**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»**
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

Наталія Аушева

«___»_____2022 р.

# Дипломна робота

**на здобуття ступеня бакалавра**

**спеціальності 122 «Комп'ютерні науки»**

**освітня програма «Комп'ютерний моніторинг та геометричне моделювання процесів та систем»**

**на тему: «Збір та первинна обробка бібліографічної інформації»**

Виконав :
студент IV курсу, групи ТР-82
Василенко Микита Андрійович _____

Керівник:
Доцент
Кузьміних Валерій Олександрович _____

Рецензент:
Директор,НВП «Символ», к.т.н.
Сенченко В'ячеслав Родіонович _____

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

Київ – 2022

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 122 «Комп’ютерні науки»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри

____ Наталія Аушева
(підпис)

“___”_____2022 р.

# ЗАВДАННЯ

## на дипломну роботу студенту

Василенко Микита Андрійович
(прізвище, ім’я, по батькові)

1. Тема роботи «Збір та первинна обробка бібліографічної інформації»

керівник роботи Кузьміних Валерій Олександрович, доцент
(прізвище, ім’я, по батькові, вчене звання)

затверджена наказом вищого навчального закладу від “__” травня 2022р. № _____

2. Строк подання студентом роботи _10 червня 2022 р._____
3. Вихідні дані до роботи персональний комп’ютер під управлінням операційної системи Windows 10, мова програмування Python, середовище розробки PyCharm, система управління базами даних SQLite_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розробити системи для пошуку та збору бібліографічної інформації для оцінки рівня міжнародної діяльності та первинної обробки даних; дослідити існуючі аналоги

5. Перелік ілюстративного матеріалу ілюстрації аналогічних систем, діаграма прецедентів системи, структурна схема відношень між таблицями бази даних, діаграма класів системи, ілюстрації роботи системи

6. Консультація розділів роботи

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

7. Дата видачі завдання 10 вересня 2021 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 25.05.2022 | |
| 2. | Вивчення та аналіз задачі | 02.03.2022-<br>08.03.2022 | |
| 3. | Розробка архітектури та загальної структури системи | 09.03.2022-<br>15.03.2022 | |
| 4. | Розробка структур окремих підсистем | 20.03.2022-<br>29.03.2022 | |
| 5. | Програмна реалізація системи | 01.04.2022-<br>16.04.2022 | |
| 6. | Оформлення пояснювальної записки | 05.05.2022-<br>19.05.2022 | |
| 7. | Захист програмного продукту | 20.05.2022 | |
| 8. | Передзахист | 06.06.2022-<br><br>09.06.2022р | |
| 9. | Захист | 10.06.2022 | |

Студент _____ Василенко М.А.
(підпис) (прізвище та ініціали.)

Керівник роботи _____ Кузьміних В.О.
(підпис) (прізвище та ініціали.)

# АНОТАЦІЯ

Структура та обсяг роботи. Робота містить 47 сторінок, 24 рисунки, 11 літературних джерел, 1 додаток.

Дипломна робота присвячена розробці програмного забезпечення, основною функцією якого є збір, зберігання та первинна обробка бібліографічної інформації.

Метою збору бібліографічної інформації є аналіз та оцінка міжнародної наукової діяльності задля подальшого її зберігання необхідних даних у базі даних. Збір даних виконується с декількох джерел для кращого порівняння даних між собою.

Програмний продукт створено у середовищі PyCharm на мові програмування Python. Для створення графічного інтерфейсу користувача було використано бібліотеку PyQt5. Для збереження даних у базу даних було використано систему керування базами даних MySQL.

Ключові слова : парсинг, бібліографічні джерела інформації, збір бібліографічної інформації, інтелектуальний аналіз інформації, статистичний аналіз.

# ABSTRACT

Structure and scope of work. The work contains 47 pages, 24 drawings, 11 literature sources, 1 appendice.

Thesis is devoted to software development, the main function of which is the collection, storage and primary processing of bibliographic information.

The purpose of collecting bibliographic information is to analyze and evaluate international scientific activities in order to further store the necessary data in the database. Data collection is performed from several sources for better comparison of data with each other.

The software product was created in the PyCharm environment with the Python programming language. The PyQt5 library was used to create the graphical user interface. A MySQL database management system was used to store the data in the database.

**Key words:** parsing, bibliographic sources of information, collection of bibliographic information, intellectual analysis of information, statistical analysis.

# Зміст

# ВСТУП

З давніх часів людство пробувало полегшити своє існування, випробовуючи щось нове, те чого раніше ніхто не робив. Проте не всі ці експерименти закінчувалися так, як на це сподівалася людина. Таким чином з'явилася необхідність запам'ятовувати результати попередніх спроб, для подальшого аналізу та виправлення помилок. Звідси людство почало документувати результати своїх досліджень, починаючи з базових " наскальних малюнків ".

Самі ж висновки залежать від спостережень, які людина отримує під час випробувань своєї гіпотези. Але ідеї та думки у кожного різні, а отже й теми досліджень, можуть відрізнятися, як й висновки щодо одного й того самого дослідження. Отже виникла необхідність обмінюватись набутим досвідом між людьми задля швидшого подальшого розвитку. Так обмінюючись досвідом люди почали поступово розвивати науку та своє суспільство.

З розвитком технологій, люди почали розвивати методи збереження інформації, а саме зберігати її більш стисло та безпечно. Починаючи з наскальних малюнків, люди почали передавати набутий досвід нащадкам. Після було винайдено писемність а з часом й було винайдено проривний на той час метод збереження інформації – папір. Так почалась накопичуватися інформація , створюватись книжки , що призвело до розвитку науково-технічного прогресу.

Зі збільшенням кількості записів про результати досліджень покращення методів зберігання інформації. Зокрема з'явилася необхідність їх розподіляти, оскільки дослідження стосувались певних питань, які інтересували лише визначене коло людей. Також з'явилася необхідність сортування цих записів, оскільки попередні спостереження можуть втратити свою актуальність у зв'язку з новими результатами досліджень у визначеному питанні, або зміною технологій вирішення у даному питанні.

З розвитком комунікаційних систем, а саме засобів передавання інформації людьми поміж собою. В наслідок цього науковці почали більш вільно поширювати

результати своєї праці, а інші – набагато легше отримувати ці роботи майже у будь якому місці та будь який час.

Видатні дослідники, результати дослідів яких були проривними - ніколи не залишалися без уваги. На їх роботи почали спиратися нові науковці та починають цитувати роботи визначних науковців, з метою доведення доречності своїх висновків, доповнення або спростування висновків інших науковців та запропонування своїх думок, теорій та рішень.

Але саме через відносно вільний доступ до інформації завжди виникає проблема, яка може повністю знецінити напрацювання, які могли коштувати науковцю багато років, місяців, років або навіть майже усе життя. З метою прославитися серед наукової спільноти , нажити собі грошей, або через звичайнісіньку лінь – одна людина може вкрасти працю іншої та видати чужі заслуги за свої.

Через велику кількість збереженої людством інформації, стає набагато складніше проаналізувати усі напрацювання інших науковців у цьому напрямі та за наявності знайти проблеми, які пов’язані зі спробою не справедливо покращити свою роботу. Саме виходячи з цієї проблеми з’явилось питання академічної доброчесності.

Академічна доброчесність – це сукупність морально-етичних принципів та визначених науковою спільнотою та законом правил, за якими керуються усі учасники наукової спільноти та освітнього процесу. Вона включає в себе:

- здобувачів освіти будь-якого рівня;
- викладачів;
- науковців;
- видавців навчальної та наукової літератури;
- установ здобуття освіти;

Грубе порушення принципів академічної доброчесності може поставити під сумнів досягнення не тільки не самого автора праці, але й установу, з якою він безпосередньо пов’язаний та створити скандал, який спричинить за собою багато

небажаних наслідків. Грубими порушеннями академічної доброчесності вважаються:

- академічний плагіат;
- самоплагіат;
- хабарництво;
- фабрикація;
- обман;
- списування;
- надання перешкод або допомоги, які не передбачені умовами;
- необ’єктивне оцінювання;

Зі створенням перших електронно-обчислювальних машин людство почало створення нового способу збереження та отримання необхідної інформації, який в перспективі був більш надійнішим та швидшим за свої аналоги. Поступово науковці почали покращувати ці технології та розповсюджувати їх в маси. Зі створенням всесвітньої системи сполучених комп’ютерних мереж Інтернет та розповсюдженням доступу до неї, почалася нова ера обміну інформацією.

Наукові журнали, видання, бібліотеки – усі можливі сховища інформації побачивши усі переваги збереження інформації у електронному вигляді почали переносити збережену людством інформацію у електронний формат збереження даних, завдяки якому ми зберігаємо та відтворюємо інформацію й досі.

# 1 ОПИС ПРОБЛЕМИ

Для успішного результату подальших дослідів науковцям необхідно проаналізувати велику кількість інформації, у сферах, які безумовно пов’язані з дослідженнями, а саме попередні дослідження інших видатних науковців, виокремити з них усе необхідне та зберегти для подальшого цитування.

Аналітику необхідно надати доступ до ресурсів, попередньо створивши графік, за яким буде можливо якісно та з урахуванням усіх можливих перешкод виконати поставлену задачу по збору та обробці необхідної інформації.

Процес аналізу великої кількості інформації аналітиками може відрізнятися в залежності від необхідної кількості матеріалів для аналізу та створення записів, моменту отримання доступу до інформації та самої компетентності аналітика у обраній сфері.

Проте також важливо пам’ятати те, що якщо занадто поквапити аналітика, потребуючи швидкий результат, є можливість отримати неточну, хибну або зовсім непотрібну інформацію, тим самим збільшуючи затримок у виконанні графіку або взагалі провалі в дослідженні.

Платформа для збирання інформації повинна містити велику кількість робіт у необхідному та пов’язаних напрямках науки, які будуть застосовані у майбутніх дослідженнях та зберігати базову інформацію про статті, їх авторів, роки публікацій та джерела інші інформації, які використовувались при написанні цих статей.

Зокрема необхідно, аби платформа вміщала в себе нові, ще не визнані науковою спільнотою належним чином роботи, з метою знаходження нових, перспективних ідей та підходів вирішення існуючих задач, в яких в майбутньому може скластися конкурентоспроможний проект та може бути цікавим для спонсорів. Також за бажанням має міститись деякий параметр, за яким можливо визначити актуальність цієї статті серед інших дослідників та аналітиків.

Але ця інформація може одночасно знадобитись багатьом аналітикам та не усі можуть знаходитись у місці, де ця інформація фактично зберігається. Для цього було створено бібліографічні бази даних, де вже зберігається велика кількість робіт та базову інформацію про них.

Бібліографічні бази даних – система на якій зберігають велику кількість інформації, яка дозволяє науковцям з усього світу обмінюватись досягненнями у науковому світі. Зазвичай вони зберігають дані у вигляді каталогу, який містить посилання на напрацювання науковців, помічаючи їх позначками для більш легкого пошуку визначених робіт серед інших.

Доступ до даних , які містять ці бібліографічні бази даних кожен може отримати через міжнародну комп'ютерну мережу Інтернет. Цими базами даних ми користуємося й по цей час.

Бібліографічні дані дуже часто використовують у сфері аналітики діяльності для певних об'єднань членів наукової спільноти. Збираючи бібліографічну інформацію аналітики можуть оцінити дослідження певних об'єднань членів наукової спільноти задля подальшого їх порівняння з іншими науковцями.

Отримуючи інформацію про дослідження цієї групи людей , аналітики можуть оцінити перспективність визначених об'єднань у певній галузі досліджень. Це дає змогу розподіляти доцільно бюджет організації та залучати зацікавлених спонсорів.

Ці бібліографічні бази даних дають змогу користувачу знаходити статті у будь якому місці та майже у будь який час. Бібліографічні бази даних зберігають усю накопичену інформацію на комп'ютерах з великими обчислювальними та дисковими ресурсами, а саме на серверах, які дозволяють одночасно працювати на визначеній базі даних багатьом користувачам одночасно.

Пошук зазвичай виконується за заздалегідь визначеними розробниками сценаріями пошуку через так названі фільтри пошуку. Чим досконаліше класифікована інформація в базі даних та чим краще реалізовані фільтри пошуку – тим швидше та більш вдало користувач отримає саме ту інформацію, яку шукає.

Але погляди на те, як система має бути реалізована, що вона має зберігати, за якими параметрами користувач буде шукати необхідну йому інформацію та як класифікувати інформацію зазвичай кардинально відрізняються, а отже може виникати проблема невизначеності деяких необхідних даних в результаті пошуку. Саме тому аналітику необхідно проводити аналіз діяльності на декількох ресурсах одночасно та зводити результати до єдиного вигляду.

Для кожного робітника у будь-якій сфері є дуже важливий ресурс, яким треба користуватись зважливо та раціонально. Цим ресурсом є час. Проведення аналітики вручну це дуже клопіткий процес, який потребую дуже багато часу та терпіння, оскільки аналітику необхідно:

- вишукувати необхідні статті на кожному ресурсі;
- переглядати кожну з статей, виокремлювати з них ті, які відповідають побажанням аналітика;
- робити первинну обробку інформації, а саме: збирати з необхідних статей лише необхідну інформацію, перевіряти на дублювання зібраної інформації, зводити інформацію до єдиного вигляду;

Звідси виникає потреба в системі, яка буде автоматизувати процес збору та первинної обробки інформації, а отже й полегшить та зробить більш продуктивним процес збору інформації, яка необхідна аналітику для подальшої роботи з бібліографічною інформацією.

# 2 ІСНУЮЧІ БІБЛІОГРАФІЧНІ БАЗИ ДАНИХ

## 2.1 Перелік найвідоміших бібліографічних баз даних

Існує багато рішень, створених спеціально для пошуку бібліографічної інформації, які широко використовуються науковцями з усього світу для зберігання своїх статей та напрацювань.

Найбільш відомими рішеннями на даний час є:

- Google Scholar
- Scopus
- Web of Science
- ScienceDirect
- ArXiv(Cornell University)
- Wolfram Alpha

У кожної бібліографічної бази даних своя структура та різна кількість статей у різних напрямках, тому виникає можливість, що деякі статті будуть ексклюзивними для певної платформи, або навпаки бути відсутніми на ній, а отже для аналізу необхідно обрати серед великої кількості бібліографічних баз даних мінімум дві діаметрально протилежні, але в той самий час доступні для широкого кола науковців та звести їх дані до єдиної системи.

## 2.2 Платформа Google Scholar

Google Академія – є наукометричною база даних з відкритим доступом, яку було створено найпотужнішою пошуковою системою Google.

Розробник сервісу – науковець індійського походження Анураг Ачарья (англ. Anurag Acharya). Метою створення бази була допомога аналітичній та академічній спільноті.

Цей інструмент дозволяє аналітикам проводити пошук серед широкого спектру наукової літератури , яка знаходиться у Інтернет-просторі , а саме:

- Тез
- Рецензованих статей
- Книг
- Дисертацій
- Презентацій
- Технічних звітів професійних спільнот
- Технічних звітів з університетів
- Наукових журналів
- Препринтів
- Рефератів
- Технічних звітів з академічних інститутів
- Технічних звітів науково-дослідницьких груп

На даний момент Google Академія має найбільшу кількість наукової літератури, а звідси й охоплює найбільшу кількість галузей досліджень. Станом на 2016 рік її база охоплювала 160 мільйонів документів з унікальним значенням. Цей показник в декілька разів більший за конкуруючих з нею баз даних Scopus та Web of Science.

Google Академія вміщає в себе як і відомості про документи, у яких доступні лише бібліографія або реферат, так і повнотекстові матеріали, якщо матеріали повністю знаходяться у вільному доступі для читача. Зокрема сервіс забезпечує користувача деякими додатковими даними як:

- індекс цитування документів;
- посилання на авторів статті;
- зовнішні посилання на матеріали, які використовувались у цій статті

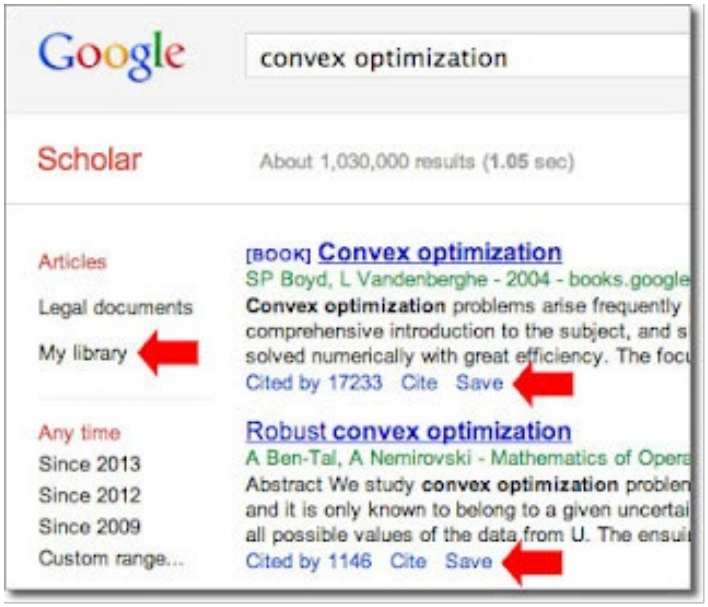



Рисунок 2.2 — Вигляд пошукової системи Google Академія

На основі вільно доступних профілів дослідників у Google Академії в постійному режимі складаються рейтинги наукових груп. Рейтинги складаються за критерієм цитування науковців, які належать до наукових установ чи інших угрупувань. Вигляд пошукової системи зображено на рисунку 2.2.

## 2.3 Платформа Scopus

Scopus – база даних, яка містить велику кількість анотацій та інформації о цитуваннях серед наукової літератури та має влаштовані системи, які відслідковують дії користувача для подальшого аналізу популярності статей з метою візуалізації статистики.

База нараховує в собі статті з більш ніж двадцяти трьох тисяч міжнародних видань та більше ніж п’яти тисяч видавців, які отримали авторитетний статус серед всього світу. Широко визнана серед міжнародних університетів за якість зібраних даних, які мають високу оцінку серед наукової спільноти.

Дана бібліографічна база охоплює такі галузі , як:

- природні науки

- гуманітарні та громадські науки
- медицини
- мистецтва
- техніки

Була створена в 2004 році власником видавничого дому Ельзевірів, який був створений ще у 1880 році у Амстердамі. База даних широко використовується між рейтинговими агенціями  для створення світових рейтингів серед університетів світу.

До публікації на ресурсі допускаються лише перевірені серед наукової спільноти видавці, а для збирання інформації необхідно бути учасником наукової спільноти та пройти процес регістрації на платформі.


Рисунок 2.3. Вигляд пошукової системи Scopus(Elsevier)

Для збирання даних необхідно фактично знаходитись у науковій установі, яка має договір з компанією Elsevier. Платформа має простий інтерфейс та об’єднаний пошук. Вигляд пошукової системи зображено на рисунку 2.3

## 2.4 Платформа Web of Science

Web of Science - бібліографічна база даних, яка вміщає в себе статті з більше ніж 12 000 наукових журналів та 148 000 наукових конференцій. База даних охоплює галузі природних , гуманітарних та громадських наук та мистецтва та надає змогу обрати користувачу найбільш релевантні у цьому напрямі.

Зокрема платформа в автоматичному режимі пов’язує посиланнями видані результати з іншими роботами, з яких було запозичено той чи інший матеріал. Платформа здатна надати користувачу:

- графічне зображення інформації про цитування обраної статті серед інших
- велику кількість пошукових сценаріїв
- автоматично надає користувачу інформацію про авторів та їх організації

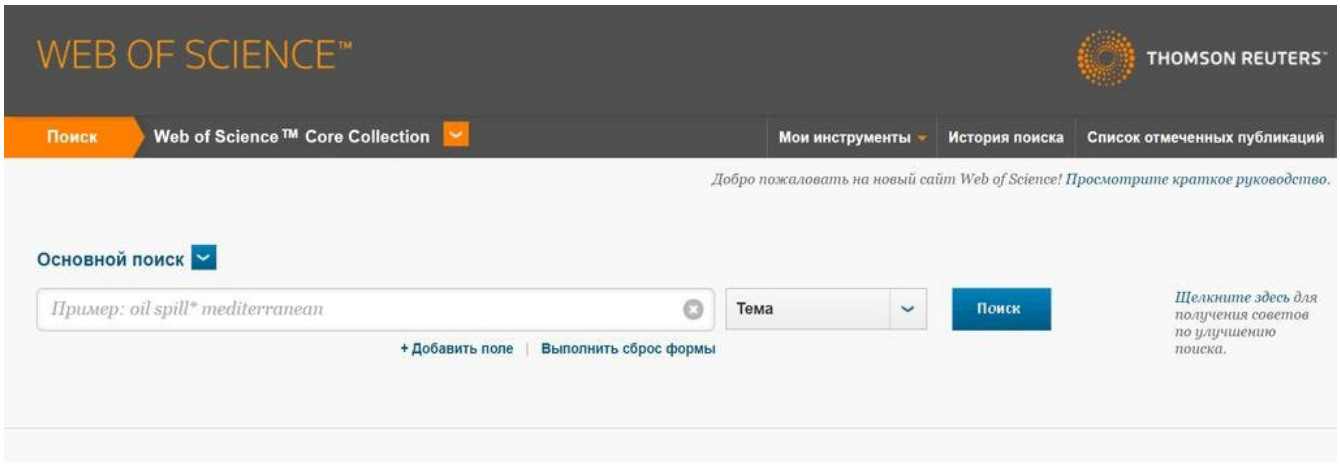



Рисунок 2.4. Вигляд пошукової системи Web of Science

Вигляд пошукової системи зображено на рисунку 2.4. Платформа доступна користувачу за підпискою на платній основі.

## 2.5 Платформа ScienceDirect

ScienceDirect – джерело медичної та науково-технічної інформації, який зберігає близько 14 мільйонів публікацій з авторитетних наукових журналів та книг, видавництва Elsevier. Студенти, викладачі та аналітики з усього світу використовують ScienceDirect для досліджень у різних областях.

Джерело містить більшість фундаментальних трудів, які було написано видатними вченими свого часу, починаючи с 1823 року. Зокрема містить більш ніж 400 публікацій від 74 Лауреатів Нобелівської премії у різних предметних областях.

Платформа містить багато інструментів для зручного пошуку, які легко дозволяють користувачу знайти необхідну інформацію, а саме:

- зручний інтерфейс;
- віддалений доступ до пошуку;
- можливість працювати на мобільних пристроях;
- рекомендації, які надають користувачу посилання на статті, що пов’язані за напрямком досліджень;
- спеціальний пошук за зображеннями або відео;
- візуалізація даних про статті у режимі реального часу;

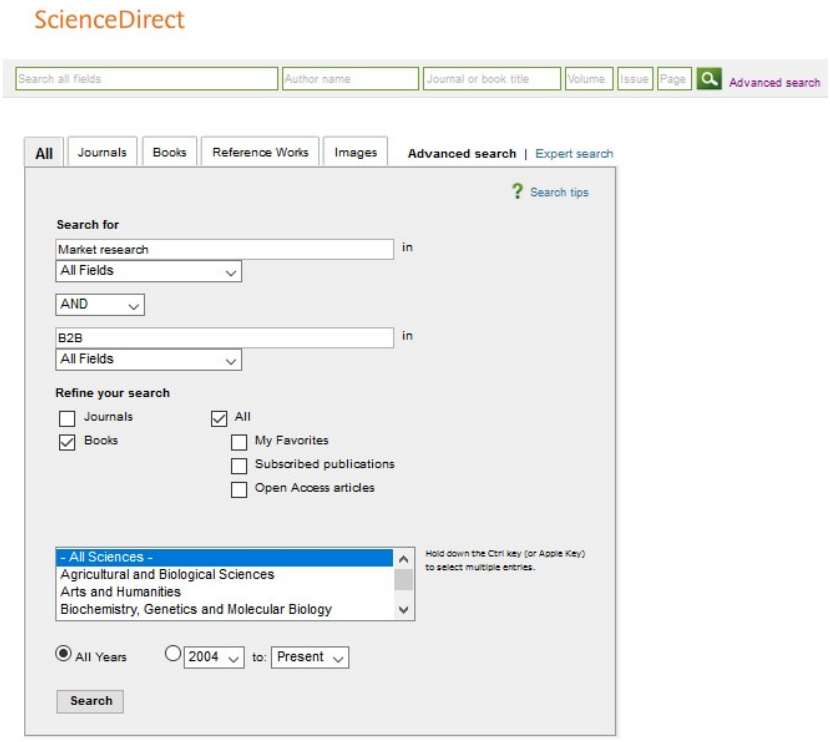



Рисунок 2.5 Вигляд пошукової системи ScienceDirect

Вигляд пошукової системи зображено на рисунку 2.5. Платформа доступна користувачу за підпискою на платній основі.

## 2.6 Платформа ArXiv

ArXiv(Архів) – база даних створена на основі бібліотеки Корнелльського університету(Cornell University), де зберігається велика кількість електронних публікацій: наукових статей та їх препринтів, рефератів та дисертацій.

Архів було створено за потребою американського астрофізика Джона Кона. До серпня 1991 року Джон отримував наукові статті на свою особисту почтову скриньку від 180 науковців лише в напрямку фізики. Звідси з'явилась ідея створення архіву наукових статей, які будуть класифікуватись за науковими напрямками.

Архів є повністю вільним для розміщення статей ким завгодно та для видавця не будуть виникати проблеми з питаннями логістики, порівняння тексту та плагіату. Також не буде питань до авторитетності видавця серед наукової спільноти, а отже усі бажаючі можуть видати свої роботи для перегляду та оцінки іншими науковцями з метою отримати думку інших про статтю. На 2018 рік Архів нараховував близько двох мільйонів статей у різних категоріях. В Архіві можна знайти роботи у напрямках:

- Фізика
- Математика
- Комп'ютерні науки
- Кількісна біологія
- Кількісні фінанси
- Статистика
- Електротехніки
- Економіки


Рисунок 2.6 Вигляд пошукової системи ArXiv

Не дивлячись на відкритість платформи, Архів надійніший за багатьох наукових журналів, оскільки доречність поданих статей до обраної категорії перевіряється науковою спільнотою. Якщо стаття не відповідає обраній категорії або не мають в собі значимості для наукової спільноти, то категорію перезначають на відповідну статті або заносять до категорії з позначкою “сміття”. Вигляд пошукової системи зображено на рисунку 2.6.

## 2.7 Платформа Wolfram Alpha

Wolfram Alpha – це об’єднана пошукова система, відома як “Вольфрамова Альфа”, яка є так званою “обчислювальною машиною знань”. Поєднуючи в собі елементи пошуку, обробники для результатів пошуку користувачем системи та алгоритми які базуються на обробці природної мови(англійської) – Вольфрамова Альфа є прямим конкурентом найпопулярнішої пошукової системи Google.

Даний ресурс не надає користувачу список посилань на літературу, яка є результатом запиту користувача, оскільки повністю спирається на власну базу даних зібраних знань, яка включає в себе велику кількість даних зокрема з областей:

- Фізики
- Астрономії
- Хімії
- Медицини
- Математики
- Кінематографу
- Музики
- Політики
- Географії
- Історії

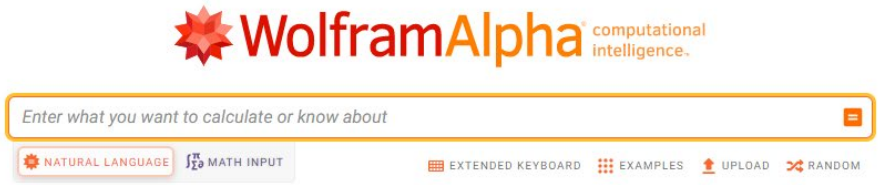



Рисунок 2.7 Вигляд пошукової системи Wolfram Alpha

Сервери пошуку Wolfram налічують близько 10 тисяч процесорів та мають постійне обслуговування, що надає користувачу системи безперебійний доступ до цієї об’єднаної бази знань. Користувачам, які бажають збирати ексклюзивні дані для подальшого аналізу, необхідно оформити підписку на платній основі. Підписка включає в себе надання доступу до хмарного сховища та статистики пошукових запитів, спираючись на їх основі яких, надає користувачу рекомендації для подальших наукових досліджень. Вигляд пошукової системи зображено на рисунку 2.7.

# 3 МЕТОД ЗБОРУ БІБЛІОГРАФІЧНОЇ ІНФОРМАЦІЇ

## 3.1 Структура веб-сайтів

Для збору бібліографічної інформації, яка знаходиться на спеціалізованих веб-сайтах, в автоматичному режимі - необхідно розуміти загальну структуру сайтів, які знаходяться у всесвітній мережі та мати базові знання Інтернет-протоколів.

Структура сайту – це схема розташування сторінок сайту, його категорій, підкатегорій та інших об’єктів. Інакше кажучи це схема плану, за яким будується логічний зв’язок між самими сторінками сайту.

Виділяють 3 основні типи структур сайтів: лінійний, довільний та ієрархічний.

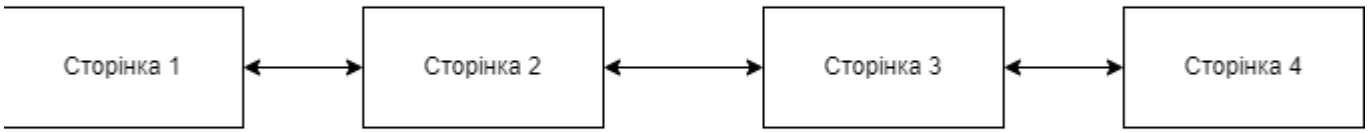



Рисунок. 3.1.1 — Приклад лінійної структури веб-сайту

Лінійна структура – використовується, якщо веб-сайт надає користувачу однакову поступову інформацію. Зазвичай перегляд інформації на цих веб-сайтах здійснюється від початкової до останньої сторінки , а кожна сторінка містить в собі лише посилання на попередню та наступні сторінки. Приблизна схема лінійного веб-сайту зображена на рисунку 3.1.1.

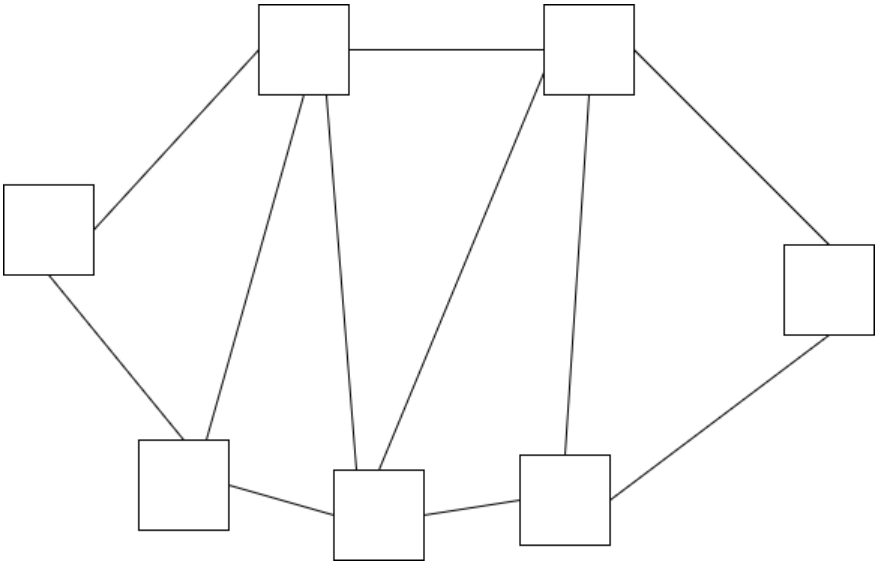


Рисунок 3.1.2 — Приклад довільної структури веб-сайту

Довільна структура – використовується для сайтів, які передбачають повну відсутність обмежень між посиланнями в середині сайтів. Інколи деякі фрагменти цих веб-сайтів можуть вміщати елементи з лінійною та ієрархічною структурою. Приблизна схема веб-сайту з довільною структурою зображена на рисунку 3.1.2.

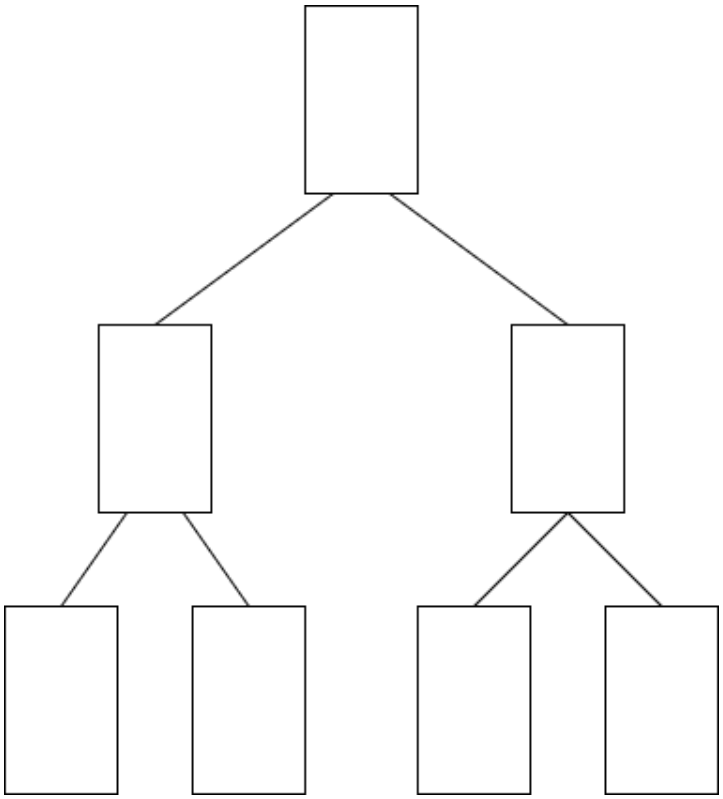


Рисунок 3.1.3 — Приклад Ієрархічної структури веб-сайту

Ієрархічна структура – вважається доцільною при використанні сторінок з різним рівнем доступу. Кожна сторінка вищого рівня має посилання на сторінку або сторінки нижчого рівня. Саме така структура використовується для каталогів та бібліографічних баз даних. Приблизна схема веб-сайту з ієрархічною структурою зображена на рисунку 3.1.3.

Більшість складних веб-сайтів складаються з:

- розмітки для даних та об’єктів, які несе в собі сайт;
- каскадну таблицю стилів, за якою оформляється веб-сайт;
- скриптів, які виконується при взаємодії користувача з веб-сайтом;
- скриптів, які передає дані з серверу на веб-сайт;
- бази даних, в якій структурується та зберігається необхідна інформація для правильної роботи веб-сайту та дані користувачів.

За розмітку веб-сайтів відповідає мова розмітки HTML. Саме в ньому вказується розташування блоків, шрифти, кольори блоків та тексту, посилання.

Каскадною таблицею є мова розмітки CSS. Використовують задля оформлення та реагування сайту на дії користувача.

Скрипти, які використовує веб-сайт можуть бути реалізовані на будь-якій мові програмування, але найчастіше використовують ті мови, які були розроблені для певних задач. Для взаємодії користувача з веб-сайтом найчастіше використовують мову програмування JavaScript. Скрипти взаємодії користувача з веб-сайтом виконуються на стороні користувача.

Для виконання процедур взаємодії серверу та веб-сайту дуже популярним є серверна мова програмування PHP. Саме через нього сервер отримує команди для виконання.

База даних – структура, яка організована для зберігання даних за визначеною концепцією, яка описує характеристику цих даних та їх взаємозв’язки. У сучасних сайтах різні дані можуть зберігатися різних серверах з метою зменшити навантаження на сервер.

## 3.2 Аналіз та збереження даних

Для автоматичного збору бібліографічної інформації програма або скрипт повинні вміти синтаксично проаналізувати сайт та зберегти усі необхідні дані для користувача. Такі скрипти або програми називають парсерами, а процес автоматичного збору інформації з інтернет ресурсів – парсингом.

Принцип роботи парсеру: створення пошукового сценарію для обраного об’єкту збирання даних, посилання цього сценарію на базу даних серверу, збирання та обробка отриманих даних, занесення даних у тимчасове сховище або іншу базу даних. Приблизна схема роботи парсеру зображена на рисунку 3.2.

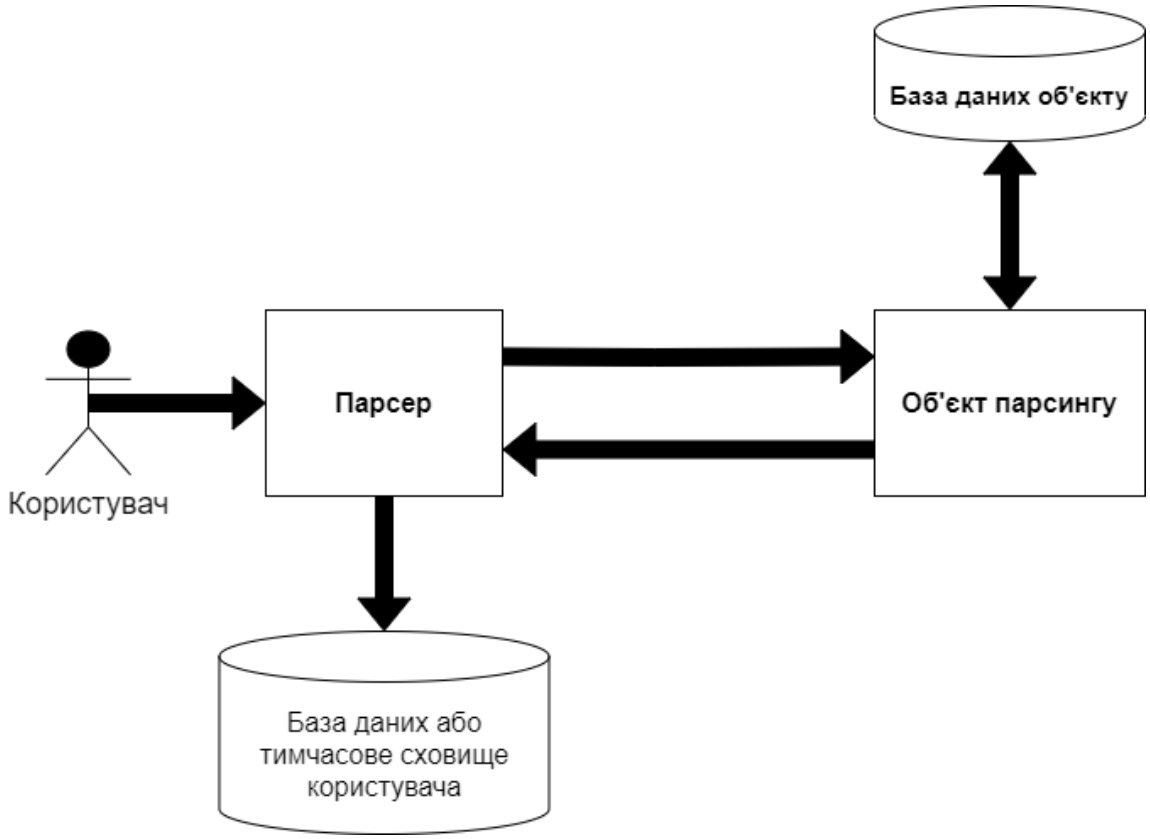



Рисунок 3.2 — Схема роботи скриптів-парсерів

Об’єктом парсингу можуть бути будь який інтернет ресурс, дані якого можуть використовуватись для подальшої обробки. Результати успішності парсингу напряму залежать від захищеності ресурсу, який є об’єктом парсингу.

Використання парсерів є темою для суперечок у всесвітній павутині, оскільки з’являються можливість збирати небажані для власника дані, без вказання власника, порушуючи права інтелектуальної власності, з корисною метою або навіть шантажу.

Парсер став потужним інструментом для аналізу даних у різних сферах життя. Головними перевагами використання парсерів:

- Можливість зібрати інформації у великому обсязі та можливість розмістити її за визначеними алгоритмами сценаріями у необхідне сховище даних.
- Автоматичне слідкування та пристосування до змін даних у джерелі, з якого інформація збирається.
- Швидкість збирання необхідної інформації.
- Точність та безпомилковість результатів при досконало написаному сценарії.
- Гнучке перетворення інформації у будь-який формат даних.

Усюди де є переваги, є й недоліки. Не дивлячись на високу популярність використання парсерів, вони мають недоліки:

- Залежить від структури кожного сайту окремо, а отже зазвичай парсер може виконувати свої задачі лише на одному ресурсі.
- Для редагування сценаріїв, або їх збільшення потребує втручання програміста.
- При зміні структури сайту, з якого збирається інформація спостерігаються помилки при виконанні поставленої задачі.
- Необхідне довге та ретельне тестування задля отримання бажаних результатів.
- Не можуть повністю замінити людський ресурс.

Ще одним значним недоліком таких скриптів є можливість відрізнити діяльність парсеру від дій звичайного користувача ресурсу. Через це парсер може бути заблокований для збирання необхідної інформації.

## 3.3 Метод зберігання бібліографічної інформації у локальному сховищі.

Однією з найпоширеніших практик зберігання інформації, яка в себе включає елементи з різними типами даних, у локальному сховищі є створення локальної бази даних. База даних є фіксованою моделлю за якою зберігаються усі необхідні дані у певному вигляді.

У випадку зберігання інформації у одній локальній загальній базі даних доречно використовувати реляційну модель бази даних. Реляційна модель бази даних являє собою подання даних у вигляді таблиць з рядками та стовпцями, де рядки являють собою записи, а стовпці – полями, де зберігаються самі дані. Сама таблиця має наступні особливості:

- відсутність повторюваних груп
- кожний елемент таблиці – один елемент даних
- кожний стовпець має унікальне ім’я
- у таблиці не має двох або більше однакових рядків

Сама база даних може зберігати інформацію, проте не може з нею ніяк взаємодіяти. Для цього було винайдено систему управління базами даних(СУБД).

## 3.4 Управління збереженими даними

Для взаємодії програми зі збереженою у базі даних інформацією широко використовують технологію ORM.

ORM (Object Relational Mapping (об'єктно-реляційне відображення)) – технологія в програмуванні, мета якої реалізувати створення віртуальної бази даних на основі об’єктно-орієнтованих можливостей мов програмування.

Об'єктно-орієнтований підхід розроблено з урахуванням основних принципів програмної інженерії (таких як зв'язок, агрегування, інкапсуляція), а реляційна база

даних - з урахуванням математичних теорій. Між двома наборами теорій є суттєві відмінності. Щоб усунути цю невідповідність, з'явилася технологія об'єктно-реляційного зіставлення.

Об'єктно-реляційне відображення дає можливість розробникам використовувати свою улюблену мову програмування для звернень до баз даних без ручного написання SQL(англ. Structured query language — мова структурованих запитів) операторів або запитів.

Використання технологій об'єктно-реляційного відображення є більш гнучким методом програмування взаємозв’язку програми та бази даних оскільки він абстрагує систему БД, тому розробник зможе змінити її у будь який час, а сама модель слабко пов'язана з рештою програми, тому розробник зможете використовувати її в інших своїх проектах.

Використання технології об'єктно-реляційного відображення заощаджує розробнику багато часу оскільки:

- Розробник створює свою модель даних лише в одному місці, і її простіше оновлювати, підтримувати та повторно використовувати код.
- Багато речей виконуються автоматично, починаючи з обробки бази даних до взаємодії елементів у ній.
- Відпадає необхідність формувати SQL запроси, які складно сприймаються людьми.
- Виклик вже заздалегідь підготовлених операторів чи транзакцій стає таким же простим, як і виклик звичайних функцій.

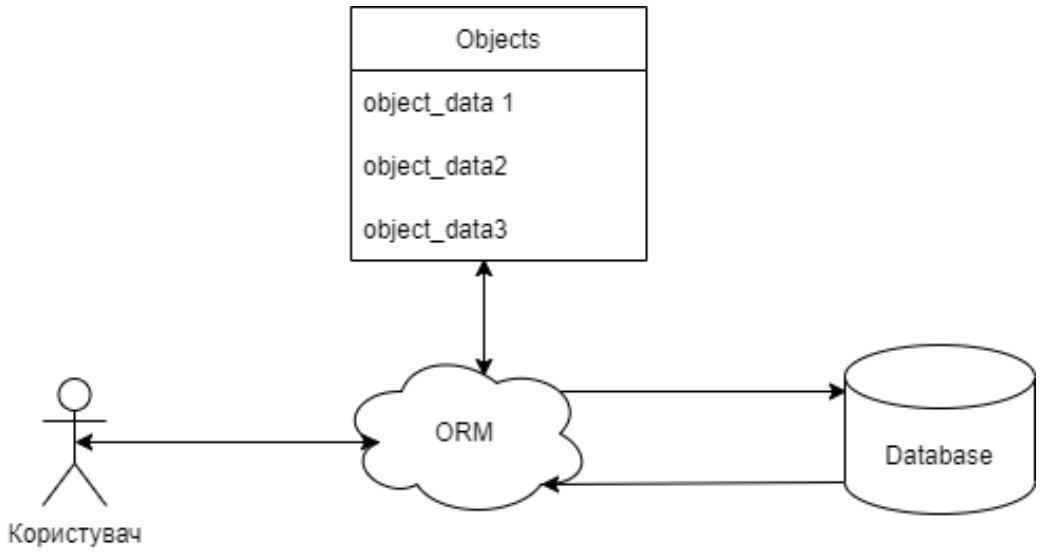



Рисунок 3.4 — Схема взаємодії користувача та бази даних через ORM

Мінусами використання даної технології є те, що використання цих інструментів потребує досконалого вивчення документації до них та пониження продуктивності роботи програми у випадку створення складних запитів через ці інструменти. Схема взаємодії ORM з базою даних зображена на рисунку 3.4.

# 4 ЗАСОБИ РОЗРОБКИ

## 4.1 Обрані джерела пошуку бібліографічної інформації

Першим джерелом збору бібліографічної інформації було обрано Google Scholar (Google Академія) через найбільшу кількість наукової літератури та велику популярність серед користувачів. Таким чином аналітик може отримати доступ до найвідоміших статей, які необхідні для аналітики та досліджень.

Другим джерелом збору бібліографічної інформації було обрано ArXiv. Через легку систему публікації статей, достатньо велику кількість наукових напрямків та повністю вільний доступ до інформації – аналітик матиме змогу отримати інформацію про нові статті, які ще не визнані науковою спільнотою, але можуть мати великий потенціал у майбутньому.

## 4.2 Опис обраної мови програмування

Python – це мова програмування високого рівня, яка широко використовується у багатьох напрямках програмування. Відноситься до мов програмування, які не вимагають попередньої компіляції, тобто не перетворюються у машинний код.

Метою створення цієї мови програмування було створити просту для розуміння, але одночасно потужну мову програмування, з можливістю швидкого вивчення новоспеченими програмістами. Саме через це Python досі залишається актуальною мовою програмування для вирішення усіх можливих задач.

Головною особливістю Python є динамічна типізація, тобто змінна набуває тип даних при присвоєні даних автоматично та може змінюватись в процесі виконання програми. Зокрема Python – мультипарадигмальна мова програмування, тому підтримує принципи об’єктно-орієнтованого програмування, принципи аспектно-орієнтованого програмування та функціонально-орієнтованого програмування.

Саме простота та широкий спектр функціоналу, велика кількість аналогів та спеціалізованих бібліотек робить мову привабливою для виконання поставленої задачі парсингу веб-ресурсів задля подальшої обробки зібраної інформації.

## 4.3 Опис обраного ORM Peewee та реляційної бази даних SQLite

Peewee – це невелике ORM, яке зараз підтримує велику кількість реляційних систем керування базами даних. Бібліотека була створена відносно недавно, якщо порівнювати з іншими, а саме у 2010 році.

У документації до Peewee користувач може знайти багато прикладів до використання, які постійно доповнюються автором проекту, а отже новачки у програмуванні з використанням цього ORM матимуть змогу навчатися, використовуючи приклади та напряму бачити результати написаного коду.

Саме ORM peewee отримало багато схвальних відгуків від широкого кола Python розробників з усього світу. Головною перевагою Peewee – невибагливість у місці на жорсткому диску та швидкість вивчення розробником за прикладами та доступною та зрозумілою документацією.

Як реляційну базу даних було обрано SQLite, оскільки її створення не потребує додаткових налаштувань та запуску на сервері. Головними перевагами SQLite є:

- висока швидкість занесення даних та виконання запитів
- мінімалізм
- збереження усіх необхідних даних на одному файлі
- надійність
- займає мінімальну кількість місця у сховищі
- кросплатформеність

## 4.4 Qt Designer та PyQt5

PyQt — це зв'язка Python для Qt, яка являє собою набір бібліотек C++ та інструментів розробки, які включають незалежні від платформи абстракції для графічних інтерфейсів користувача (GUI), а також мережу, потоки, регулярні вирази, бази даних та багато інших потужних функцій.

Зокрема, для пришвидшення процесу розробки програм з використанням бібліотеки PyQt створили QtDesigner – середу розробки графічних інтерфейсів , результат якої можна зберігати у форматах xml або вже згенерованому в автоматичному режимі Python коді. Вигляд інтерфейсу програми зображений на рисунку 4.4.

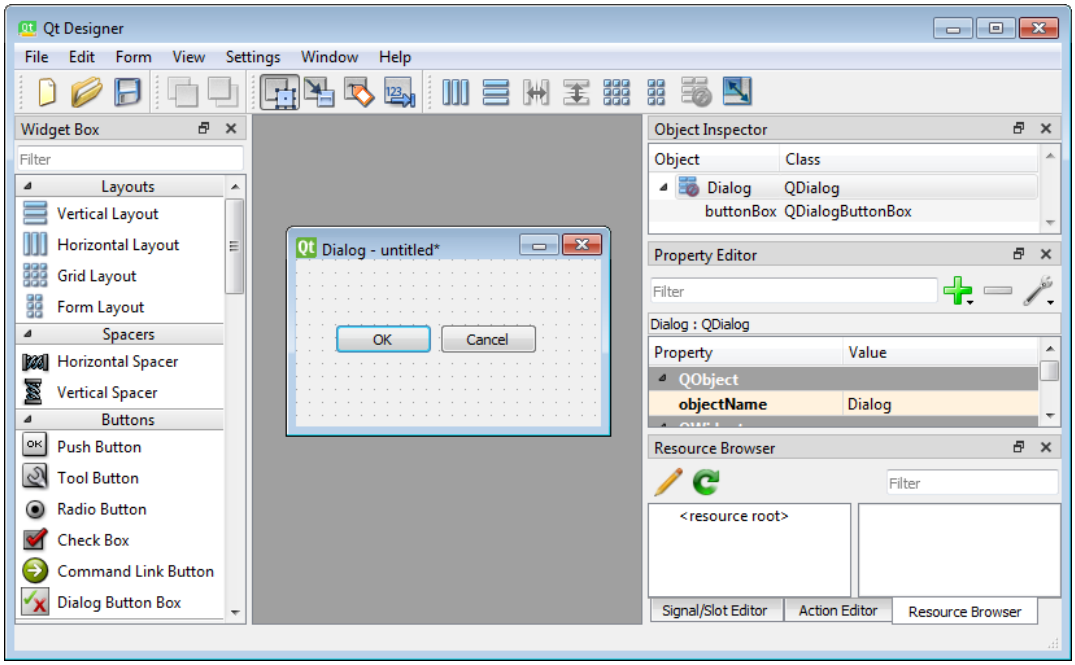



Рисунок 4.4 — Інтерфейс програми QtDesigner

Як продукт PyQt отримав широку користувальницьку аудиторію, включаючи відомі корпорації як: Disney, Dreamworks, Pixar, Industrial Light and Magic та Sony Pictures.

# 5 ТЕХНІЧНИЙ ОПИС СИСТЕМИ

## 5.1 Опис архітектури

Архітектуру програмного забезпечення а саме всі процеси, які проходять всередині цієї програми, взаємодію користувачів з програмою та результати цієї взаємодії можна графічно зобразити спроектувавши діаграму прецедентів(англ. Use Case diagram)

Діаграма прецедентів часто використовується у проектуванні системи поведінки системи у визначеному сценарії, який напряму пов’язаний зі взаємодією користувача в системі.

Іншими словами проектується схематична інструкція тих процесів, які будуть виконуватись у системі, а також прецеденти – учасники, які безпосередньо взаємодіють з системою.

Таким чином проектування діаграми прецедентів дає змогу розробити концептуальну модель системи для подальшого огляду замовником системи з метою внесення побажань, правок або затвердження системи у вигляді концепції.

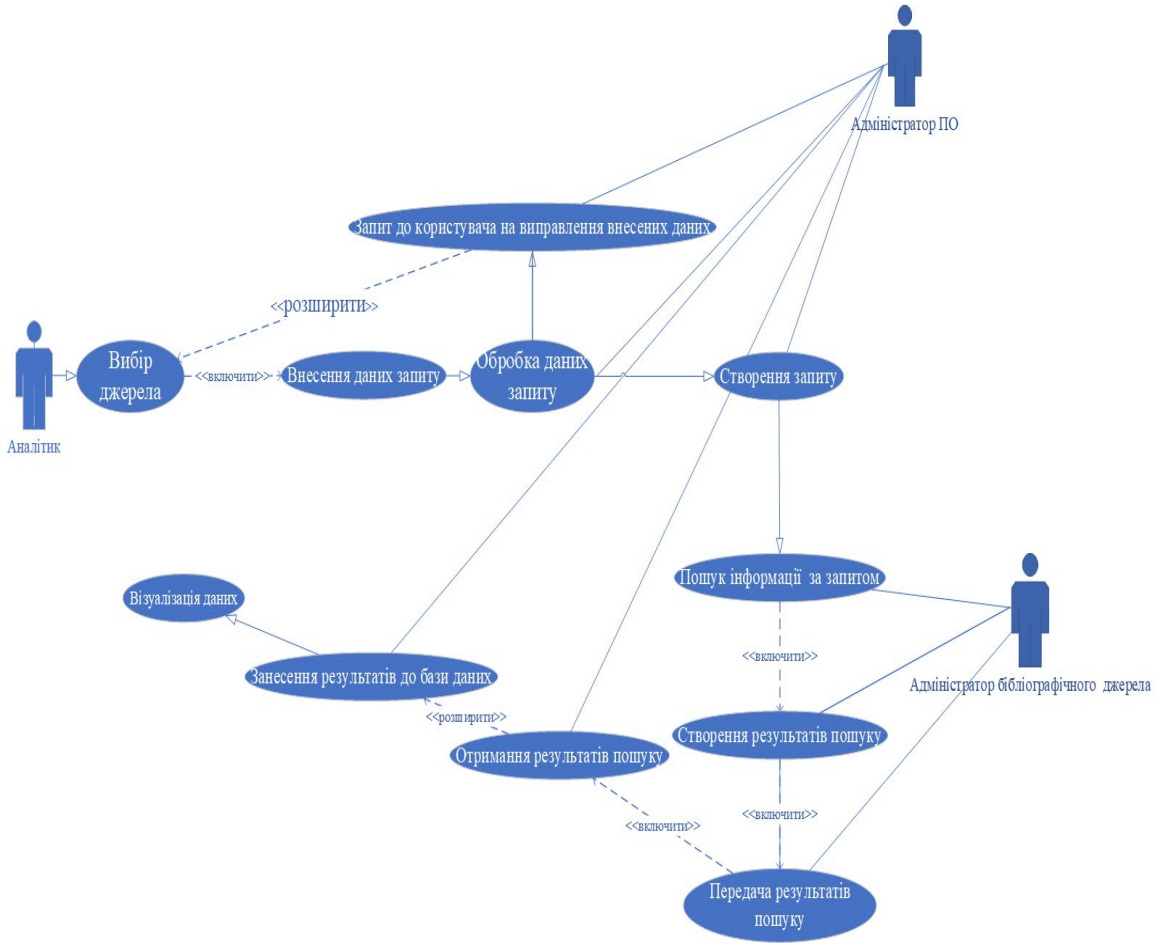



Рисунок 5.1 — Діаграма прецедентів

Архітектура даного програмного забезпечення схематично зображена на рисунку 5.1. Система проектується з урахуванням можливостей кожного з її прецедентів, які будуть застосовувати цю систему, а отже може проглядатись певна ієрархія, за якою різні прецеденти мають різні сценарії використання системи.

## 5.2 Структура БД

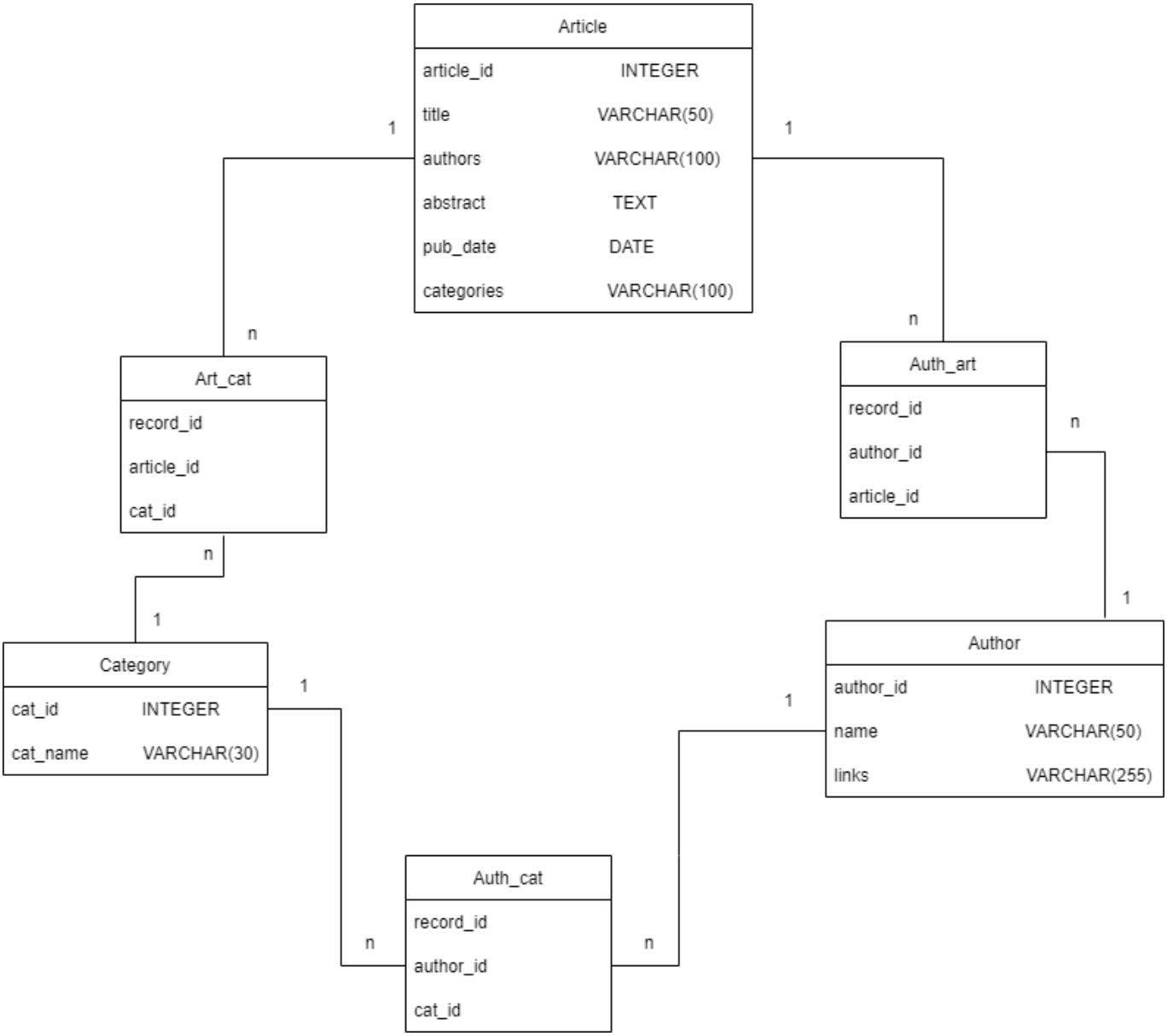



Рисунок 5.2— Структура бази даних

Концептуальна модель бази даних дає змогу спроектувати базу даних в залежності від потреб програмного продукту, який буде використовувати її. На рисунку 5.2 зображено саму концептуальну модель, взаємозв’язок між її елементами, які будуть використовуватись у проекті.

# 6 ПРОГРАМНА РЕАЛІЗАЦІЯ

## 6.1 Загальна структура проекту

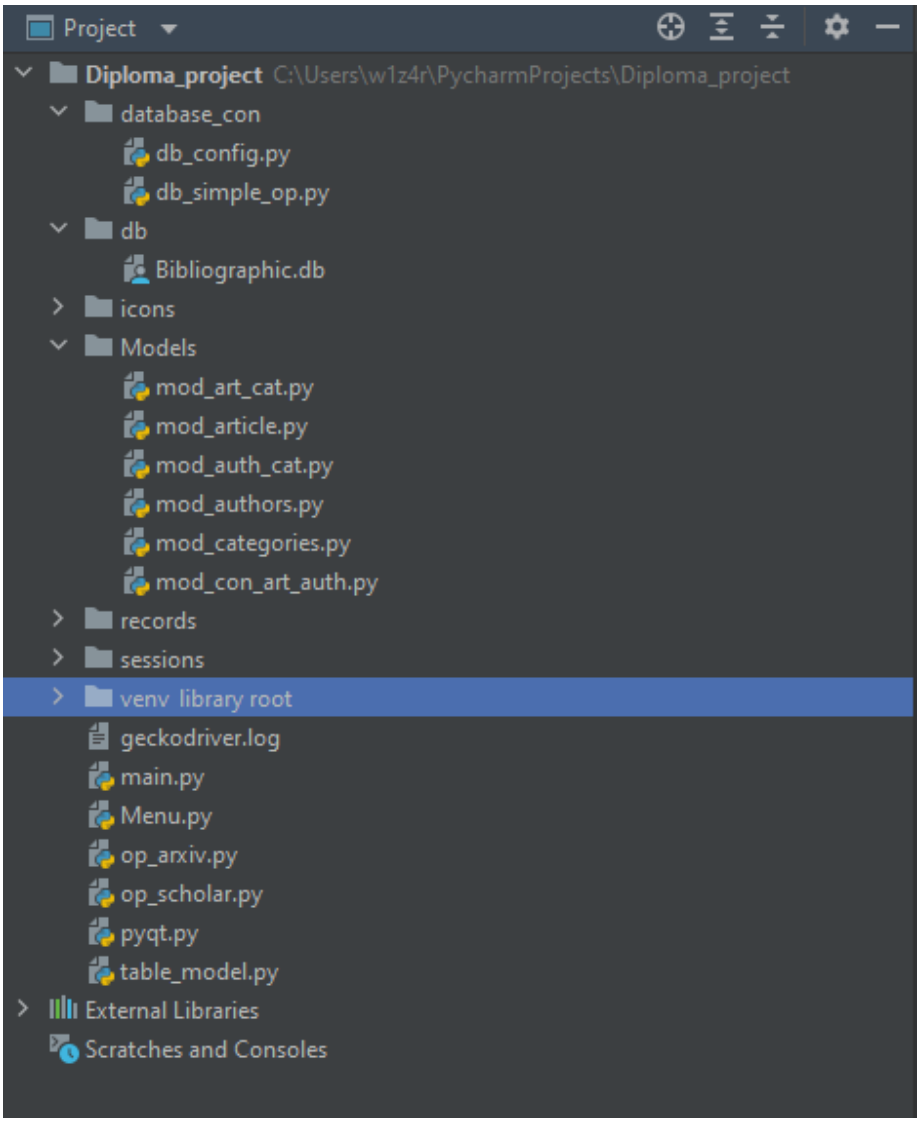



Рисунок 6.1 Загальна структура проекту

На рисунку 6.1 приведена повний список файлів, які необхідні для стабільної роботи програми. Кожен файл відповідає за свою частину у всьому процесі від запуску програми та з’єднання з базою даних до збереження та створення зв’язку між збереженими даними.

Таблиця 6.1 — Перелік файлів та їх загального призначення

| Файл | Призначення у системі |
|---|---|
| db_config.py | Зберігає конфігураційну інформацію як: шлях для зберігання файлів, шлях для підключення до внутрішньої бази даних, ключ для підключення проксі генератора |
| db_simple_op.py | Зберігає функції для роботи з базою даних такі як: підключення , зберігання даних, відключення від неї, створення нової бази даних |
| Bibliographic.db | Файл у якому зберігається база даних |
| Mod_art_cat.py<br>Mod_article.py<br>Mod_auth_cat.py<br>Mod_authors.py<br>Mod_categories.py<br>Mod_con_art_auth.py | Зберігають моделі для взаємодії до відповідних за ім’ям таблиць у базі даних. Моделі вміщають в себе типи даних для кожного елементу в базі даних, імена таблиць та параметр сортування даних у базі. |
| main.py | Файл у якому прописані сценарії взаємодії інтерфейсу та функцій у інших файлах |
| Menu.py | Файл у якому прописаний прототип інтерфейсу користувача |
| op_arxiv.py | Файл у якому прописані відповідні до сценаріїв у main.py функції пошуку у ресурсі ArXiv |

| | |
|---|---|
| op_scholar.py | Файл у якому прописані відповідні до сценаріїв у main.py функції пошуку у ресурсі Google Scholar |
| pyqt.py | Файл, який запускає інтерпретатор, має в собі клас, який наслідує прописаний інтерфейс. Має прописані функції, які будуть визивати відповідні сценарії при взаємодії користувача з інтерфейсом |
| table_model.py | Файл, який зберігає функції , що відповідають за відображення інформації у інтерфейсі користувача |

## 6.2 Алгоритм роботи проекту та його результати


Рисунок 6.2.1. Зображення початку процесу пошуку бібліографічної інформації

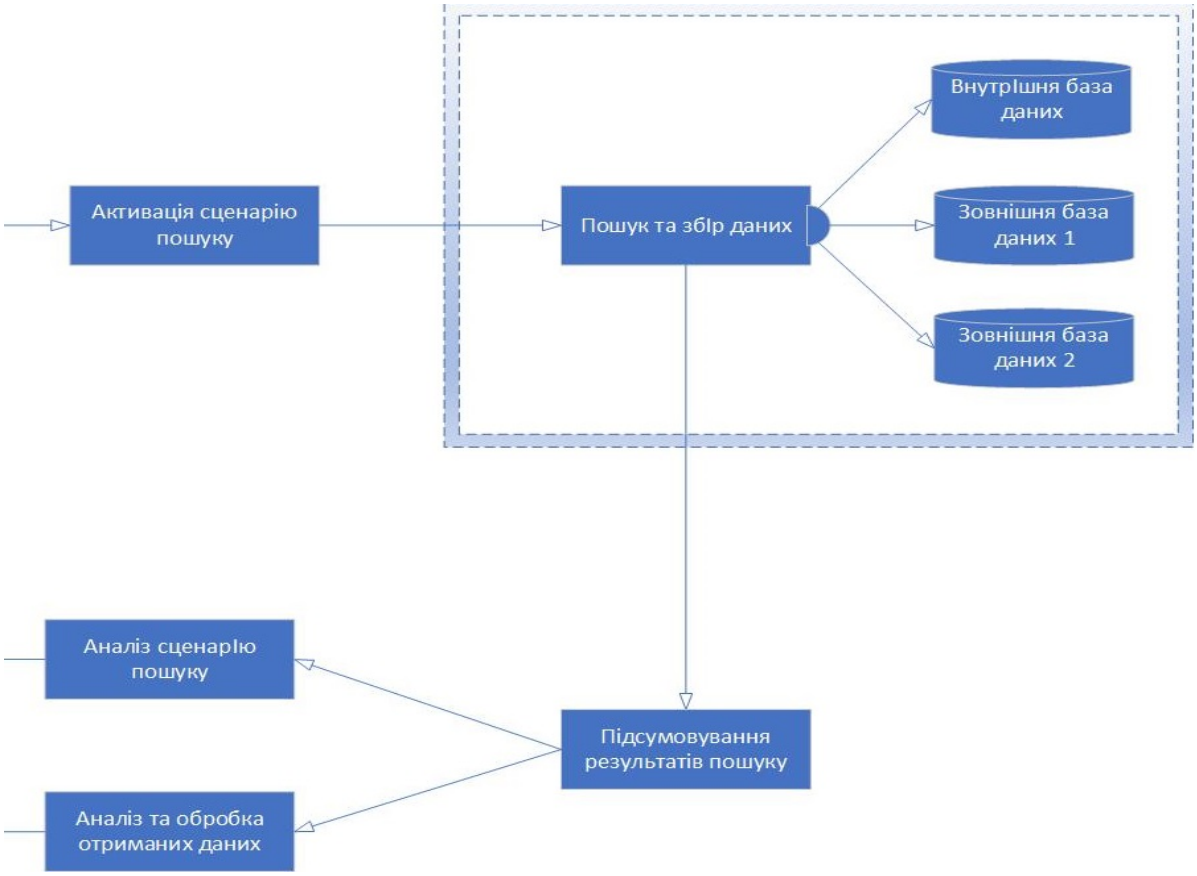



Рисунок 6.2.2 Процес зберігання результатів пошуку

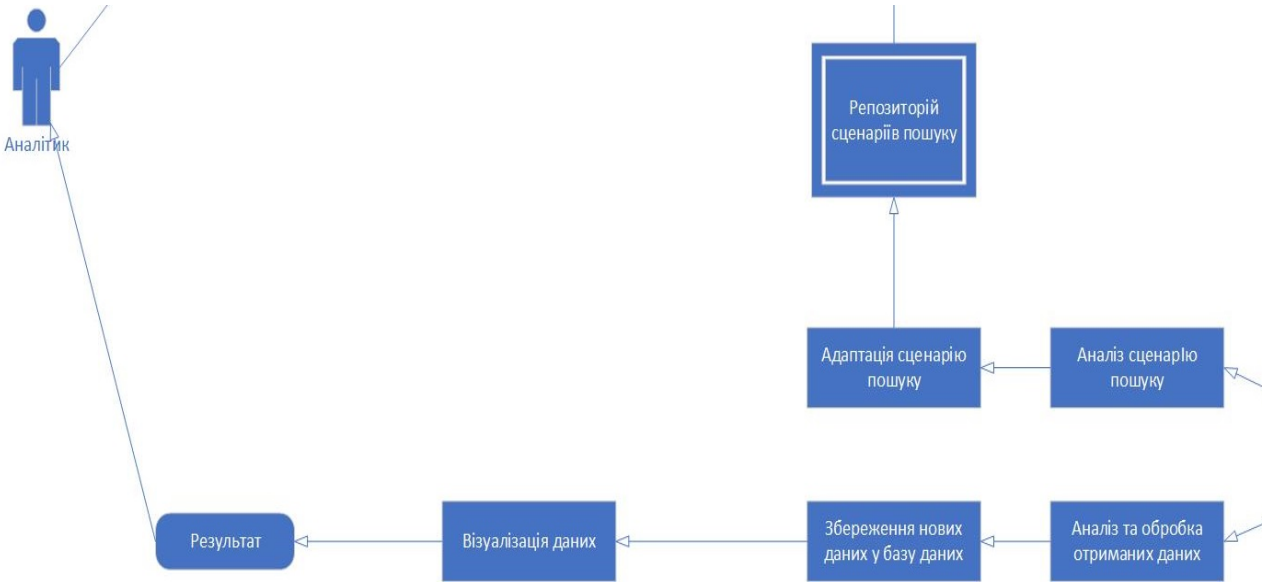



Рисунок 6.2.3 Процес збереження необхідних даних та їх візуалізації

Процес виконання програми користувачем було зображено на рисунку 6.2.1-6.2.3 Після запуску програми, перш ніж користувач побачить інтерфейс, система в автоматичному режимі підключиться до існуючого файлу з базою даних або створить новий файл з необхідним шаблоном для заповнення.

Після завершення процесу підключення користувача до бази даних користувач побачить графічний інтерфейс. Інтерфейс має поля:

- Два поля з прапорцями для обирання ресурсу, який буде задіяно при пошуку
- Поле “Search by” для вибору сценарію(критерію) пошуку
- Поле “Searching parameter” для вводу користувачем даних, які будуть використані при запиті
- Поле “Maximum result” для визначення максимальної кількості результатів пошуку для зберігання у базі даних
- Поле “Category” для визначення наукових напрямків, за якими буде здійснено пошук відповідних статей.
- Кнопки: “Search” – для здійснення сценарію пошуку, “Save current state” – для збереження копії бази даних під спеціальним ім’ям, “Exit” – для безпечного виходу з програми.

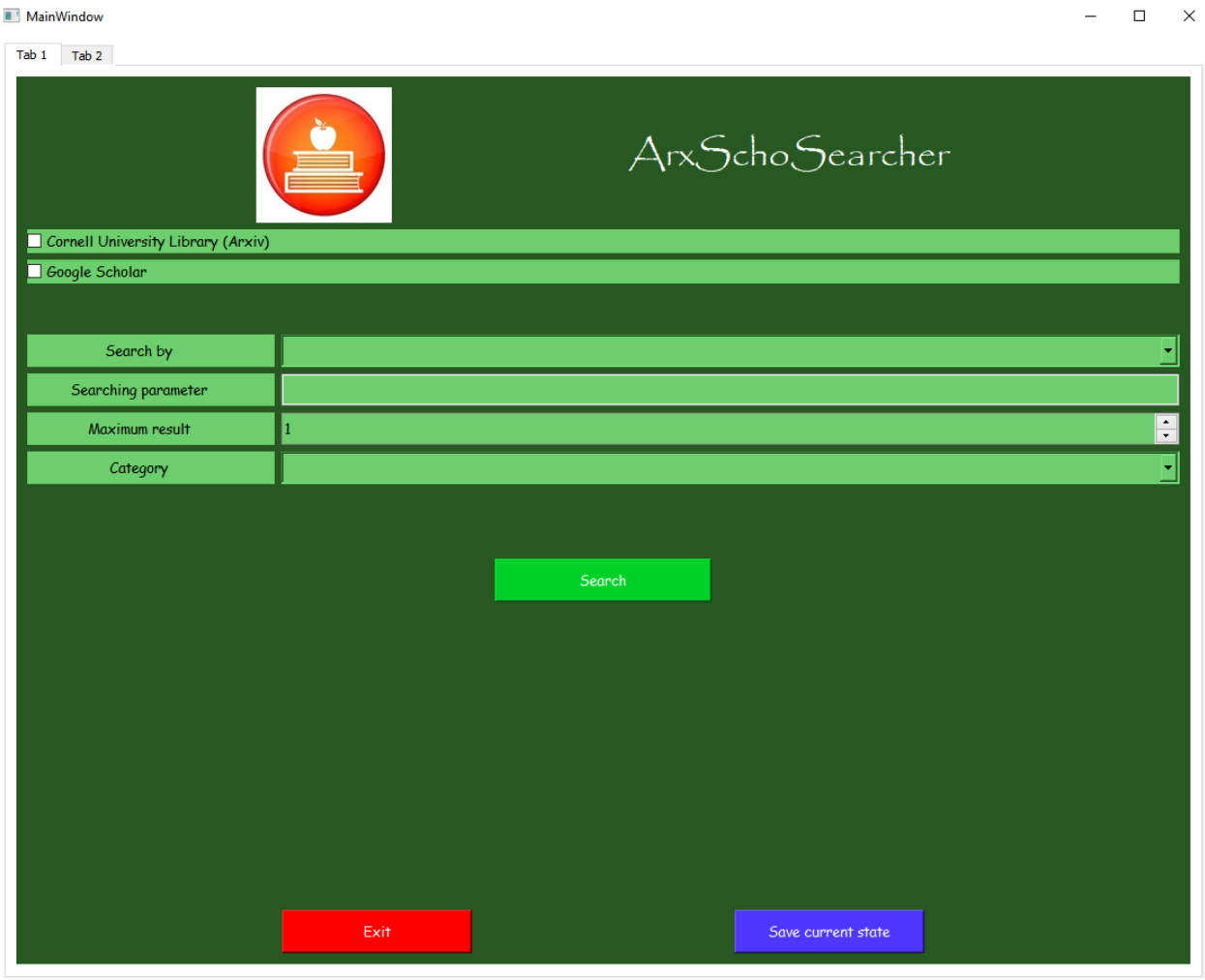



Рисунок 6.2.2— Інтерфейс головного меню пошуку

Інтерфейс має поля для заповнення, за якими буде визначений сценарій пошуку на ресурсах, з яких користувач хоче отримати бібліографічну інформацію. Вигляд інтерфейсу зображено на рисунку 6.2.2.

В залежності від необхідних даних від кожного сценарію пошуку програма буде зчитувати дані, які ввів користувач у визначених полях. В разі відсутності параметрів у обов’язковому полі для заповнення користувач отримує сповіщення від програми. Приклад сповіщення зображено на рисунку 6.2.3.

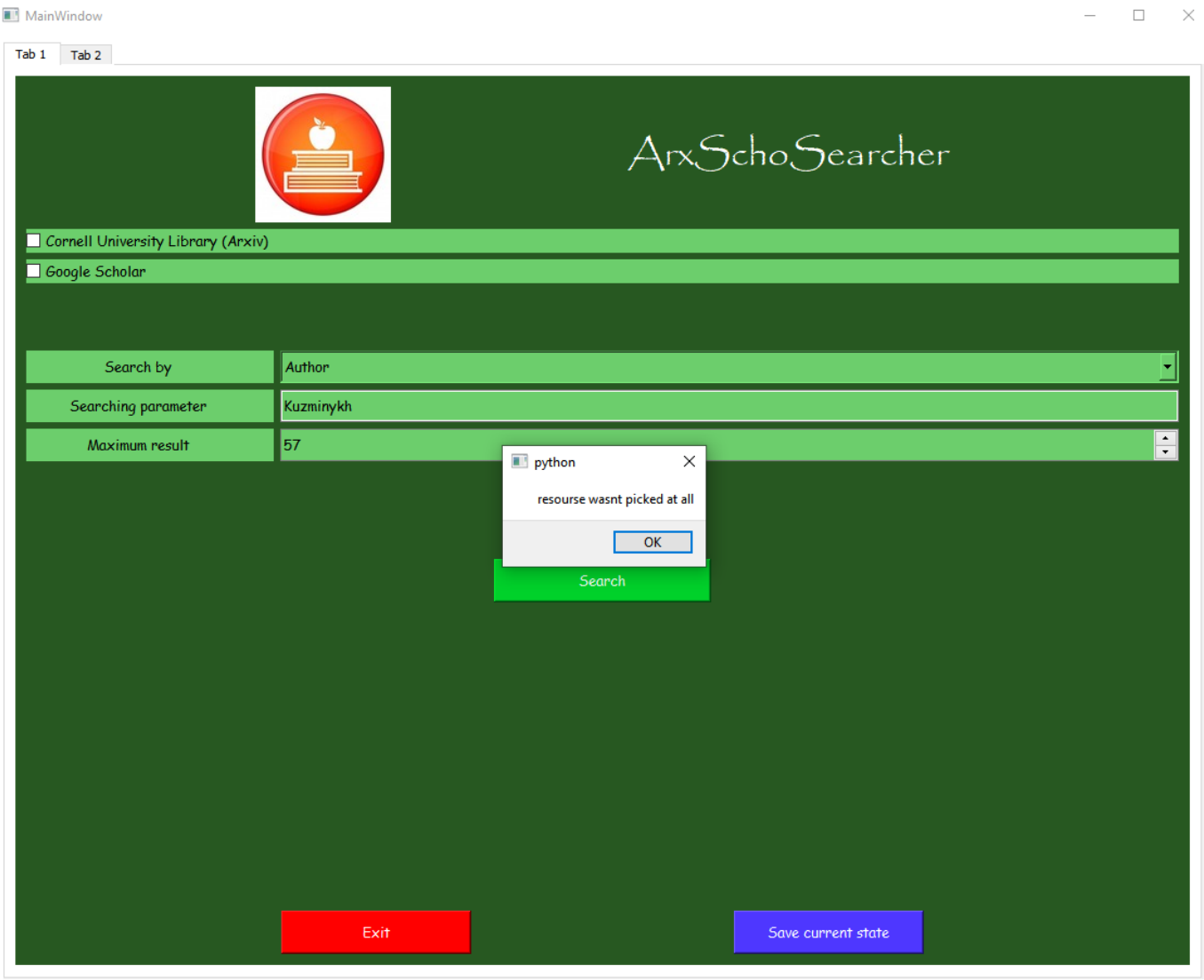



Рисунок 6.2.3 — Приклад сповіщення від програми. Сценарій коли користувач не обрав жодного ресурсу для пошуку інформації.

Користувач може здійснювати пошук бібліографічної інформації на двох ресурсах. На першому ресурсі Cornell University Library(ArXiv) користувач може здійснювати пошук за параметрами: Tittle – назва статті, Author – ім’ям автора публікації та Category – напрямками науки , в яких статті були написані. На другому ресурсі користувач має змогу шукати за параметрами Tittle та Author. Також користувач може обрати бажану кількість найперших знайдених результатів для їх збереження.

Після проведення пошуку користувач отримає повідомлення про кінець процесу пошуку та занесення інформації до бази даних. Приклад сповіщення зображено на рисунку 6.2.4.

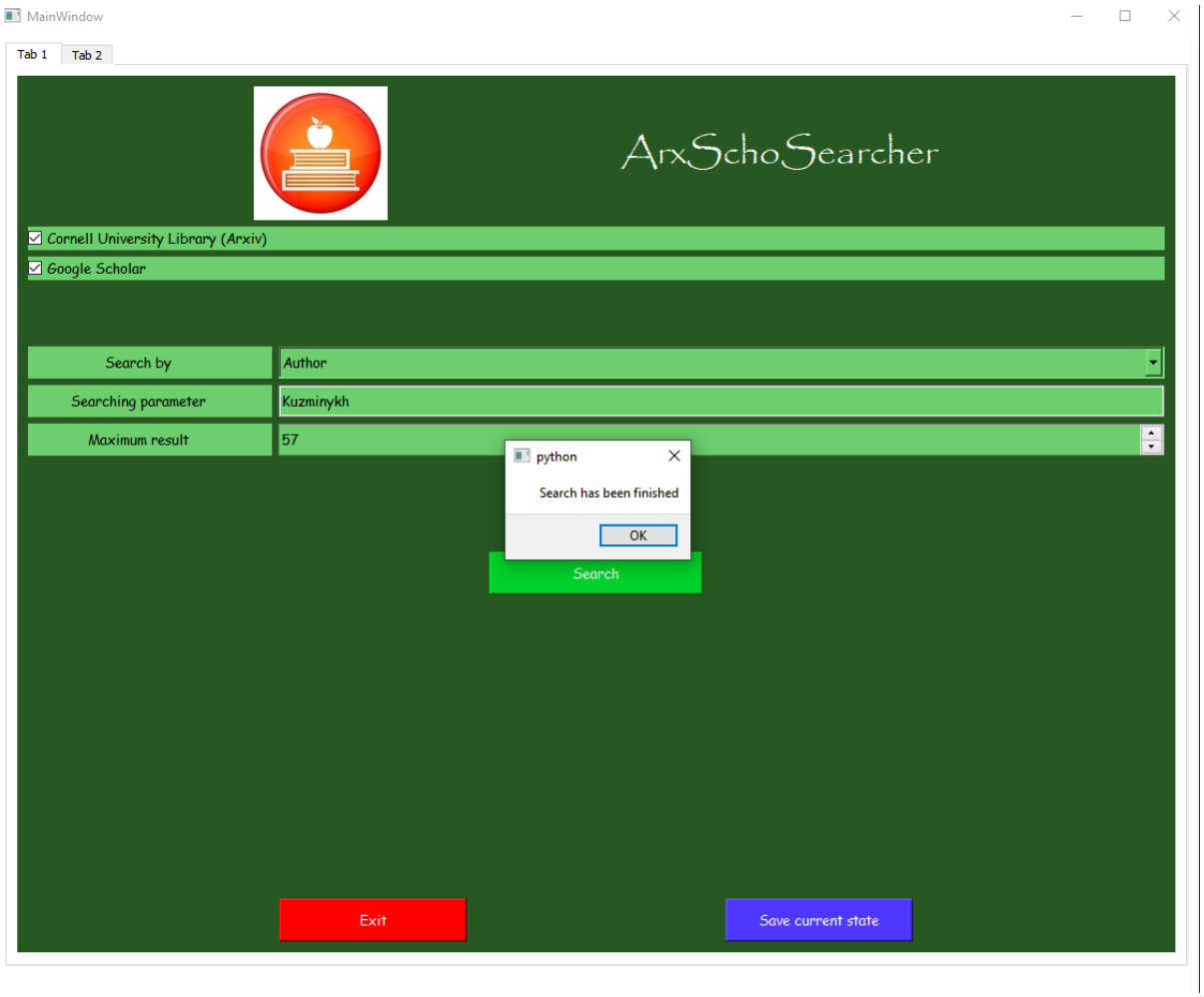



Рисунок 6.2.4 — Приклад сповіщення про закінчення пошуку бібліографічної інформації

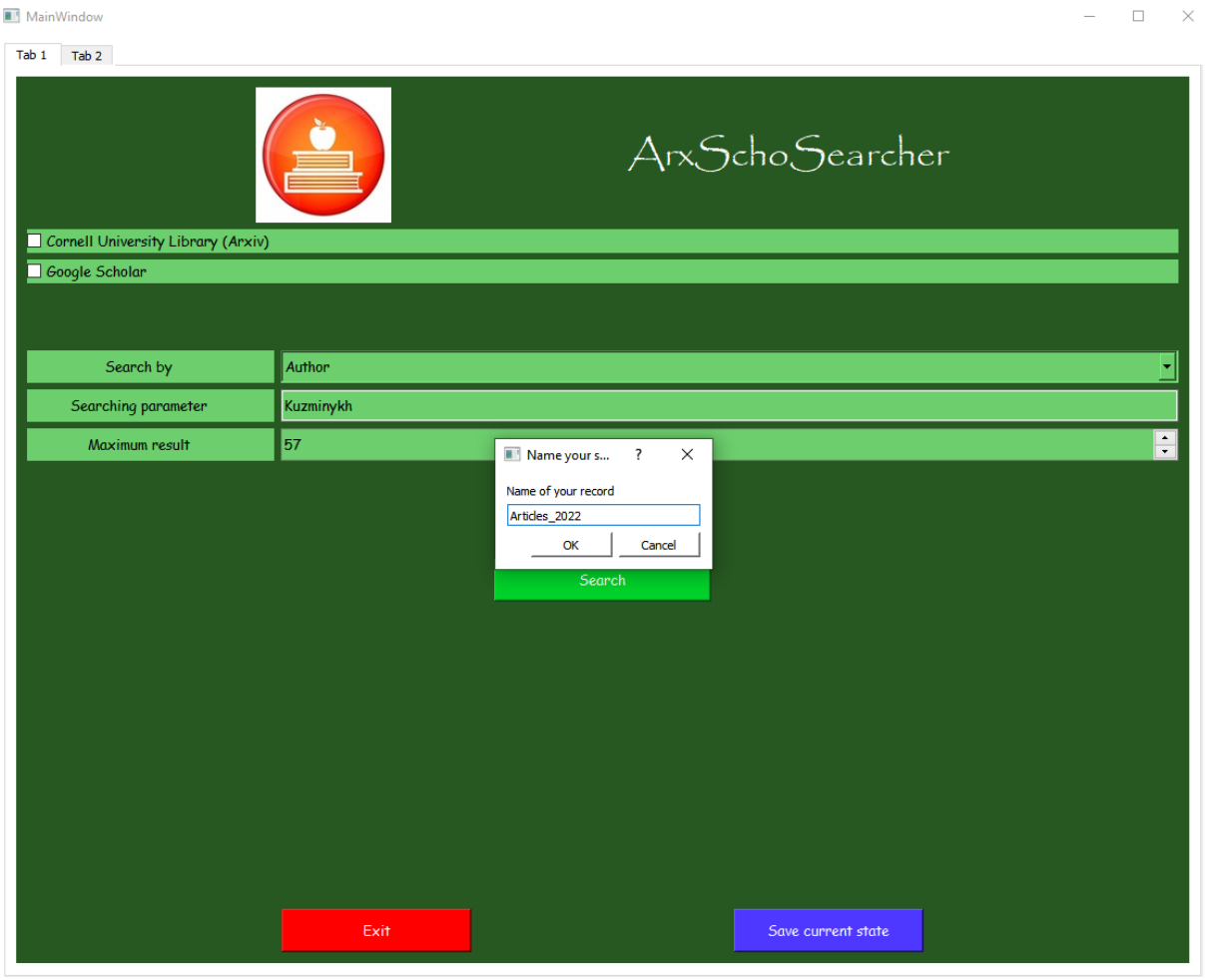
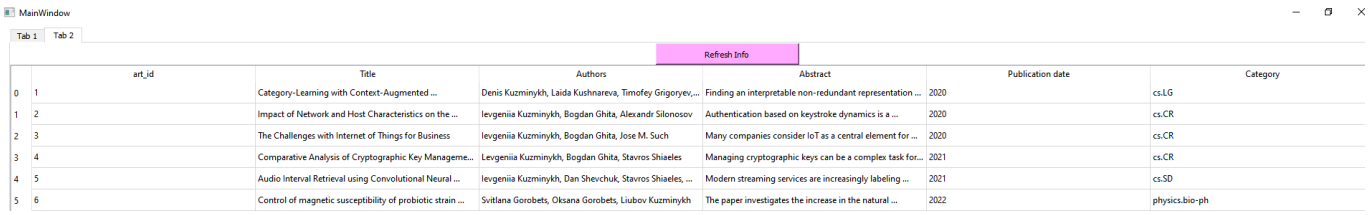



Рисунок 6.2.5 — Діалогове вікно для назви файлу користувачем

Програмне забезпечення дає змогу користувачу копію файлу з базою даних. Файл з базою даних зберігається користувачем у папці “records” під ім’ям, яке дає йому користувач, або під стандартним ім’ям record_ з точним вказанням часу, коли цей файл було збережено. Вигляд діалогового вікна зображено на рисунку 6.2.5.

Результатами виконання програми є зібрані бібліографічні дані, які було зібрано парсером та розміщено до відповідних таблиць у базі даних програми. Перегляд інформації, яка знаходиться у базі даних можливий в режимі реального часу. Приклади збереженої інформації зображено на рисунках 6.2.6 та 6.2.7.




| | art_id | Title | Authors | Abstract | Publication date | Category |
|---|---|---|---|---|---|---|
| 0 | 1 | Category-Learning with Context-Augmented ... | Denis Kuzminykh, Laida Kushnareva, Timofey Grigoryev,... | Finding an interpretable non-redundant representation ... | 2020 | cs.LG |
| 1 | 2 | Impact of Network and Host Characteristics on the ... | Ievgeniia Kuzminykh, Bogdan Ghita, Alexandr Silonosov | Authentication based on keystroke dynamics is a ... | 2020 | cs.CR |
| 2 | 3 | The Challenges with Internet of Things for Business | Ievgeniia Kuzminykh, Bogdan Ghita, Jose M. Such | Many companies consider IoT as a central element for ... | 2020 | cs.CR |
| 3 | 4 | Comparative Analysis of Cryptographic Key Manageme... | Levgeniia Kuzminykh, Bogdan Ghita, Stavros Shiaeles | Managing cryptographic keys can be a complex task for... | 2021 | cs.CR |
| 4 | 5 | Audio Interval Retrieval using Convolutional Neural ... | Ievgeniia Kuzminykh, Dan Shevchuk, Stavros Shiaeles, ... | Modern streaming services are increasingly labeling ... | 2021 | cs.SD |
| 5 | 6 | Control of magnetic susceptibility of probiotic strain ... | Svitlana Gorobets, Oksana Gorobets, Liubov Kuzminykh | The paper investigates the increase in the natural ... | 2022 | physics.bio-ph |

Рисунок 6.2.6 — Приклад виведення зібраної інформації користувачу системи у вікні програми

| | | |
|---|---|---|
| 17 | Jose M. Such | Cornell University Library (arXiv) article: http://... |
| 18 | Levgeniia Kuzminykh | Cornell University Library (arXiv) article: http://... |
| 19 | Stavros Shiaeles | Cornell University Library (arXiv) article: http://... |
| 20 | Dan Shevchuk | Cornell University Library (arXiv) article: http://... |
| 21 | Svitlana Gorobets | Cornell University Library (arXiv) article: http://... |
| 22 | Oksana Gorobets | Cornell University Library (arXiv) article: http://... |
| 23 | JG Brookshear | no Author Account link |
| 24 | PJ Denning | Google Scholar Author Account link: https://... |
| 25 | A McGettrick | no Author Account link |
| 26 | G Dodig-Crnkovic | Google Scholar Author Account link: https://... |
| 27 | A Ralston | no Author Account link |
| 28 | ED Reilly | no Author Account link |
| 29 | D Hemmendinger | no Author Account link |
| 30 | SM Ross | Google Scholar Author Account link: https://... |

Рисунок 6.2.7 – Приклад зображення зібраної інформації через стороннє програмне забезпечення

Зібрана під час виконання алгоритму інформація зберігається в трьох головних таблицях Article, Author, Category. Зокрема є три допоміжні таблиці, які містять в собі інформацію про унікальні номери в системі (ID) для кожного ключового елементу, а саме: статті, автора та категорії. Допоміжні таблиці створені з метою зберігання зв’язку між: статтями та їх категоріями, статтями та їх авторами, авторами та категоріями(науковими напрямками), в яких автор має напрацювання. В разі відсутності необхідної інформації у джерелі, інформація заміняється на статичне значення.

# 7 СИСТЕМНІ ВИМОГИ ТА ДОДАТКОВЕ ПРОГРАМНЕ ЗАБЕСПЕЧЕННЯ

Для коректної роботи програми користувачу необхідно задовільнити наступні умови:

- Процесор з двома ядрами тактовою частотою від 2.1-2.5 або краще.
- Вільне місце на накопичувачу від 2 гігабайтів.
- Оперативна пам’ять не меньше 2 гігабайтів.
- Операційна система Windows 7 / Ubuntu 16.04 або новіші.

Для роботи з розробленим програмним забезпеченням необхідно інсталювати інтерпретатор мови програмування Python версії не нижче 3.9 та вказані у вкладеному файлі бібліотеки, які необхідні для програмного забезпечення.

Під час користування програмою необхідне якісне з’єднання з інтернетом, оскільки програма взаємодіє з Інтернет-ресурсами.

Для взаємодії з даними у базі даних необхідно встановити СУБД з можливістю взаємодіяти з базами даних на структурі SQLite

# ВИСНОВКИ

В ході виконання поставленої задачі було створено систему та отримано наступні результати:

- Проведено аналіз існуючих аналогів для виконання поставленої задачі. Було вирішено створити власне програмне забезпечення з урахуванням усіх потреб та побажань можливих користувачів системи.
- Досліджено методи парсингу для ресурсів з аналогічною структурою веб-сайтів та з подібними даними, які зберігаються в середині цих ресурсів.
- Було обрано зручну та ефективну мову програмування, середу розробки програмного забезпечення та бібліотеки для розробки програмного забезпечення, яке буде виконувати поставлену задачу.

В ході розробки було досліджено предметну область та проаналізовано спеціалізовану літературу. Були здобуті навички праці зі спеціалізованими бібліотеками мови програмування, які були створенні для виконання подібних задач.

На основі отриманих навичок було побудовано відповідну систему, яка виконує поставлену задачу. Система створена без потреби опанування особливих навичок для користування повним функціоналом системи, з метою виконання задачі по збору та первинної обробки бібліографічних даних.

Спроектована система розроблена з урахуванням її подальшого можливого розвитку. Були проаналізовані можливі сценарії подальшого розвитку систему, які можуть покращити результати роботи.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Етапи створення веб-сайтів [Електронний ресурс]— Режим доступу до ресурсу: http://alextexnok.blogspot.com/p/blog-page_85.html (дата звернення 10.04.2022).
2. Поняття, структура та різновиди веб-сайтів. Автоматизоване розроблення веб-сайтів: [Електронний ресурс] — Режим доступу до ресурсу: http://www.ndu.edu.ua/liceum/web.pdf (дата звернення 14.04.2022).
3. Python documentation [Електронний ресурс] — Режим доступу до ресурсу: https://docs.python.org/3/ (дата звернення 15.04.2022).
4. Peewee documentation [Електронний ресурс] — Режим доступу до ресурсу: http://docs.peewee-orm.com/en/latest/ (дата звернення 11.04.2022).
5. PyQt5 documentation [Електронний ресурс] — Режим доступу до ресурсу: https://doc.qt.io/qtforpython/ (дата звернення 12.04.2022).
6. PyQt5 керівництво [Електронний ресурс] — https://pythonist.ru/rukovodstvo-po-pyqt5/ (дата звернення 18.04.2022).
7. SQLite tutorial [Електронний ресурс] — https://unetway.com/tutorial/sqlite (дата звернення 19.04.2022).
8. ORM [Електронний ресурс] — https://simpleone.ru/glossary/orm/ (дата звернення 10.04.2022).
9. Google Академія для науковців [Електронний ресурс] — Режим доступу до ресурсу: http://www.library.univ.kiev.ua/ukr/res/google_scholar.pdf (дата звернення 14.04.2022).
10. Wolfram Alpha [Електронний ресурс] — Режим доступу до ресурсу: https://www.wolfram.com/education/?source=nav (дата звернення 17.04.2022).
11. ScienceDirect [Електронний ресурс] — https://kai.ru/documents/678086/10143360/ScienceDirect%D0%B1%D1%83%D0%BA%D0%BB%D0%B5%D1%82.pdf/57c4d12d-cc9f-4f10-88ca-146977720980 (дата звернення 18.04.2022).

Додаток А

Класифікація наукових текстових масивів за параметрами

Текст програмного модулю

УКР.НТУУ“КПІ ім. Ігоря Сікорського”.ТР82357_22Б 12-1

Аркушів 35

Київ 2022

Файл op_scholar.py – виконання усіх пошукових операцій на платформі Google Scholar

```
# імпортування модулів
import scholarly
from scholarly import ProxyGenerator
from scholarly import scholarly

# Імпортування моделей головних таблиць
from Models.mod_authors import Author
from Models.mod_article import Article
from Models.mod_categories import Category

#Імпортування моделей допоміжних таблиць
from Models.mod_auth_cat import Auth_Cat
from Models.mod_art_cat import Art_Cat
from Models.mod_con_art_auth import Auth_Art

#Імпортування конфігураційних даних
from database_con.db_config import sch_key, db

# Підключення проксі генератора
pg = ProxyGenerator()
success = pg.ScraperAPI(str(sch_key))
scholarly.use_proxy(pg)
#Функція пошуку на платформі Google Scholar за ключовими словами або повною назвою у назві статті
def search_by_keyword(parametr,search_amount):
    res_count = 0
```

```
    pub_Links_separ = "  "
    raw_cat = "Category unknown"

# Створення запиту та занесення його результатів у 1 змінну
    search_query = scholarly.search_pubs(parametr)
    search_number = 0

#Перебирання кожного результату пошуку циклом
        for res in search_query:
            res_count += 1
            search_number += 1
            if search_number == search_amount:
                break
            else:
                authors_list = []
                authors_id = []

#Занесення даних у потрібному вигляді у змінні
            result_text = res['bib']
            raw_title = result_text['title']
            raw_abstract = result_text['abstract']
            raw_year = result_text['pub_year']
            raw_category = raw_cat

            result_author_id = res['author_id']
            result_author = result_text['author']
            for item in result_author:
                authors_list.append(item)
```

```
#Присвоєння кожному автору посилань на профіль(якщо існують)
        for item in result_author_id:
            link = "no Author Account link"
            if (item !=("")):
                lin = "Google Scholar Author Account link:
https://scholar.google.com/citations?hl=uk&user="
                link = lin + item
                authors_id.append(link)
            else:
                authors_id.append(link)
        i = 0
        while (i < len(authors_list)):
            author_data = authors_list[i]
            auth_acc_link = (authors_id[i])

      #Перевірка на існування записів про даного автора у базі даних

   author_check = None

            aut_check = (Author.select(Author.name).where(Author.name == author_data))
            for r in aut_check:
                author_check = r

#Занесення у базу даних , якщо запис відсутній

            if (author_check == None):
                links = auth_acc_link + pub_Links_separ
```

```
            record_author = Author.create(name=author_data, links=links)
            record_author.save()
        i += 1

    raw_authors = ', '.join(authors_list)

#Перевірка на існування записів про дану статтю у базі даних
    art_title_check = None

    art_check = (Article.select(Article.title).where((Article.title == raw_title) &
(Article.authors == raw_authors)))
    for r in art_check:
        art_title_check = r

#Занесення у базу даних , якщо запис відсутній

    if (art_title_check == None):
        record_articles = Article.create(title=raw_title,
                                         authors=raw_authors,
                                         abstract=raw_abstract,
                                         pub_date=raw_year,
                                         categories=raw_category)
        record_articles.save()
        db.commit()

#Перевірка на існування записів про дану категорію у базі даних
    cat_check = None
```

```
        aut_check = (Category.select(Category.cat_name).where(Category.cat_name ==
raw_category))

        for r in aut_check:
            cat_check = r



#Занесення у базу даних , якщо запис відсутній
        if (cat_check == None):
            record_cat = Category.create(cat_name=raw_category)
            record_cat.save()

#Перевірка на існування записів про зібрану та вже наявну інформацію в базі даних
та присвоєння їм унікальних номерів(ID)

      for authors in authors_list:

        query_art_id_extr = (Article.select(Article.art_id).where(Article.title == raw_title))
        for id in query_art_id_extr:
            raw_art_id = id.art_id

        query_auth_id_extr = (Author.select(Author.author_id).where(Author.name ==
authors))

        for id in query_auth_id_extr:
            raw_auth_id = id.author_id
```

```
        query_auth_id_extr = (Category.select(Category.cat_id).where(Category.cat_name
== raw_category))

        for id in query_auth_id_extr:
            raw_cat_id = id.cat_id

        aut_cat_check = None

        aut_cat = (Auth_Cat.select(Auth_Cat.record_id).where(
            (Auth_Cat.author_id == raw_auth_id) & (Auth_Cat.cat_id == raw_cat_id)))
        for r in aut_cat:
            aut_cat_check = r

#Створення нових зв’язків, з урахуванням вже наявних у базі даних
        if (aut_cat_check == None):
            record_Auth_cat = Auth_Cat.create(author_id=raw_auth_id, cat_id=raw_cat_id)
            record_Auth_cat.save()

        art_cat_check = None

        art_cat = (Art_Cat.select(Art_Cat.record_id).where(
            (Art_Cat.art_id == raw_art_id) & (Art_Cat.cat_id == raw_cat_id)))
        for r in art_cat:
            art_cat_check = r

        if (art_cat_check == None):
            record_Art_cat = Art_Cat.create(art_id=raw_art_id, cat_id=raw_cat_id)
            record_Art_cat.save()
```

```
            auth_art_check = None

            auth_art = (Auth_Art.select(Auth_Art.record_id).where(
                (Auth_Art.art_id == raw_art_id) & (Auth_Art.author_id == raw_auth_id)))

            for r in auth_art:
                auth_art_check = r

            if (auth_art_check == None):
                record_Auth_Art = Auth_Art.create(author_id=raw_auth_id, art_id=raw_art_id)
                record_Auth_Art.save()
    return (res_count)

#Функція пошуку на платформі Google Scholar за авторами статей
def author_searcher(parameter, max_res):
    res_count = 0

# Створення запиту та занесення його результатів у 1 змінну
    search_query = scholarly.search_author(parameter)

#Перебирання кожного результату пошуку циклом
    for res in search_query:
        res_count += 1
        if(res_count == max_res):
            break
        else:
```

```
#Занесення даних у потрібному вигляді у змінні
        raw_category = "Category unknown"
        raw_authors = res["name"]
        link_id = res["scholar_id"]
        link_pre = "Google Scholar Author Account link:
https://scholar.google.com/citations?hl=uk&user="
        link = link_pre + link_id
        filled_author = scholarly.fill(res, sections=["publications"])
        pubs = filled_author["publications"]

#Перевірка на наявність записів про цього автора
        author_check = None

        aut_check = (Author.select(Author.name).where(Author.name == raw_authors))
        for r in aut_check:
            author_check = r

#Занесення у разі відсутності записів
        if (author_check == None):
            links = link
            record_author = Author.create(name=raw_authors, links=links)
            record_author.save()

#Перебирання статей, які написані цим автором та присвоєння зміним необхідних
даних
        for p in pubs:
            r = p["bib"]
            raw_title = (r["title"])
```

```
            try:
                raw_year = (r["pub_year"])
            except KeyError:
                raw_year = "1111"

#Перевірка на наявність даних про цю статтю у базі
            art_title_check = None
            art_check = (
                Article.select(Article.title).where((Article.title == raw_title) &
(Article.authors == raw_authors)))
            for r in art_check:
                art_title_check = r

#Занесення у разі відсутності записів
            if (art_title_check == None):
                record_articles = Article.create(title=raw_title,
                                                 authors=raw_authors,
                                                 pub_date=raw_year,
                                                 categories=raw_category)
                record_articles.save()
                db.commit()


#Перевірка на наявність записів про категорію
            cat_check = None
            aut_check = (Category.select(Category.cat_name).where(Category.cat_name ==
raw_category))
```

```
        for r in aut_check:
            cat_check = r
#Занесення у разі відсутності записів
        if (cat_check == None):
            record_cat = Category.create(cat_name=raw_category)
            record_cat.save()

#Перевірка на існування записів про зібрану та вже наявну інформацію в базі даних та присвоєння їм унікальних номерів(ID)

        query_art_id_extr = (Article.select(Article.art_id).where(Article.title == raw_title))
        for id in query_art_id_extr:
            raw_art_id = id.art_id

        query_auth_id_extr = (Author.select(Author.author_id).where(Author.name == raw_authors))

        for id in query_auth_id_extr:
            raw_auth_id = id.author_id

        query_auth_id_extr = (Category.select(Category.cat_id).where(Category.cat_name == raw_category))

        for id in query_auth_id_extr:
            raw_cat_id = id.cat_id
```

```
#Створення нових зв’язків, з урахуванням вже наявних у базі даних
        aut_cat_check = None

        aut_cat = (Auth_Cat.select(Auth_Cat.record_id).where(
          (Auth_Cat.author_id == raw_auth_id) & (Auth_Cat.cat_id == raw_cat_id)))
        for r in aut_cat:
          aut_cat_check = r

        if (aut_cat_check == None):
          record_Auth_cat = Auth_Cat.create(author_id=raw_auth_id,
cat_id=raw_cat_id)
          record_Auth_cat.save()

        art_cat_check = None

        art_cat = (Art_Cat.select(Art_Cat.record_id).where(
          (Art_Cat.art_id == raw_art_id) & (Art_Cat.cat_id == raw_cat_id)))
        for r in art_cat:
          art_cat_check = r

        if (art_cat_check == None):
          record_Art_cat = Art_Cat.create(art_id=raw_art_id, cat_id=raw_cat_id)
          record_Art_cat.save()

        auth_art_check = None

        auth_art = (Auth_Art.select(Auth_Art.record_id).where(
          (Auth_Art.art_id == raw_art_id) & (Auth_Art.author_id == raw_auth_id)))
```

```
            for r in auth_art:
                auth_art_check = r

            if (auth_art_check == None):
                record_Auth_Art = Auth_Art.create(author_id=raw_auth_id,
art_id=raw_art_id)
                record_Auth_Art.save()
```

Файл op_arxiv.py – файл відповідає за усі пошукові операції для платформи ArXiv

```
# імпортування модулів

import arxiv
# Імпортування моделей головних таблиць
from Models.mod_article import Article
from Models.mod_authors import Author
from Models.mod_categories import Category

#Імпортування моделей допоміжних таблиць
from Models.mod_auth_cat import Auth_Cat
from Models.mod_art_cat import Art_Cat
from Models.mod_con_art_auth import Auth_Art

#Імпортування конфігураційних даних
from database_con.db_config import db



#Загальна функція пошуку за категоріями та ключовими словами або  назвою статті
def searcher(parametr,search_amount,search_category):
```

```
    res_count = 0
    platform = "Cornell University Library (arXiv) article: "
    searching = search_category + parametr
    search = arxiv.Search(
        query = searching,
        max_results = search_amount,
        sort_by=arxiv.SortCriterion.Relevance
    )
    for result in search.results():
        res_count += 1
        authors_list = []
        raw_title = result.title
        raw_year = result.published
        raw_year = raw_year.year
        raw_category = result.primary_category
        for author in result.authors:
            authors_list.append(str(author))
        raw_authors = ', '.join(authors_list)
        raw_abstract = result.summary

        art_title_check = None

        art_check = (Article.select(Article.title).where((Article.title == raw_title) &
(Article.authors == raw_authors)))
        for r in art_check:
            art_title_check = r

        if (art_title_check == None):
```

```
        record_articles = Article.create(title=raw_title,
                                         authors=raw_authors,
                                         abstract= raw_abstract,
                                         pub_date=raw_year,
                                         categories=raw_category)
        record_articles.save()
        db.commit()

    for authors in authors_list:
        info = platform + result.entry_id

        author_check = None

        aut_check = (Author.select(Author.name).where(Author.name == authors))
        for r in aut_check:
            author_check = r

        if (author_check == None):

            record_author = Author.create(name=authors, links=info)
            record_author.save()

        cat_check = None
        cat_recheck = (Category.select(Category.cat_name).where(Category.cat_name ==
raw_category))

        for r in cat_recheck:
            cat_check = r
```

```
        if (cat_check == None):

            record_cat = Category.create(cat_name=raw_category)
            record_cat.save()

    for authors in authors_list:

        query_art_id_extr = (Article.select(Article.art_id).where(Article.title == raw_title))
        for id in query_art_id_extr:
            raw_art_id = id.art_id

        query_auth_id_extr = (Author.select(Author.author_id).where(Author.name ==
authors))

        for id in query_auth_id_extr:
            raw_auth_id = id.author_id

        query_auth_id_extr = (Category.select(Category.cat_id).where(Category.cat_name
== raw_category))

        for id in query_auth_id_extr:
            raw_cat_id = id.cat_id

        aut_cat_check = None

        aut_cat = (Auth_Cat.select(Auth_Cat.record_id).where((Auth_Cat.author_id ==
raw_auth_id) & (Auth_Cat.cat_id == raw_cat_id)))
```

```
        for r in aut_cat:
            aut_cat_check = r

        if (aut_cat_check == None):
            record_Auth_cat = Auth_Cat.create(author_id=raw_auth_id, cat_id=raw_cat_id)
            record_Auth_cat.save()

        art_cat_check = None

        art_cat = (Art_Cat.select(Art_Cat.record_id).where((Art_Cat.art_id == raw_art_id)
& (Art_Cat.cat_id == raw_cat_id)))
        for r in art_cat:
            art_cat_check = r

        if (art_cat_check == None):
            record_Art_cat = Art_Cat.create(art_id=raw_art_id, cat_id=raw_cat_id)
            record_Art_cat.save()

        auth_art_check = None

        auth_art = (Auth_Art.select(Auth_Art.record_id).where((Auth_Art.art_id ==
raw_art_id) & (Auth_Art.author_id == raw_auth_id)))

        for r in auth_art:
            auth_art_check = r

        if (auth_art_check == None):
            record_Auth_Art = Auth_Art.create(author_id=raw_auth_id, art_id=raw_art_id)
```

```
                record_Auth_Art.save()
    return(res_count)
# функція яка визиває пошук та надає параметр (назва статті або ключові слова)
def searched_by_title_func(parametr,search_amount):
    search_category = "ti:"
    res_count =searcher(parametr,search_amount,search_category)
    return (res_count)

# функція яка визиває пошук та надає параметр (категорії)
def searched_by_category(parametr,search_amount):
    search_category = "cat:"
    res_count = searcher(parametr, search_amount, search_category)
    return (res_count)

#функція яка здійснює пошук за автором статті
def searcher_authors(parametr,search_amount):
    res_count = 0
    platform = "Cornell University Library (arXiv) article: "
    search_category = "au:"
    searching = search_category + parametr
    search = arxiv.Search(
        query = searching,
        max_results = search_amount,
        sort_by=arxiv.SortCriterion.Relevance
    )
    for result in search.results():
        res_count += 1
        authors_list = []
```

```
        raw_title = result.title
        raw_year = result.published
        raw_year = raw_year.year
        raw_category = result.primary_category
        for author in result.authors:
            authors_list.append(str(author))
        raw_authors = ', '.join(authors_list)
        raw_abstract = result.summary

        art_title_check = None

        art_check = (Article.select(Article.title).where((Article.title == raw_title) &
(Article.authors == raw_authors)))
        for r in art_check:
            art_title_check = r

        if (art_title_check == None):
            record_articles = Article.create(title=raw_title,
                                             authors=raw_authors,
                                             abstract= raw_abstract,
                                             pub_date=raw_year,
                                             categories=raw_category)
            record_articles.save()



        for authors in authors_list:
            info = platform + result.entry_id
```

```
        author_check = None

        aut_check = (Author.select(Author.name).where(Author.name == authors))
        for r in aut_check:
            author_check = r

        if (author_check == None):

            record_author = Author.create(name=authors, links=info)
            record_author.save()

        cat_check = None
        aut_check = (Category.select(Category.cat_name).where(Category.cat_name ==
raw_category))

        for r in aut_check:
            cat_check = r

        if (cat_check == None):

            record_cat = Category.create(cat_name=raw_category)
            record_cat.save()




    for authors in authors_list:
```

```
        query_art_id_extr = (Article.select(Article.art_id).where(Article.title == raw_title))
        for id in query_art_id_extr:
            raw_art_id = id.art_id

        query_auth_id_extr = (Author.select(Author.author_id).where(Author.name ==
authors))

        for id in query_auth_id_extr:
            raw_auth_id = id.author_id

        query_auth_id_extr = (Category.select(Category.cat_id).where(Category.cat_name
== raw_category))

        for id in query_auth_id_extr:
            raw_cat_id = id.cat_id



        aut_cat_check = None

        aut_cat = (Auth_Cat.select(Auth_Cat.record_id).where((Auth_Cat.author_id ==
raw_auth_id) & (Auth_Cat.cat_id == raw_cat_id)))
        for r in aut_cat:
            aut_cat_check = r

        if (aut_cat_check == None):
            record_Auth_cat = Auth_Cat.create(author_id=raw_auth_id, cat_id=raw_cat_id)
            record_Auth_cat.save()
```

```
        art_cat_check = None

        art_cat = (Art_Cat.select(Art_Cat.record_id).where((Art_Cat.art_id == raw_art_id)
& (Art_Cat.cat_id == raw_cat_id)))
        for r in art_cat:
            art_cat_check = r

        if (art_cat_check == None):
            record_Art_cat = Art_Cat.create(art_id=raw_art_id, cat_id=raw_cat_id)
            record_Art_cat.save()



        auth_art_check = None

        auth_art = (Auth_Art.select(Auth_Art.record_id).where((Auth_Art.art_id ==
raw_art_id) & (Auth_Art.author_id == raw_auth_id)))

        for r in auth_art:
            auth_art_check = r

        if (auth_art_check == None):
            record_Auth_Art = Auth_Art.create(author_id=raw_auth_id, art_id=raw_art_id)
            record_Auth_Art.save()
    return (res_count)
```

Файл db_simple_op – Зберігає функції для роботи з базою даних через ORM такі як: підключення , зберігання даних, відключення від неї, створення нової бази даних

```
#Підключення моделей усіх таблиць
from Models.mod_auth_cat import Auth_Cat
from Models.mod_con_art_auth import Auth_Art
from Models.mod_article import Article
from Models.mod_authors import Author
from Models.mod_categories import Category
from Models.mod_art_cat import  Art_Cat
from database_con.db_config import db

#Функція створення нової бази даних та підключення до неї
def create_new_database():
    db.connect()
    db.create_tables([Article, Author, Category, Auth_Art, Art_Cat, Auth_Cat])
    db.commit()
    db.close()

#Функція підключення до існуючої бази даних
def connect_to_database():
    db.connect()
    print('connected')
#Функція збереження змін у базі даних
def commit_database():
    db.commit()
    print('changes was saved')
#функція відключення бази даних
```

```
def disconect_database():
    print('exit')
    db.close()
```

Файл main.py – зберігає усі сценарії , які будуть виконуватись при виклику за відповідними кнопками у інтерфейсі

```
#Імпортування модулів та файлів
import os
import shutil

from datetime import datetime

import op_arxiv
import op_scholar

from database_con.db_simple_op import *
from database_con.db_config import db_standart_path , db_record_path ,db_session_path

#Функція , яка перевіряє чи існує база даних у відповідній папці, якщо ні то створює автоматично
def db_checker():
    path = db_standart_path
    if (os.path.exists(path)):
        pass
    else:
        if (os.path.exists('db')):
            create_new_database()
        else:
            os.mkdir('db')
            create_new_database()
#Функція, що зберігає копію бази даних під ім’ям що задав користувач, або використовує шаблон: record_час зберігання копії.db

def record_saver(file_name):
```

```
    extension = ".db"
    if(len(file_name) < 3):
        record_time = datetime.now().strftime('%d_%m_%Y_%H_%M_%S')
        record = "Record-"
        timer = str(record_time)
        new_record_path = db_record_path + record + timer + extension
    else:
        record = file_name
        new_record_path = db_record_path + record + extension
    file_oldname = os.path.join(db_standart_path)
    file_newname_newfile = os.path.join(new_record_path)
    shutil.copy(file_oldname, file_newname_newfile)

#Функція яка зберігає копію бази даних при коректному виході з програми. Копія зберігається як session_часвиходу.db
def session_saver():
    record_time = datetime.now().strftime('%d_%m_%Y_%H_%M_%S')
    timer = str(record_time)
    extension = ".db"
    new_record_path = db_session_path + timer + extension
    file_oldname = os.path.join(db_standart_path)
    file_newname_newfile = os.path.join(new_record_path)
    shutil.copy(file_oldname, file_newname_newfile)

#Функція , що викликає пошук за назвою статті на платформі ArXiv
def searched_by_title_arx(user_parametr,user_max_res):
    parametr = user_parametr
    search_amount = user_max_res
```

```
    res_arx_count = op_arxiv.searched_by_title_func(parametr, search_amount)
    print(res_arx_count)
    commit_database()
#Функція , що викликає пошук за назвою статті на платформі Google Scholar
def searched_by_title_sch(user_parametr,user_max_res):
    parametr = user_parametr
    search_amount = user_max_res
    res_sch_count = op_scholar.search_by_keyword(parametr, search_amount)
    print(res_sch_count)
    commit_database()

#Функція , що викликає пошук за автором на платформі ArXiv
def searched_by_author_arx(user_parametr,user_max_res):

    parametr= user_parametr
    search_amount = user_max_res
    res_arx_count = op_arxiv.searcher_authors(parametr,search_amount)
    print(res_arx_count)
    commit_database()

#Функція , що викликає пошук за автором на платформі Google Scholar
def searched_by_author_sch(user_parametr,user_max_res):
    parametr = user_parametr
    op_scholar.author_searcher(parametr,user_max_res)
    #op_scholar.search_author(parametr)
    commit_database()
#Функція , що викликає пошук за категорією на платформі ArXiv
```

```
def searched_by_category_arx(user_parametr,user_max_res):
    parametr= user_parametr
    search_amount = user_max_res
    res_arx_count = op_arxiv.searched_by_category(parametr, search_amount)
    print(res_arx_count)
    commit_database()
```

Файл pyqt.py – Файл, який запускає інтерпретатор, має в собі клас, який наслідує прописаний інтерфейс. Має прописані функції, які будуть визивати відповідні сценарії при взаємодії користувача з інтерфейсом

```
#Імпорт модулів та класу, в якому знаходиться прототип інтерфейсу
import table_model
from main import *
import sys
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QMessageBox, qApp, QHeaderView, QInputDialog
import Menu

#Клас , який наслідує прототип інтерфейсу. Пов’язує виклик відповідних функцій при взаємодії користувача з інтерфейсом
class Form(QtWidgets.QMainWindow, Menu.Ui_MainWindow):
    def __init__(self):
        db_checker()
        connect_to_database()
        super(Form, self).__init__()
        self.setupUi(self)
        self.Search_button.clicked.connect(self.inputdata)
        self.Save_session.clicked.connect(self.save_session)
        self.Save_record.clicked.connect(self.save_record)
        self.Refresh_info.clicked.connect(self.based)
        self.based()
        self.categories = {…}
        self.Category_box.addItems(self.categories.keys())
        self.Search_by_box.currentTextChanged.connect(self.categ_ch
```

```
#Функція , яка зчитує дані, що знаходяться у інтерфейсі під час її виклику.
Перевіряє внесені дані та викликає відповідний сценарій
    def inputdata(self):
        self.Save_record.setEnabled(False)
        self.Save_session.setEnabled(False)
        self.Search_button.setEnabled(False)
        try:
            if(self.Parametr_line.text() == ""):
                parameter = "No"
            else:
                parameter = self.Parametr_line.text()

            if (self.Category_box.currentText() == ""):
                category = "Nothing"
            else:
                category = self.categories[str(self.Category_box.currentText())]

            max_res = int(self.Max_results_box.value())
            search_by = self.Search_by_box.currentText()
            qApp.processEvents()

            if not self.CheckBox_Arxiv.isChecked() and not
self.Check_box_goo_sch.isChecked():
                error = QMessageBox()
                error.setText("resourse wasnt picked at all")
                error.setStandardButtons(QMessageBox.Ok)
                error.exec()
```

```
        elif self.CheckBox_Arxiv.isChecked() and not
self.Check_box_goo_sch.isChecked():
            if search_by == "Title":
                if (parameter == "No") or (len(parameter) < 3):
                    error = QMessageBox()
                    error.setText("Parameter has no data")
                    error.setStandardButtons(QMessageBox.Ok)
                    error.exec()
                else:
                    error = QMessageBox()
                    error.setText("Search has been finished")
                    error.setStandardButtons(QMessageBox.Ok)
                    searched_by_title_arx(parameter, max_res)
                    error.exec()
            elif search_by == "Category":
                if (category == "Nothing"):
                    error = QMessageBox()
                    error.setText("Category wasnt picked")
                    error.setStandardButtons(QMessageBox.Ok)
                    error.exec()
                else:
                    error = QMessageBox()
                    error.setText("Search has been finished")
                    error.setStandardButtons(QMessageBox.Ok)
                    searched_by_category_arx(category, max_res)
                    error.exec()
            elif search_by == "Author":
                if (parameter == "No") or (len(parameter) < 3):
```

```
                error = QMessageBox()
                error.setText("Parameter has no data")
                error.setStandardButtons(QMessageBox.Ok)
                error.exec()
            else:
                error = QMessageBox()
                error.setText("Search has been finished")
                error.setStandardButtons(QMessageBox.Ok)
                searched_by_author_arx(parameter, max_res)
                error.exec()
        else:
            error = QMessageBox()
            error.setText("Search_by wasn`t picked")
            error.setStandardButtons(QMessageBox.Ok)
            error.exec()

    elif not self.CheckBox_Arxiv.isChecked() and
self.Check_box_goo_sch.isChecked():
        if search_by == "Title":
            if (parameter == "No") or (len(parameter) < 3):
                error = QMessageBox()
                error.setText("Parameter has no data")
                error.setStandardButtons(QMessageBox.Ok)
                error.exec()
            else:
                error = QMessageBox()
                error.setText("Search has been finished")
                error.setStandardButtons(QMessageBox.Ok)
```

```
            searched_by_title_sch(parameter, max_res)
            error.exec()

    elif search_by == "Author":
        if (parameter == "No") or (len(parameter) < 3):
            error = QMessageBox()
            error.setText("Parameter has no data")
            error.setStandardButtons(QMessageBox.Ok)
            error.exec()
        else:
            error = QMessageBox()
            error.setText("Search has been finished")
            error.setStandardButtons(QMessageBox.Ok)
            searched_by_author_sch(parameter, max_res)
            error.exec()
    else:
        error = QMessageBox()
        error.setText("Search_by wasn`t picked")
        error.setStandardButtons(QMessageBox.Ok)
        error.exec()
elif self.CheckBox_Arxiv.isChecked() and self.Check_box_goo_sch.isChecked():
    if search_by == "Title":
        if (parameter == "No") or (len(parameter) < 3):
            error = QMessageBox()
            error.setText("Parameter has no data")
            error.setStandardButtons(QMessageBox.Ok)
            error.exec()
        else:
```

```
            error = QMessageBox()
            error.setText("Search has been finished")
            error.setStandardButtons(QMessageBox.Ok)
            searched_by_title_arx(parameter, max_res)
            searched_by_title_sch(parameter, max_res)
            error.exec()
    elif search_by == "Author":
        if (parameter == "No") or (len(parameter) < 3):
            error = QMessageBox()
            error.setText("Parameter has no data")
            error.setStandardButtons(QMessageBox.Ok)
            error.exec()
        else:
            error = QMessageBox()
            error.setText("Search has been finished")
            error.setStandardButtons(QMessageBox.Ok)
            searched_by_author_sch(parameter, max_res)
            searched_by_author_arx(parameter, max_res)
            error.exec()
    elif search_by == "Category":
        if (category == "Nothing"):
            error = QMessageBox()
            error.setText("Category wasnt picked")
            error.setStandardButtons(QMessageBox.Ok)
            error.exec()
        else:
            error = QMessageBox()
            error.setText("Search has been finished")
```

```
                error.setStandardButtons(QMessageBox.Ok)
                searched_by_category_arx(category, max_res)
                error.exec()
            else:
                error = QMessageBox()
                error.setText("Search_by wasn`t picked")
                error.setStandardButtons(QMessageBox.Ok)
                error.exec()
        except Exception as err:
            print(err)

        self.Save_session.setEnabled(True)
        self.Save_record.setEnabled(True)
        self.Search_button.setEnabled(True)

#Функція, що викликає сценарій збереження сесії кориистувача
    def save_session(self):
        self.Save_record.setEnabled(False)
        self.Save_session.setEnabled(False)
        self.Search_button.setEnabled(False)
        session_saver()
        disconect_database()
        App.exit(0)



#Функція , що викликає сценарій збереження запису користувачем
    def save_record(self):
        self.Save_record.setEnabled(False)
```

```
        self.Save_session.setEnabled(False)
        self.Search_button.setEnabled(False)
        text, ok = QInputDialog.getText(self,'Name your saving file', 'Name of your record')
        if ok:
            file_name = str(text)
            record_saver(file_name)
        self.Save_session.setEnabled(True)
        self.Save_record.setEnabled(True)
        self.Search_button.setEnabled(True)

    def based(self):
self.tableView.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        articles = Article.select().tuples()
        model = table_model.MyTableModel(articles)
        self.tableView.setModel(model)
#Функція, яка проводить зміни у інтерфейсі, в залежності від обраних параметрів
    def categ_ch(self):
        if(self.Search_by_box.currentText() == "Title"):
            self.Label_Searching_parametr.show()
            self.Parametr_line.show()
            self.Label_category.hide()
            self.Category_box.hide()
            self.Check_box_goo_sch.show()

        elif(self.Search_by_box.currentText() == "Author"):
            self.Label_Searching_parametr.show()
            self.Parametr_line.show()
            self.Label_category.hide()
```

```
            self.Category_box.hide()
            self.Check_box_goo_sch.show()

        elif (self.Search_by_box.currentText() == "Category"):
            self.Category_box.show()
            self.Label_category.show()
            self.Label_Searching_parametr.hide()
            self.Parametr_line.hide()
            self.Check_box_goo_sch.hide()

        else:
            self.Label_Searching_parametr.show()
            self.Label_category.show()
            self.Label_Search_by.show()
            self.Check_box_goo_sch.show()
            self.Parametr_line.show()
            self.Category_box.show()
            self.Check_box_goo_sch.show()

#Функції , що відкривають інтерфейс при запуску програми
App = QtWidgets.QApplication(sys.argv)
window = Form()
window.show()
App.exec()
```