти і науки України

Національний університет "Одеська політехніка"

Навчально-науковий інститут комп'ютерних систем

Кафедра інженерії програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Комлева Н.О.

«___» _____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Комлевої Ганни Олександрівни, група АС-171

1. Тема роботи: Програмна система для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз

Керівник роботи: Тройніна Анастасія Сергіївна, канд. техн. наук, доцент

затверджені наказом ректора від «20» жовтня 2022 р. № 399-в.

2. Зміст роботи: проведення огляду структури наукових публікацій та аналіз потреб користувачів щодо пошуку наукових статей; розробка програмної системи, побудова правил формування бібліографічних посилань; специфікація вимог до системи; проектування системи, визначення архітектури; програмна реалізація системи; визначення властивостей програмної системи.

3. Перелік ілюстративного матеріалу: згідно зі слайдами презентації.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання: «01» вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз існуючих рішень та створення канви ціннісної пропозиції	1.09.2022 – 10.09.2022	вик.
2	Розробка програмної системи	11.09.2022 – 25.09.2022	вик.
3	Специфікація вимог до програмної системи	26.09.2022 – 10.10.2022	вик.
4	Проектування системи	11.10.2022 – 20.10.2022	вик.
5	Програмна реалізація системи	21.10.2022 – 15.11.2022	вик.
6	Визначення властивостей системи	16.11.2022 – 20.11.2022	
7	Оформлення пояснювальної записки та графічного матеріалу	21.11.2020 – 29.11.2021	вик.

Здобувач вищої освіти _____ Г.О. Комлева

Керівник роботи _____ А. С. Тройніна

АНОТАЦІЯ

Метою роботи є скорочення часу на пошук наукових статей, визначення фрагментів для цитування та отримання бібліографічного посилання на статті шляхом створення програмної системи для управління репозиторієм наукових статей, що має доступ до міжнародних наукометричних баз.

Методи розробки базуються на технологіях мов програмування Python та TypeScript, інтегрованому середовищі розробки PyCharm, базі даних MySQL та форматі обміну даними JSON, можливостях HTML та CSS, засобах проектування програмних проектів StarUML та Drawio. Пошук наукових статей здійснюється у репозиторіях міжнародних наукометричних баз Scopus, Web of Science та Index Copernicus.

Ключові слова: наукова стаття, наукометрична база, Python, TypeScript, MySQL, JSON, StarUML, Drawio, Scopus, Web of Science, Index Copernicus.

ABSTRACT

The purpose of the work is to reduce the time spent on searching for scientific articles, identifying fragments for citation and obtaining bibliographic references to articles by creating a software system for managing a repository of scientific articles that has access to international scientometric databases.

Development methods are based on Python and TypeScript programming language technologies, PyCharm integrated development environment, MySQL database and JSON data exchange format, HTML and CSS capabilities, StarUML and Drawio software project design tools. The search for scientific articles is carried out in the repositories of international scientometric databases Scopus, Web of Science and Index Copernicus.

Keywords: scientific article, scientometric database, Python, TypeScript, MySQL, JSON, StarUML, Drawio, Scopus, Web of Science, Index Copernicus.

ВСТУП

Наукова стаття як результат пізнавальної діяльності відіграє важливу роль як при викладанні основ наук, так і при організації дослідницької роботи фахівців, що націлені як на вивчення теоретичних основ певних питань, так і на надбання та поступовий розвиток практичних навичок та вмінь. Сучасний підхід до освіти передбачає тісний зв'язок науки з процесом навчання протягом всього життя. Ця тенденція сприяє вихованню активності, посиленню індивідуальних здібностей, формуванню навичок дослідницької діяльності та вміння застосовувати теоретичні знання на практиці.

Будь-яка наукова стаття повинна бути повноцінним дослідженням, яке підпорядковується певним вимогам стосовно об'єму, особливостей структури, оформлення та переліку літературних джерел. В залежності від специфіки статті вона розміщується в певному журналі чи репозиторії. Після цього стаття може бути включена до міжнародних наукометричних баз з метою покращення розповсюдження ідей, наукової новизни, результатів досліджень та аналітичних висновків, які лежать у її основі.

Якісно створена наукова стаття, яка несе максимальну користь читачеві, повинна відповідати наступним загальним вимогам:

- назва статті повинна відображати основну її ідею, бо найчастіше пошук статті ведеться саме по фрагменту назви;
- чітка структура статті допомагає читачеві швидко зорієнтуватись та оцінити ступінь її корисності, в тому числі доцільно чи ні зберігати статтю у власному репозиторії;
- спеціальні терміни та скорочення повинні бути поясненими та розшифрованими, зловживання великою кількістю скорочень ускладнює сприйняття статті;
- матеріали статті не повинні містити плагіат, цитування без посилань та самоплагіат, лексичні і граматичні помилки, речення повинні містити завершені думки, мати розумну кількість слів тощо.

Всі ці вимоги повинна враховувати розроблювана у даній кваліфікаційній роботі програмна система для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз.

Метою цієї роботи є скорочення часу на пошук наукових статей, визначення фрагментів для цитування та отримання бібліографічного посилання на статті шляхом створення програмної системи для управління репозиторієм наукових статей, що має доступ до міжнародних наукометричних баз.

У першому розділі проведено огляд структури наукових публікацій. Створено канву ціннісної пропозиції. Проведено аналіз існуючих програмних аналогів та визначено загальні вимоги до цієї системи.

У другому розділі формалізовано підставу для розробки, призначення та основні можливості програмної системи. Побудовано правила формування бібліографічних посилань на наукові матеріали для форматів ACM, ACS, APA, ABNT, Chicago, Harvard, IEEE, MLA, Turabian та Vancouver.

Третій розділ містить специфікацію функціональних та нефункціональних вимог до програмної системи.

У четвертому розділі виконано проектування програмної системи з використанням діаграми компонентів. Проведено детальне проектування системи з використанням діаграм послідовностей та діяльності. Розроблена структура бази даних, спроектовано діаграму програмних класів.

П’ятий розділ містить вибір та обґрунтування інструментів та мов розробки, а також приклади інтерфейсу роботи програмної системи.

У шостому розділі виконано тестування генератора посилань для цитування наукових статей. Проведені експерименти по визначенню часу, який витрачається на пошук наукових статей та на формування цитувань.

1 КРИТИЧНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд структури наукових публікацій

Наукова публікація, або стаття, є результатом ретельних наукових досліджень, які завдяки цієї публікації не тільки розповсюджуються у науковому світі, але й отримують верифікацію.Часто у кількості та якості статей вимірюють ефективність наукової роботи певного науковця чи взагалі окремого структурного підрозділу.

Публікація має певну структуру. Ця структура включає:

- назву;
- автора (чи список авторів);
- анотацію – текст, у якому коротко висвітлюються напрямки роботи та основні отримані результати;
- ключові слова;
- основну частину;
- список використаних джерел;
- детальні відомості про авторів.

Основна частина містить постановку мети, аналіз поточного стану вирішення проблеми, розв’язання проблеми з використанням методів, моделей та методик, демонстрацію практичних результатів та висновки.

Зміст наукової статті містить текст, формули та ілюстрації відповідно до прийнятих у виданні вимог. Традиційно статті приймаються у редакцію у форматах doc чи docx. Після узгодження статті та прийняття до публікації вона зазвичай стискається, переводиться у формат pdf та зберігається у репозиторії видавництва саме в такому форматі.

В результаті перетворення у формат pdf можливі спотворення та погіршення якості контенту статті. Особливо це впливає на якість формул та ілюстрацій. Тому в першу чергу вони повинні бути представлені у високій якості для того, щоб читачі мали можливість ознайомлення з науковими матеріалами.

Для читача наукової статті може представляти цінність посилання на неї цілком, або ж на її фрагмент, у власній публікації. При оформленні посилань можна вказувати номер джерела у списку літератури, або ж крім номера вказувати номери певних сторінок. Зазвичай посилання на джерело (в обох випадках) приводиться наприкінці речення у науковій статті. Посилання робляться тільки на ті праці, які є завершеними та опублікованими. Всі наукові джерела, на які були зроблені посилання у тексті статті, повинні бути приведеними наприкінці статті у списку літератури. В залежності від вимог список літератури може включати від декількох посилань до декількох десятків посилань.

Застосування запозичених матеріалів без відповідних посилань є неприпустимим та класифікується як прояв академічної недоброчесності. До розповсюджених варіантів академічної недоброчесності відносять списування, при якому авторський текст використовується без посилання на автора, та плагіат, при якому авторські ідеї будь-яка особа приписує як власні.

«Цитата береться в лапки і обов’язково має посилання на джерело із зазначеним номером сторінки.

Цитати звичайно наводять:

- для підтвердження власних аргументів;
- як посилання на авторитетне джерело;
- для критичного аналізу того чи іншого твердження.

В інших випадках краще робити непряме цитування.» [1]

Джерела у списку літератури можна розміщувати одним з двох способів: у тому порядку, як посилання з’являються у тексті наукової статті, або в алфавітному порядку прізвищ авторів статей.

1.2 Аналіз потреб користувачів щодо пошуку наукових статей

При пошуку наукової статті користувач стикається з різними способами доступу до матеріалів статті: підписка чи відкритий доступ.

Якщо науковий матеріал опублікований у журналі, що розповсюджується за підпискою (paywall), то для доступу до самої статті потрібно внести оплату, яка може досягати десятків доларів. Крім того, підписка працює тільки з корпоративних IP-адрес університету, що унеможливлює доступ до матеріалів з інших місць перебування науковців-читачів.

При публікації матеріалу у режимі відкритого доступа статтю оплачує той, хто її публікує, тобто автор (чи колектив авторів). Ціна оплати може досягати тисяч доларів, але така форма публікації дозволяє всім бажаючим прочитати статтю та створити на неї посилання.

Все це безумовно впливає на проблему пошуку наукової статті.

«На сьогоднішній день існує велика кількість міжнародних систем цитування (бібліографічних баз): Web of Science, Scopus, Index Copernicus, Astrophysics, PubMed, Mathematics, Chemical Abstracts, Springer, Agris, GeoRef. Найавторитетнішими з них, індекси яких визнаються в усьому світі, є «Web of Science» і «Scopus». [2]

Будь-яке програмне забезпечення реалізує певні, зазначені майбутніми користувачами, бізнес-вимоги. Отже, для логічного обґрунтування розробки програмного забезпечення доцільно створювати бізнес-модель. Саме бізнес-модель відповідає за те, як саме буде працювати система бізнесу, що забезпечується розроблюваним програмним продуктом.

Універсальним алгоритмом запуску та налаштування реалізації типових завдань є канва ціннісної пропозиції [3].

Розглянемо канву ціннісної пропозиції для процеса та відповідного програмного продукту для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз (рис. 1.1).

Для побудови канви потрібно створити шість блоків та рознести їх на дві частини: клієнтський профіль та карту цінності.

Клієнтський профіль розроблюваної програмної системи містить:

− клієнтські завдання – «Визначення параметрів пошуку», «Обробка результатів»;

Рисунок 1.1 – Канва ціннісної пропозиції

– проблеми клієнтів – «Важко відсіювати не релевантні статті», «Проблеми сприйняття англомовних анотацій статей»;

– вигоди для клієнта – «Зберегти час на пошук статей», «Отримати посилання у потрібному форматі».

Карта цінності містить наступні елементи:

– продукти та послуги – «Програмна система для пошуку наукових статей»;

– знеболювальні – «Автоматизація пошуку у міжнародних наукометричних базах»;

– створення переваг – «Використання API баз та структурування даних по науковим статтям».

1.3 Аналіз існуючих програмних аналогів

Система BASE (Bielefeld Academic Search Engine) є однією з найбільш розповсюджених пошукових систем, націлених на пошук наукових статей, які є у вільному доступі у мережі Інтернет. За результатами пошуку користувачі можуть отримати анотації статей та бібліографічні метадані, однак повнотекстові статті не надаються. На рис. 1.2 показано приклад роботи системи BASE для пошуку за

запитом «time series forecasting» («прогнозування часових рядів»). За результатами пошуку надається назва статті, автор, тип документу, дані щодо цитування. Результати запиту можна фільтрувати за ріком публікації, мовою, типом документу та доступу до документу та ін. Суттєвим недоліком є те, що отримані результати неможливо інтегрувати у власний репозиторій та використовувати багаторазово.

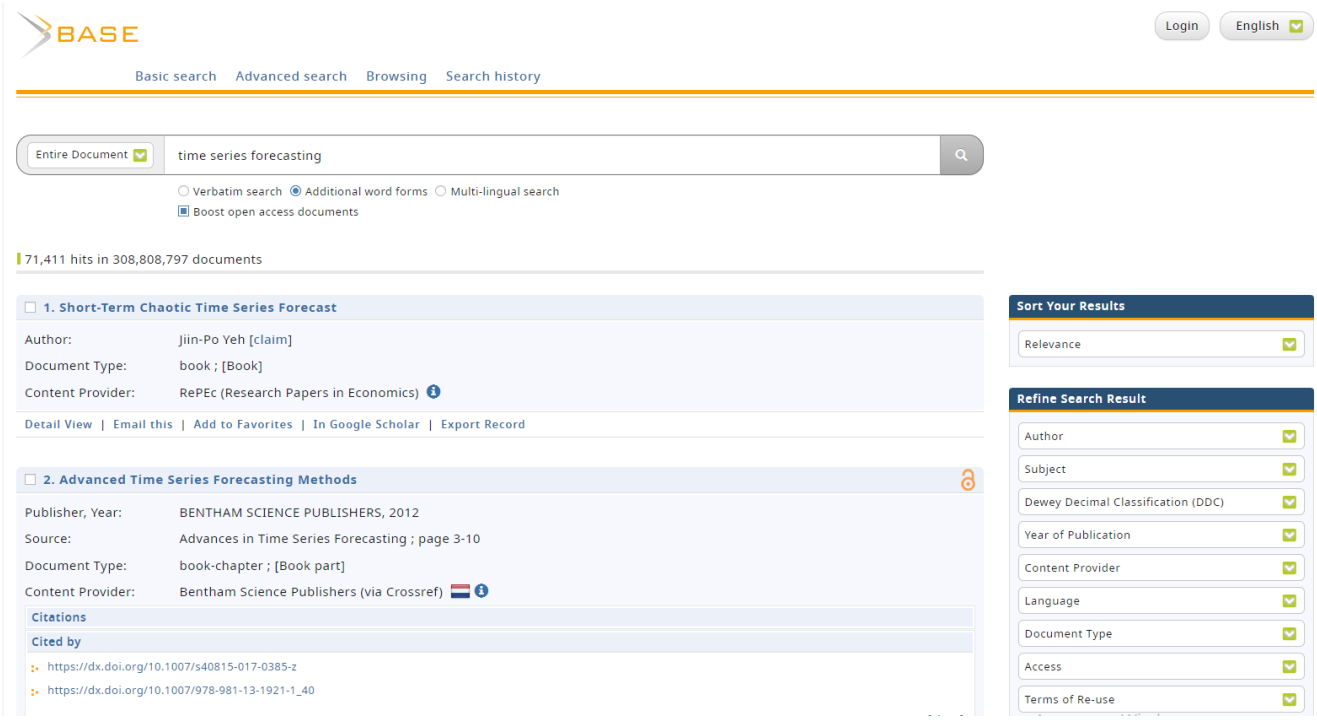

Рисунок 1.2 – Приклад використання пошукової системи BASE

Міжнародна наукометрична база Scopus є найбільш відомою базою цитування наукових статей і матеріалів [4]. Інтерфейс системи підтримує багато режимів пошуку: за тематикою, за назвою статті, автором, ріком випуску статті і т.д. Крім того, статті можна переглядати з упорядкуванням індексу цитування. На рис. 1.3 показано приклад відображення категорій статей в порядку спадання індексу цитування (індекс Гірша) статей, що входять в ту чи іншу категорію. База Scopus містить велику кількість наукових матеріалів, в тому числі і у відкритому доступі. Крім того, існують функції API для можливості взаємодії зовнішніх застосувань з базою Scopus, що є дуже корисним та повинно бути використано у власній розробці для забезпечення пошуку статей у цій базі.

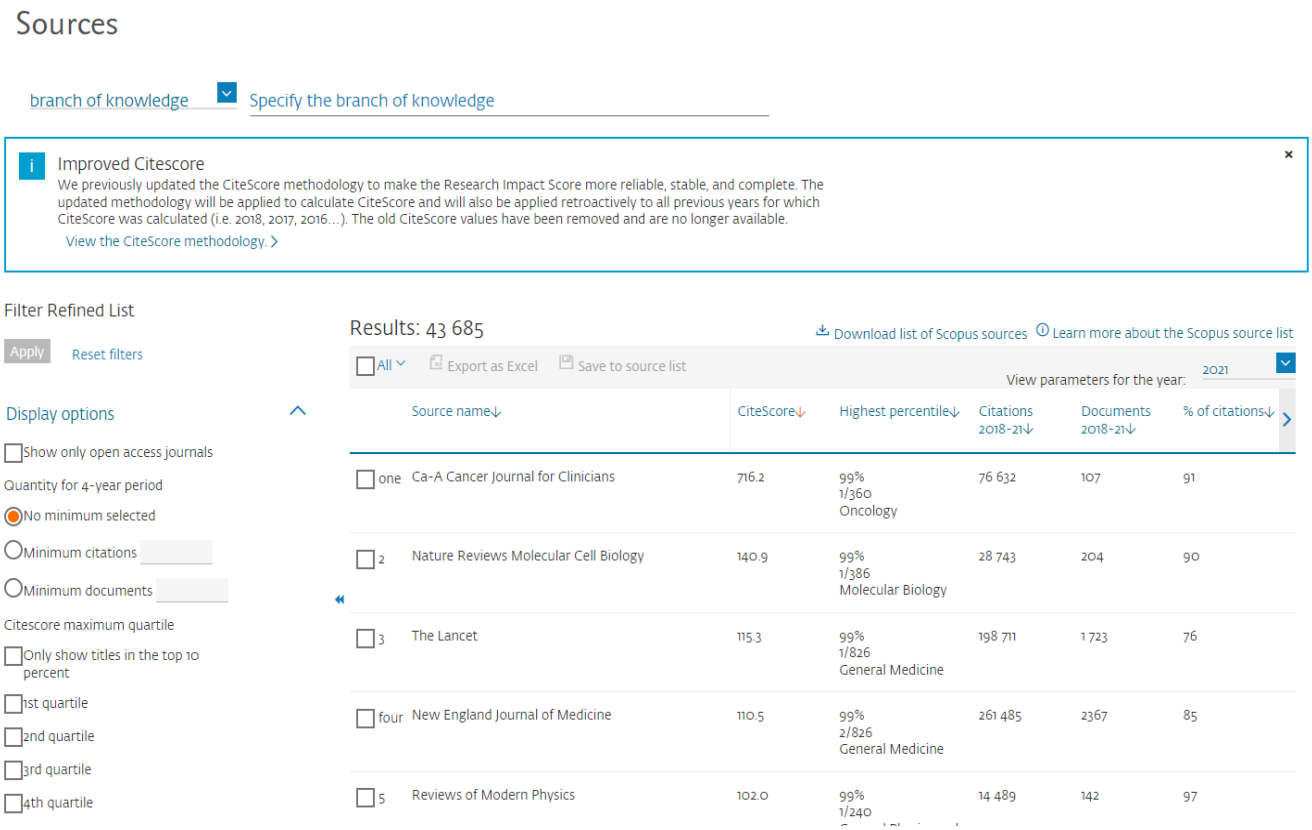

Рисунок 1.3 – Приклад роботи з науковою базою Scopus

Іншою розповсюдженою міжнародною наукометричною базою є Web of Science Core Collection [5]. На цій платформі розміщуються як наукові статті, так і патенти на винахід. Для доступу до переліку матеріалів потрібна реєстрація, після цього матеріали доступни за підпискою. Можливі два варіанти реєстрації: як окрема фізична особа чи як член організації.

Користувачі платформи Web of Science (WoS) можуть виконувати пошук наукових матеріалів за назвою, автором, року чи діапазону років публікації, користуватись реферативними базами. В основі роботи WoS лежить обчислення індексу цитування, який розраховується виходячи з кількості цитувань наукових статей, книг, рецензованих журналів.

Платформа WoS містить багато вбудованих інструментів для аналізу рівня цитувань та рейтингу публікацій, наприклад, JSR для збору імпакт-факторів, ESI для підрахунку значущості публікацій, InCites для аналізу взаємозв’язку між дослідженнями. Так же і Scopus, платформа WoS має API для отримання з неї інформації.

На рис. 1.4 наведений приклад пошуку на платформі WoS публікацій за запитом «software engineering» («програмна інженерія»). Входження слів, які задані як ключові, підсвічується жовтим кольором у назвах та анотаціях публікацій.

Поряд з кожним науковим матеріалом вказується кількість його цитувань та кількість посилань на літературні джерела.

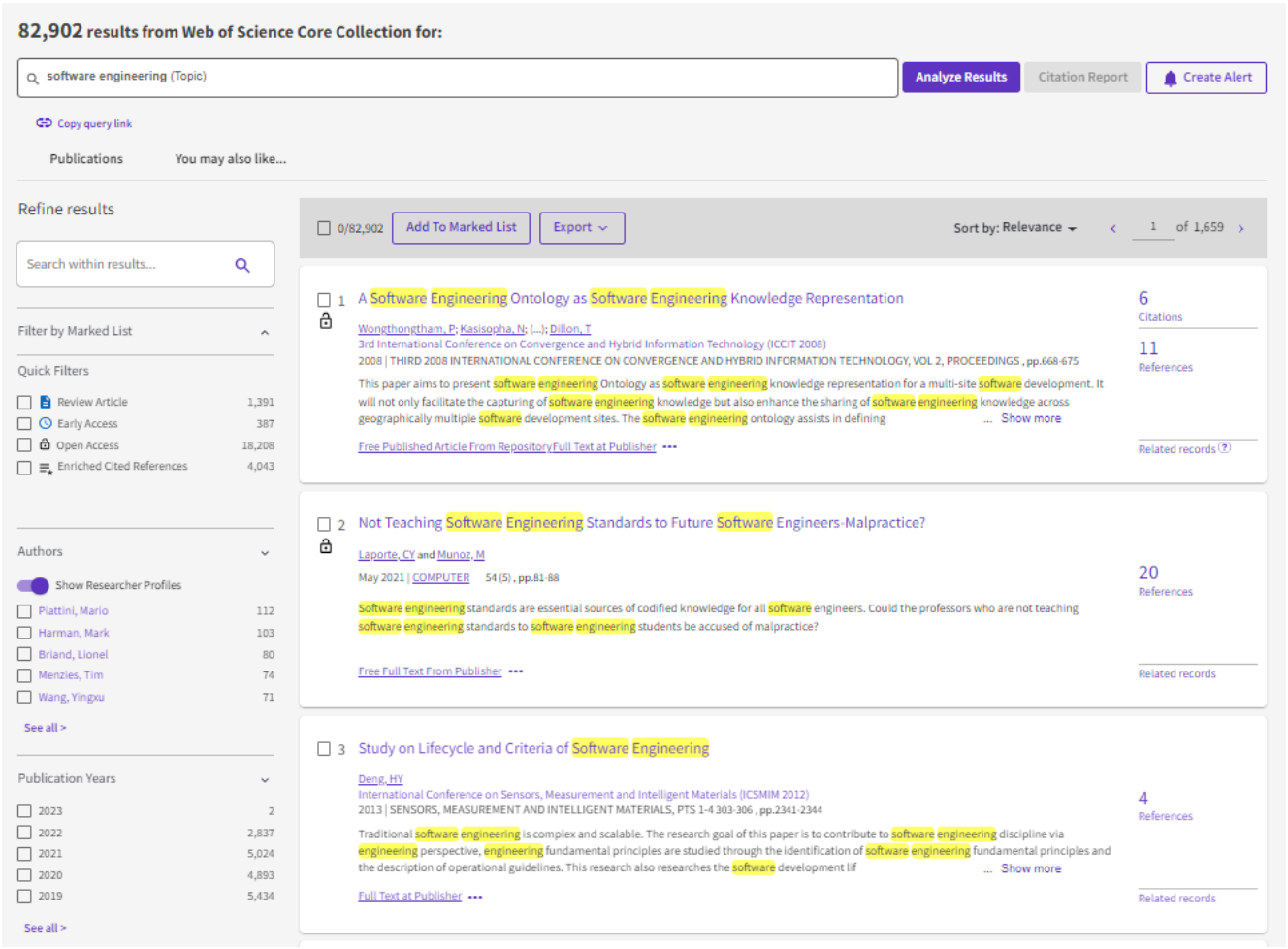

Рисунок 1.4 – Результати пошуку матеріалів на платформі Web of Science

Ще однією з розповсюджених наукометричних наукових баз є польська база Index Copernicus [6]. Вона також містить наукові матеріали та інформацію з друкованих видань та проектів. Index Copernicus вміє обчислювати продуктивність та наукометричні індекси матеріалів та відстежувати вплив на науковий розвиток науково-технічних установ (рис. 1.5). У базу Index Copernicus інтегровано багато оцінок продуктивності та наукового розвитку.

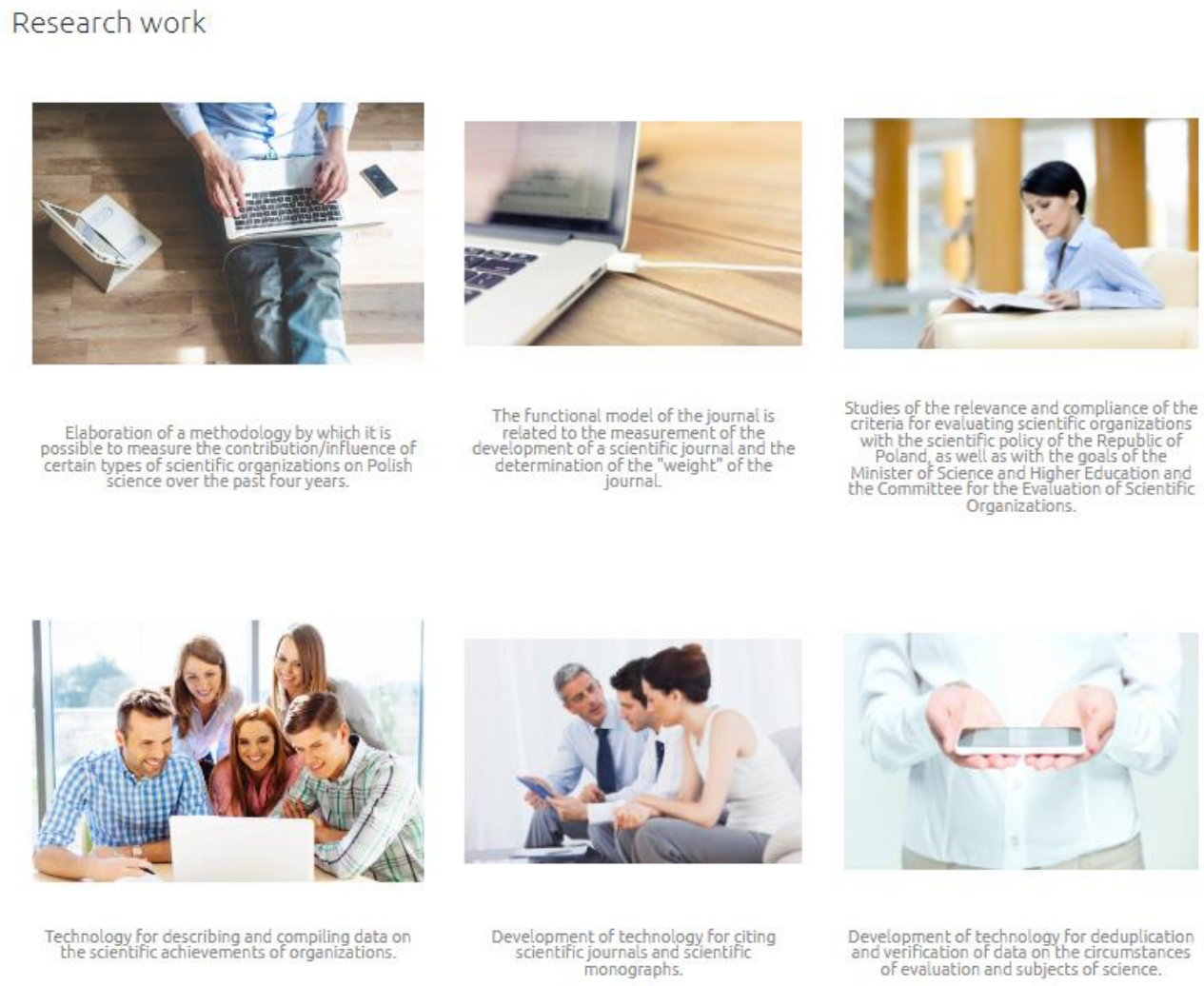

Рисунок 1.5 – Приклад роботи з польською базою Index Copernicus

Наприкінці розглянемо облікову систему Fosslook, що працює з елементами програмного забезпечення, у тому числі текстовими файлами [7]. На рис. 1.6 показана робота з договорами щодо створення програмного забезпечення у якості елементів. Але працювати можна з будь-якими текстовими файлами, в тому числі і науковими статтями.

Як можна побачити, облікова система Fosslook має застарілий інтерфейс. Вміст інформаційного елементу відображується у системі під назвою «картка». Позитивним моментом системи Fosslook є активний журнал версій, використання якого пояснює які зміни щодо списку програмного забезпечення були зроблені та хто є відповідальним за ці зміни. Кожен елемент має актуальний номер, автора версії та номера версії.

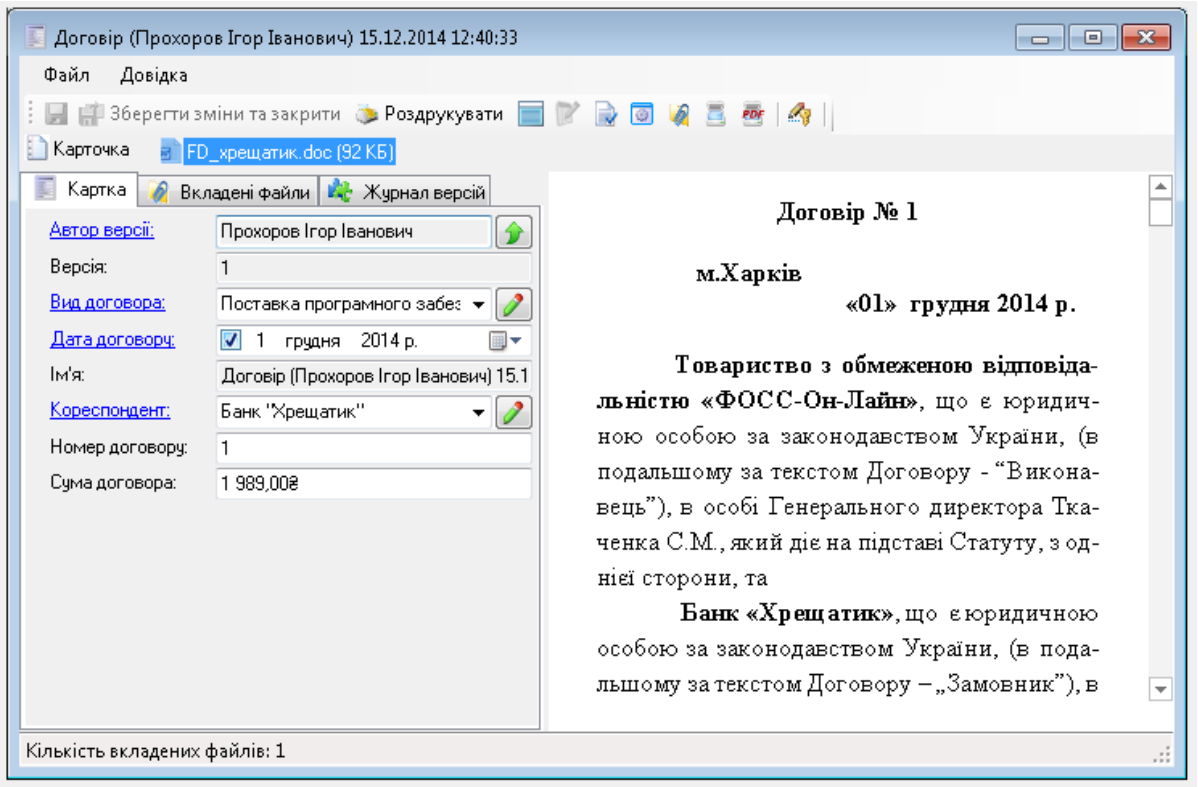

Рисунок 1.6 – Приклад використання облікової системи Fosslook

Результати аналізу програмних систем для управління репозиторієм наукових статей наведені у табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика програмних систем та баз

Назва продукту	Властивості продукту				
	Можливість роботи з науковими матеріалами	Можливість зберігання у власному репозиторії	Можливість визначення фрагментів для цитування	Можливість генерації посилання	Сучасний інтерфейс
1	2	3	4	5	6
BASE	+	-	-	+	+
Scopus	+	-	-	+	+
Web of Science	+	-	-	+	+

Продовження таблиці 1.1

1	2	3	4	5	6
Index Copernicus	+	-	-	+	+
Fosslook	+	+	+	-	-
Власна система	+	+	+	+	+

Аналіз характеристик аналогів дає підставу вважати розробку власної програмної системи актуальним завданням.

1.4 Висновки до розділу

У першому розділі проведено огляд структури наукових публікацій. Підкреслена важливість правильного оформлення цитувань статей та їх фрагментів. Проведено аналіз потреб користувачів щодо пошуку наукових статей.

Створено канву ціннісної пропозиції, що має частини клієнтського профілю та карти цінності.

Проведено аналіз існуючих програмних аналогів: системи BASE, наукометричних баз Scopus та Index Copernicus, платформи Web of Science та облікової системи Fosslook. За результатами аналізу зроблено висновок про актуальність розробки власної система управління репозиторієм.

2 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ РЕПОЗИТОРІЄМ НАУКОВИХ СТАТЕЙ

2.1 Підстава для розробки

Підставою розробки є завдання на кваліфікаційну роботу магістра. Темою роботи є «Програмна система для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз».

2.2 Призначення розробки

Призначенням розробки є створення програмної системи, яка дозволяє користувачеві керувати власним репозиторієм наукових статей, в тому числі додавати в репозиторій нові статті з пошуком їх у міжнародних наукометричних базах, а також редагувати список статей. Потрібно зазначити, що у репозиторій можуть входити статті будь-яких авторів. Для статей відкритих для доступу, у репозиторій можуть включатись повні тексти статті, в іншому випадку – вихідні дані статті та анотація.

Програмна система повинна надавати користувачеві наступні можливості:

- здійснювати пошук наукової статті у міжнародних наукометричних базах;
- додавати знайдену статтю до власного репозиторію;
- здійснювати пошук статті у власному репозиторії та керувати списком статей у репозиторії;
- дозволяти користувачеві автоматично отримувати бібліографічне посилання відповідно до певного формату (ACM, ACS, APA, ABNT, Chicago, Harvard, IEEE, MLA, Turabian, Vancouver) на статтю, яка присутня у власнім репозиторії;
- керувати списком фрагментів наукових статей, присутніх у власнім репозиторії, для цитування, та допомагати додаванню потрібної цитати у певну наукову статтю;
- дозволяти користувачеві визначати статистичні характеристики цитувань:

абсолютні статистичні, відносні статистичні та середні величини.

2.3 Побудова правил формування бібліографічних посилань на наукові статті

В залежності від вимог посилання на наукову статтю може бути створено відповідно до різних форматів.

Визначимо правила Rules формування бібліографічних посилань Ref в залежності від обраного формату F:

Rules = {R_A, R_S, R_P, R_T, R_C, R_H, R_I, R_M, R_U, R_V}

F = «ACM» | «ACS» | «APA» | «ABNT» | «Chicago» | «Harvard» | «IEEE» | «MLA» | «Turabian» | «Vancouver»

Наведемо формалізацію елементів R_A, R_S, R_P, R_T, R_C, R_H, R_I, R_M, R_T, R_V в залежності від затребуваного в публікації формату.

Формат «ACM»:

R_A: if (F=«ACM») then Ref = $П_1$ «, » I_1 O_1 [«, » $П_i$ «, » I_i O_i] і $П_N$ «, » I_N O_N «. » Year «. » Name «. » Journal «. » Volume «, » Number «(» Month « » Year «), » P_{begin} «-» P_{end} «.»

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Year – рік публікації у форматі хххх;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі х;

Number – номер журналу у форматі х;

Month – три перші букви назви місяця виходу журнала;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка.

Формат «ACS»:

R_S: if (F=«ACS») then Ref = $П_1$ «, » I_1 «. » O_1 «.» [«; » $П_i$ «, » I_i «. » O_i «.»] і $П_N$ «, » I_N «. » O_N «. » Name «. » ArticleLink Year «, » Volume «, » P_{begin} «-» P_{end} «.»,

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Name – назва наукової статті;

ArticleLink – посилання на публікацію;

Year – рік публікації у форматі хххх;

Volume – том журналу у форматі х;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка.

Формат «APA»:

R_P: if (F=«APA») then Ref = $П_1$ «, » I_1 «. » O_1 «.» [«, » $П_i$ «, » I_i «. » O_i] & $П_N$ «, » I_N «. » O_N «. » «(»Year «). » Name «. » Journal «, » Volume «, » Number «(» Month « » Year «), » P_{begin} «-» P_{end} «.» «вилучено із » ArticleLink

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Year – рік публікації у форматі хххх;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі х;

Number – номер журналу у форматі х;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію.

Формат «ABNT»:

R_T: if (F=«ABNT») then Ref = $П_1$ «, » $І_1$ «. » $О_1$ «.» [«; » $П_і$ «, » $І_і$ «. » $О_і$] «; » $П_N$ «, » $І_N$ «. » $О_N$ «. » Name «. » Journal «, [S. » Series «.], » «v. » Volume «, n. » Number «, p. » P_{begin} «-» P_{end} «, » Year «. » «Disponível em: » ArticleLink «. Acesso em: » DayAccess « » MonthAccess «. » YearAccess

де $П_1$ – прізвище першого автора, великими літерами, $І_1$ та $О_1$ – перші літери імені та по-батькові першого автора;

$П_і$ – прізвище не першого та не останнього авторів, великими літерами, $І_і$ та $О_і$ – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, великими літерами, $І_N$ та $О_N$ – перші літери імені та по-батькові останнього автора;

Name – назва наукової статті;

Journal – назва журналу;

Series – номер серії;

Volume – том журналу у форматі х;

Number – номер журналу у форматі х;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

Year – рік публікації у форматі хххх;

ArticleLink – посилання на публікацію;

DayAccess – дата звернення, день у форматі х;

MonthAccess – дата звернення, місяць у рядковому форматі xxx;

YearAccess – дата звернення, рік у форматі xxxx.

Формат «Chicago»:

R_C: if (F=«Chicago») then Ref = $П_1$ «, » I_1 O_1 «.» [«, » $П_i$ «, » I_i O_i] «, i » $П_N$ «, » I_N O_N «. » Year «. «» Name «». » Journal « » Volume « (» Number «): » P_{begin} «-» P_{end} «. » ArticleLink «.»

де $П_1$ та I_1 – прізвище та ім'я першого автора, O_1 – перша літера по-батькові першого автора;

$П_i$ та I_i – прізвище та ім'я не першого та не останнього авторів, O_i – перша літера по-батькові не першого та не останнього авторів;

$П_N$ та I_N – прізвище та ім'я останнього автора, O_N – перша літера по-батькові останнього автора;

Year – рік публікації у форматі xxxx;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі x;

Number – номер журналу у форматі x;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію.

Формат «Harvard»:

R_H: if (F=«Harvard») then Ref = $П_1$ «, » I_1 O_1 «.» [«, » $П_i$ «, » I_i O_i] «i » $П_N$ «, » I_N O_N «. » Year «. «» Name «», » Journal « » Volume « (» Number «), с. » P_{begin} «-» P_{end} «. доступний у: » ArticleLink «(дата звернення: » DayAccess « » MonthAccess «. » YearAccess «).»

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Year – рік публікації у форматі хххх;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі х;

Number – номер журналу у форматі х;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію;

DayAccess – дата звернення, день у форматі х;

MonthAccess – дата звернення, місяць у рядковому форматі;

YearAccess – дата звернення, рік у форматі хххх.

Формат «IEEE»:

R_I: if (F=« IEEE») then Ref = I_1 «. » O_1 «. » $П_1$ [«, » I_i «. » O_i «. » $П_i$] «, » «i » I_N «. » O_N «. » $П_N$ «, » ««Name», » ArticleLink «, » «том » Volume «, вип. » Number «, с. » P_{begin} «-» P_{end} «, » MonthAccess YearAccess «.»

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Name – назва наукової статті;

ArticleLink – посилання на публікацію;

Volume – том журналу у форматі х;

Number – номер журналу у форматі х;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

MonthAccess – дата звернення, місяць у рядковому форматі xxx;

YearAccess – дата звернення, рік у форматі xxxx.

Формат «MLA»:

R_M: if (F=« MLA») then Ref = $П_1$ «, » I_1 «. » O_1 «.» [«, » $П_i$ «, » I_i «. » O_i] «, і » $П_N$ «, » I_N «. » O_N «. » ««Name». » Journal «, » «том » Volume «, вип. » Number «, » MonthAccess YearAccess «, с. » P_{begin} «-» P_{end} «, » ArticleLink «.»

де $П_1$ – прізвище першого автора, I_1 та O_1 – перші літери імені та по-батькові першого автора;

$П_i$ – прізвище не першого та не останнього авторів, I_i та O_i – перші літери імені та по-батькові не першого та не останнього авторів;

$П_N$ – прізвище останнього автора, I_N та O_N – перші літери імені та по-батькові останнього автора;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі x;

Number – номер журналу у форматі x;

MonthAccess – дата звернення, місяць у рядковому форматі;

YearAccess – дата звернення, рік у форматі xxxx;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію.

Формат «Turabian»:

R_U: if (F=« Turabian») then Ref = $П_1$ «, » I_1 O_1 «.» [«, » $П_i$ «, » I_i O_i «.»] «, і » $П_N$ «, » I_N O_N «. » ««Name». » Journal Volume «, no. » Number « (» Month « » Day «, » Year «): » P_{begin} «-» P_{end} «, дата звернення » MonthAccess « » DayAccess «, » YearAccess «, » ArticleLink «.»

де $П_1$ та $І_1$ – прізвище та ім’я першого автора, $О_1$ – перша літера по-батькові першого автора;

$П_i$ та $І_i$ – прізвище та ім’я не першого та не останнього авторів, $О_i$ – перша літера по-батькові не першого та не останнього авторів;

$П_N$ та $І_N$ – прізвище та ім’я останнього автора, $О_N$ – перша літера по-батькові останнього автора;

Name – назва наукової статті;

Journal – назва журналу;

Volume – том журналу у форматі x;

Number – номер журналу у форматі x;

Month – дата публікації, місяць у рядковому форматі;

Day – день публікації, у форматі x;

Year – рік публікації, у форматі xxxx;

MonthAccess – дата звернення, місяць у рядковому форматі;

DayAccess – дата звернення, день у форматі x;

YearAccess – дата звернення, рік у форматі xxxx;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію.

Формат «Vancouver»:

R_V: if (F=«Vancouver») then Ref = $П_1$ « » $І_1$ $О_1$ [«, » $П_i$ « » $І_i$ $О_i$] «, » $П_N$ « » $І_N$ $О_N$ «. » Name «. » JournalLink «[інтернет]. » DayAccess «, » MonthAccess YearAccess « [цит. за » Day «, » Month « » Year «]; » Volume «(» Number «): » P_{begin} «-» P_{end} «, доступний у: » ArticleLink

де $П_1$ та $І_1$ – прізвище та ім’я першого автора, $О_1$ – перша літера по-батькові першого автора;

$П_i$ та $І_i$ – прізвище та ім’я не першого та не останнього авторів, $О_i$ – перша літера по-батькові не першого та не останнього авторів;

$П_N$ та $І_N$ – прізвище та ім’я останнього автора, $О_N$ – перша літера по-батькові останнього автора;

Name – назва наукової статті;

JournalLink – посилання на сайт журналу;

DayAccess – дата звернення, день у форматі x;

MonthAccess – дата звернення, місяць у рядковому форматі;

YearAccess – дата звернення, рік у форматі xxxx;

Day – день публікації, у форматі x;

Month – дата публікації, місяць у рядковому форматі;

Year – рік публікації, у форматі xxxx;

Volume – том журналу у форматі x;

Number – номер журналу у форматі x;

P_{begin} – початкова сторінка;

P_{end} – кінцева сторінка;

ArticleLink – посилання на публікацію.

2.4 Визначення функцій пошуку матеріалів у власному репозиторії

Для пошуку матеріалів у власному репозиторії визначені наступні функції:

max(article_1, article_2, …, article_n) – повертає статтю чи набір статей, для яких найчастіше створювались посилання, з групи визначених у аргументах статей;

min(article_1, article_2, …, article_n) – повертає статтю чи набір статей, для яких створювалось найменше посилань, з групи визначених у аргументах статей;

count() – повертає загальну кількість статей у репозиторії;

count(article) – повертає кількість фрагментів для коментування, визначених у статті article;

count_with_comment() – повертає загальну кількість статей, які мають фрагменти для коментування;

count_with_comment(k) – повертає загальну кількість статей, які мають k фрагментів для коментування (ця ж функція дозволяє визначити статті, які не мають жодного фрагмента для коментування);

get_comment(article, k) – повертає визначений фрагмент для цитування під номером k зі статті article;

length(article, k) – повертає кількість символів, яку містить фрагмент для цитування під номером k зі статті article;

get_comment(article, string) – повертає фрагмент для цитування, в який входить підрядок string зі статті article;

length(article, string) – повертає кількість символів, яку містить фрагмент для цитування, в який входить підрядок string зі статті article;

get_date(date_1, date_2) – повертає статті, що були завантажені у репозиторій у діапазоні від date_1 до date_2;

get_before(date) – повертає статті, що були завантажені у репозиторій до вказаної дати date;

get_after(date) – повертає статті, що були завантажені у репозиторій після вказаної дати date;

get_date_comment(article, date_1, date_2) – повертає фрагмент чи набір фрагментів для цитування, які були створені у діапазоні від date_1 до date_2 для статті article;

get_before_date_comment(article, date) – повертає фрагмент чи набір фрагментів для цитування, які були створені до вказаної дати date для статті article;

get_after_date_comment(article, date) – повертає фрагмент чи набір фрагментів для цитування, які були створені після вказаної дати date для статті article;

equal_comment(k_1, k_2) – перевіряє чи співпадають збережені у репозиторії фрагменти для цитування k_1 та k_2;

isin_comment(k_1, k_2) – перевіряє чи входить збережений у репозиторії фрагмент для цитування k_1 в k_2;

is_empty(k) – перевіряє чи є фрагмент для цитування k порожнім.

2.5 Висновки до розділу

У другому розділі формалізовано підставу для розробки, призначення та основні можливості програмної системи.

Побудовано правила формування бібліографічних посилань на наукові матеріали для форматів ACM, ACS, APA, ABNT, Chicago, Harvard, IEEE, MLA, Turabian та Vancouver. Це потрібно для автоматичного формування необхідних посилань.

Визначено функції, які складають пошукову базу для опрацювання наукових статей та визначених у них фрагментів для цитування, що зберігаються у власному репозиторії.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

3.1 Варіанти використання програмної системи

Програмна система для управління репозиторієм наукових статей працює з зареєстрованими користувачами. Саме реєстрація та подальша авторизація користувача надає йому доступ до власного репозиторію.

На рис. 3.1 представлена діаграма варіантів використання програмного проекту.

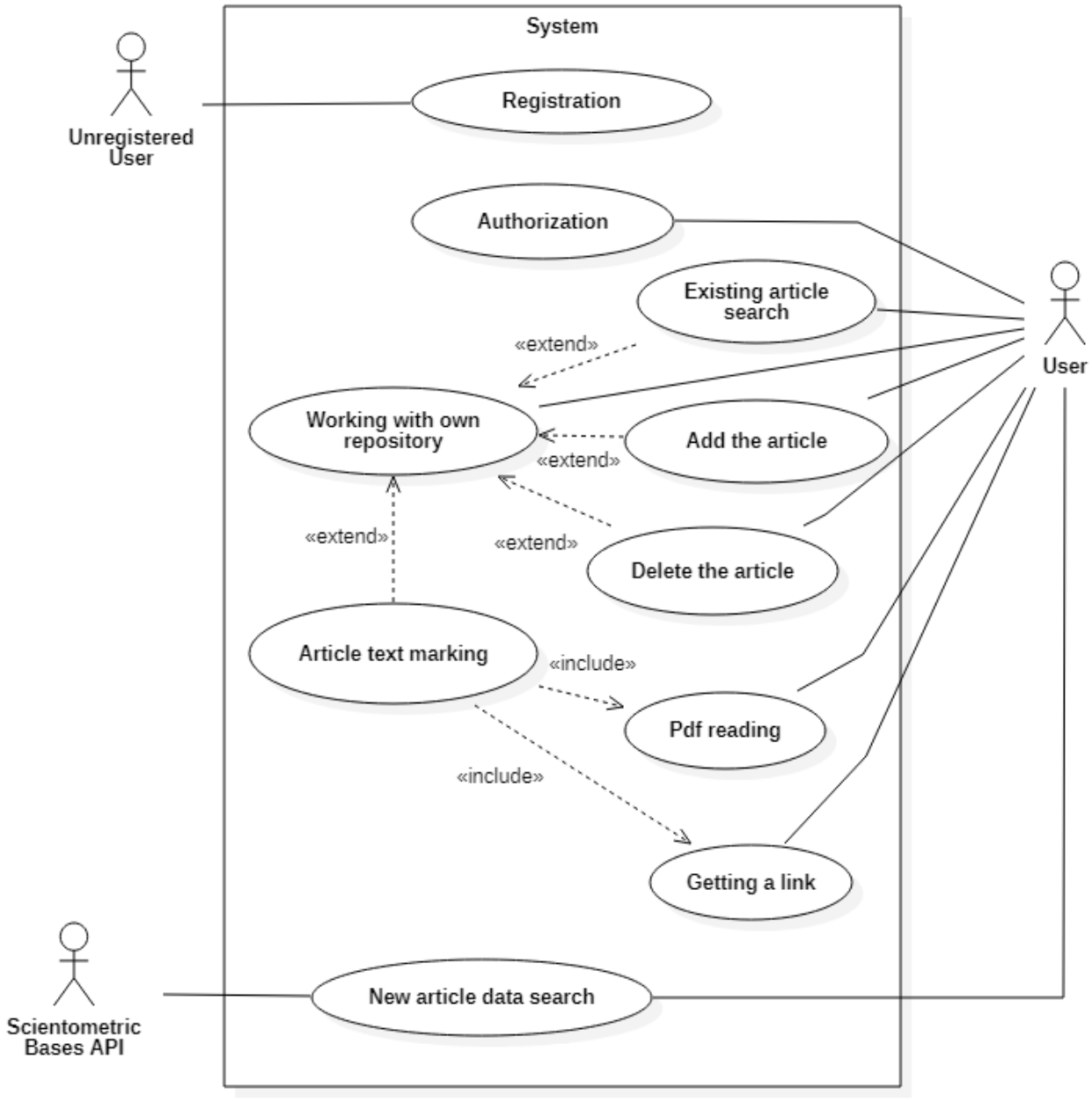

Рисунок 3.1 – Діаграма UML Use Case системи

Варіант використання № 1 – «Registration»

Основний виконавець: Unregistered User.

Передумови: Користувач системи Unregistered User має намір зареєструватись, щоб отримати доступ до власного репозиторію та повного функціоналу системи.

Післяумови: Користувач системи User отримав доступ до повного функціоналу та власного репозиторію.

Основний успішний сценарій.

1. Користувач Unregistered User обирає функцію «Реєстрація» для реєстрації у системі.
2. Система надає користувачеві Unregistered User форму, у яку необхідно ввести власний логін та пароль, які повинні бути унікальними у системі та ідентифікувати користувача.
3. Користувач Unregistered User вводить логін та пароль. Логін повинен містити не менш ніж 16 символів, в тому числі хоча б 1 символ верхнього регістру, хоча б 1 символ нижнього регістру та хоча б 1 спеціальний символ: «*», «_» чи «-». Пароль не повинен повторювати логін, рекомендована довжина паролю – 10 символів та довше. Пароль вводиться двічі задля впевненості коректного вводу.
4. Програмна система перевіряє введені дані реєстрації. Персональні дані кодуються для того, щоб не зберігати їх у базі даних у відкритому вигляді.
5. Програмний система звертається до бази даних, активує новий запис та зберігає у ньому введені реєстраційні дані.
6. Завершення прецеденту.

Альтернативні сценарії.

3а) Користувач Unregistered User натискає «Назад», він не хоче реєструватись та відмінив запит.

1) Програмна система повідомляє, що реєстрацію відмінено.

2) Завершення прецеденту без зміни складу користувачів.

4а) Unregistered User вводить невірний логін.

1) Програмна система видає повідомлення про правильний формат.
2) Завершення прецеденту без реєстрації нового користувача.

4б) Unregistered User вводить логін, який належить іншому користувачу.

1) Програмна система повідомляє про це та надає припустимі варіанти логінів.
2) Завершення прецеденту без реєстрації нового користувача.

4в) Unregistered User визначає помилку формату при введенні логіну.

1) Програмна система видає інформацію про вимоги до формату логіну.
2) Завершення прецеденту без реєстрації нового користувача.

4г) Unregistered User при повторному введенні паролю вводить пароль, що не співпадає з початково введеним.

1) Програмна система повідомляє про неспівпадіння паролів.
2) Завершення прецеденту без реєстрації нового користувача.

5а) Помилка у збереженні реєстраційних даних нового користувача.

1) Повідомлення про помилку збереження реєстраційних даних. Завершення прецеденту реєстрації.

Варіант використання № 2 – «Authorization»

Основний виконавець: User.

Передумови: Користувач User має реєстрацію у програмній системі та бажає авторизуватись.

Післяумови: Користувач User отримав доступ до власного репозиторію та функціоналу програмної системи.

Основний успішний сценарій.

1. Користувач User обирає функцію «Авторизація» для авторизації у програмній системі.
2. Програмна система запитує логін та пароль.

3. Користувач User вводить логін та пароль у відповідні поля. При вводі логіну система відображає його звичайним чином (не приховує). При вводі паролю символи з метою підвищення безпеки відображаються точками. При бажанні користувач User може натиснути кнопку з зображенням «ока» для відображення символів паролю.
4. Програмна система ідентифікує користувача User по введеним даним та виконує його авторизацію.
5. Програмна система надає користувачеві доступ до власного репозиторію та повного функціоналу.
6. Успішне завершення авторизації.

Альтернативні сценарії – «Authorization».

3а) Користувач User натискає «Назад», він не хоче авторизуватись та відмінив запит.

1) Програмна система повідомляє, що авторизацію відмінено.
2) Завершення прецеденту «Authorization».

3б) Користувач User не ввів логін.

1) Програмна система повідомляє, що це обов’язкове поле.

3в) Користувач User не ввів пароль.

1) Програмна система повідомляє, що це обов’язкове поле.

3г) Користувач User задав неприпустимі символи у логіні.

1) Програмна система наводить перелік припустимих символів.
2) Система повторно надає поля для вводу.

3д) Користувач User задав логін чи пароль неприпустимої довжини.

1) Програмна система наводить вимоги до довжини.
2) Система повторно надає поля для вводу.

3е) Користувач User ввів невірний логін чи пароль.

1) Програмна система повідомляє про помилку.
2) Система повторно надає поля для вводу.

Варіант використання № 3 – «Existing article search»

Основний виконавець: User.

Передумови: Користувач User авторизований у системі та може переглядати власний репозиторій. Користувач User бажає знайти певний науковий матеріал в репозиторії.

Післяумови: Користувач User отримав доступ до наукового матеріалу та переглядає його.

Основний успішний сценарій.

1. Користувач User обирає функцію «Пошук статті» для того, щоб знайти науковий матеріал у власному репозиторії.
2. Програмна система відкриває форму з фільтрами для статей.
3. Користувач User задає дані, відповідно до яких він бажає здійснювати пошук:
   - назва (чи фрагмент назви);
   - автори (чи фрагмент переліку авторів);
   - рік публікації (чи діапазон років);
   - назва журналу (чи фрагмент назви журналу);
   - джерело (наукометрична база, з якої потрапила у репозиторій користувача стаття – обирається з переліку).

   Користувач може вказувати певне значення чи діапазон значень. Крім того, користувач може заповнювати одне поле фільтру чи декілька полів для уточнення пошуку.
4. Програмна система виконує пошук та надає користувачеві перелік знайдених матеріалів у вигляді списку. Якщо під умови підпадає тільки одна публікація, вона одразу ж відкривається для перегляду.
5. Якщо знайдено декілька публікацій, користувач обирає потрібну та двічі натискає на неї для перегляду.
6. Для виходу з режиму перегляду користувач обирає функцію «Close».
7. Завершення прецеденту.

Альтернативні сценарії – «Existing article search».

3а) Користувач User не задав жодного фільтру.

1) Програмна система виводить всі наукові статті, які є у репозиторії.
2) Користувач може налаштовувати перегляд, встановлюючи режими відображення: «Сортування по назві», «Сортування по автору» (розглядаються тільки перші автори), «Сортування по року видання», «Зворотнє сортування по назві», «Зворотнє сортування по автору» (тільки перші автори), «Зворотнє сортування по року видання».

4а) Немає доступу до бази даних.

1) Програмна система повідомляє про помилку.
2) Завершення прецеденту.

4б, 5а) Файл статті пошкоджений.

1) Програмна система повідомляє про неможливість відображення.
2) Завершення прецеденту.

5б) Користувач User не обрав жодної наукової статті для перегляду.

1) Завершення прецеденту.

Деталізація прецеденту «Existing article search» показана на рис. 3.2: для пошуку матеріалів можна встановлювати фільтри («Select filters»), налаштовувати сортування статей («Select display mode») та відкривати для перегляду потрібну статтю («Open article»).

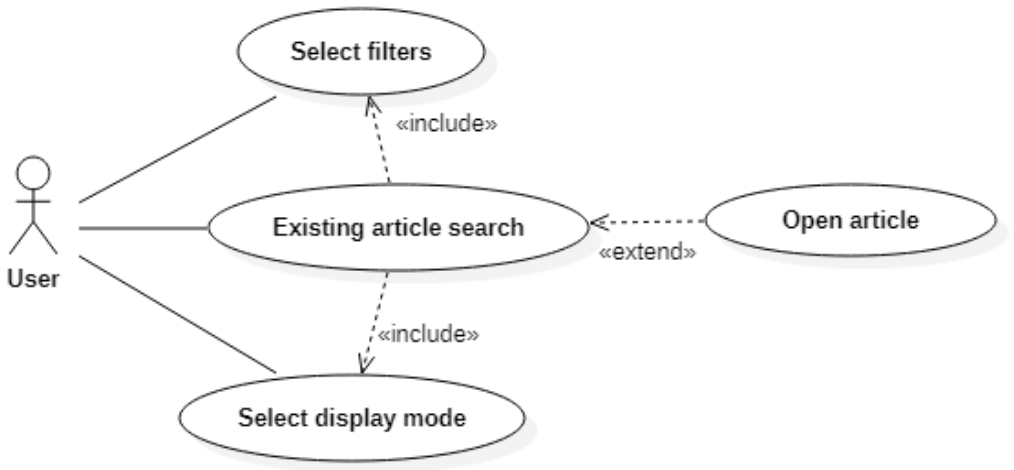

Рисунок 3.2 – Деталізація прецеденту «Existing article search»

Варіант використання № 4 – «Add the article»

Основний виконавець: User.

Передумови: Користувач User авторизований у системі та може переглядати власний репозиторій. Користувач User бажає знайти новий науковий матеріал та зберегти його у власний репозиторії.

Післяумови: Репозиторий користувача оновлений. Користувач User має доступ до нової статті.

Основний успішний сценарій.

1. Користувач User обирає функцію «Додати статтю».
2. Програмна система відкриває форму з полями для визначення вимог до статті.
3. Користувач User задає критерії, відповідно до яких він бажає здійснювати пошук:
   - назва (чи фрагмент назви);
   - ключові слова (повне співпадіння з хоча б з одним ключовим словом);
   - фрагмент анотації (повне співпадіння з хоча б з одним словом анотації);
   - тематика наукової роботи;
   - автори (чи фрагмент переліку авторів);
   - рік публікації (чи діапазон років, може бути обмежений тільки верхньою межею, чи тільки нижньою межею);
   - назва журналу (чи фрагмент назви журналу);
   - джерело (наукометрична база, у якій потрібно шукати статтю; якщо не вказано – виконується пошук статті в усіх доступних базах).

   Користувач User може задавати один чи декілька критеріїв для пошуку нового наукового матеріалу.
4. Програмна система виконує пошук наукових матеріалів, які є у вільному доступі, відповідно до вказаних критеріїв, та надає користувачеві перелік знайдених матеріалів у вигляді списку. Якщо під умови підпадає тільки одна публікація, вона одразу ж відкривається для перегляду.

5. Якщо знайдено декілька публікацій, користувач обирає потрібну та двічі натискає на неї для перегляду.
6. Якщо користувач User бажає додати знайдену статтю у власний репозиторій, він обирає функцію «Зберегти».
7. Програмна система зберігає новий науковий матеріал у репозиторії користувача. З використанням стандартних API наукометричних баз витягується вся інформація стосовно знайденої статті.
8. Завершення прецеденту.

Альтернативні сценарії – «Add the article».

3а) Користувач User не обрав жодного критерію.

1) Завершення прецеденту без оновлення репозиторію користувача.

4а) Жодного матеріалу відповідно до введених критеріїв не знайдено.

1) Програмна система повідомляє про порожній результат пошуку.
2) Завершення прецеденту.

4б) Для завантаження знайденого матеріалу потрібна реєстрація через університет.

1) Програмна система повідомляє про обов’язкову реєстрацію.
2) Завершення прецеденту.

4в) Для завантаження знайденого матеріалу потрібно сплатити певні кошти.

1) Програмна система повідомляє про розмір оплати за доступ до повного обсягу статті.
2) Завершення прецеденту.

7а) Немає доступу до бази даних.

1) Програмна система повідомляє про помилку.
2) Завершення прецеденту.

Деталізація прецеденту «Add the article» показана на рис. 3.3: для пошуку нових матеріалів можна встановлювати критерії пошуку («Entering search criteria»). В свою чергу, при здійсненні пошуку виконується перевірка доступу до повного тексту статті («Checking access to material»).

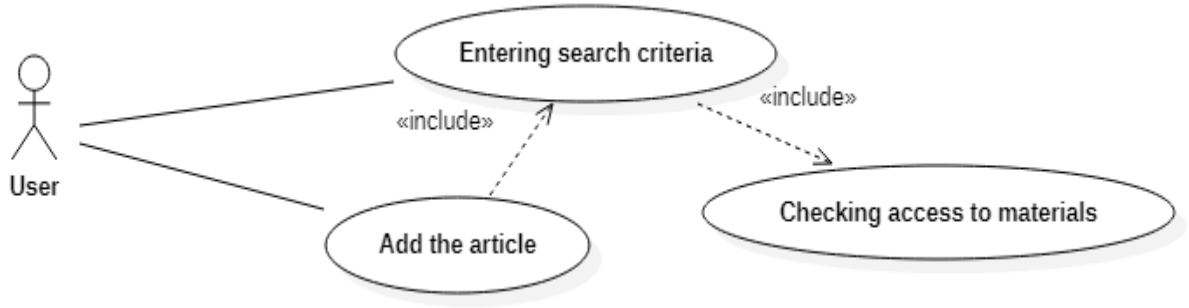

Рисунок 3.3 – Деталізація прецеденту «Add the article»

Варіант використання № 5 – «Delete the article».

Основний виконавець: User.

Передумови: Користувач User авторизований у системі та може переглядати власний репозиторій. Користувач User бажає видалити непотрібний йому новий науковий матеріал та оновити власний репозиторій.

Післяумови: Репозиторий користувача оновлений. Непотрібна стаття видалена.

Основний успішний сценарій.

1. Користувач User обирає функцію «Видалити статтю».
2. Програмна система відкриває форму з полями для визначення вимог до статті.
3. Користувач User задає критерії, відповідно до яких він бажає здійснювати пошук:
   - назва (чи фрагмент назви);
   - автори (чи фрагмент переліку авторів);
   - назва журналу (чи фрагмент назви журналу).

   Користувач User може задавати один чи декілька критеріїв для визначення наукового матеріалу, який потрібно видалити.
4. Програмна система виконує пошук наукових матеріалів у власному репозиторії відповідно до вказаних критеріїв, та надає користувачеві перелік знайдених матеріалів у вигляді списку.

5. Якщо знайдено декілька публікацій, користувач обирає потрібну та двічі натискає на неї для видалення.
6. Система повторно запитує у користувача, чи згоден він видалити знайдений науковий матеріал з власного репозиторію, для підтвердження він обирає функцію «Підтвердити».
7. Програмна система видаляє вказаний науковий матеріал з репозиторію користувача.
8. Завершення прецеденту.

Альтернативні сценарії – «Delete the article».

3а) Користувач User не обрав жодного критерію для видалення статті.

1) Завершення прецеденту без оновлення репозиторію користувача.

4а) Жодного матеріалу для видалення відповідно до введених критеріїв не знайдено.

1) Програмна система повідомляє про порожній результат пошуку.
2) Завершення прецеденту.

5а) Користувач не уточнює яку саме публікацію він бажає видалити.

1) Завершення прецеденту без оновлення репозиторію користувача.

6а) Користувач не підтверджує видалення матеріалу.

1) Завершення прецеденту без оновлення репозиторію користувача.

7а) Немає доступу до бази даних.

1) Програмна система повідомляє про помилку.

Варіант використання № 6 – «Pdf reading».

Основний виконавець: User.

Передумови: Користувач User обрав статтю та бажає переглянути її у форматі pdf.

Післяумови: Обрана стаття переглянута.

Основний успішний сценарій.

1. Користувач User обирає функцію «Переглянути статтю».

2. Програмна система відкриває обрану статтю та обирає функцію «Конвертувати у pdf».
3. Програмна система обирає конвертацію у формат pdf із застосуванням стандартних API для конвертації.
4. Програмна система виконує збереження поточного наукового матеріалу у форматі pdf.
5. Програмна система виконує завантаження обраного наукового матеріалу з репозиторію у тимчасову пам’ять для подальшої роботи.
6. Завершення прецеденту.

Альтернативні сценарії – «Pdf reading».

2a) Немає доступу до статті.

1) Завершення прецеденту без конвертації.

4a) Немає доступу до бази даних.

1) Програмна система повідомляє про неможливість збереження.
2) Завершення прецеденту.

Варіант використання № 7 – «Getting a link».

Основний виконавець: User.

Передумови: Користувач User обрав статтю та бажає отримати на неї посилання.

Післяумови: Посилання на обрану статтю отримано.

Основний успішний сценарій.

1. Користувач User обирає функцію «Отримати посилання».
2. Програмна система відкриває обрану статтю та надає варіанти посилання для одного з наступних форматів: ACM, ACS, APA, ABNT, Chicago, Harvard, IEEE, MLA, Turabian та Vancouver.
3. Користувач User обирає потрібний формат.
4. Програмна система формує посилання відповідно до обраного користувачем формату.

5. Програмна система надає користувачеві сформоване посилання.
6. Завершення прецеденту.

Альтернативні сценарії – «Getting a link».

2а) Немає доступу до статті.

1) Завершення прецеденту без формування посилання.
2) Завершення прецеденту

3а) Користувачем не обрано жодного формату для формування посилання.

1) Повідомлення про обов’язковість вибору формату.

Варіант використання № 8 – «Article text marking».

Основний виконавець: User.

Передумови: Користувач User обрав статтю та бажає отримати посилання на фрагмент тексту, який у ній міститься.

Післяумови: Посилання на фрагмент статті отримано.

Основний успішний сценарій.

1. Користувач User бажає виконати маркування фрагменту обраної статті.
2. Програмна система запускає ВВ 6 «Pdf reading».
3. Користувач User виділяє фрагмент тексту статті.
4. Користувач User обирає потрібний формат.
5. Програмна система запускає ВВ 7 «Getting a link».
6. Програмна система надає користувачеві сформоване посилання на фрагмент статті.
7. Завершення прецеденту.

Варіант використання № 9 – «Working with own repository»

Основний виконавець: User.

Передумови: Користувач User авторизований у системі та може переглядати власний репозиторій. Користувач User бажає працювати з власним репозиторієм.

Післяумови: Дія з власним репозиторієм виконана.

Основний успішний сценарій.

1. Користувач User обирає функцію «Додати статтю».
2. Програмна система запускає ВВ 4 – «Add the article».
3. Перехід на п. 8.
4. Користувач обирає функцію «Видалити статтю».
5. Програмна система запускає ВВ 5 – «Delete the article».
6. Перехід на п. 8.
7. Програмна система запускає ВВ 6 – «Article text marking».
8. Завершення прецеденту.

Варіант використання № 10 – «New article data search»

Основний виконавець: User.

Передумови: Користувач User авторизований у системі та може переглядати власний репозиторій. Користувач User бажає знайти новий науковий матеріал та переглянути його.

Післяумови: Користувач User знайшов та переглянув нову статтю.

Основний успішний сценарій.

1. Користувач User обирає функцію «Знайти статтю».
2. Програмна система відкриває форму з полями для визначення вимог до статті.
3. Користувач User задає критерії, відповідно до яких він бажає здійснювати пошук:
   - назва (чи фрагмент назви);
   - ключові слова (повне співпадіння з хоча б з одним ключовим словом);
   - фрагмент анотації (повне співпадіння з хоча б з одним словом анотації);
   - тематика наукової роботи;
   - автори (чи фрагмент переліку авторів);
   - рік публікації (чи діапазон років, може бути обмежений тільки верхньою межею, чи тільки нижньою межею);

- назва журналу (чи фрагмент назви журналу);
- джерело (наукометрична база, у якій потрібно шукати статтю; якщо не вказано – виконується пошук статті в усіх доступних базах).

Користувач User може задавати один чи декілька критеріїв для пошуку нового наукового матеріалу.

4. Програмна система виконує пошук наукових матеріалів, які є у вільному доступі, відповідно до вказаних критеріїв, та надає користувачеві перелік знайдених матеріалів у вигляді списку. Якщо під умови підпадає тільки одна публікація, вона одразу ж відкривається для перегляду.
5. Якщо знайдено декілька публікацій, користувач обирає потрібну та двічі натискає на неї для перегляду.
6. Завершення прецеденту.

Альтернативні сценарії – «New article data search».

3а) Користувач User не обрав жодного критерію.

1) Завершення прецеденту без здійснення пошуку.

4а) Жодного матеріалу відповідно до введених критеріїв не знайдено.

1) Завершення прецеденту.

3.2 Вимоги до нефункціональних характеристик

Безпека

1. Якщо користувач буде намагатись отримати доступ до власного репозиторію, в нього це вийде не менш ніж у 98% випадків.

2. Якщо користувач спробує отримати доступ до репозиторіїв інших користувачів системи, у нього це не вийде більш ніж у 99% випадків.

Надійність

1. Якщо користувач завантажує у власний репозиторій науковий матеріал, то це виходить успішно не менш ніж в 96% випадків.

2. Якщо користувач намагається створити посилання на фрагмент статті, то це виходить успішно не менш ніж в 95% випадків.

3. Якщо користувач видаляє науковий матеріал з репозиторію, то це відбувається успішно не менше ніж в 97% випадків.

Продуктивність

1. Якщо користувач завантажує у власний репозиторій науковий матеріал, то він зберігається у репозиторії не більше ніж через 3 секунди.

2. Якщо користувач створює нове посилання, то воно генерується не більше ніж через 1 секунду.

3. Якщо користувач намагається зберегти нове посилання, то це відбувається не більше ніж через 1,5 секунди.

4. Якщо користувач створює вимоги щодо пошуку наукового матеріалу, то вони перевіряється на коректність не більше ніж через 1 секунду.

Супровід

1. Якщо користувачеві знадобиться підтримка іншої наукометричної бази, то для доопрацювання функціоналу знадобиться не більше ніж 2 місяці.

2. Якщо користувачеві знадобиться статистична обробка результатів пошуку наукових матеріалів, то для надання йому таких функцій знадобиться не більше ніж 2 тижні.

3. Якщо користувачу з особливими потребами знадобяться зміни в інтерфейсі, то система зможе надати йому таку можливість не більше ніж через місяць.

4. Якщо знадобиться мобільна версія програмної системи, то на доопрацювання знадобиться не більш ніж 3 місяці.

Юзабіліті

1. Якщо користувач бажає завантажити науковий матеріал, то це йому вдається не більше чим за 1 хвилину.

2. Для завантаження нового наукового матеріалу користувачеві знадобиться виконання не більш ніж 5 команд (вибір команд меню та натискання кнопок управління).

2. Якщо користувач бажає переглянути науковий матеріал, то це йому вдасться не більше чим за 1,5 хвилини.

3. Якщо користувач бажає виділити фрагмент тексту у науковому матеріалі для генерації посилання на нього, то це йому вдасться не більше чим за 1 хвилину.

3.3 Висновки до розділу

У третьому розділі виконано специфікацію вимог до програмної системи для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз. Для цього визначено функціональні вимоги, описано 10 варіантів використання. Також визначено нефункціональні вимоги – безпеку, надійність, продуктивність, супровід та юзабіліті.

4 ПРОЕКТУВАННЯ СИСТЕМИ ДЛЯ УПРАВЛІННЯ РЕПОЗИТОРІЄМ НАУКОВИХ СТАТЕЙ

4.1 Проектування архітектури програмної системи

Програмна система є багатофункціональною та містить сукупність бізнес-логік для реалізації роботи з міжнародними базами (пошук та завантаження матеріалів), управління науковими статтями у власному репозиторії, формування посилань відповідно до визначеного набору форматів, налаштування профілю та характеристик цитування (рис. 4.1).

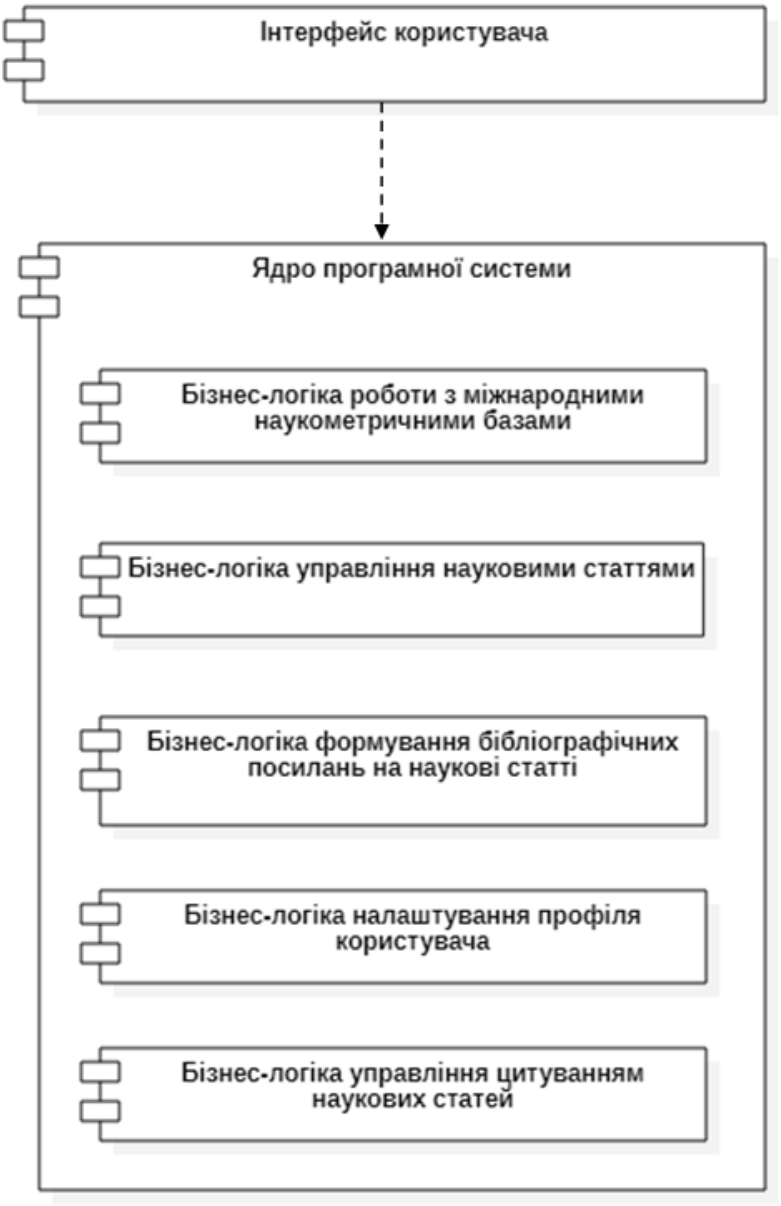

Рисунок 4.1 – Архітектура програмної системи у вигляді діаграми компонентів

«Компонент у UML представляє модульну частину системи. Поведінка визначається з точки зору обов'язкових і наданих інтерфейсів. Компонент має зовнішнє подання із загальнодоступними властивостями та операціями, і він має внутрішнє подання з приватними властивостями» [8].

4.2 Детальне проектування програмної системи

Для детального проектування програмної системи застосуємо можливості, які надає діаграма послідовностей, та продемонструємо за її допомогою виконання основних процесів системи з плином часу.

Рисунок 4.2 показує роботу програмної системи для прецеденту «Registration». Користувач задає логін та пароль, згідно з якими буде у подальшому авторизуватись у системі. Контролер перевіряє введені дані на відповідність вимогам та повертає error у разі невідповідності. У разі успіху створюється новий (порожній) репозиторій користувача, до якого йому надається доступ. Створюється акаунт користувача та до нього підвантажується посилання на репозиторій.

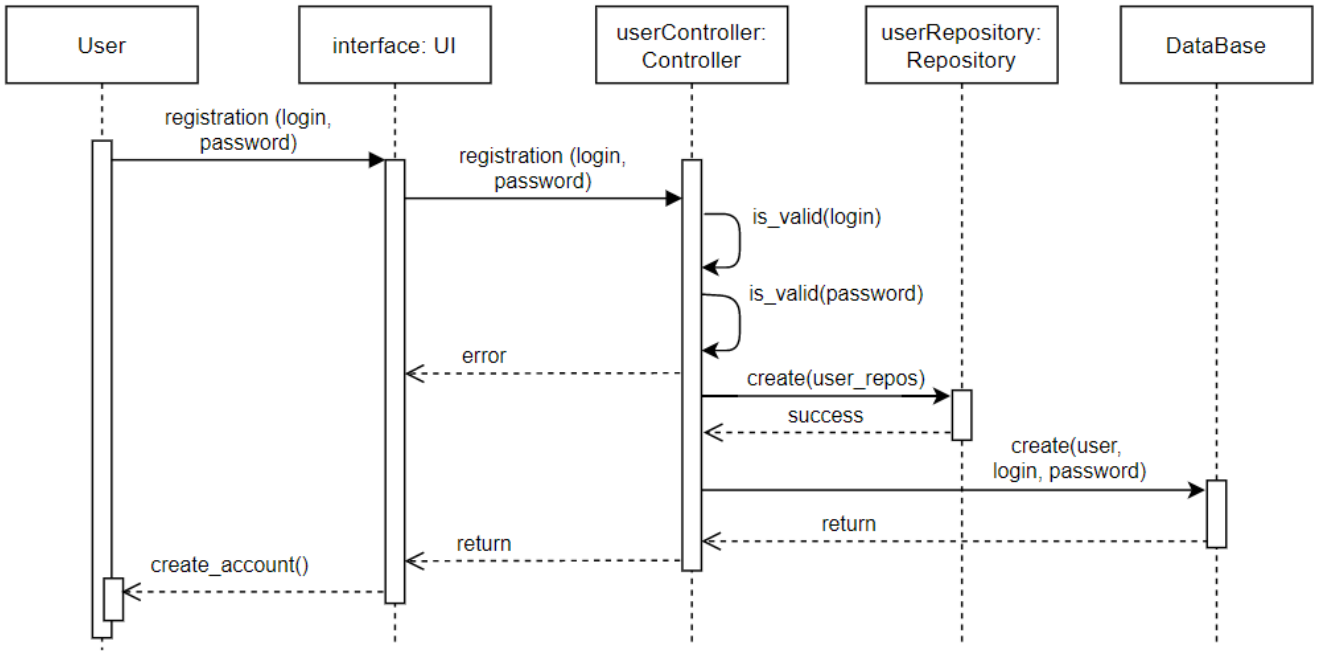

Рисунок 4.2 – Діаграма послідовностей прецеденту «Registration»

Рисунок 4.3 демонструє роботу прецеденту «Authorization». Після вводу користувачем авторизаційних даних вони перевіряються контролером та у разі успіху передаються на DateBase для пошуку акаунту. Якщо відповідний акаунт знайдено, користувачеві надається доступ до репозиторію user_repos та до всього функціоналу системи.

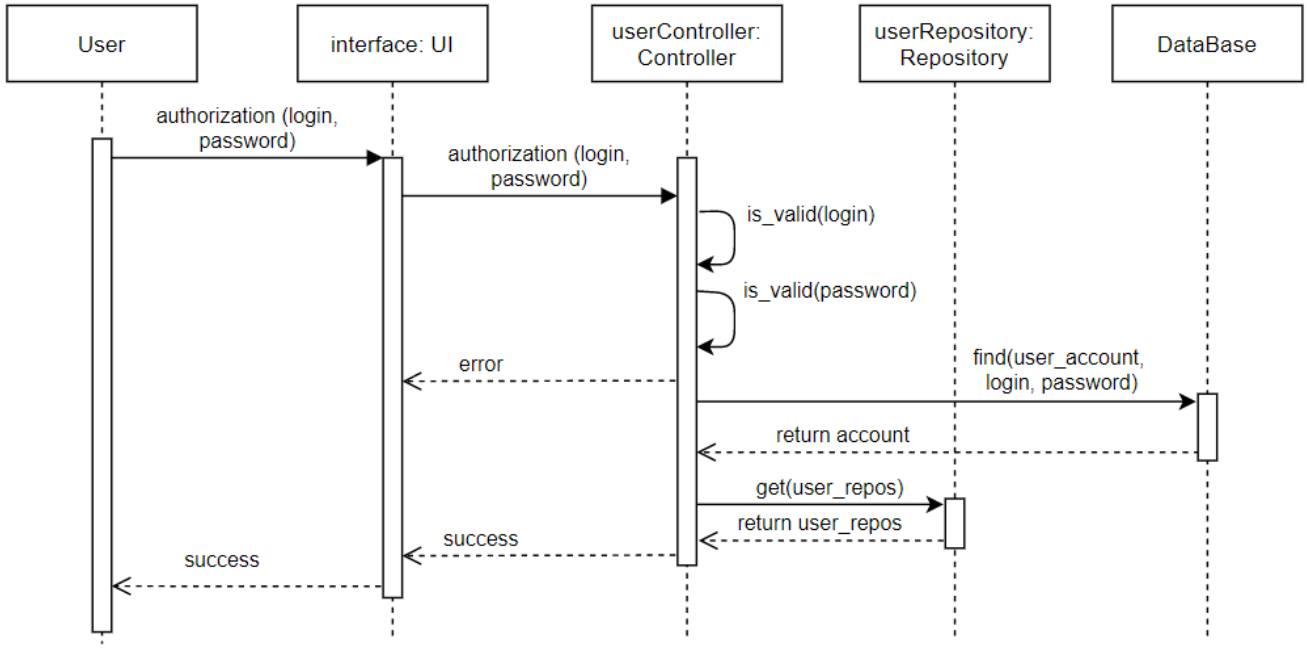

Рисунок 4.3 – Діаграма послідовностей прецеденту «Authorization»

На рис. 4.4 наведена діаграма діяльності для пошуку наукової статті у наукометричних базах та додавання її у репозиторій. Спочатку користувач визначає всі вимоги до пошуку та задає параметри статті, чи параметри тематики, чи параметри журналу та бази. Якщо визначені не всі вимоги, процес продовжується. Якщо вимоги визначені некоректно, система повідомляє користувача про необхідність виправлення помилок.

Коли перевірка на коректність вимог пройдена, за допомогою API наукометричних баз виконується пошук статей. Всі знайдені статті надаються користувачеві у вигляді списку. Він обирає зі списку ту статтю (чи декілька статей), які бажає включити у власний репозиторій. Вибрані матеріали зберігаються у репозиторії.

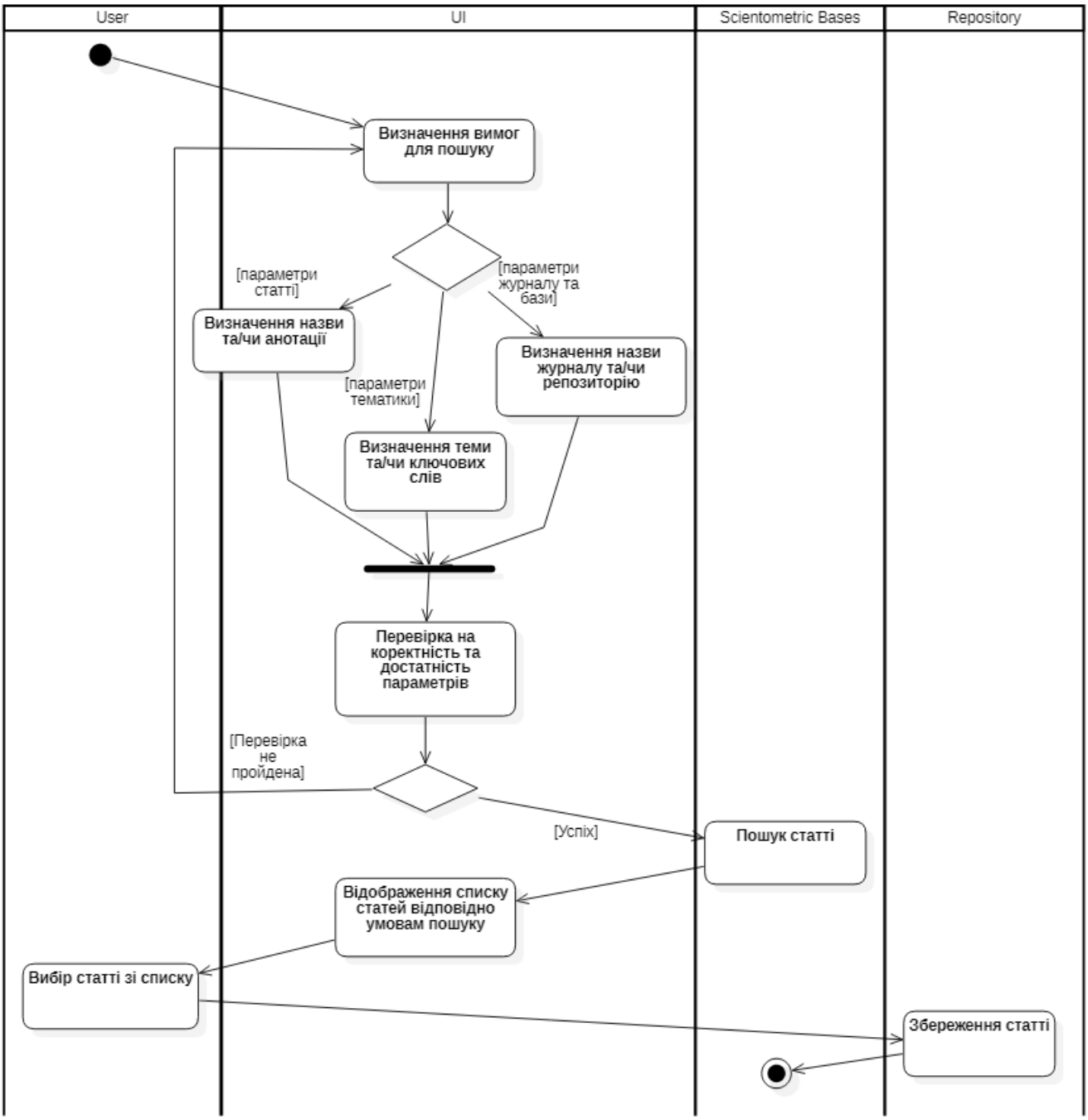

Рисунок 4.4 – Діаграма діяльності для пошуку та додавання статті

На рис. 4.5 наведена діаграма діяльності для формування посилання на фрагмент статті. Відповідно до цієї діаграми, користувач виконує пошук статті у репозиторії, виділяє потрібний для цитування фрагмент та формат цитування. Відповідно до обраного формату виконується генерація цитати. Система надає користувачеві вигляд сгенерованого цитування. Користувач зберігає цитату у власному репозиторії.

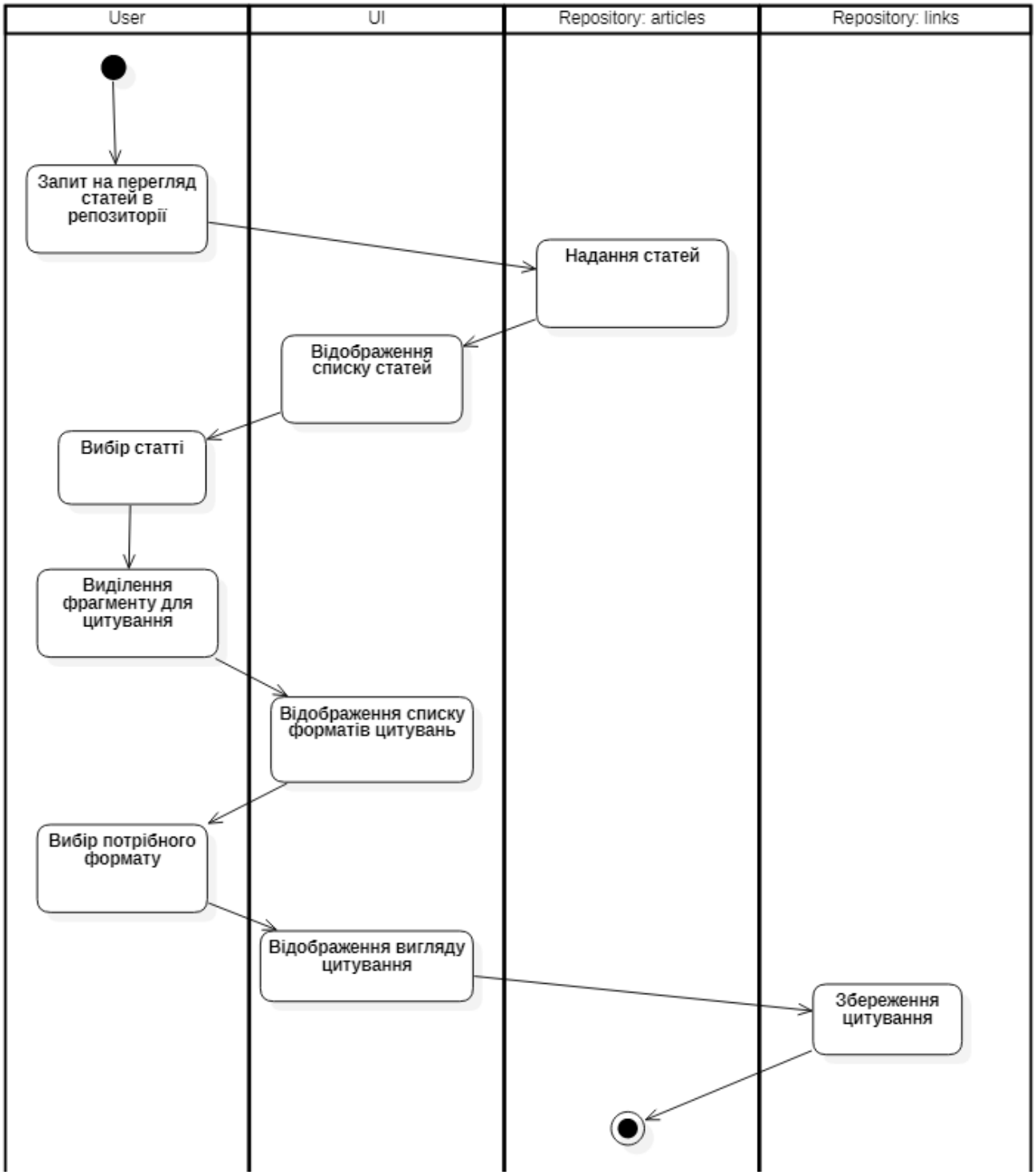

Рисунок 4.5 – Діаграма діяльності для формування посилання на фрагмент статті

Ця діаграма діяльності поєднує у собі процеси, що належать варіантам використання «Article text marking» та «Getting a link», які підпорядковуються варіанту використання «Work with own repository». Крім того, якщо знайдена стаття знаходиться у форматі, який відрізняється від pdf, виконується перетворення у цей формат.

4.3 Структура бази даних

Для збереження даних у програмній системі використовується реляційна база даних. «База даних – це інтегрована сукупність структурованих і взаємозалежних даних, організована за певними правилами, які передбачають загальні принципи опису, зберігання і обробки даних.

Реляційна база даних — це тип бази даних, що зберігає інформацію в електронних таблицях і здійснює пошук даних в одній таблиці на підставі визначених ключових полів іншої таблиці.» [9] Реляційна БД повинна бути нормалізованою.

«Принципи нормалізації:

- в кожній таблиці БД не повинно бути повторюваних полів;
- в кожній таблиці повинен бути унікальний ідентифікатор (первинний ключ);
- кожному значенню первинного ключа повинна відповідати достатня інформація про тип суті або про об’єкт таблиці;
- зміна значень в полях таблиці не повинна впливати на інформацію в інших полях (крім змін у полях ключа).» [9]

Розглянемо таблиці, які необхідні для збереження всіх даних:

- Users – таблиця даних для збереження даних користувачів програмної системи;
- Seance – таблиця з даними щодо сеансів роботи користувача;
- Articles – таблиця для збереження даних щодо наукових матеріалів;
- Links – таблиця для збереження посилань на фрагменти наукових статей у потрібних форматах;
- Format – таблиця з інформацією про формати цитувань, які підтримуються програмною системою.

Для вказаних таблиць визначені поля, типи даних, що зберігаються у цих полях, наведений опис та позначка чи є поле ключовим (табл. 4.1 – 4.5).

Таблиця 4.1 – Реляційна таблиця «Users»

Поле	Тип даних	Опис	Ключ
IdUser	integer (8)	Id користувача програмної системи	PK
Login	varchar (30)	Хеш логіну користувача програмної системи	
Password	varchar (50)	Хеш паролю користувача програмної системи	
RepositoryLink	String(1000)	Посилання на власний репозиторій	

Таблиця 4.2 – Реляційна таблиця «Seance»

Поле	Тип даних	Опис	Ключ
IdSeance	integer (8)	Id сеансу роботи користувача з системою	PK
IdUser	integer (8)	Id користувача програмної системи	
DateSeance	date(dd.mm.yy)	Дата початку роботи користувача з системою	

Продовження табл. 4.2

Поле	Тип даних	Опис	Ключ
TimeSeance	time(hh.mm.ss)	Час початку роботи користувача з системою	

Таблиця 4.3 – Реляційна таблиця «Articles»

Поле	Тип даних	Опис	Ключ
IdArticle	integer (4)	Id наукової статті	PK
NameArticle	varchar (100)	Назва статті	
Keywords	varchar (100)	Ключові слова	
IdLink	integer (4)	Id посилання	
Abstract	string(1000)	Анотація до статті	
Theme	varchar (100)	Тематика	
Author	varchar (500)	Перелік авторів	
Year	integer (4)	Рік публікації	
Journal	varchar (500)	Назва журналу	

Таблиця 4.4 – Реляційна таблиця «Links»

Поле	Тип даних	Опис	Ключ
IdLink	integer (4)	Id посилання	PK
IdFormat	integer (4)	Id формата для генерації посилання	
Description	varchar (500)	Коментарі до посилання (при необхідності)	

Таблиця 4.5 – БД: Реляційна таблиця Seance

Поле	Тип даних	Опис	Ключ
IdFormat	integer (4)	Id формата для генерації посилання	PK
FormatName	varchar (100)	Назва формату посилання на науковий матеріал	

ER-діаграма програмної системи для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз наведена на рис. 4.6. Вона містить міжтабличні зв’язки один-до-багатьох.

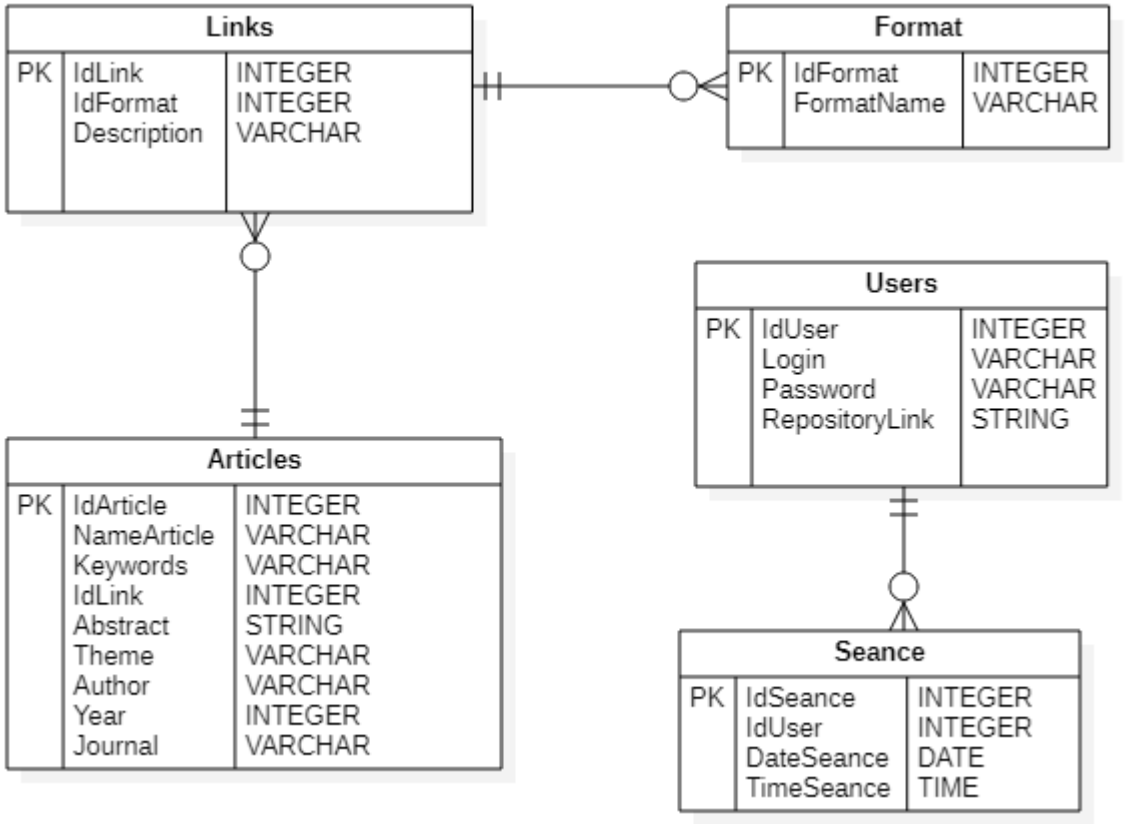

Рисунок 4.6 – Реляційна модель даних програмної системи

4.4 Проектування структури програмних класів

На рис. 4.7 наведена діаграма програмних класів.

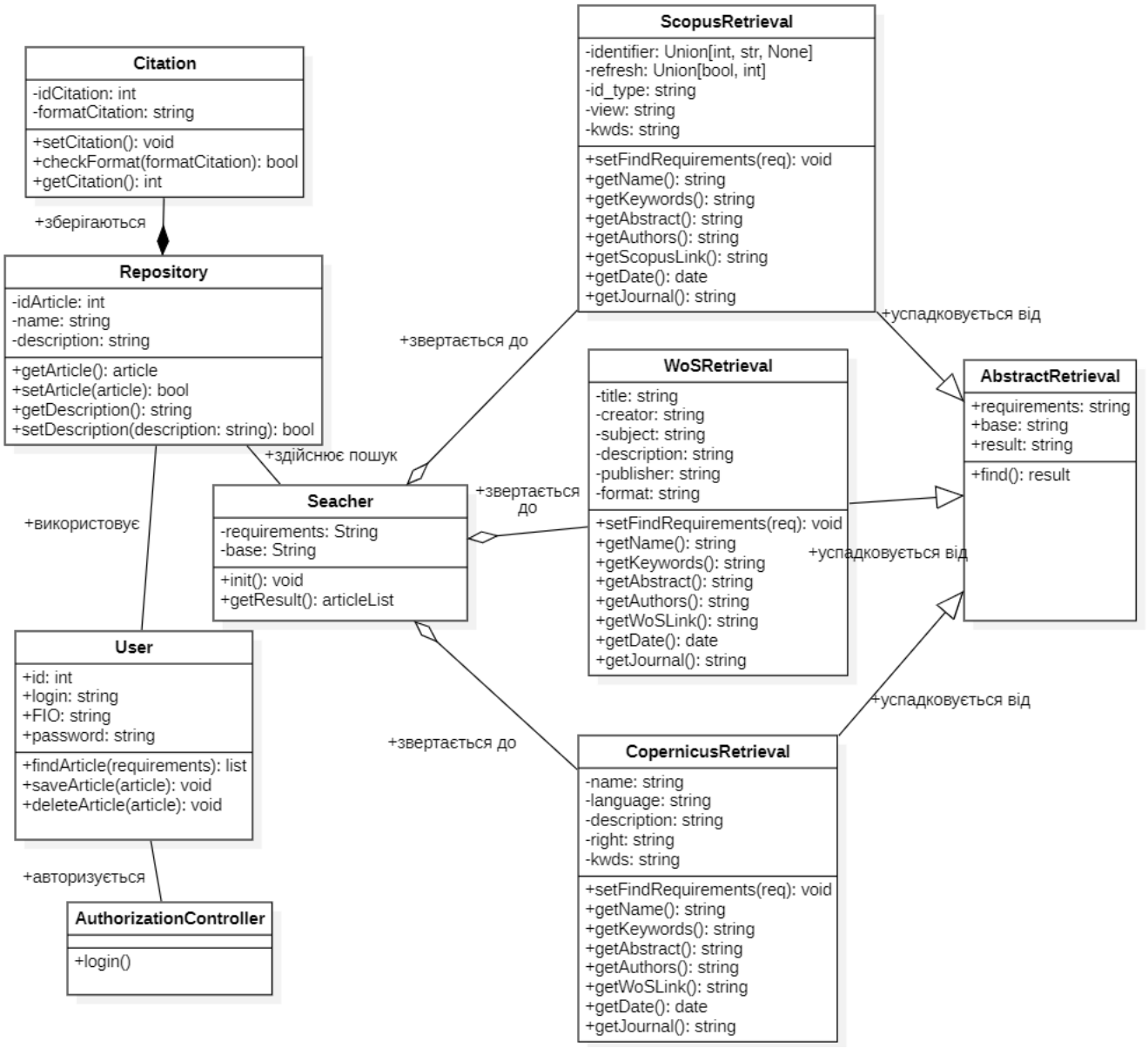

Рисунок 4.7 – Діаграма програмних класів

Клас ScopusRetrieval використовується для реалізації пошуку наукових статей у наукометричній базі Scopus.

Атрибути класу ScopusRetrieval:

– identifier: Union[int, str, None] - ідентифікатор наукової статті, тобто ідентифікатор Scopus або DOI (digital object identifier);

- refresh (Union[bool, int]) – прапорець щодо оновлення існуючого кешованого файлу, спочатку встановлюється False;
- id_type (string) – тип використовуваного ідентифікатора - Scopus чи DOI, спочатку встановлюється None;
- view (string) – формат статті, яка завантажується. Дозволено, в тому числі, і завантаження всієї інформації;
- kwds (str) – ключові слова, які входять до складу запиту як параметри та використовуються API.

Методи класу ScopusRetrieval:

- setFindRequirements(in req): void – встановлення та передача вимог ло пошуку матеріалів у науковометричних базах;
- getName(): string – отримання назви статті за фрагментом назви чи іншими вимогами;
- getKeywords(): string – отримання ключових слів;
- getAbstract(): string – отримання анотації статті за фрагментом анотації чи іншими вимогами;
- getAuthors(): string – отримання авторів статті за прізвищем одного з авторів чи іншими вимогами;
- getScopusLink(): string – отримання посилання на статтю у наукометричній базі;
- getDate(): date – отримання дати виходу статті;
- getJournal(): string – отримання назви журналу за його фрагментом чи іншими вимогами.

Класи WoSRetrieval та CopernicusRetrieval використовуються для пошуку наукових матеріалів в двох інших базах. Вони мають такі ж самі методи, як і клас ScopusRetrieval, але дещо відрізняються наборами атрибутів.

Атрибути класу WoSRetrieval:

- title: string – назва наукової статті;
- creator: string – автор (перший автор, якщо авторів декілька);
- subject: string – тематика публікації;

- description: string – опис, у який входить анотація статті;
- publisher: string – видавництво, у якому публікується журнал з науковими статтями;
- format: string – формат статті.

Атрибути класу CopernicusRetrieval:

name: string – назва статті;

language: string – мова, на якій подано статтю;

description: string – опис статті, включаючи анотацію;

right: string – ліцензія, відповідно до якої розповсюджується повний текст статті;

kwds: string – набір ключових слів.

Клас Seacher здійснює пошук статті у наукометричних базах:

- requirements: String – вимоги щодо пошуку;
- base: String – назва наукометричної бази;
- init(): void – метод ініціалізації пошуку;
- getResult(): articleList – метод отримання результату.

Клас User відповідає за дані та дії користувача системи:

- id: int – ідентифікатор користувача;
- login: string – логін (в зашифрованому вигляді);
- password: string – пароль користувача (в зашифрованому вигляді);
- FIO: string – персональні дані користувача;
- findArticle(requirements): list – метод пошуку статті;
- saveArticle(article): void – метод збереження статті в репозиторії;
- deleteArticle(article): void – метод видалення статті з репозиторію.

Клас Repository відповідає за репозиторій користувача системи:

- idArticle: int – ідентифікатор статті;
- name: string – назва статті;
- description: string – повний опис статті;
- getArticle(): article – метод для отримання статті;
- setArticle(in article): bool – метод для визначення статті;

- getDescription(): string – метод для отримання опису статті;
- setDescription(in description:string): bool – метод для визначення опису статті.

Клас Citation забезпечує роботу з цитатами фрагментів статей:

- idCitation: int – ідентифікатор цитати;
- formatCitation: string – формат цитування;
- setCitation(): void – запит на цитування;
- checkFormat(formatCitation): bool – перевірка формату цитування;
- getCitation(): int – отримання цитування відповідно до обраного формату.

4.5 Висновки до розділу

У четвертому розділі виконано проектування програмної системи з використанням діаграми компонентів, визначено компоненти для реалізації роботи з міжнародними базами, управління науковими статтями у власному репозиторії, формування посилань відповідно до визначеного набору форматів, налаштування профілю та характеристик цитування.

Проведено детальне проектування системи з використанням діаграм послідовностей та діяльності.

Розроблена структура бази даних, що використовується для збереження даних у програмній системі. Створено відповідну реляційну модель даних.

Спроектовано діаграму програмних класів. Наведено опис класів, їх атрибутів та методів.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЮВАНОЇ СИСТЕМИ

5.1 Особливості створення програмних модулів з урахуванням мови програмування

Для реалізації програмної система обрана сучасна мова програмування Python. «Python – це широко використовувана, інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою, яка використовується для програмування загального призначення.

Існує велика кількість факторів, які роблять Python доцільним для застосування:

− його легко вивчити – час, необхідний для вивчення Python, менший, ніж для багатьох інших мов; це означає, що можна швидше розпочати фактичне програмування;

− він простий у використанні для написання нового програмного забезпечення – часто можна писати код швидше, використовуючи Python;

− його легко отримати, встановити та розгорнути – Python є безкоштовним, відкритим і багатоплатформним; не всі мови можуть цим похвалитися.» [10]

Python підтримується Python Software Foundation, некомерційною членською організацією та спільнотою, яка займається розробкою, вдосконаленням, розширенням і популяризацією мови Python та її середовища.

Для збереження даних (власного репозиторію користувача) обрано та застосовано реляційну базу MySQL. «MySQL є однією з найбільш впізнаваних технологій у сучасній екосистемі великих даних. Її часто називають найпопулярнішою базою даних, яка наразі користується широким та ефективним використанням незалежно від галузі. Зрозуміло, що будь-хто, хто займається корпоративними даними чи загальними IT-технологіями, повинен прагнути принаймні до базових знань MySQL. З MySQL навіть новачки в реляційних системах можуть відразу створювати швидкі, потужні та безпечні системи зберігання даних. Програмний синтаксис та інтерфейси MySQL також є ідеальними шлюзами до широкого світу інших популярних мов запитів і структурованих сховищ даних.» [11]

Реляційна база даних – це цифрове сховище, яке збирає дані та організовує їх відповідно до реляційної моделі. У цій моделі таблиці складаються з рядків і стовпців, а зв'язки між елементами даних підтримують сувору логічну структуру.

5.2 Налаштування середовища для використання API

Для отримання доступу до можливостей API потрібно їх попередньо підключити. Для підключення PyScopus (Scopus API) найбільш простий шлях – це використання pip (рис. 5.1).

```
$ pip install pyscopus==1.0.3a2
```

Рисунок 5.1 – Підключення PyScopus за допомогою pip

Інший шлях – клонувати та інсталювати PyScopus з Git-репозиторію (рис. 5.2). Аналогічно з відповідного Git-репозиторію можна завантажувати і API для Index Copernicus.

```
$ git clone https://github.com/python-scopus.git
$ cd python-scopus/
$ python setup.py install
```

Рисунок 5.2 – Підключення PyScopus за допомогою Git-репозиторію

Крім того, PyScopus можна завантажити зі стандартної сторінки PyPi [12], як це показано на рис. 5.3.

```
$ tar -xvzf python-scopus-1.0.3a2.tar.gz
$ cd python-scopus-1.0.3a2.tar.gz/
$ python setup.py install
```

Рисунок 5.3 – Підключення PyScopus зі сторінки PyPi

Підключення API Web of Science виконується за допомогою Client URL (рис. 5.4).

```
$ curl -H 'X-ApiKey: (your API Key)' https://api.clarivate.com/desired_api/some_endpoint
```

Рисунок 5.4 – Доступ до WoS API

5.3 Реалізація інтерфейсу користувачів системи

Розглянемо декілька прикладів роботи інтерфейсу системи Science guide. На рис. 5.5 показано стартове вікно системи, що дозволяє зареєструватись чи авторизуватись для входу.

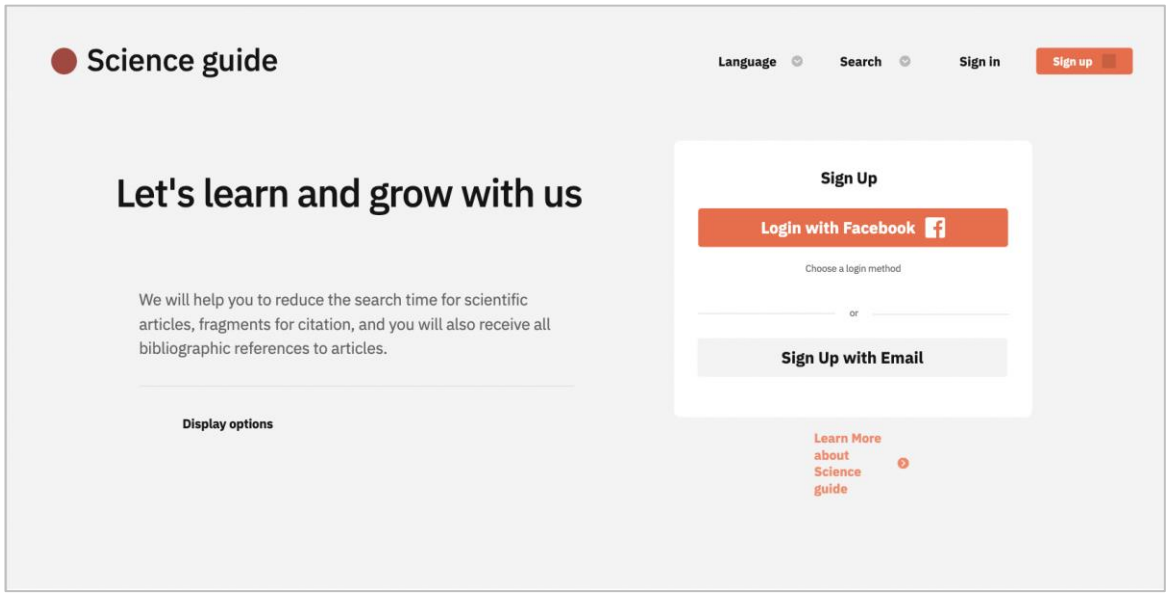

Рисунок 5.5 – Стартове вікно системи Science guide

На рис. 5.6 показано початок пошуку певного слова у власному репозиторії. Для введеного слова «Algoritms» результат пошуку є успішним, це слово міститься у 82% документів, збережених у репозиторії. При натисканні на кнопку «Next» система переходить до відображення списку результатів (рис. 5.7). Пошук слова здійснювався у книгах та статтях.

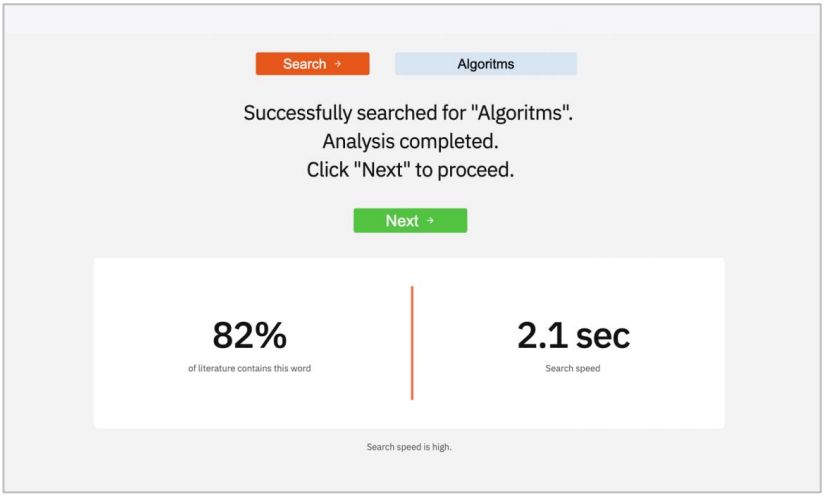

Рисунок 5.6 – Вікно пошуку слова чи фрагмента у документах репозиторія

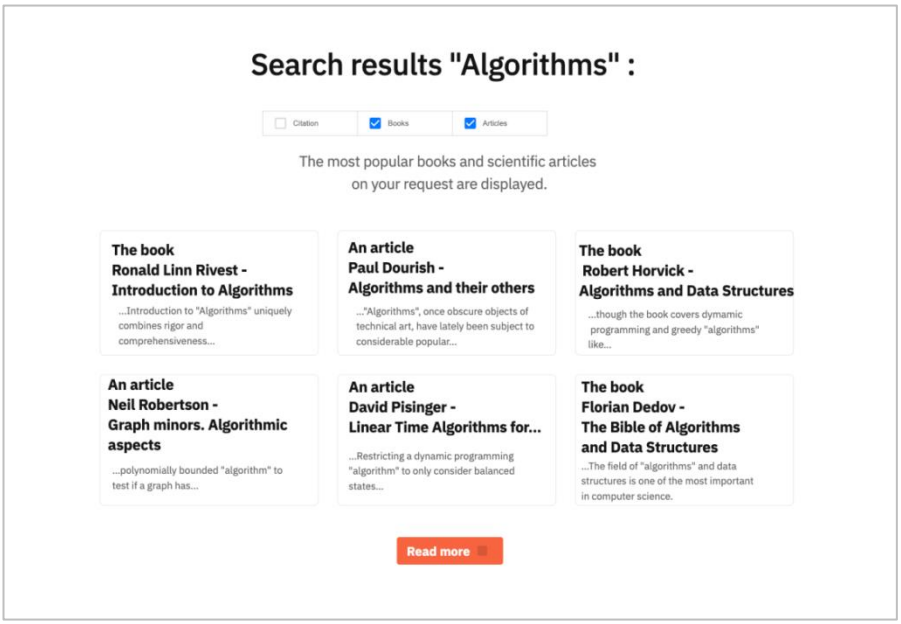

Рисунок 5.7 – Відображення результатів пошуку

5.4 Висновки до розділу

У розділі проведено вибір та обґрунтування інструментів та мов розробки. Показано, що мова Python є доцільною для використання у цьому проекті. Розглянуто механізми підключення та використання API наукометричних баз даних. Наведені приклади інтерфейсу роботи програмної системи.

6 ВИЗНАЧЕННЯ ВЛАСТИВОСТЕЙ ПРОГРАМНОЇ СИСТЕМИ

6.1 Тестування механізму генерації посилань

Проведемо тестування правильності генерації бібліографічних посилань в залежності від обраного користувачем формату. Результати тестування наведені у табл. 6.1.

Таблиця 6.1 – Верифікація бібліографічних посилань

Формат	Бібліографічне посилання	Результат
ACM	Popov, V.M., Petrov, R.K. і Honcharuk S.O. 2022. Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників. Інформаційні технології та моделювання. 3, 1 (Сер 2022), 30-45.	Успіх
ACS	Popov, V. M.; Petrov, R. K.; Honcharuk, S. O. Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників. https://journal.itmod.ua 2022, 3, 30-45.	Успіх
APA	Popov, V.M., Petrov, R.K. & Honcharuk, S.O. (2022). Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників. Інформаційні технології та моделювання, 3(1), 30-45. вилучено із https://journal.itmod.ua /index.php/ /article/view/3-1-30	Успіх
ABNT	POPOV, V. M.; PETROV, R. K.; HONCHARUK S. O. Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників. Інформаційні технології та моделювання, [S. l.], v. 3, n. 1, p. 30-45, 2022. Disponível em: http://journal.itmod.ua/index.php/article/view/3-1-30. Acesso em: 5 сер. 2022.	Успіх

Продовження табл. 6.1

Формат	Бібліографічне посилання	Результат
Chicago	Popov, Victor V., Petrov, Robert K., і Honcharuk, Stanislav O. 2022. «Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників». *Інформаційні технології та моделювання* 3 (1): 30-45. http:// journal.itmod.ua/index.php/article/view/3-1-30.	Успіх
Harvard	Popov, V. V., Petrov, R. K. і Honcharuk S. O. (2022) «Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників», *Інформаційні технології та моделювання*, 3(1), с. 30-45. доступний у: http:// journal.itmod.ua/index.php/article/view/3-1-30 (дата звернення: 5 Серпня 2022).	Успіх
IEEE	V. V. Popov, R. K. Petrov, і S. O. Honcharuk, «Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників», *http:// journal.itmod.ua*, том 3, вип. 1, с. 30-45, Сер 2022.	Успіх
MLA	Popov, V. V., Petrov, R. K., і Honcharuk S. O. «Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників». *Інформаційні технології та моделювання*, том 3, вип. 1, Серпень 2022, с. 30-45, http:// journal.itmod.ua/index.php/article/view /3-1-30.	Успіх
Turabian	Popov, Victor V., Petrov, Robert K., і Honcharuk Stanislav O. «Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників». *Інформаційні технології та моделювання* 3, no. 1 (Серпень 5, 2022): 30-45, дата звернення Серпень 10, 2022, http:// journal.itmod.ua/index.php/article/view/3-1-30.	Успіх

Продовження табл. 6.1

Формат	Бібліографічне посилання	Результат
Vancouver	Popov VV, Petrov RK, Honcharuk SO. Ієрархічні моделі побудови складних структур даних для проведення моніторингу економічних показників. http:// journal.itmod.ua [інтернет]. 11, Вересень 2022 [цит. за 12, Серпень 2022]; 3(1): 30-45, доступний у: http:// journal.itmod.ua/index.php/article/view/3-1-30	Успіх

6.2 Перевірка часових характеристик роботи системи

Так як метою роботи є скорочення часу на пошук наукових статей, визначення фрагментів для цитування та отримання бібліографічного посилання на статті, проведемо 2 серії експериментів.

Будемо визначати та усереднювати час, який потрібен користувачам для виконання певних дій без використання системи та з нею.

У першому експерименті приймали участь 5 користувачів, кожен з яких повинен був знайти 3 статті без системи та інші 3 статті з розробленої системою. Усереднені результати часу, витраченого кожним користувачем, наведені на рис. 6.1.

Аналіз усереднених результатів витраченого часу по всім користувачам показав економію часу приблизно у 2.5 разів.

У другому експерименті приймали участь ті ж самі 5 користувачів, кожен з яких повинен був обрати по 3 будь-яких фрагмента певної статті та створити посилання за різними форматами (формат користувачі могли обрати за власним бажанням). При застосування системи користувачі використовувати статтю з власного репозиторію, без системи могли задіювати будь-які інструменти чи роботи власноруч.

Усереднені результати часу, витраченого кожним користувачем у другому експерименті, наведені на рис. 6.2.

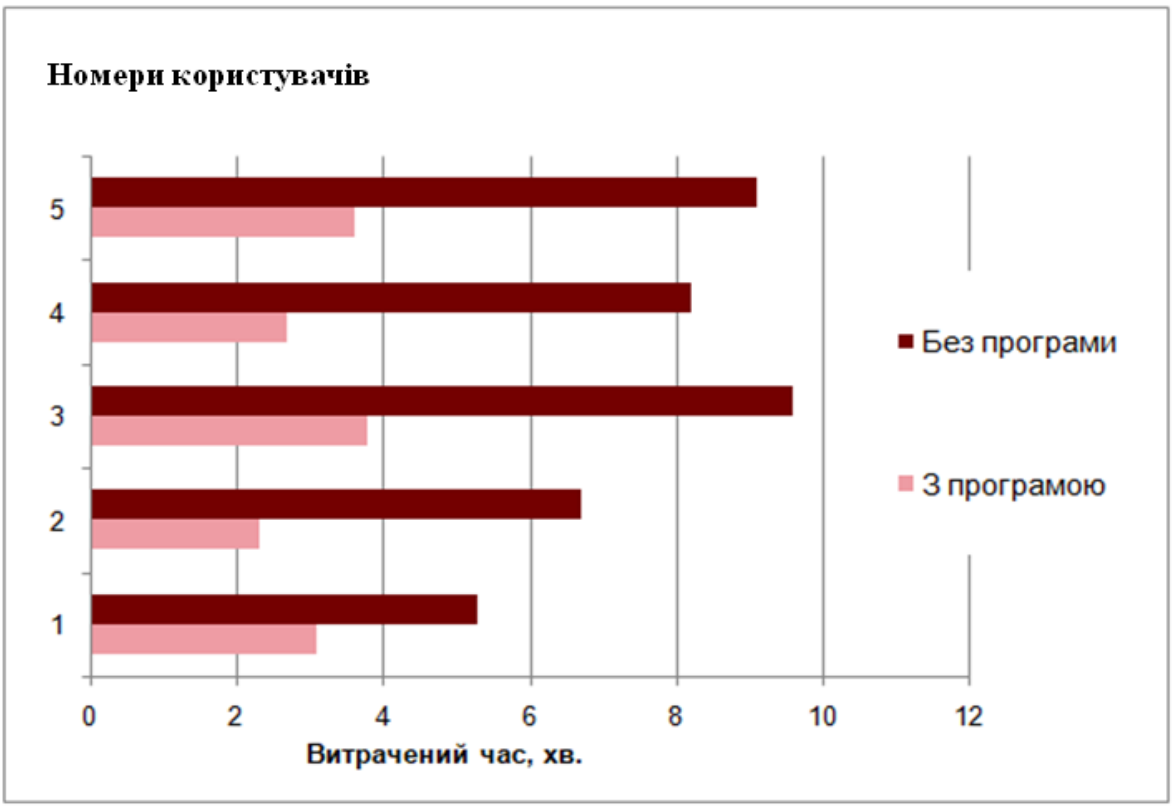

Рисунок 6.1 – Результати витрат часу при пошуку статті

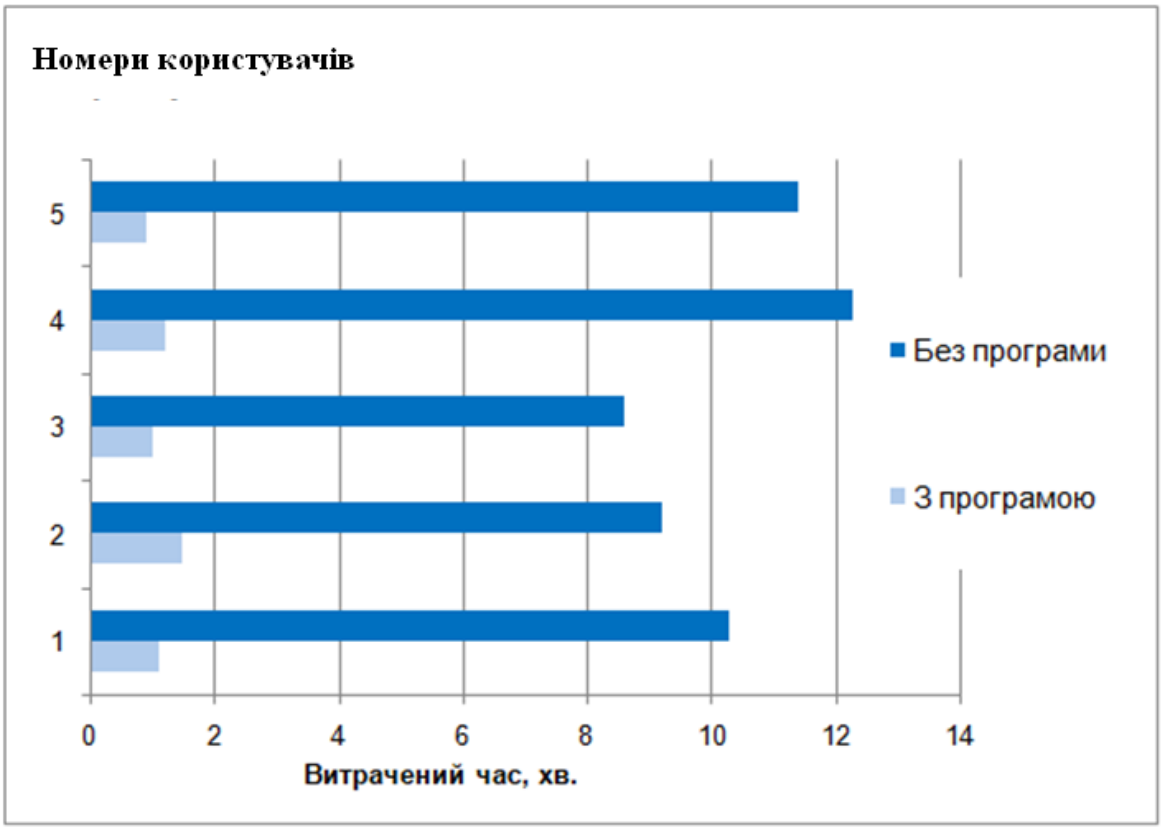

Рисунок 6.2 – Результати витрат часу при формуванні цитування

Як можна побачити, в усіх користувачів час, витрачений на формуванні цитування з використанням програми, сягав близько 1 хвилини. Аналіз усереднених результатів витраченого часу по всім користувачам показав економію часу більш ніж у 9 разів.

6.3 Висновки до розділу

У шостому розділі проведено тестування генератора посилань для цитування наукових статей чи їх фрагментів. За результатами верифікації успішно згенеровані посилання за форматами ACM, ACS, APA, ABNT, Chicago, Harvard, IEEE, MLA, Turabian та Vancouver.

Проведені експерименти по визначенню часу, який витрачається на пошук наукових статей та на формування цитувань без використання системи та з її допомогою. Результати показали, що використання системи дозволило знизити час пошуку статті приблизно у 2.5 разів та час формування цитування більш ніж у 9 разів.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмну систему для управління репозиторієм наукових статей з використанням міжнародних наукометричних баз. Експериментальна перевірка роботи з системою показала, що її використання дозволяє зменшити час пошуку статей у наукометричних базах Scopus, Web of Science та Index Copernicus приблизно у 2.5 разів та знизити час формування цитування більш ніж у 9 разів.

Для досягнення поставленої мети були виконані наступні кроки:

− вивчення предметної області, в тому числі особливостей взаємодії з міжнародними наукометричними базами;

− побудова правил для формування бібліографічних посилань на наукові статті;

− визначення переліку та способу обчислення статистичних характеристик цитувань наукових статей;

− створення канви ціннісної пропозиції з визначенням потреб використання API баз даних та структурування даних по науковим статтям, запитів на збереження часу на пошук статей та отримання бібліографічних посилань у потрібному форматі, набору робіт з визначення параметрів пошуку статей та обробка отриманих результатів, автоматизація пошуку у міжнародних наукометричних базах;

− створення бізнес-вимог та варіантів використання програмна система для управління репозиторієм, нефункціональних характеристик;

− визначення архітектури, алгоритмів роботи, структур даних та програмних класів системи;

− аналіз сучасних засобів розробки та вибір необхідних інструментів розробки, інтеграції та тестування;

− створення усіх програмних модулів з використанням зручних інструментів та фреймворків;

− функціональне тестування та визначення часових характеристик при пошуку наукових статей, визначення фрагментів для цитування та отримання бібліографічного посилання.

Впровадження створеної програмної системи надає можливість читачеві створювати власні репозиторії з наукових статей, управляти цими репозиторіями, визначати фрагменти для цитування та автоматично отримувати бібліографічне посилання відповідно до потрібного формату.

Результати роботи висвітлювались у матеріалах XXII Всеукраїнської науково–технічної конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій», що відбулась 21-22 квітня 2022 р.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вимоги до оформлення бібліографічного опису літературних джерел [Електронний ресурс]: – Режим доступу: https://www.narodnaosvita.kiev.ua/?page_id=109, вільний. – Загл. з екрану. (01.09.2022)

2. Міжнародні наукометричні бази даних та індекси цитування [Електронний ресурс]: – Режим доступу: http://www.vtei.com.ua/index.php/ua/2-uncategorised/606-mizhnarodni-naukometrychni-bazy-danykh-ta-indeksy-tsytuvannia, вільний. – Загл. з екрану. (01.09.2022)

3. Остервальдер О., Піньє Ів, Бернарда Г., Смит А. Розробляємо ціннісні пропозиції. Як створити продукти та послуги, яких хочуть клієнти. – Київ: Наш формат, 2018. – 324 с.

4. Scopus Preview [Електронний ресурс]: – Режим доступу:https://www.scopus.com/sources.uri?zone=TopNavBar&origin=AuthorProfile, вільний. – Загл. з екрану. (11.09.2022)

5. Web of Science [Електронний ресурс]: – Режим доступу: https://www.webofscience.com/wos/woscc/summary/01f5d711-e7dd-49fd-b5ba-c7894c809197-4e9f91be/relevance/1. – Загл. з екрану. (11.09.2022)

6. Indexcopernicus [Електронний ресурс]: – Режим доступу: https://indexcopernicus.com/index.php/ru/badania-3/badawczorozwojowe-3. – Загл. з екрану. (11.09.2022)

7. Fosslook [Електронний ресурс]: – Режим доступу: https://fosslook.com.ua/articles/use-fosslook. – Загл. з екрану. (11.09.2022)

8. Microsoft: створення схеми компонентів UML [Електронний ресурс]: – Режим доступу: http://surl.li/dkzdu – Загл. з екрану. (22.10.2022)

9. Основні поняття реляційних БД: нормалізація, зв’язок та ключі [Електронний ресурс]: – Режим доступу: https://bondarenko.dn.ua/osnovni-ponyattya-relyatsijnih-bd-normalizatsiya-zv-yazok-ta-klyuchi/ – Загл. з екрану. (22.10.2022)

10. Python Institute. Python® – the language of today and tomorrow [Електронний ресурс]: – Режим доступу: https://pythoninstitute.org/about-python – Загл. з екрану. (23.10.2022)

11. What is MySQL? Everything You Need to Know [Електронний ресурс]: – Режим доступу: https://ua.talend.com/resources/what-is-mysql/ – Загл. з екрану. (23.10.2022)

12. Pyscopus 1.0.3 [Електронний ресурс]: – Режим доступу: https://pypi.org/project/pyscopus/ – Загл. з екрану. (23.10.2022)

ДОДАТОК А. ЛІСТИНГ ПРОГРАМИ

```
import ipaddress
import uuid
import weakref
from datetime import date, datetime, time, timedelta
from decimal import Decimal
from enum import Enum
from pathlib import Path
from typing import (
    AbstractSet,
    Any,
    Callable,
    ClassVar,
    Dict,
    List,
    Mapping,
    Optional,
    Sequence,
    Set,
    Tuple,
    Type,
    TypeVar,
    Union,
    cast,
)

from pydantic import BaseConfig, BaseModel
from pydantic.errors import ConfigError, DictError
from pydantic.fields import SHAPE_SINGLETON
from pydantic.fields import FieldInfo as PydanticFieldInfo
from pydantic.fields import ModelField, Undefined, UndefinedType
from pydantic.main import ModelMetaclass, validate_model
from pydantic.typing import ForwardRef, NoArgAnyCallable, resolve_annotations
from pydantic.utils import ROOT_KEY, Representation
from sqlalchemy import Boolean, Column, Date, DateTime
from sqlalchemy import Enum as sa_Enum
from sqlalchemy import Float, ForeignKey, Integer, Interval, Numeric, inspect
from   sqlalchemy.orm   import   RelationshipProperty,   declared_attr,   registry,
relationship
from sqlalchemy.orm.attributes import set_attribute
from sqlalchemy.orm.decl_api import DeclarativeMeta
from sqlalchemy.orm.instrumentation import is_instrumented
from sqlalchemy.sql.schema import MetaData
from sqlalchemy.sql.sqltypes import LargeBinary, Time

from .sql.sqltypes import GUID, AutoString

_T = TypeVar("_T")


def __dataclass_transform__(
    *,
    eq_default: bool = True,
    order_default: bool = False,
    kw_only_default: bool = False,
    field_descriptors: Tuple[Union[type, Callable[..., Any]], ...] = ((),),
) -> Callable[[_T], _T]:
    return lambda a: a


class FieldInfo(PydanticFieldInfo):
```

```
    def __init__(self, default: Any = Undefined, **kwargs: Any) -> None:
        primary_key = kwargs.pop("primary_key", False)
        nullable = kwargs.pop("nullable", Undefined)
        foreign_key = kwargs.pop("foreign_key", Undefined)
        unique = kwargs.pop("unique", False)
        index = kwargs.pop("index", Undefined)
        sa_column = kwargs.pop("sa_column", Undefined)
        sa_column_args = kwargs.pop("sa_column_args", Undefined)
        sa_column_kwargs = kwargs.pop("sa_column_kwargs", Undefined)
        if sa_column is not Undefined:
            if sa_column_args is not Undefined:
                raise RuntimeError(
                    "Passing sa_column_args is not supported when "
                    "also passing a sa_column"
                )
            if sa_column_kwargs is not Undefined:
                raise RuntimeError(
                    "Passing sa_column_kwargs is not supported when "
                    "also passing a sa_column"
                )
        super().__init__(default=default, **kwargs)
        self.primary_key = primary_key
        self.nullable = nullable
        self.foreign_key = foreign_key
        self.unique = unique
        self.index = index
        self.sa_column = sa_column
        self.sa_column_args = sa_column_args
        self.sa_column_kwargs = sa_column_kwargs


class RelationshipInfo(Representation):
    def __init__(
        self,
        *,
        back_populates: Optional[str] = None,
        link_model: Optional[Any] = None,
        sa_relationship: Optional[RelationshipProperty] = None,  # type: ignore
        sa_relationship_args: Optional[Sequence[Any]] = None,
        sa_relationship_kwargs: Optional[Mapping[str, Any]] = None,
    ) -> None:
        if sa_relationship is not None:
            if sa_relationship_args is not None:
                raise RuntimeError(
                    "Passing sa_relationship_args is not supported when "
                    "also passing a sa_relationship"
                )
            if sa_relationship_kwargs is not None:
                raise RuntimeError(
                    "Passing sa_relationship_kwargs is not supported when "
                    "also passing a sa_relationship"
                )
        self.back_populates = back_populates
        self.link_model = link_model
        self.sa_relationship = sa_relationship
        self.sa_relationship_args = sa_relationship_args
        self.sa_relationship_kwargs = sa_relationship_kwargs


def Field(
    default: Any = Undefined,
    *,
    default_factory: Optional[NoArgAnyCallable] = None,
    alias: Optional[str] = None,
```

```
    title: Optional[str] = None,
    description: Optional[str] = None,
    exclude: Union[
        AbstractSet[Union[int, str]], Mapping[Union[int, str], Any], Any
    ] = None,
    include: Union[
        AbstractSet[Union[int, str]], Mapping[Union[int, str], Any], Any
    ] = None,
    const: Optional[bool] = None,
    gt: Optional[float] = None,
    ge: Optional[float] = None,
    lt: Optional[float] = None,
    le: Optional[float] = None,
    multiple_of: Optional[float] = None,
    min_items: Optional[int] = None,
    max_items: Optional[int] = None,
    min_length: Optional[int] = None,
    max_length: Optional[int] = None,
    allow_mutation: bool = True,
    regex: Optional[str] = None,
    primary_key: bool = False,
    foreign_key: Optional[Any] = None,
    unique: bool = False,
    nullable: Union[bool, UndefinedType] = Undefined,
    index: Union[bool, UndefinedType] = Undefined,
    sa_column: Union[Column, UndefinedType] = Undefined,  # type: ignore
    sa_column_args: Union[Sequence[Any], UndefinedType] = Undefined,
    sa_column_kwargs: Union[Mapping[str, Any], UndefinedType] = Undefined,
    schema_extra: Optional[Dict[str, Any]] = None,
) -> Any:
    current_schema_extra = schema_extra or {}
    field_info = FieldInfo(
        default,
        default_factory=default_factory,
        alias=alias,
        title=title,
        description=description,
        exclude=exclude,
        include=include,
        const=const,
        gt=gt,
        ge=ge,
        lt=lt,
        le=le,
        multiple_of=multiple_of,
        min_items=min_items,
        max_items=max_items,
        min_length=min_length,
        max_length=max_length,
        allow_mutation=allow_mutation,
        regex=regex,
        primary_key=primary_key,
        foreign_key=foreign_key,
        unique=unique,
        nullable=nullable,
        index=index,
        sa_column=sa_column,
        sa_column_args=sa_column_args,
        sa_column_kwargs=sa_column_kwargs,
        **current_schema_extra,
    )
    field_info_validate()
    return field_info
```

```
def Relationship(
    *,
    back_populates: Optional[str] = None,
    link_model: Optional[Any] = None,
    sa_relationship: Optional[RelationshipProperty] = None,  # type: ignore
    sa_relationship_args: Optional[Sequence[Any]] = None,
    sa_relationship_kwargs: Optional[Mapping[str, Any]] = None,
) -> Any:
    relationship_info = RelationshipInfo(
        back_populates=back_populates,
        link_model=link_model,
        sa_relationship=sa_relationship,
        sa_relationship_args=sa_relationship_args,
        sa_relationship_kwargs=sa_relationship_kwargs,
    )
    return relationship_info


@__dataclass_transform__(kw_only_default=True,           field_descriptors=(Field,
FieldInfo))
class SQLModelMetaclass(ModelMetaclass, DeclarativeMeta):
    __sqlmodel_relationships__: Dict[str, RelationshipInfo]
    __config__: Type[BaseConfig]
    __fields__: Dict[str, ModelField]

    # Replicate SQLAlchemy
    def __setattr__(cls, name: str, value: Any) -> None:
        if getattr(cls.__config__, "table", False):
            DeclarativeMeta.__setattr__(cls, name, value)
        else:
            super().__setattr__(name, value)

    def __delattr__(cls, name: str) -> None:
        if getattr(cls.__config__, "table", False):
            DeclarativeMeta.__delattr__(cls, name)
        else:
            super().__delattr__(name)

    # From Pydantic
    def __new__(
        cls,
        name: str,
        bases: Tuple[Type[Any], ...],
        class_dict: Dict[str, Any],
        **kwargs: Any,
    ) -> Any:
        relationships: Dict[str, RelationshipInfo] = {}
        dict_for_pydantic = {}
        original_annotations = resolve_annotations(
            class_dict.get("__annotations__",   {}),   class_dict.get("__module__",
None)
        )
        pydantic_annotations = {}
        relationship_annotations = {}
        for k, v in class_dict.items():
            if isinstance(v, RelationshipInfo):
                relationships[k] = v
            else:
                dict_for_pydantic[k] = v
        for k, v in original_annotations.items():
            if k in relationships:
                relationship_annotations[k] = v
            else:
```

```
                pydantic_annotations[k] = v
        dict_used = {
            **dict_for_pydantic,
            "__weakref__": None,
            "__sqlmodel_relationships__": relationships,
            "__annotations__": pydantic_annotations,
        }
        # Duplicate logic from Pydantic to filter config kwargs because if they
are
        # passed directly including the registry Pydantic will pass them over to
the
        # superclass causing an error
        allowed_config_kwargs: Set[str] = {
            key
            for key in dir(BaseConfig)
            if not (
                key.startswith("__") and key.endswith("__")
            )  # skip dunder methods and attributes
        }
        pydantic_kwargs = kwargs.copy()
        config_kwargs = {
            key: pydantic_kwargs.pop(key)
            for key in pydantic_kwargs.keys() & allowed_config_kwargs
        }
        new_cls = super().__new__(cls, name, bases, dict_used, **config_kwargs)
        new_cls.__annotations__ = {
            **relationship_annotations,
            **pydantic_annotations,
            **new_cls.__annotations__,
        }

        def get_config(name: str) -> Any:
            config_class_value = getattr(new_cls.__config__, name, Undefined)
            if config_class_value is not Undefined:
                return config_class_value
            kwarg_value = kwargs.get(name, Undefined)
            if kwarg_value is not Undefined:
                return kwarg_value
            return Undefined

        config_table = get_config("table")
        if config_table is True:
            # If it was passed by kwargs, ensure it's also set in config
            new_cls.__config__.table = config_table
            for k, v in new_cls.__fields__.items():
                col = get_column_from_field(v)
                setattr(new_cls, k, col)
            # Set a config flag to tell FastAPI that this should be read with a
field
            # in orm_mode instead of preemptively converting it to a dict.
            # This could be done by reading new_cls.__config__.table in FastAPI,
but
            # that's very specific about SQLModel, so let's have another config
that
            # other future tools based on Pydantic can use.
            new_cls.__config__.read_with_orm_mode = True

        config_registry = get_config("registry")
        if config_registry is not Undefined:
            config_registry = cast(registry, config_registry)
            # If it was passed by kwargs, ensure it's also set in config
            new_cls.__config__.registry = config_table
            setattr(new_cls, "_sa_registry", config_registry)
            setattr(new_cls, "metadata", config_registry.metadata)
```

```
            setattr(new_cls, "__abstract__", True)
        return new_cls

    # Override SQLAlchemy, allow both SQLAlchemy and plain Pydantic models
    def __init__(
        cls, classname: str, bases: Tuple[type, ...], dict_: Dict[str, Any], **kw:
Any
    ) -> None:
        # Only one of the base classes (or the current one) should be a table
model
        # this allows FastAPI cloning a SQLModel for the response_model without
        # trying to create a new SQLAlchemy, for a new table, with the same name,
that
        # triggers an error
        base_is_table = False
        for base in bases:
            config = getattr(base, "__config__")
            if config and getattr(config, "table", False):
                base_is_table = True
                break
        if getattr(cls.__config__, "table", False) and not base_is_table:
            dict_used = dict_.copy()
            for field_name, field_value in cls.__fields__.items():
                dict_used[field_name] = get_column_from_field(field_value)
            for rel_name, rel_info in cls.__sqlmodel_relationships__.items():
                if rel_info.sa_relationship:
                    #  There's  a  SQLAlchemy  relationship  declared,  that  takes
precedence
                    # over anything else, use that and continue with the next
attribute
                    dict_used[rel_name] = rel_info.sa_relationship
                    continue
                ann = cls.__annotations__[rel_name]
                temp_field = ModelField.infer(
                    name=rel_name,
                    value=rel_info,
                    annotation=ann,
                    class_validators=None,
                    config=BaseConfig,
                )
                relationship_to = temp_field.type_
                if isinstance(temp_field.type_, ForwardRef):
                    relationship_to = temp_field.type_.__forward_arg__
                rel_kwargs: Dict[str, Any] = {}
                if rel_info.back_populates:
                    rel_kwargs["back_populates"] = rel_info.back_populates
                if rel_info.link_model:
                    ins = inspect(rel_info.link_model)
                    local_table = getattr(ins, "local_table")
                    if local_table is None:
                        raise RuntimeError(
                            "Couldn't find the secondary table for "
                            f"model {rel_info.link_model}"
                        )
                    rel_kwargs["secondary"] = local_table
                rel_args: List[Any] = []
                if rel_info.sa_relationship_args:
                    rel_args.extend(rel_info.sa_relationship_args)
                if rel_info.sa_relationship_kwargs:
                    rel_kwargs.update(rel_info.sa_relationship_kwargs)
                rel_value: RelationshipProperty = relationship(  # type: ignore
                    relationship_to, *rel_args, **rel_kwargs
                )
                dict_used[rel_name] = rel_value
```

```
                setattr(cls, rel_name, rel_value)  # Fix #315
            DeclarativeMeta.__init__(cls, classname, bases, dict_used, **kw)
        else:
            ModelMetaclass.__init__(cls, classname, bases, dict_, **kw)


def get_sqlalchemy_type(field: ModelField) -> Any:
    if issubclass(field.type_, str):
        if field.field_info.max_length:
            return AutoString(length=field.field_info.max_length)
        return AutoString
    if issubclass(field.type_, float):
        return Float
    if issubclass(field.type_, bool):
        return Boolean
    if issubclass(field.type_, int):
        return Integer
    if issubclass(field.type_, datetime):
        return DateTime
    if issubclass(field.type_, date):
        return Date
    if issubclass(field.type_, timedelta):
        return Interval
    if issubclass(field.type_, time):
        return Time
    if issubclass(field.type_, Enum):
        return sa_Enum(field.type_)
    if issubclass(field.type_, bytes):
        return LargeBinary
    if issubclass(field.type_, Decimal):
        return Numeric(
            precision=getattr(field.type_, "max_digits", None),
            scale=getattr(field.type_, "decimal_places", None),
        )
    if issubclass(field.type_, ipaddress.IPv4Address):
        return AutoString
    if issubclass(field.type_, ipaddress.IPv4Network):
        return AutoString
    if issubclass(field.type_, ipaddress.IPv6Address):
        return AutoString
    if issubclass(field.type_, ipaddress.IPv6Network):
        return AutoString
    if issubclass(field.type_, Path):
        return AutoString
    if issubclass(field.type_, uuid.UUID):
        return GUID
    raise ValueError(f"The field {field.name} has no matching SQLAlchemy type")


def get_column_from_field(field: ModelField) -> Column:  # type: ignore
    sa_column = getattr(field.field_info, "sa_column", Undefined)
    if isinstance(sa_column, Column):
        return sa_column
    sa_type = get_sqlalchemy_type(field)
    primary_key = getattr(field.field_info, "primary_key", False)
    index = getattr(field.field_info, "index", Undefined)
    if index is Undefined:
        index = False
    nullable = not primary_key and _is_field_noneable(field)
    # Override derived nullability if the nullable property is set explicitly
    # on the field
    if hasattr(field.field_info, "nullable"):
        field_nullable = getattr(field.field_info, "nullable")
        if field_nullable != Undefined:
```

```
            nullable = field_nullable
    args = []
    foreign_key = getattr(field.field_info, "foreign_key", None)
    unique = getattr(field.field_info, "unique", False)
    if foreign_key:
        args.append(ForeignKey(foreign_key))
    kwargs = {
        "primary_key": primary_key,
        "nullable": nullable,
        "index": index,
        "unique": unique,
    }
    sa_default = Undefined
    if field.field_info.default_factory:
        sa_default = field.field_info.default_factory
    elif field.field_info.default is not Undefined:
        sa_default = field.field_info.default
    if sa_default is not Undefined:
        kwargs["default"] = sa_default
    sa_column_args = getattr(field.field_info, "sa_column_args", Undefined)
    if sa_column_args is not Undefined:
        args.extend(list(cast(Sequence[Any], sa_column_args)))
    sa_column_kwargs = getattr(field.field_info, "sa_column_kwargs", Undefined)
    if sa_column_kwargs is not Undefined:
        kwargs.update(cast(Dict[Any, Any], sa_column_kwargs))
    return Column(sa_type, *args, **kwargs)  # type: ignore


class_registry = weakref.WeakValueDictionary()  # type: ignore

default_registry = registry()


def _value_items_is_true(v: Any) -> bool:
    # Re-implement Pydantic's ValueItems.is_true() as it hasn't been released as
of
    # the current latest, Pydantic 1.8.2
    return v is True or v is ...


_TSQLModel = TypeVar("_TSQLModel", bound="SQLModel")


class SQLModel(BaseModel, metaclass=SQLModelMetaclass, registry=default_registry):
    # SQLAlchemy needs to set weakref(s), Pydantic will set the other slots values
    __slots__ = ("__weakref__",)
    __tablename__: ClassVar[Union[str, Callable[..., str]]]
    __sqlmodel_relationships__: ClassVar[Dict[str, RelationshipProperty]]  # type:
ignore
    __name__: ClassVar[str]
    metadata: ClassVar[MetaData]

    class Config:
        orm_mode = True

    def __new__(cls, *args: Any, **kwargs: Any) -> Any:
        new_object = super().__new__(cls)
        # SQLAlchemy doesn't call __init__ on the base class
        # Ref: https://docs.sqlalchemy.org/en/14/orm/constructors.html
        # Set __fields_set__ here, that would have been set when calling __init__
        # in the Pydantic model so that when SQLAlchemy sets attributes that are
        # added (e.g. when querying from DB) to the __fields_set__, this already
exists
        object.__setattr__(new_object, "__fields_set__", set())
```

```
        return new_object

    def __init__(__pydantic_self__, **data: Any) -> None:
        # Uses something other than `self` the first arg to allow "self" as a
        # settable attribute
        values, fields_set, validation_error = validate_model(
            __pydantic_self__.__class__, data
        )
        # Only raise errors if not a SQLModel model
        if (
            not getattr(__pydantic_self__.__config__, "table", False)
            and validation_error
        ):
            raise validation_error
        # Do not set values as in Pydantic, pass them through setattr, so
SQLAlchemy
        # can handle them
        # object.__setattr__(__pydantic_self__, '__dict__', values)
        for key, value in values.items():
            setattr(__pydantic_self__, key, value)
        object.__setattr__(__pydantic_self__, "__fields_set__", fields_set)
        non_pydantic_keys = data.keys() - values.keys()
        for key in non_pydantic_keys:
            if key in __pydantic_self__.__sqlmodel_relationships__:
                setattr(__pydantic_self__, key, data[key])

    def __setattr__(self, name: str, value: Any) -> None:
        if name in {"_sa_instance_state"}:
            self.__dict__[name] = value
            return
        else:
            # Set in SQLAlchemy, before Pydantic to trigger events and updates
            if getattr(self.__config__, "table", False) and is_instrumented(self,
name):
                set_attribute(self, name, value)
            # Set in Pydantic model to trigger possible validation changes, only
for
            # non relationship values
            if name not in self.__sqlmodel_relationships__:
                super().__setattr__(name, value)

    @classmethod
    def from_orm(
        cls: Type[_TSQLModel], obj: Any, update: Optional[Dict[str, Any]] = None
    ) -> _TSQLModel:
        # Duplicated from Pydantic
        if not cls.__config__.orm_mode:
            raise ConfigError(
                "You must have the config attribute orm_mode=True to use from_orm"
            )
        obj = {ROOT_KEY: obj} if cls.__custom_root_type__ else
cls._decompose_class(obj)
        # SQLModel, support update dict
        if update is not None:
            obj = {**obj, **update}
        # End SQLModel support dict
        if not getattr(cls.__config__, "table", False):
            # If not table, normal Pydantic code
            m: _TSQLModel = cls.__new__(cls)
        else:
            # If table, create the new instance normally to make SQLAlchemy create
            # the _sa_instance_state attribute
            m = cls()
        values, fields_set, validation_error = validate_model(cls, obj)
```

```
        if validation_error:
            raise validation_error
        # Updated to trigger SQLAlchemy internal handling
        if not getattr(cls.__config__, "table", False):
            object.__setattr__(m, "__dict__", values)
        else:
            for key, value in values.items():
                setattr(m, key, value)
        # Continue with standard Pydantic logic
        object.__setattr__(m, "__fields_set__", fields_set)
        m._init_private_attributes()
        return m

    @classmethod
    def parse_obj(
        cls: Type[_TSQLModel], obj: Any, update: Optional[Dict[str, Any]] = None
    ) -> _TSQLModel:
        obj = cls._enforce_dict_if_root(obj)
        # SQLModel, support update dict
        if update is not None:
            obj = {**obj, **update}
        # End SQLModel support dict
        return super().parse_obj(obj)

    def __repr_args__(self) -> Sequence[Tuple[Optional[str], Any]]:
        # Don't show SQLAlchemy private attributes
        return [(k, v) for k, v in self.__dict__.items() if not
k.startswith("_sa_")]

    # From Pydantic, override to enforce validation with dict
    @classmethod
    def validate(cls: Type[_TSQLModel], value: Any) -> _TSQLModel:
        if isinstance(value, cls):
            return value.copy() if cls.__config__.copy_on_model_validation else
value

        value = cls._enforce_dict_if_root(value)
        if isinstance(value, dict):
            values, fields_set, validation_error = validate_model(cls, value)
            if validation_error:
                raise validation_error
            model = cls(**value)
            # Reset fields set, this would have been done in Pydantic in __init__
            object.__setattr__(model, "__fields_set__", fields_set)
            return model
        elif cls.__config__.orm_mode:
            return cls.from_orm(value)
        elif cls.__custom_root_type__:
            return cls.parse_obj(value)
        else:
            try:
                value_as_dict = dict(value)
            except (TypeError, ValueError) as e:
                raise DictError() from e
            return cls(**value_as_dict)

    # From Pydantic, override to only show keys from fields, omit SQLAlchemy
attributes
    def _calculate_keys(
        self,
        include: Optional[Mapping[Union[int, str], Any]],
        exclude: Optional[Mapping[Union[int, str], Any]],
        exclude_unset: bool,
        update: Optional[Dict[str, Any]] = None,
```

```
    ) -> Optional[AbstractSet[str]]:
        if include is None and exclude is None and not exclude_unset:
            # Original in Pydantic:
            # return None
            # Updated to not return SQLAlchemy attributes
            # Do not include relationships as that would easily lead to infinite
            # recursion, or traversing the whole database
            return          self.__fields__.keys()                          #           |
self.__sqlmodel_relationships__.keys()

        keys: AbstractSet[str]
        if exclude_unset:
            keys = self.__fields_set__.copy()
        else:
            # Original in Pydantic:
            # keys = self.__dict__.keys()
            # Updated to not return SQLAlchemy attributes
            # Do not include relationships as that would easily lead to infinite
            # recursion, or traversing the whole database
            keys        =        self.__fields__.keys()                    #         |
self.__sqlmodel_relationships__.keys()
        if include is not None:
            keys &= include.keys()

        if update:
            keys -= update.keys()

        if exclude:
            keys -= {k for k, v in exclude.items() if _value_items_is_true(v)}

        return keys

    @declared_attr  # type: ignore
    def __tablename__(cls) -> str:
        return cls.__name__.lower()


def _is_field_noneable(field: ModelField) -> bool:
    if not field.required:

        return field.allow_none and (
            field.shape != SHAPE_SINGLETON or not field.sub_fields
        )
    return False
```