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

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення

на тему Аналіз характеристик даних, що збираються з відібраних джерел науково – технічної інформації

Виконав: студент 4 курсу, групи ТІ-62

Квасницький Роман Ігорович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник Воронько Максим Платонович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент Реуцький Микола Олександрович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення веб-технологій та мобільних пристроїв

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

ЗАВДАННЯ
на дипломну роботу студенту

Квасницькому Роману Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Аналіз характеристик даних, що збираються з відібраних джерел науково–технічної інформації
керівник роботи асистент Воронько Максим Платонович_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № 1168-с

2. Строк подання студентом роботи 10 червня 2020 р.
3. Вихідні дані до роботи: мова програмування – Java, середовище розробки Intellij IDEA
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): проаналізувати задачу створення аналітичної платформи для аналізу науково-технічої літератури, спроектувати та розробити програмне забезпечення для аналізу та протестувати його
5. Перелік ілюстративного матеріалу: схеми архітектури додатку, знімки екранних форм, діаграма прецедентів системи, діаграма структури системи, зразки розробленого інтерфейсу додатку.
6. Дата видачі завдання: ”25” листопада 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	12.02.2020 – 18.03.2020	
2.	Вивчення та аналіз задачі	27.03.2020 – 03.04.2020	
3.	Розробка архітектури та загальної структури системи	09.04.2020 – 16.04.2020	
4.	Розробка структур окремих підсистем	17.04.2020 – 29.04.2020	
5.	Програмна реалізація системи	30.04.2020 – 12.05.2020	
6.	Оформлення пояснювальної записки	03.05.2020 – 05.06.2020	
7.	Захист програмного продукту	09.06.2020	
8.	Передзахист	09.06.2020	
9.	Захист	16.06.2020	

Студент _____ Квасницький Р.І.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Воронько М.П.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Дана робота присвячена розробці аналітичної платформи для аналізу науково-технічної літератури.

Метою роботи є аналіз застосунків-конкурентів й розробка з подальшим впровадженням власної системи, в якій відсутні недоліки аналогів.

В ході роботи було вирішено наступні задачі:

1. Провести аналіз застосунків-конкурентів.
2. Визначити засоби розробки для створення аналітичної платформи.
3. Здійснити проєктний аналіз і розробити каркас застосунку.
4. Забезпечити безперебійну роботу платформи.

В роботі було використано платформу JVM, для якої розробка здійснювалась засобами мови програмування Java.

Обсяг роботи складає 84 сторінки, 46 зображень, 3 використаних джерела та 3 додатки.

Ключові слова: граф цитованості, наукові публікації, бібліографія, наукометрика.

ABSTRACT

This thesis is devoted to development of an analytical platform for scientific literature.

The purpose of the work is to examine existing analogues of such system, design and develop own solution without disadvantages of analogues.

In order to achieve this goal several tasks were solved:

1. Analyze existing analogues.
2. Define tools for development of an analytical platform.
3. Design and implement application framework.
4. Ensure uninterrupted operation of the platform.

The JVM platform was used in the work, for which the development was carried out by means of Java programming language.

Explanatory note consists of 84 pages, 46 figures, 3 references and 3 appendixes.

Keywords: citation graph, scientific publications, bibliography, scientometrics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface

DAG – Directed Acyclic Graph

DOI – Digital Object Identifier

IDE – Integrated Development Environment

JAR – Java ARchive

JDK – Java Development Kit

JRE – Java Runtime Environment

JVM – Java Virtual Machine

MVC – Model-View-Contoller

REST – Representational State Transfer

XML – Extensible Markup Language

ЕОМ – Електронно-обчилювальна машина

ЕОП – Електронно-обчислювальний пристрій

НТЛ – Науково-технічна література

ПК – Персональний комп’ютер

ВСТУП

Пошук науково-технічної літератури може бути вкрай витратним за часом заняттям. Наприклад, пошук літератури для оглядової статті, яка має намір освітити певну тему чи область наукових досліджень часто займає багато часу. Також, необхідно докласти чималих зусиль на ознайомлення із науковими матеріалами для того аби точно визначити джерело певного наукового концепту чи методології дослідження. Більш того, в деяких сферах це завдання стає практично неможливим для виконання через неймовірно високу кількість різноманітних наукових напрямів, та більш вузьких сфера, а отже й значно більшу кількість дослідницьких матеріалів, що потребують детального вивчення.

В багатьох випадках, аналіз зв'язків(цитувань) між науковими працями допомагає в пошуках релевантних матеріалів. Наприклад, якщо було визначено релевантну відносно пошукових запитів наукову працю, то додаткові корисні матеріали можуть бути описані як джерела цієї наукової праці. Після аналізу цих праць, може знайтись публікація(чи публікації), на які посилається як наша первинна стаття, так і додаткові матеріали, що може вказувати на те, що публікація має дещо вищу наукову цінність в данній області через частоту її використання як джерела.

Хоча аналіз зв'язків між науковими матеріалами є вкрай ефективним методом для пошуку релевантної наукової літератури, аналіз великої кількості наукових праць забирає надзвичайно багато часу та сил. Саме з метою спрощення та автоматизації цієї рутинної роботи було розроблено такі сервіси як Scopus, Semantic Scholar, Core, Google Scholar та багато інших бібліографій, в яких наявні механізми для пошуку наукових праць. Тим не менш, бібліографічні сервіси надають вкрай обмежені ресурси для користувачів, які прагнуть використовувати подібні сервіси для аналізу зв'язків між науковими працями. В таких сервісах відсутня можливість візуалізації зв'язків між публікаціями, та їх аналізу в необхідній мірі. Хоча в таких сервісах і існує можливість перегляду необхідної інформаціїї, все ще не надається

жодних автоматизованих методів аналізу із зручним інтерфейсом та гнучкою конфігурацією.

Саме тому, метою цієї роботи є розробка інформаційної системи, яка дозволила би користувачам здійснювати пошук необхідних для них матеріалів, отримувати аналітичну інформацію у зручній візуальній інтепретації, що надавала би можливість оцінити користь певної наукової праці, її вплив на область наукового пізнання, та визначити підрозділи й напрями наукової області, отримавши інформацію про наукові роботи, які тісно пов'язані між собою великою кількістю прямих чи транзитивних цитувань(тобто таких, що не вказані в переліку матеріалів, які використовувались у шуканій статті, але вказані в статті(або статтях), які напряму описані у списку джерел шуканної статті чи вказані транзитивно в списках використаної літератури її джерел). Оскільки інструмент має бути універсальним, то в ньому існує підтримка імпортування різних наукометричних баз. Таким чином застосунок залишається незалежним від конкретного постачальника публікацій і увага зосереджується безпосередньо на засобах аналізу.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмний продукт для пошуку наукових праць та аналізу зв’язків між публікаціями.

1.1 Функціональні вимоги до програмного продукту

1. Існує можливість пошуку наукових праць та перегляд метаінформації знайденних матеріалів
2. Існує можливість відображення зв’язків між науковими працями у вигляді неорієнтованного графу, де вершинами виступають статті, а ребрами позначається наявність цитування між працями
3. Існує можливість перегляду результатів пошуку у вигляді деревовидної структури, де внутрішніми вузлами виступають автори наукових праць, чи наукові журнали, а зовнішніми вузлами виступають наукові праці
4. Існує можливість обчислення індексу цитованості для наукових праць, які були визначені як пов’язані із працею, що аналізується, де під індексом цитованості мається на увазі міра частоти використання проміжних чи кінцевих результатів дослідження описаного в статі
5. Існує можливість забарвлення графу відповідно до належності до певної компоненти зв’язності
6. Існує можливість збереження результатів у форматі, що дозволяє відтворити результати аналізу при наступному сеансі роботи із програмним продуктом
7. Існує можливість завантаження структури зв’язків попереднього аналізу для візуалізації його результатів

1.2 Нефункціональні вимоги до програмного продукту

1. Має бути забезпечено належну роботу програмного продукту на наступних операційних системах для ПК:
1. Сімейство Microsoft Windows. Версії XP, 7, Vista, 8, 10.

2. Сімейство Linux. Для мажорних версій ядра 3+

3. Сімейство Solaris. Для мажорних версій 9+

4. Операційна система FreeBSD. Для мажорних версій 9+

2. Графічний інтерфейс має бути виконаний згідно із принципами побудови Material Design застосунків

3. Підбір кольорів для системи має бути гармонійним, та таким що не суперечить правилам естетики

4. Графічний інтерфейс має бути виконано згідно до принципу Оккама, тобто бути мінімальним, але таким що покриває функціональні вимоги

5. Програмний продукт повинен бути розробленим з урахуванням технічних характеристик сучасних робочих станцій та забезпечити стабільну продуктивність і належну утилізацію ресурсів системи.

6. Інтерфейс повинен бути доступним 100% часу роботи програмного продукту, тобто в будь який момент часу користувач може обрати альтернативну операцію користуючись елементами графічного інтерфейсу(наприклад вийти із програми)

1.3 Вхідні та вихідні дані системи

1. До вхідних даних системи відноситься:

1.1. Інформація необхідна для пошуку наукових статей, а саме назва чи автор.

1.2. Натискання на інтерактивні елементи графічного інтерфейсу, наприклад кнопки, чи випадаючі списки.

2. До вихідних даних системи відноситься:

2.1. Список знайдених наукових праць

2.2. Граф із зв’язками між науковими працями

2 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Задачі аналізу НТЛ

Аналіз науково-технічної літератури хоч і є підзадачею аналізу природньої мови, має свої характерні особливості. Для найбільш повного розкриття теми нам необхідно класифікувати методи аналізу науково-технічої літератури за цілями аналізу.

В рамках вищевказаної класифікації ми маємо 2 категорії:

1. Семантичний аналіз вмісту. Данний вид аналізу дозволяє визначати якісні характеристики тексту, такі як:
   1.1. Орфографічні, граматичні та стилістичні помилки.
   1.2. Обсяг тексту.
   1.3. Кількість слів-паразитів, кількість сталих оборотів використанних у тексті.
   1.4. Частота вживання слів.
   1.5. Сенс написанного (техніки NLP, машинне навчання, покроковий семантичний розбір для пошуку лінгвістичних конституент тексту).
2. Аналіз метаінформації тексту, тобто інформації що описує походження, призначення та тематику тексту, включаючи такі методи, але не обмежуючись ними:
   2.1. Хронологічний частотний аналіз наукових праць в певній області. Визначення періодів високої та низької активності відносно наукової теми чи напрямку.
   2.2. Аналіз активності відносно локації чи організації (визначення кількості робіт на певну тему в певній географічній зоні чи в певних дослідницьких центрах – університетах, інститутах, тощо)
   2.3. Аналіз цитат наукових робіт. Визначення зв’язків між роботами, пошук схожих дослідницьких матеріалів, визначення впливу окремо взятих публікацій на розвиток наукового напряму.

В цій роботі ми розглядаємо лише третю групу другої категорії, а саме “Аналіз цитат наукових робіт”.

2.2 Наявні засоби аналізу

На момент дослідження існує велика кількість різноманітних бібліографій та наукометричних баз із відкритим, або частково відкритим доступом до наукових публікацій. Розглянемо найпопулярніші:

Scopus – популярна бібліографія статей опублікованих у наукових журналах та програмах наукових конференцій. Є частиною програмного комплексу SciVerse компанії Elsveir. Scopus регулярно індексує наукові видання, матеріали конференцій і книги. Доступ до бази данних здійснюється через веб-браузер. Також в систему інтегровану гнучку систему пошуку наукових публікацій . Плюси: Велика кількість наукових публікацій, широкі можливості фільтрації результатів пошуку та наявність API для інтеграції із своїми застосунками. Серед мінусів: відсутність відкритого доступу до мета інформації, обмеження пошуку на безоплатній основі, відсутність можливості безлімітної інтеграції(на інтеграцію накладаються обжемення у вигляді кількості запитів в хвилину). Також відсутні будь які інструменти для аналізу графів цитованності. На рисунку 2.1 представлено користувацький інтерфейс сервісу Scopus.

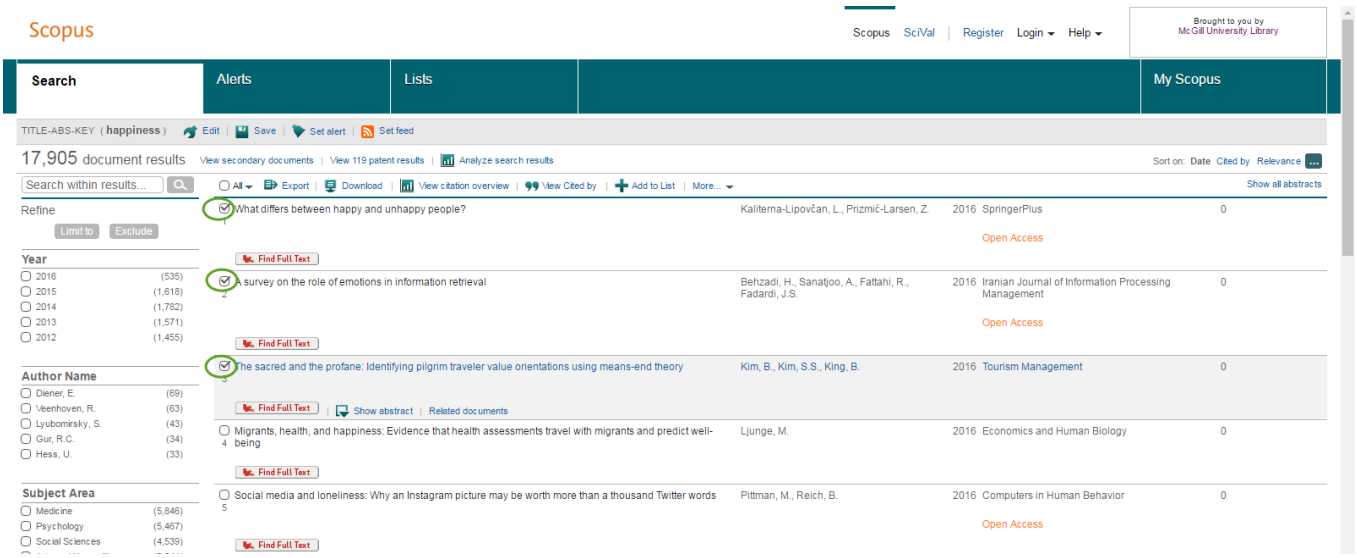

Рисунок 2.1 – користувацький інтерфейс системи Scopus

Google Scholar – бібліографія та частина пошукової системи, яка регулярно індексує наукові публікації широкого спектру дисциплін. Перша версія системи датується 2004 роком. Система здатная індексувати та збагачувати публікації метаінформацією стосовно походження матеріалу. Система є аналогом та конкурентом системи Scopus. Плюси: можливість пошуку найновіших наукових публікацій, широкі можливості по ранжуванню результатів пошуку, висока доступність сервісу. До мінусів можна віднести відсутність відкритого API для взаємодії із сервісом, та відсутність будь яких інструментів аналізу тексту. На рисунку 2.2 зображено користувацький інтерфейс системи Google Scholar.

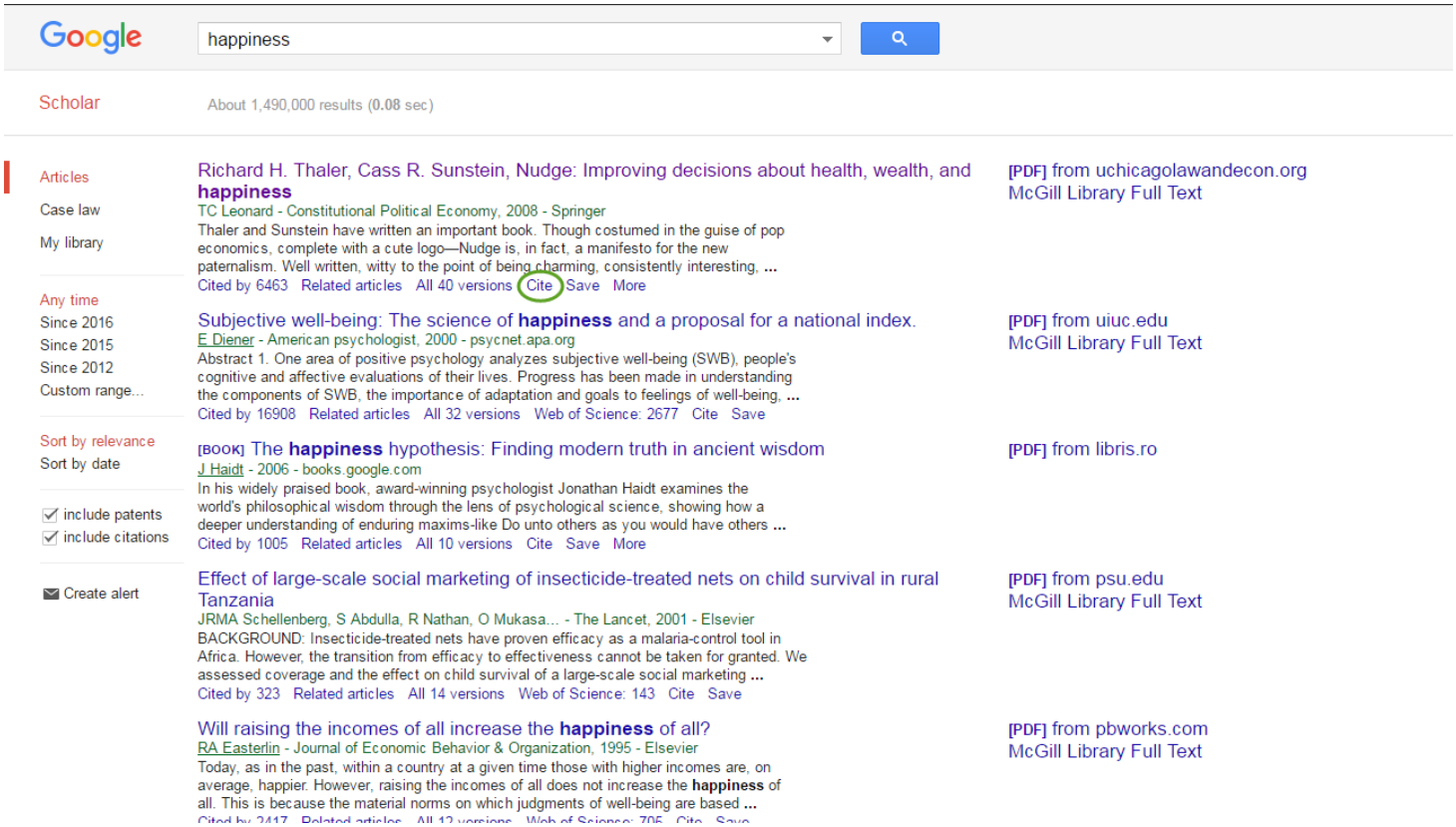

Рисунок 2.2 – користувацький інтерфейс системи Google Scholar

ArXiv - архів публікацій наукових статей який працює на безоплатній основі. Система створена і підтримується Корнелльським університетом а також широкою спільнотою редакторів. Архів не має обмежень відносно дисциплін, проте здебільшого там знаходяться наукові праці на тему астрономії, математики, фізики, статистики, комп'ютерних наук, біології та хімії. Публікації доступні у форматах

TeX та PDF. Користувачі мають змогу отримувати публікації як з веб-сайту так і через REST API що надається архівом. Автори мають право вільно викладати свої дослідження та редагувати їх після публікації, але попередні версії залишаються в системі і доступні для перегляду.

Архів заборонено використовувати для розповсюдження не технічної інформації, а також для подання лише анотації без самого тексту статті. Серед плюсів – можливість пошуку та безкоштовного перегляду всіх матеріалів що розміщено на хостингу. Архів зарекомендував себе як найпопулярніший сервіс для розміщення публікацій в області математики, фізики та комп'ютерних наук, тому пошук літератури на ці теми є найбільш перспективним саме на цьому ресурсі. До мінусів можна віднести відсутність будь яких інструментів аналізу публікацій та невисоку гнучкість та можливості API. На рисунку 2.3 зображено користувацький інтерфейс системи ArXiV.

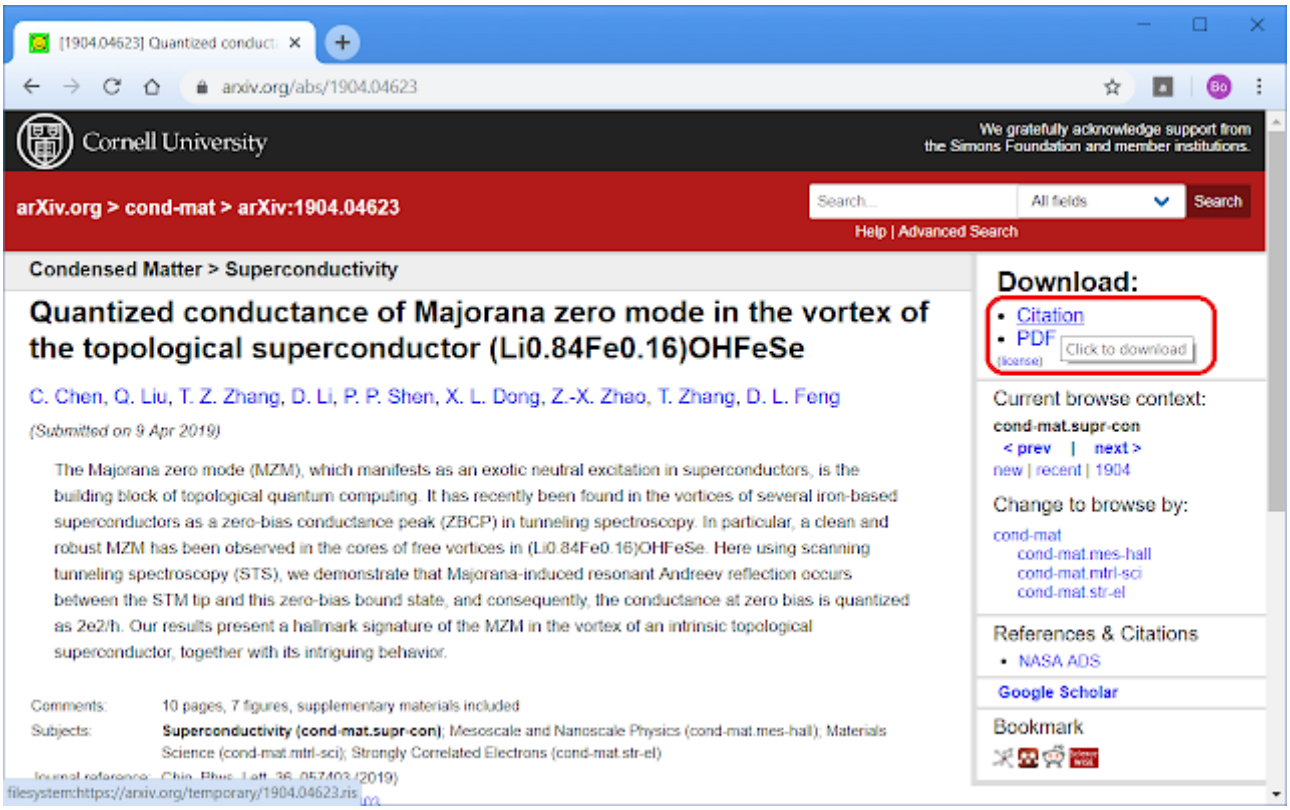

Рисунок 2.3 – користувацький інтерфейс системи ArXiV

2.3 Висновки до розділу

Із аналізу предметної області можна зробити висновок про різноманітність напрямів наукових досліджень в області аналізу науково-технічної літератури, та невелику кількість практичних застосунків для аналітики. Це може бути обумовлено складністю технічної реалізації.

Також, можливо, що рішення цих задач існують у вигляді програмного забезпечення, що є закритим для широкої публіки і розроблені всередині великих корпорацій для власного використання.

Розробка відкритої аналітичної платформи для агрегування аналітичних засобів НТЛ може мати велику цінність для бібліографії та науки про дані у розрізі наукових публікацій.

3 ЗАСОБИ РОЗРОБКИ

3.1 Опис технологій

Java – мова програмування орієнтована на розробку в межах ООП парадигми. Розроблювалась компанією “Sun Microsystems”, з 2009 року підтримується та розвивається за підтримки і кураторством компанії “Oracle”. Напряму розробкою займається компанія Red Hat. Програмний код компілюється в проміжний машинно-незалежний байт-код для віртуальної машини Java – JVM. Під час запуску програми запускається віртуальна машина та код інтерпретується за допомогою JIT компілятора в машинний код, готовий до виконання. Під час роботи JIT компілятору код оптимізується шляхом аналізу використання того чи іншого функціоналу. Рекомендовано використовувати Oracle JRE для запуску цієї программи, оскільки в цьому середовищі є гарантована підтримка апаратного відео-прискорення яке значно пришвидшує аналіз, та дозволяє використовувати меншу кількість процесорного часу.

Для керуваня залежностями використовується технологія Gradle. Gradle – система автоматичного компонування програмного забезпечення, що побудована на досвіді аналогічних система Apache Ant та Apache Maven, але замість громіздких XML конфігурацій, користувач описує потік обробки за домогою DSL на мовах Kotlin чи Groovy. На відміну від своїх попердників, використовує орієнтований ациклічний граф для представлення порядку виконная задач збірки продукту. Gradle може використовуватись для збору будь якої мови, що працює на базі JVM, проте планується підтримка й інших мов програмування.

Для розробки графічного інтерфейсу в якості базової графічної платформи використовується бібліотека JavaFX яка постачалась разом із JRE компанії Oracle з 7 по 10 версії, починаючи з 11 версії необхідно встановлювати як окреме розширення. JavaFX – платформа та набір інструментів для створення насичених інтернет-застосунків з можливістю підвантаження медіа додатків, а також для розробки насиченних застосунків для ПК. JavaFX включає в себе набір базових классів із простими елементами керування програмою, допоміжні класи для роботи із

мережею та файловою системою та різноманітні інтеграції із робочим столом користувача(незалежно від операційної системи на якій працює користувач). Також в роботі використано розширення для JavaFX, а саме FXML – мова розмітки користувацького інтерфейсу на базі XML, створена компанією Oracle для опису окремих елементів користувацького інтерфейсу для JavaFX програм. Приклад користувацького інтерфейсу створеного за допомогою JavaFX показано на рисунку 3.1.

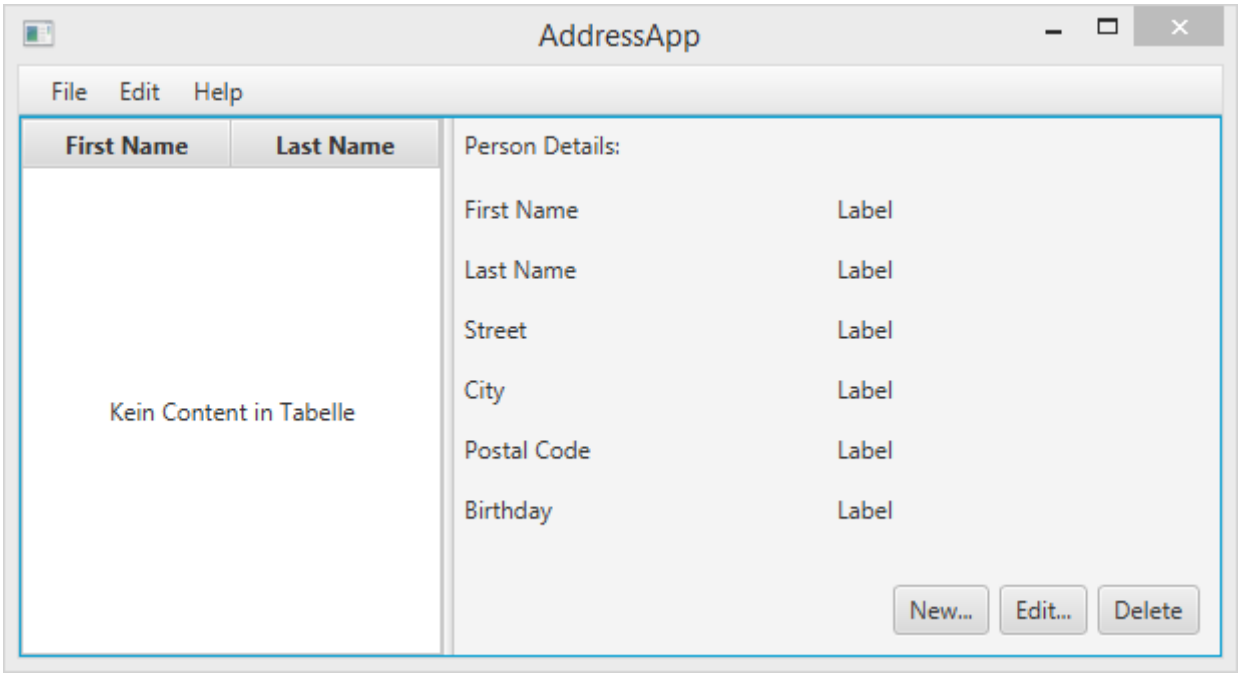

Рисунок 3.1 – приклад інтерфейсу програми створеної з JavaFX

JavaFX – це потужний інструмент для створення найбільш складних користувацьких інтерфейсів, проте вона не відповідає сучасним вимогам до дизайну застосунків. Оскільки метою розробки є програма виконана в стилі Material компанії Google було використано бібліотеку JFoenix. Jfoenix – це бібліотека із відкритим програмним кодом для JavaFX програм, яка містить в собі класси та окремі елементи інтерфейсу які відповідають Material Design принципам. Окремі елементи бібліотеки можна використувати у FXML конфігураціях.

Після визначення базових класів та елементів інфраструктури для побудови графічного інтерфейсу знадобився інструмент, який дозволяє здійснювати переходи між завершеними сутностями графічного інтерфейсу, а саме – вікнами. Це дозволило би масштабувати систему та легко її розширювати, описавши загальні принципи переходів між вікнами один раз, та зосередившись на наповненні цих самих вікон. В якості такої бібліотеки було обрано DataFX. Цей проект із відкритим кодом має намір спростити отримання, обробку, перегляд та редагування даних JavaFX застосунках. Це досягається шляхом надання різних адаптерів джерел даних для забезпечення зручної передачі інформації між різними незалежними елементами користувацького інтерфейсу. DataFX також надає зручні інтерфейси для роботи із данними такими як сортування, фільтрування, підтримка завантаження на вимогу та повернення даних, які змінилися локально, та їх миттєве відображення на усіх пов'язаних елементах користувацького інтерфейсу. Також DataFX надає інструментарій для описання взаємодії користувача із програмним комплексом у вигляді потоку, дозволяючи швидко та якісно розробляти застосунки які легко змінюються та розширюються, адже всі переваги розробки з допомогою декларативного опису елементів користувацького інтерфейсу за допомогою FXML залишаються в силі. Приклад графічного інтерфейсу розробленого за допомогою бібліотек Jfoenix та DataFX показано на рисунку 3.2.

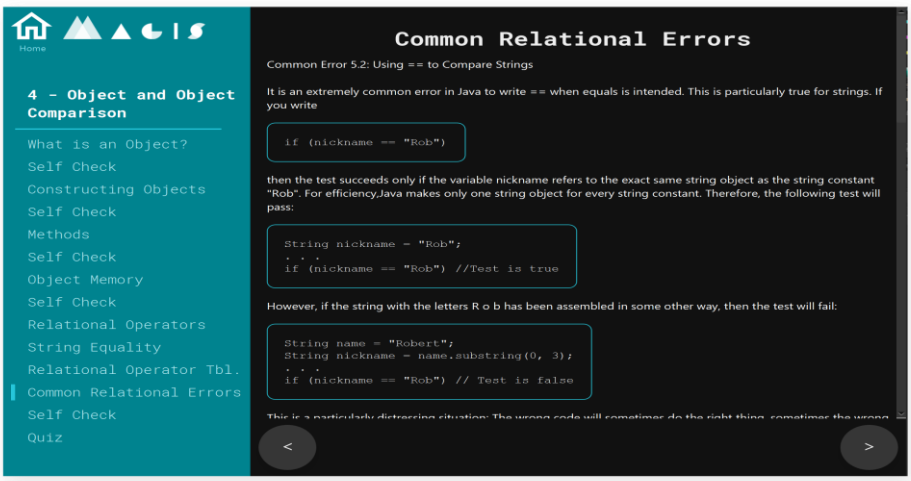

Рисунок 3.2 – інтерфейс програми Magis створений на базі Jfoenix і DataFX

Оскільки мережа публікацій може бути вкрай великою необхідно використувати швидкий та компактний інструментарій для збереження та читання таких данних. Для вирішення цієї задачі було обрано бібліотеку Kryo. Kryo - це швидка та ефективна бібліотека серіалізації об'єктів, що можуть бути представленні у виглядів графу для мови програмування Java. Цілі проекту - це висока швидкість, низький розмір та простий у користуванні API. Проект корисний у будь-який час, коли потрібно зберігати об'єкти, будь-які файли великої розмірності. Kryo також може виконувати автоматичне глибоке та неглибоке копіювання / клонування. Є варіанти клонування об'єктів один в одного в оперативній пам'яті, так і збереження на енерго-незалежні диски. Ця бібліотека є найшвидшою серед бібліотек серіалізації та десеріалізації для JVM, і ця теза підтверджується великою кількістю незалежних тестів. Також Kryo дозволяє зберігати дані займаючи вкрай невеликий простір, як показано на порівняльному графіку бібліотек серіалізації Java, що на рисунку 3.3

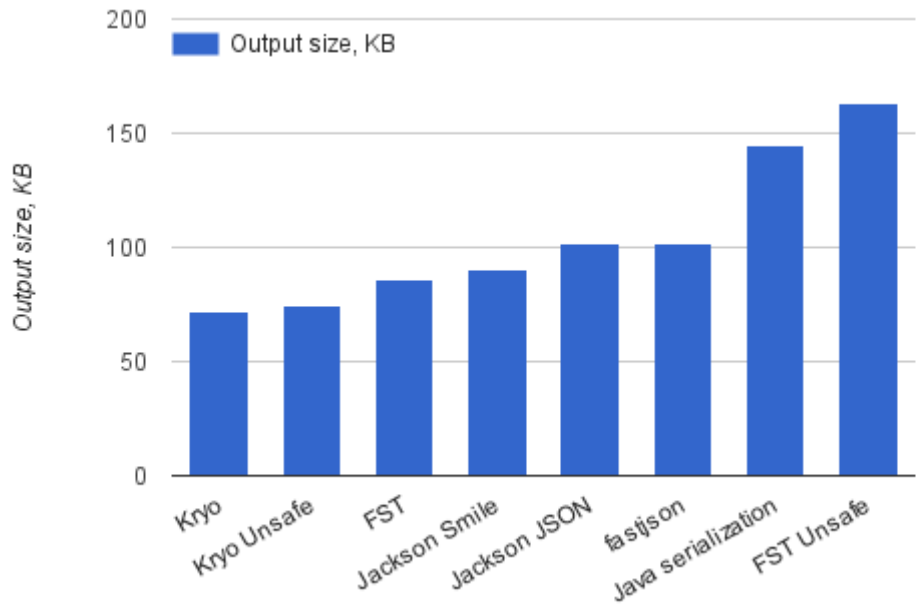

Рисунок 3.3 – графік порівняння розміру збережених даних бібліотек JVM

Як добре видно на рисунку 3.3 бібліотека Kryo дозволяє зберігати дані в найбільш компактному форматі серед можливих, що і стало вирішальним фактором

при виборі цієї бібліотеки як основної для вирішення задачі збереження проміжних та кінцевих результатів аналізу.

Для забезпечення комунікації між застосунком та зовнішнім сховищем публікацій використовується бібліотека Retrofit. Ця бібліотека дозволяє швидко і безпечно з точки зору розробки на програмних типах розробляти HTTP клієнтів для обміну інформацією з сервісом. Комунікація відбувається за допомогою протоколу HTTP. Усі запити та відповіді формуються до JSON об'єктів. Для десеріалізації цих об'єктів використовується бібліотека Jackson, яка дозволяє швидко та ефективно транслювати об'єкти описані мовою Java до JSON чи XML об'єктів.

Для логування кожного з етапів роботи використовується бібліотека Logback. Logback дозволяє визначати декілька рівнів повідомлення:

1. INFO – інформаційні повідомлення
2. WARN – повідомлення про неочікуванну поведінку
3. ERROR – повідомлення про критичну для роботи программи поведінку, зазвичай супроводжується виходом із програми
4. DEBUG –розширена інформація про роботу програми, використовується при відладці програмного забезпечення.
5. TRACE – спеціальний тип повідомлень для відслідковування запитів та відповідей на коммунікацію між сервісами по мережі.

Logback дозволяє конфігурувати розмір лог-файлів, стратегію їх іменування, автоматичного видалення, переміщення та форматування повідомлень. Бібліотека також може бути розширена для більш детального керування повідомленнями – наприклад, для відправки в агрегаційні сервіси із логами, такі як Logstash, для подальшої передачі до сервісів візуалізації лог-повідомлень, такі як Kibana.

3.2 Опис середовища розробки

При розробці використувувалось інтегроване середовище розробки Intellij IDEA від компанії JetBrains. Ця IDE містить в собі велику кількість інтеграцій із веб-бібліотеками, графічними фреймворками, базами даних і тд.

На рисунку 3.4 показано інтерфейс програми Intellij IDEA.

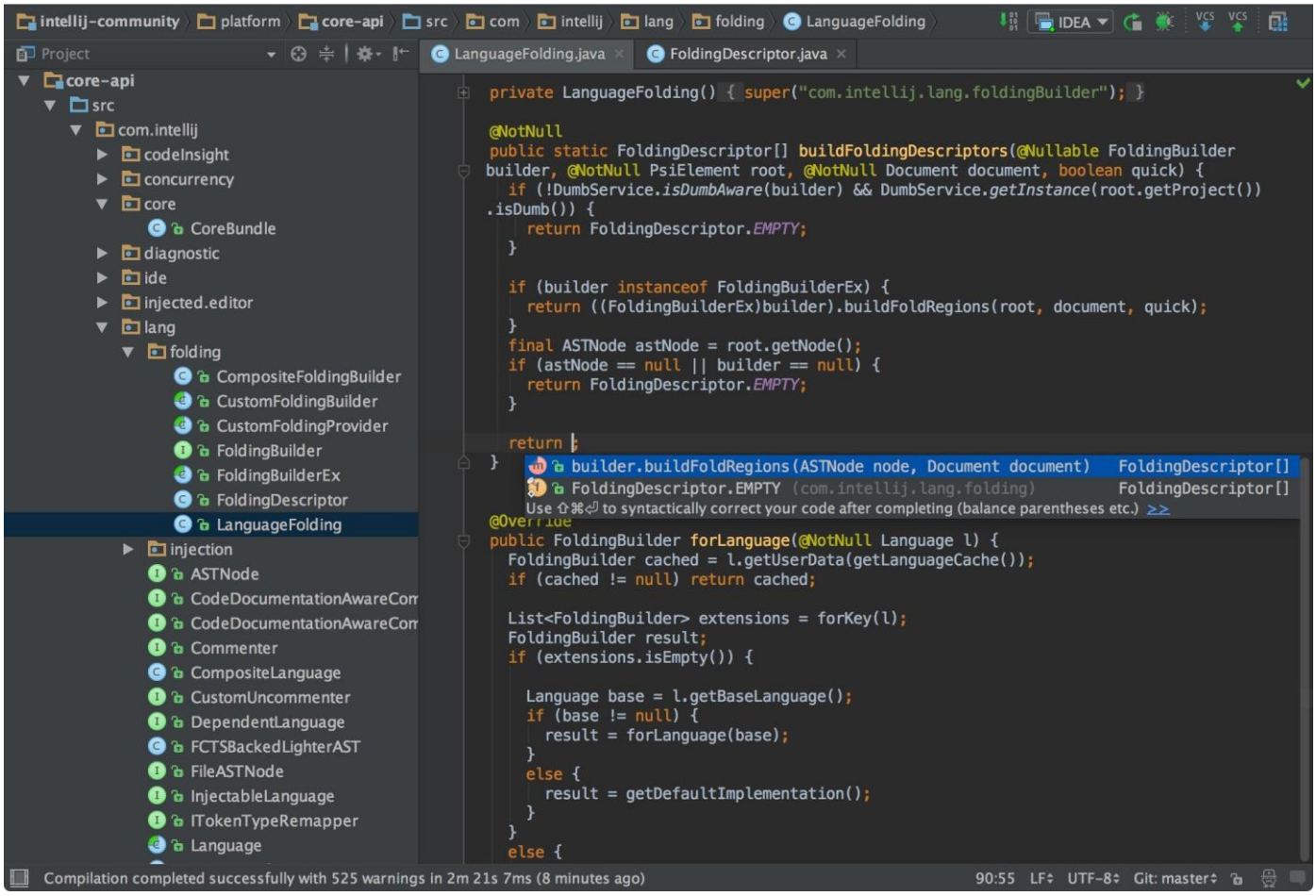

Рисунок 3.4 – інтерфейс програми Intellij IDEA

Intellij IDEA було обрано за основний інструмент розробки через наявність інтеграцій із JavaFX у вигляді вбудованного до середовища розробки редактора сцен JavaFX – Scene Builder.

3.3 Висновки до розділу

У цьому розділі було розглянуто засоби для виконання задачі такі як Java, Kryo, JafaFX.

Технології підбирались за критеріями:

1. Швидкість розробки
2. Кросплатформенність

3. Оптимальність та повнота інструментарію

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Алгоритм розробки

При розробці програмного забезпечення за стратегію розробки було обрано наступну послідовність дій:

1. Аналіз функціональних та нефункціональних вимог
2. Розробка технічого завдання на базі вимог
3. Прототипування та моделювання розробки, що включає в себе вибір технологій розробки, стратегію розгортання застосунку, деталізація вимог до апаратного та програмного забезпечення ЕВМ на яких передбачається експлуатація програмного комплексу, опис варіантів використання програмного забезпечення(та візуалізація за допомогою Use-Case діаграм), опис компонентної моделі.
4. Безпосередня реалізація на базі програмної моделі
5. Тестування застосунку

Необхідно визначити основні задачі які повинна вирішувати інформаційна система:

1. Визначення індексу цитованності
2. Побудова графу цитувань

4.2 Визначення індексу цитованності

Ми можемо визначити ряд корисних метрик для об'єктивної оцінки впливу та суб'єктивної оцінки якості дослідження описаного в конкретно взятій публікації. Використовувати велику кількість показників при порівнянні статей не зручно, тому ми введемо та спробуємо формалізувати поняття індекс цитованості, та будем його використовувати для визначення впливу публікації.

Зазначимо, що детермінувати об'єктивний індекс цитованості ми не в змозі внаслідок великої кількості суб'єктивних факторів. Наукові журнали відрізняються в популярності, і як наслідок праці, які публікуються в більш популярних журналах однозначно отримують більше уваги від науковців, що приводить до більшої кількості цитат цих статей в їх роботах. Також кількість цитувань залежить від

суб'єктивних факторів таких як – стиль письма автора, його репутація в науковому світі, тощо. [1]

В цій роботі, припустимо що вплив наукової праці залежить лише від кількості посилань на цю роботу.

Очевидно, що навіть при вищевказанному припущенні можна ввести нескінченну кількість альтернативних визначень терміну індекс цитованості, тому для простоти постулюємо наступне:

Індекс цитованості – міра впливу наукової праці, що чисельно дорівнює кількістю цитувань роботи.

4.3 Побудова графу цитувань

Окрім визначення міри впливу наукової праці, ми можемо визначити авторів, що часто цитують один одного і сегментувати науковців та їх публікації до певних "кластерів" тобто до угрупувань, які поєднані спільною темою або науковим напрямком, як зображено на рисунку 4.1. Наприклад, науковці, що працюють в області біоінформатики будуть значно частіше цитувати праці своїх колег по науковому напрямку ніж публікації з області математики.

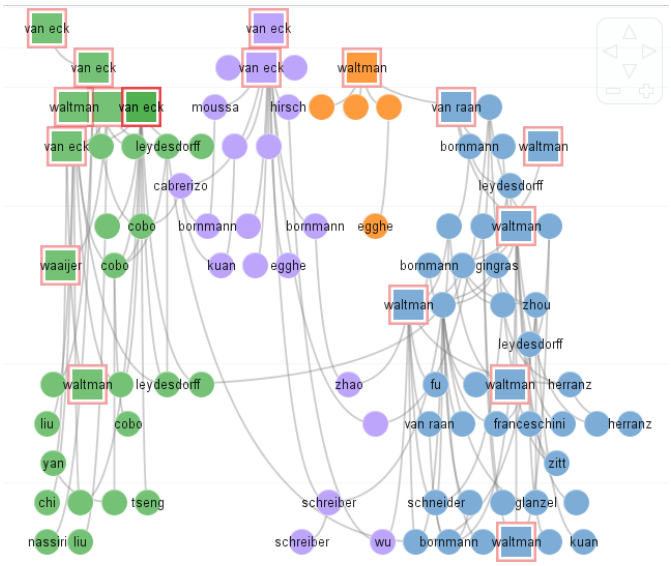

Рисунок 4.1 – сегментація вчених за кількістю перехресних цитувань

Для того, аби згрупувати публікації, та інформативно їх відобразити перейдемо до математичної абстракції, а саме до теорії графів. [2]

Граф – сукупність об'єктів із зв'язками між ними. Найчастіше, графічно зображується як набір точок на площині, які можуть бути пов'язані відрізками, як показано на рисунку 4.2.

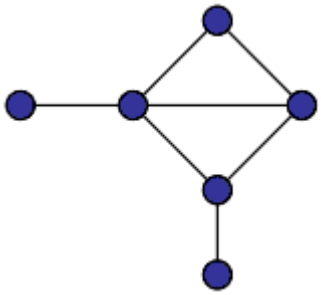

Рисунок 4.2 – графічна інтерпретація графу

Об'єкти називаються вершинами, а зв'язки між ними ребрами, тому формальне визначення графу набуває форми:

Граф G – це впорядкована пара $G := (V, E)$, для якої виконуються наступні умови:

- V – множина вершин,
- E – множина пар вершин з V, що називаються *ребрами.*

Нам підходить незважений та орієнтований граф, тобто граф вага зв'язків якого чисельно дорівнює 1, та в разі наявності між будь якими двома його вершинами існує зв'язок, то існує "перехід" із першої вершини в другу, але не навпаки.

Введемо також додаткові поняття відносно якісної характеристики зв'язків між вершинами графу, а саме "компонента зв'язності графа" та "компонента сильної зв'язності графа".

Компонента зв'язності графа – це підграф(ненульова підмножина вершин та ребер початкового графу, що їх пов'язують та утворють граф), в якому будь-які дві вершини пов'язані одна з одною шляхами(тобто послідовністю ребер), та не зв'язані з ніякими додатковими вершинами. На рисунку 4.3 зображено граф із трьома компонентами зв'язності

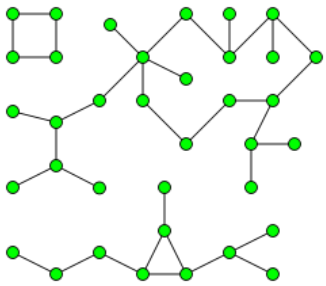

Рисунок 4.3 – граф з трьома компонентами зв'язності

Компонента сильної зв'язності орієнтованого графу G — це найбільший сильно зв'язаний підграф. Якщо кожну компонента сильної зв'язності стягнути до однієї вершини, отримаємо орієнтований ациклічний граф, ущільнення (англ. condensation) G. Орієнтований граф є ациклічним тоді і лише тоді, коли він не має компонент сильної зв'язності з більш як однією вершиною, бо орієнтований цикл є сильно зв'язним і кожна нетривіальна компонента сильної зв'язності містить щонайменше один орієнтований цикл. На рисунку 4.4 зображено приклад такого графу.

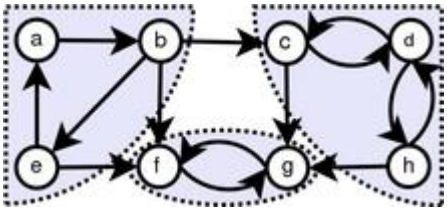

Рисунок 4.4 – граф з позначеними трьома компонентами сильної зв'язності

Ми можемо орієнтувати граф G, керуючись наступними міркуваннями. Нехай маємо 2 публікації – A та B. Якщо публікація A посилається на публікацію B, то можемо описати деякий двовершинний граф з вершинами P та Q, що відповідають публікаціям A та B, та орієнтованим незваженим ребром з A в B. Узагальнивши цей підхід ми можемо побудувати орієнтований граф, що складається з будь-якої кількості публікацій, оскільки очевидно, що в нас завжди буде інформація про напрямок цитування. Такая стратегія побудови графу не єдина, але вона підходить нам більш за все, оскільки дозволяє зосередитись лише на характеристики цитованості. Якби нам було необхідно враховувати інші характеристики(такі як автора, рік публікації, видавництво, країна видання та інше), то нам довелось додавати би до кожної вершини метаінформацію, що відповідала би цим характеристикам, і відповідно орієнтувати наш граф.

Тепер, коли ми визначили стратегію побудови графів, що ілюструють зв'язки між публікаціями, необхідно пояснити яким чином ми можемо знайти компоненти зв'язності та компоненти сильної зв'язності. Для пошуку компонентів зв'язності ми будемо використовувати алгоритм пошуку в глибину, до тих пір поки не пройдемо усі вершини. Для пошуку компонентів сильної зв'язності ми будемо використовувати алгоритм Тарджана[3], базований на пошуку в глибину.

4.4 Візуалізація програмної моделі

У користувача програмного продукту є декілька різних варіантів взаємодії із інтерфейсом.

На рисунку 4.5 показано приклади роботи користувача із системою у вигляді use-case діаграми

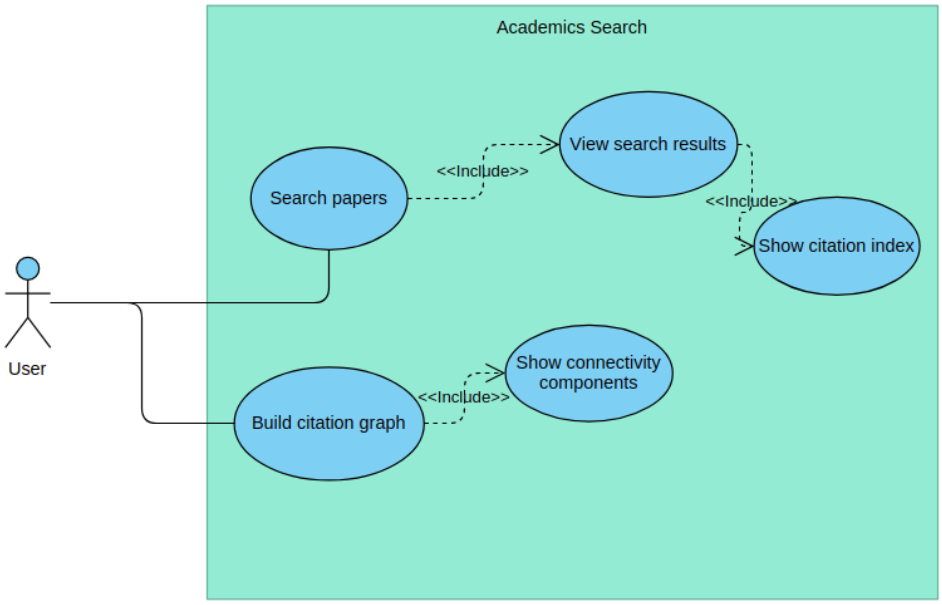

Рисунок 4.5 – use-case діаграмма

Користувач може використовувати інтерфейс програми для того аби шукати наукові публікації. В результаті користувач отримає декілька статей(або жодної), та зможе детально переглянути інформацію відносно наукової статі, наприклад автора, рік публікації, видавництво, розмір публікації, її тематику, призначення, а найголовніше обрахований індекс цитованості. Результати ніяким чином не відсортовані, якщо користувач не визначив інакше.

Система умовно поділяється на 2 компоненти – клієнтську частину, та зовнішнє сховище данних.

На рисунку 4.6 показано компонентну модель застосунку із двома компонентами – графічним інтерфейсом, та системою агрегатором публікацій

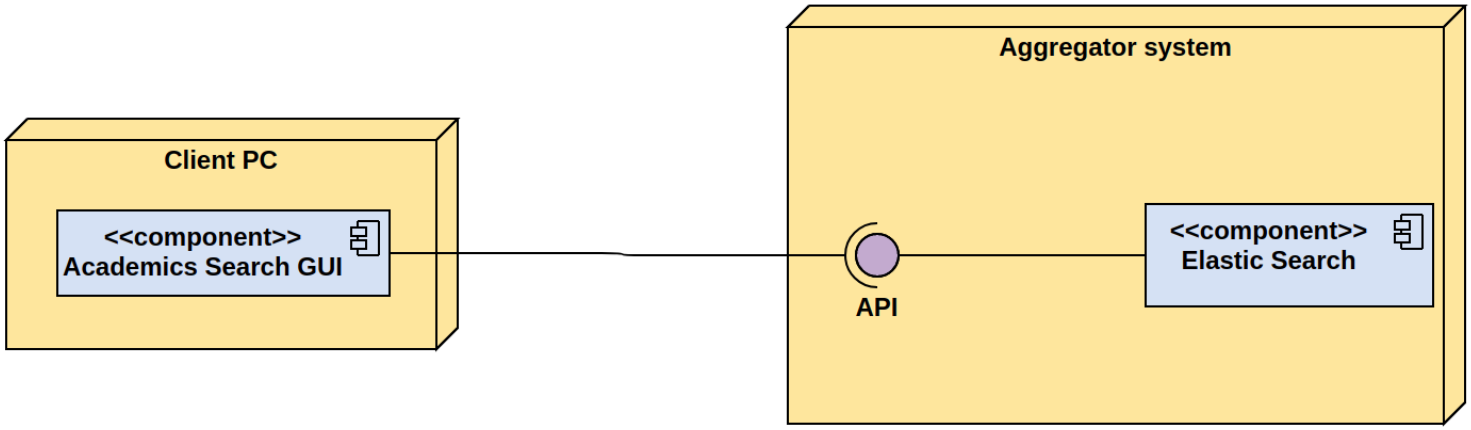

Рисунок 4.6 - компонента модель застосунку

Клієнтський застосунок умовно поділяється на 3 модулі:

1. Графічний застосунок, який відображає результати пошуку, елементи керування програмним забезпеченням, перехоплює користувацький ввід та супроводжує користувача підказками стосовно функціоналу програми
2. Набір бібліотек для взаємодії із зовнішніми системами зберігання публікацій. До стандартного набору входить пакет взаємодії із Web Of Science(у вигляді імпорту файлів) та веб-інтеграція із системою Core
3. Аналітичний модуль, який будує, фарбує та аналізує графи залежностей між публікаціями.

Всі модулі незалежні один від одного та легко заміняються. Це необхідно для гнучкої розробки, ми можемо швидко замінити одну реалізацію на іншу, розширити існуючий функціонал одного з модулів не завдаючи шкоди іншим модулям до тих пір допоки наші розробки не порушують контрактів взаємодії.

Розглянемо діаграму послідовності(sequence diagrams) для імпортування публікацій із системи Core на рисунку 4.7:

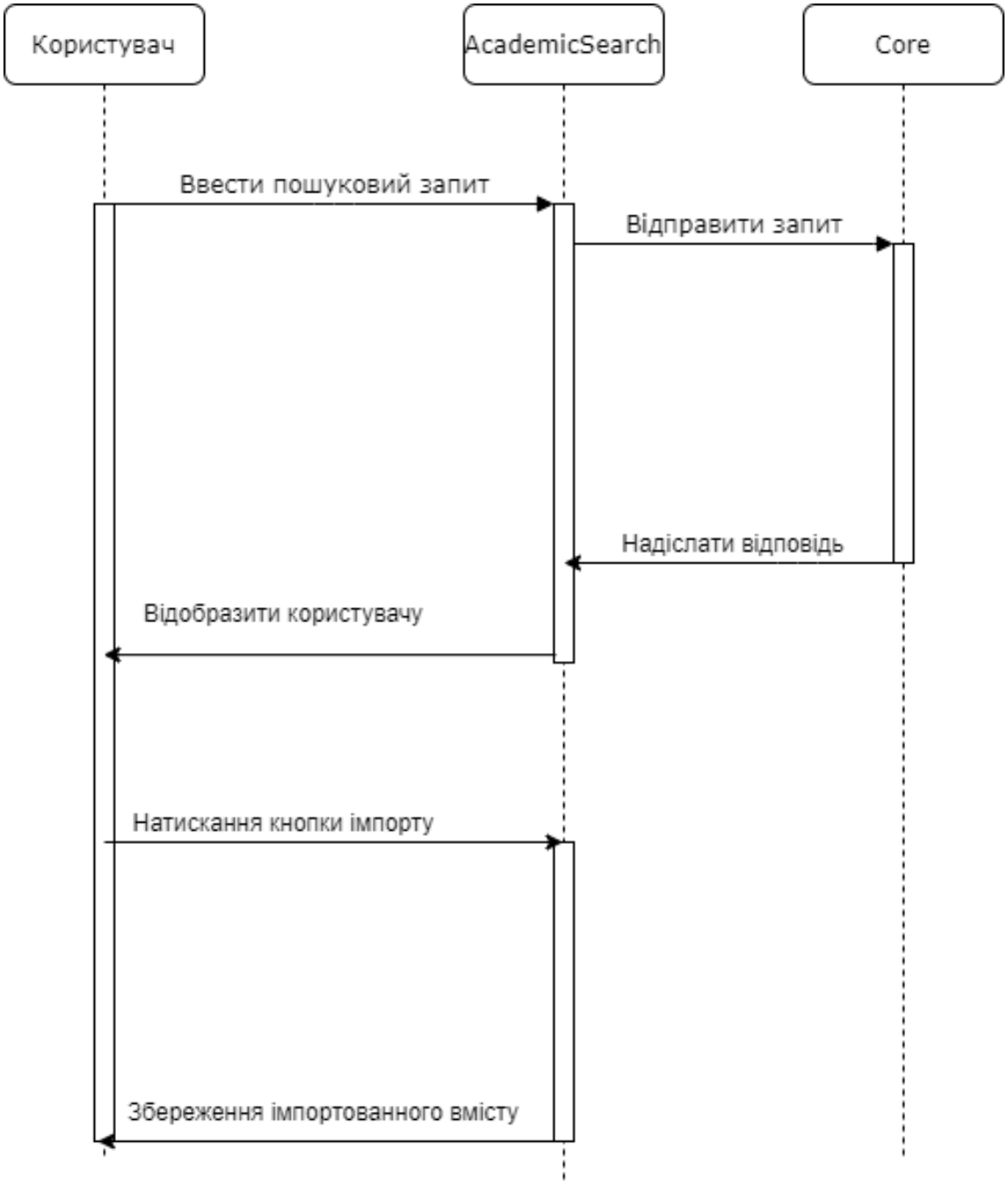

Рисунок 4.7 – послідовна діаграма імпорту публікацій

1. Користувач вводить пошуковий запит із автором, роком випуску публікації, назвою праці, її DOI, OAI чи будь яким іншим ідентифікатором зформувавши пошуковий запит у форматі, який сформульовано для ElasticSearch та натискає кнопку пошуку публікації та очікує відповіді програми

2. Застосунок формує пошуковий запит за правилами описаними в Core API, та очікує відповіді від Core
3. Core приймає запит від нашого застосунку, валідує наш пошуковий запит, та аутентифікує запит. У випадку успішної аутентифікації, здійснюється пошук в внутрішньому ElasticSearch системи Core.
4. Застосунок обробляє відповідь Core, та формує представлення результату користувачеві. Користувачу відображається результат пошуку та кнопка, яка провокує збереження знайдених публікацій до файлу.
5. Користувач переглядяє результати пошуку, та у тому випадку, якщо знайдені публікації відповідають його вимогам натискає кнопку імпортування.
6. Застосунок трансформує публікації до Kryo сумісного бінарного формату, та пропонує користувачу обрати місце для збереження публікацій. Після вибору папки для збереження та імені файлу користувач отримує повідомлення про успішний імпорт публікацій із цитуванням з системи Core.

Розглянемо послідовну діаграму взаємодії користувача із підсистемою аналізу графів, яку зображену на рисунку 4.8:

1. Користувач відкриває менеджер файлів своєї системи та обирає файл.
2. Якщо файл не може бути розпізнано як файл публікацій програмного комплексу користувачу відображається повідомлення про неможливість використання файлу, та користувач має змогу знову обрати файл. Якщо файл розпізнано як коректний файл із публікаціями користувача перенаправляють до наступного вікна.
3. Користувач отримує змогу переглянути список імпортований публікацій із короткою інформацією про публікації у вигляді таблиці, зміст якої не може бути модифіковано. В таблиці користувач може сортувати публікації за кожним полем окремо.

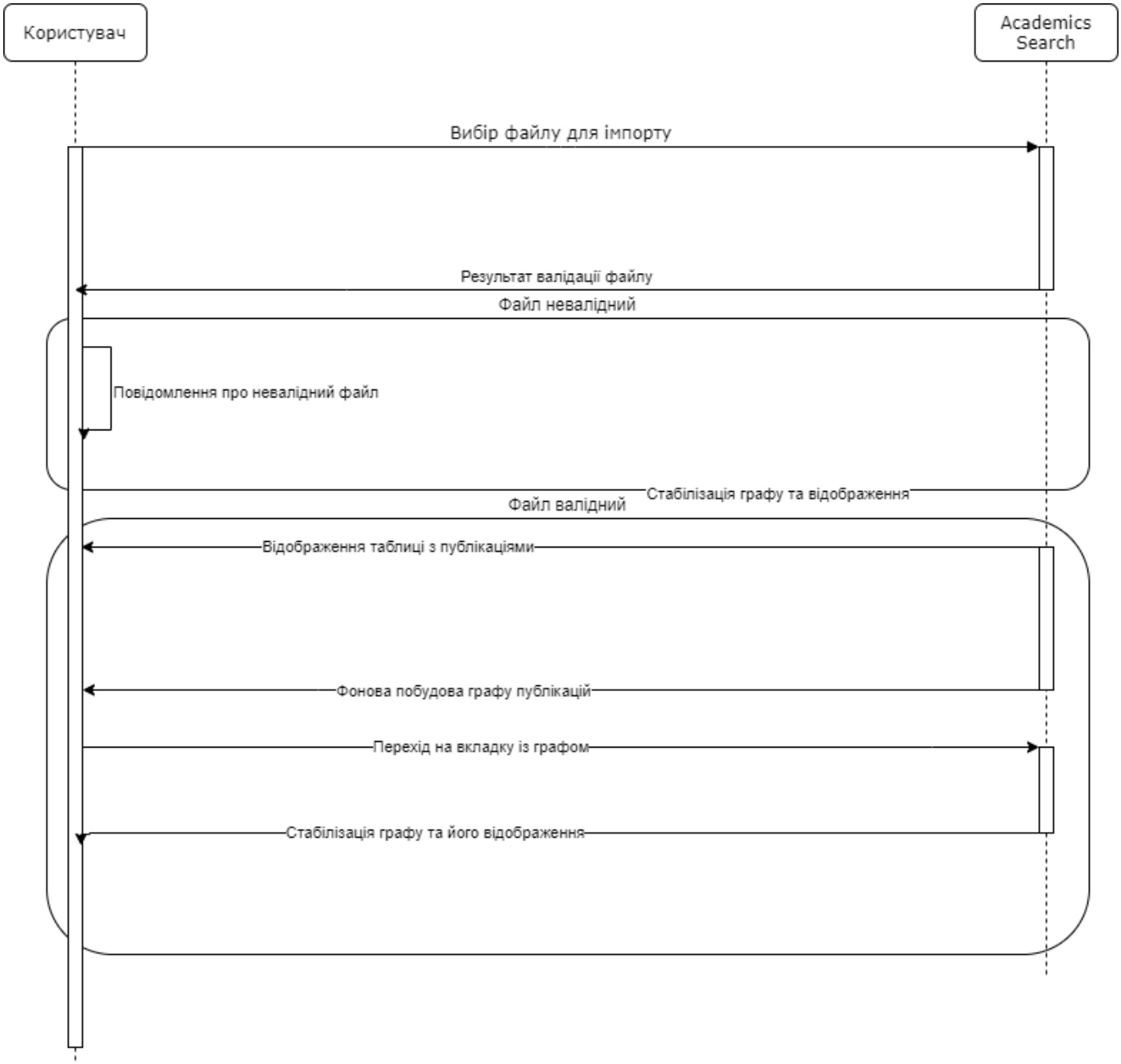

Рисунок 4.8 – послідовна діаграма модулю аналізу графів

4. Користувач відкриває вкладку Graph Analysis та може побачити побудований граф, який знаходиться в стані стабілізації, адже для кожного конкретного випадку час стабілізації графу невідомий і може продовжуватись тривалий час.

4.5 Структура програми

Розглянемо загальну структуру пакетів на рисунку 4.8:

1. Core – пакет інтеграцій із зовнішньою системою Core
2. GUI – пакет в якому знаходиться весь код пов’язаний із відображенням користувацького інтерфейсу
3. Model – пакет в якому знаходиться базові структури та класси із сутностями предметної області
4. Smartgraph – пакет в якому описані допоміжні структури для відображення та обрахування графів
5. Wos – пакет, в якому описані класи для імпортування Web Of Science документів до вигляду який може бути відображений системою

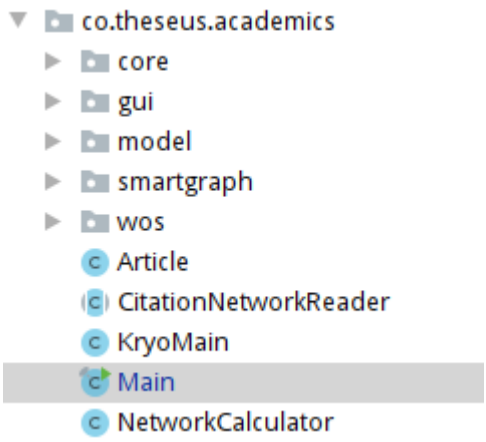

Рисунок 4.8 – загальна структура пакетів

У вигляді UML діаграми ми можемо пакети наступним чином на рисунку 4.9

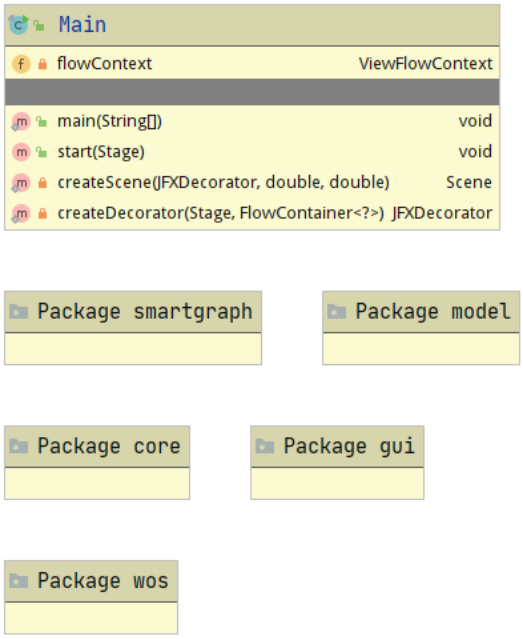

Рисунок 4.9 – загальна діаграма класів

4.5.1 Структура пакету інтеграції із Core

Детальніше розглянемо інтеграцією із зовнішім агрегатором науково-технічої літератури Core на рисунку 4.10

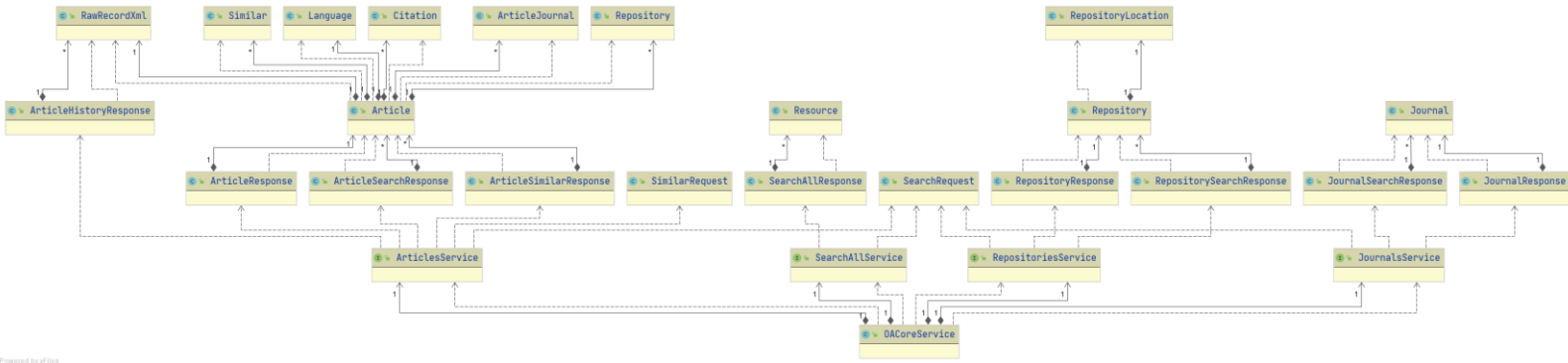

Рисунок 4.10 – UML діаграма класів Core

Уся взаємодія із цим відбувається за допомогою фасаду OACoreService. Він агрегує 4 доступних для користувача сервіси:

1. ArticlesService – пошуковий сервіс для статей
2. SearchAllService – загальний пошуковий сервіс

3. RepositoryService – пошуковий сервіс за агрегаторами(репозиторіями)
4. JournalService – пошуковий сервіс за науковими журналами

Розглянемо структуру OACoreService на рисунку 4.11

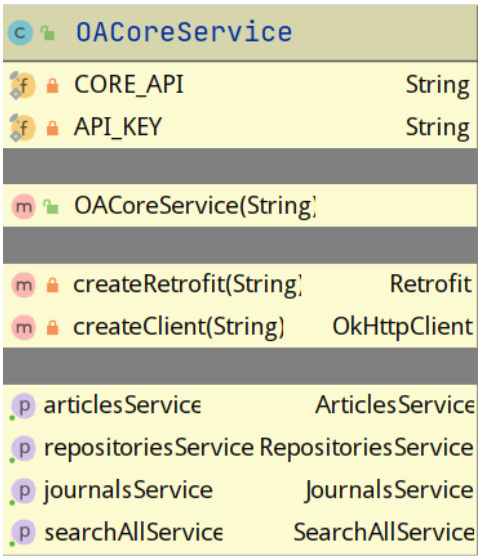

Рисунок 4.11 – структура OACoreService

Фасад складається із чотирьох вищевказаних сервісів, статичних конфігураціних параметрів для аутентифікації запитів із Core, та методи створення фасаду. Детальніше розглянемо кожен з внутрішніх сервісів.

ArticlesService спеціалізований сервіс пошуку за назвою публікацій, як показано на рисунку 4.12

Рисунок 4.12 – структура ArticleService

Він містить методи для пошуку публікацій за внутрішнім ідентифікатором, методи завантаження повного тексту публікації(за наявності), історію змін публікації в сервісі, методи пошуку подібних публікацій, це може бути використано в майбутньому для альтернативних методів аналізу, що не представлені у цій роботі.

JournalService – спеціалізований сервіс для пошуку наукових журналів, та вибірок з них, як показано на рисунку 4.13

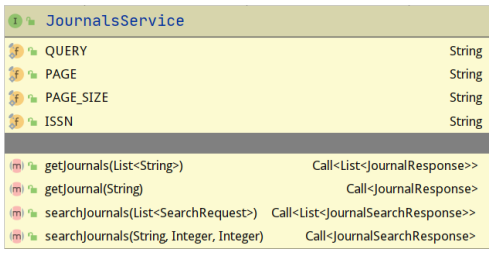

Рисунок 4.13 - структура JournalsService

JournalService містить методи для пошуку наукових видань за їх назвою. Передбачає можливість пошуку як конкретного журналу, так і списку, за допомогою спеціально визначених регулярних виразів.

RepositoriesService це сервіс, який дозволяє здійснювати пошук за різноманітними агрегаторами НТЛ, такими як Scopus, ArXiV та ін. Його методи ми можемо бачити на рисунку 4.14

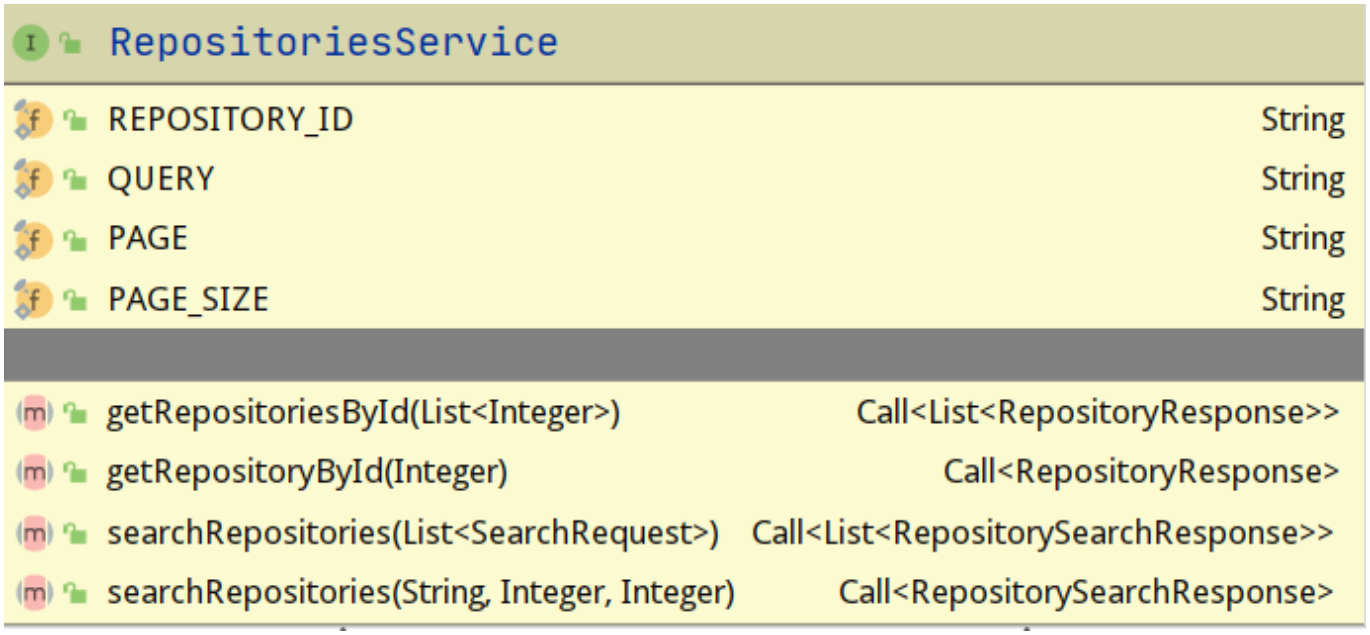

Рисунок 4.14 – структура RepositoriesService

RepositoriesService містить методи для пошуку, цілеспрямованої вибірки та статичні поля конфігурації пошуку.

Останній з вищевказанних – SearchAllService відповідає за загальні пошукові запити, які виходять за рамки стандартних, які описані в інших сервісах. Сервіс описано на рисунку 4.15

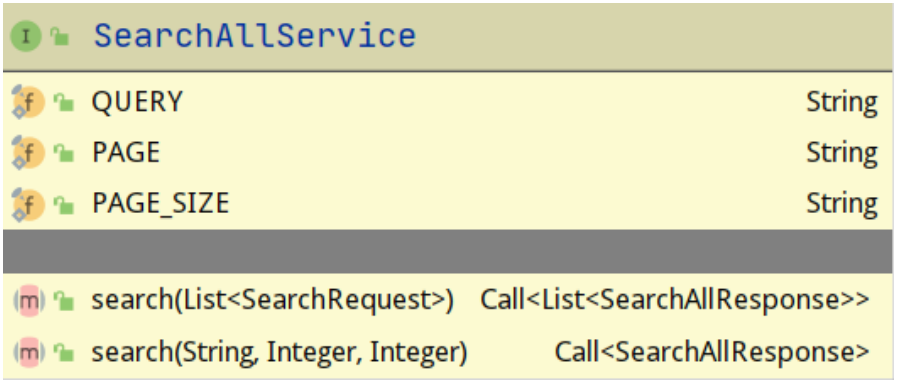

Рисунок 4.15 – структура SearchAllService

SearchAllService містить методи для загального пошуку інформації за допомогою регулярних виразів ElasticSearch.

4.5.2 Структура пакету графічного відображення

Детальніше розглянемо структурні елементи графічної підсистеми на рис. 4.16

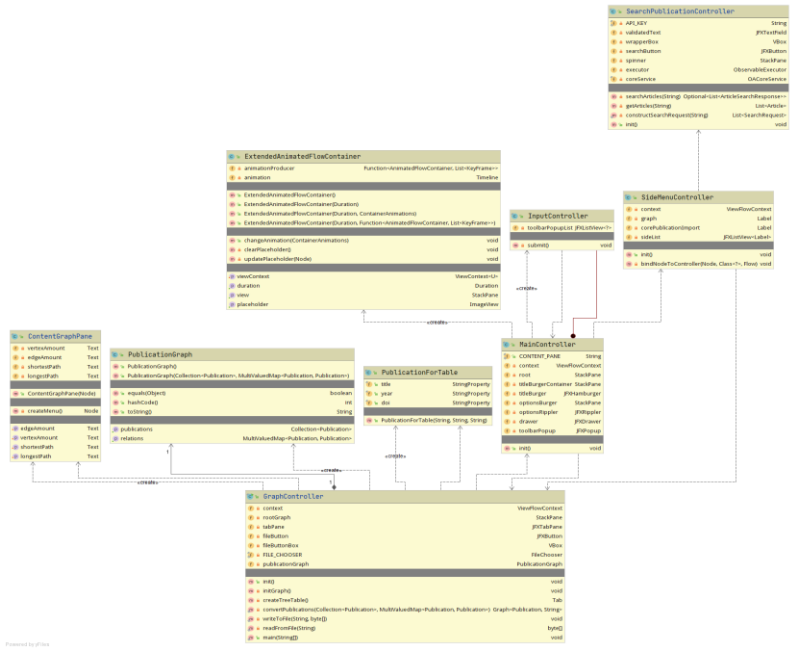

Рисунок 4.16 – UML діаграма класів GUI модуля

Взаємодія між різними вікнами та обмін даними реалізується за патерном MVC. В підсистемі графічного інтерфейсу визначено 4 контролера:

1. GraphController – контролер вікна, що відповідає за взаємодію із вікном побудови та аналізу графів
2. SearchPublicationController – контролер вікна, що відповідає за взаємодію із вікном пошуку та імпортування наукових публікацій у системі Core
3. SideMenuController – контролер вікна, що відповадє за навігацію між іншими вікнами застосунку. Містить в собі посилання на інші вікна
4. MainController – контролер, який бере на себе завдання первинної ініціалізації застосунку, завантаження усіх графічних ресурсів та декларативних описів структурних елементів користувацького інтерфейсу

4.5.3 Структура пакету із базовими класами

Детальніше розглянемо ієрархію базових алгоритмічних структур на рисунку 4.17

Цей пакет складається із великої кількості допоміжних структур для обчислюючого классу NetworkCalculator, який безпосередньо виконує усі обчислення індексу цитованності, силу взаємозв'язків між публікаціями, впливовість тієї чи іншої роботи за кількістю посилань на неї і тд.

Всі сутності цього пакету поділяються на 3 категорії:

1. Базові сутності – ті, що містять в собі інформацію необхідну для розрахунків
2. Уточнюючі сутності – ті, що необхідні для корегування, фільтрації або впливу на хід обчислень
3. Утримувачі результати – ті сутності, які зберігають в собі результат певних обчислень

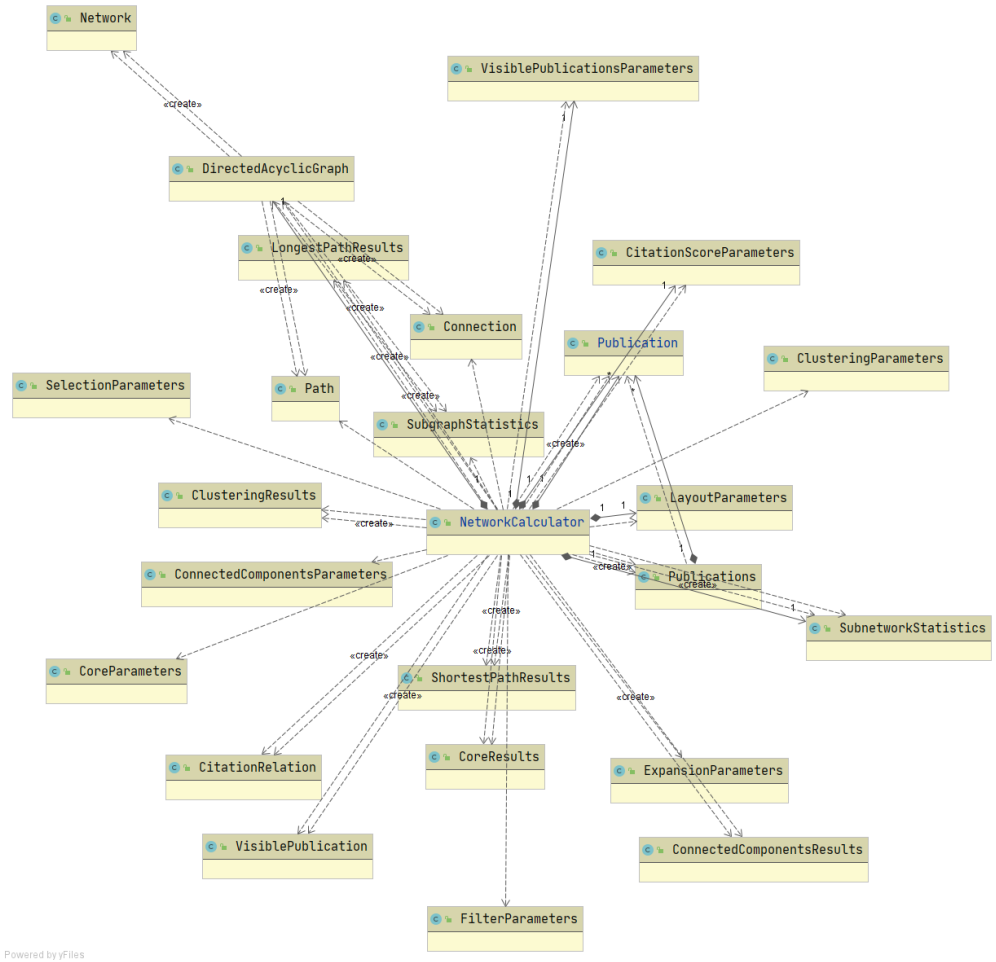

Рисунок 4.17 – ієрархія базових структур

Нетиповими є класи DirectedAcyclicGraph та Network Calculator. DirectedAcyclicGraph це сутність яка інкапсулює в собі всю інформацію стосовного графового представлення інформації, що подана на аналіз.

Розглянемо детальніше структуру DirectedAcyclicGraph на рисунку 4.19

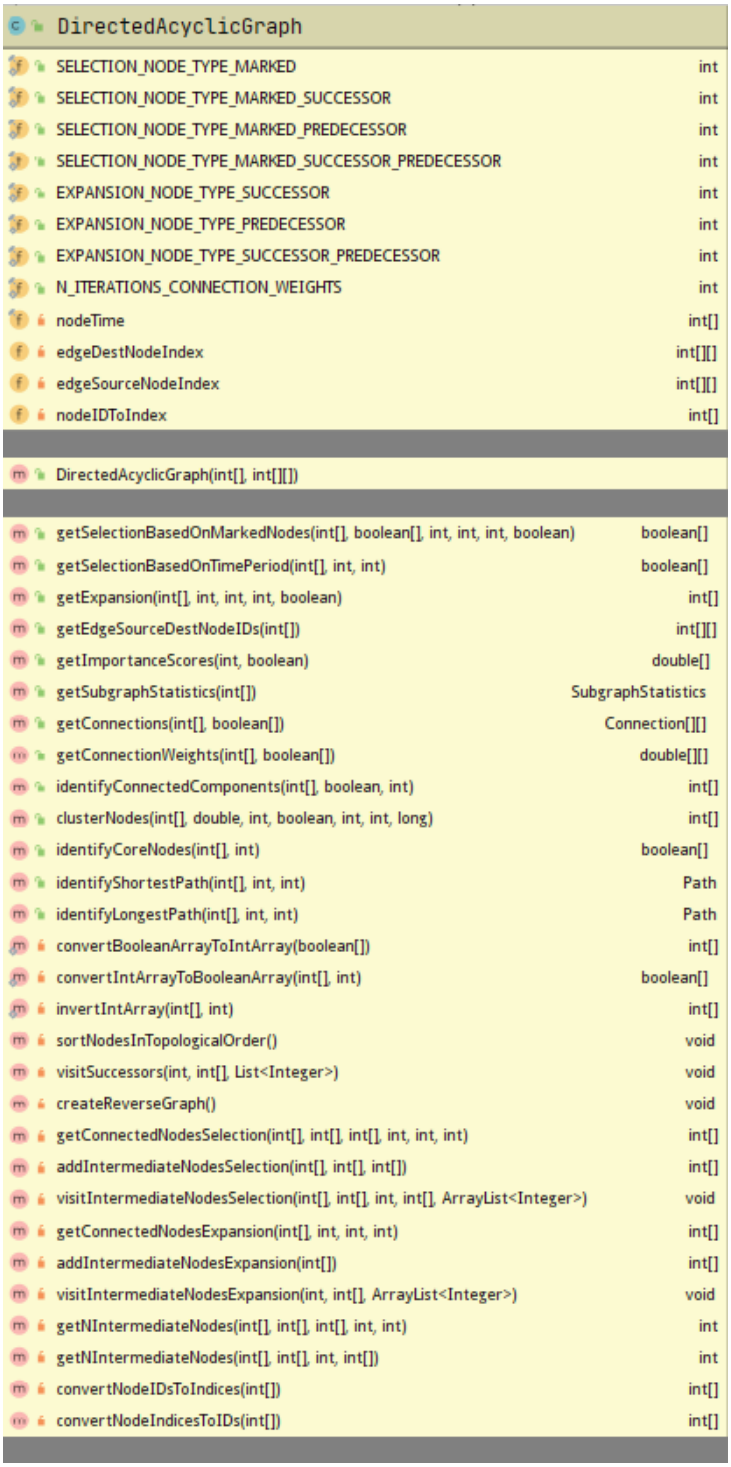

Рисунок 4.19 – внутрішня структура DirectedAcyclicGraph

Цей клас містить в собі методи які дозволяють:

1. Давати кількісну оцінку важливості публікації
2. Видавати статистику за підграфом, таку як кількість вершин та кількість зв’язків між ними
3. Розвертати граф, тобто розвертати усі напрямки ребер

4. Будувати матрицю інцидентності
5. Надавати інструментарій для розширення, звуження та редагування будь якої з вершин графу
6. Розширювати граф новими графами, тобто поєднувати декілька графів
7. Обчислювати кількість зв'язків між вершинами графа
8. Оцінювати загальну вагу зв'язків між вершинами графа, у випадку зваженності графа

Також цей клас інкапсулює структури, що описують стан графу. Цими структурами виступають матриці інцидентності

Опишемо головний клас модуля, який проводить усі необхідні розрахунки за допомогою вищевказаних допоміжних структур, а саме NetworkCalculator:

1. Містить в собі методи для визначення індексу цитованості кожної з публікацій
2. Містить в собі методи для кластеризації даних та встановлення вершинам ваги для подальшої візуалізації
3. Містить в собі методи для пошуку найкоротшого шляху між вершинами
4. Містить в собі методи для пошуку найдовшого шляху між вершинами
5. Містить в собі методи для визначення найбільш довгого шляху між вершинами

Детальніше розглянем структуру і функціональне призначення модулю Smart Graph. Для початку розглянемо структуру пакету Graph, який містить в собі сутності, що описують стан графу, яку зображено на рис 4.20, а саме:

1. Vertex – вершина, інтерфейс який описує абстрактну поведінку вершин графу
2. Edge – ребро, інтерфейс який описує абстрактну поведінку ребер графу
3. Graph – інтерфейс, який описує взаємозв'язки між вершинами за допомогою ребер
4. Digraph – більш спеціалізований інтерфейс, який описує поведінку орієнтованого графа

5. DigraphEdgeList – конкретна імплементація орієнтованого графу
6. GraphEdgeList – конкретна імплементація неорієнтованого графу
7. InvalidVertexException – сутність, яка описує некорректний стан вершини
8. InvalidEdgeException – сутність, яка описує некоректний стан ребра

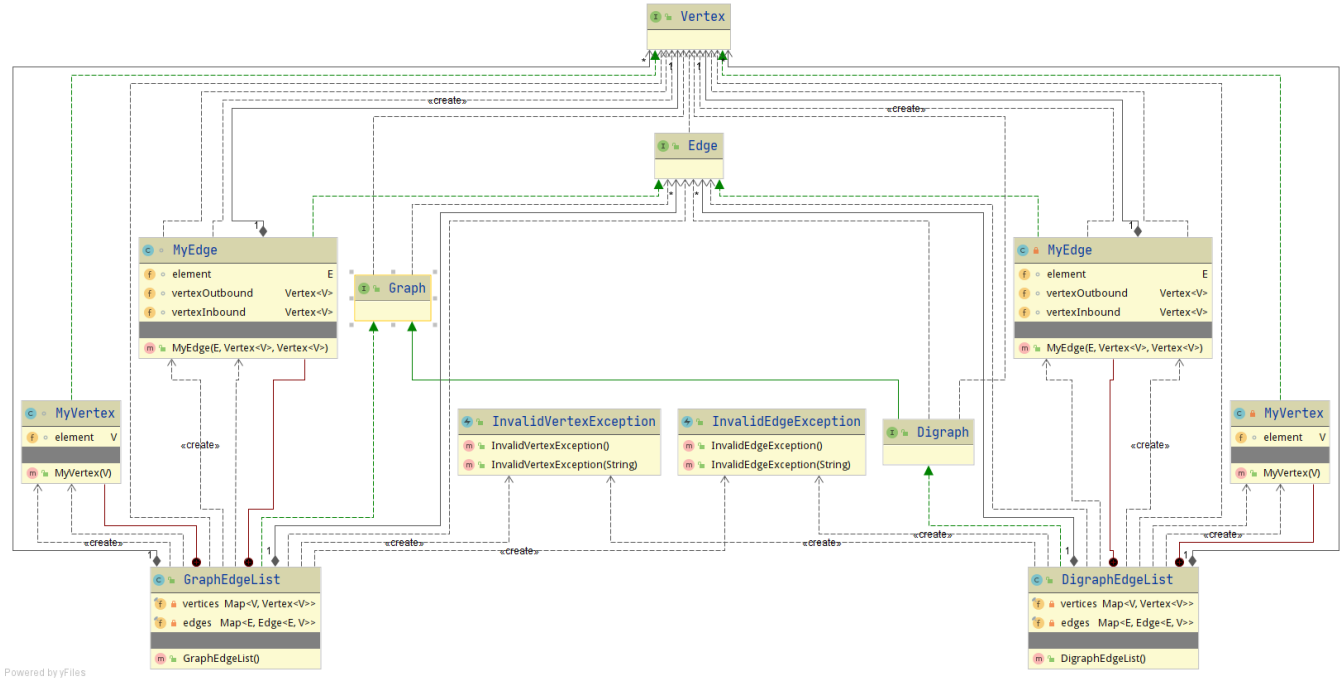

Рисунок 4.20 – структура пакету Graph

Варто зауважити, що сутності, які описані в цьому пакеті не мають жодного відношення до сутностей пакету Model, адже їх призначення у відображенні тих графів, якими ми оперуємо в пакеті Model.

Детальніше розглянемо структуру пакету GraphView. Призначення сутностей цього пакету у побудові відображення та розміщення графу на екрані, як зображено на рисунку 4.21.

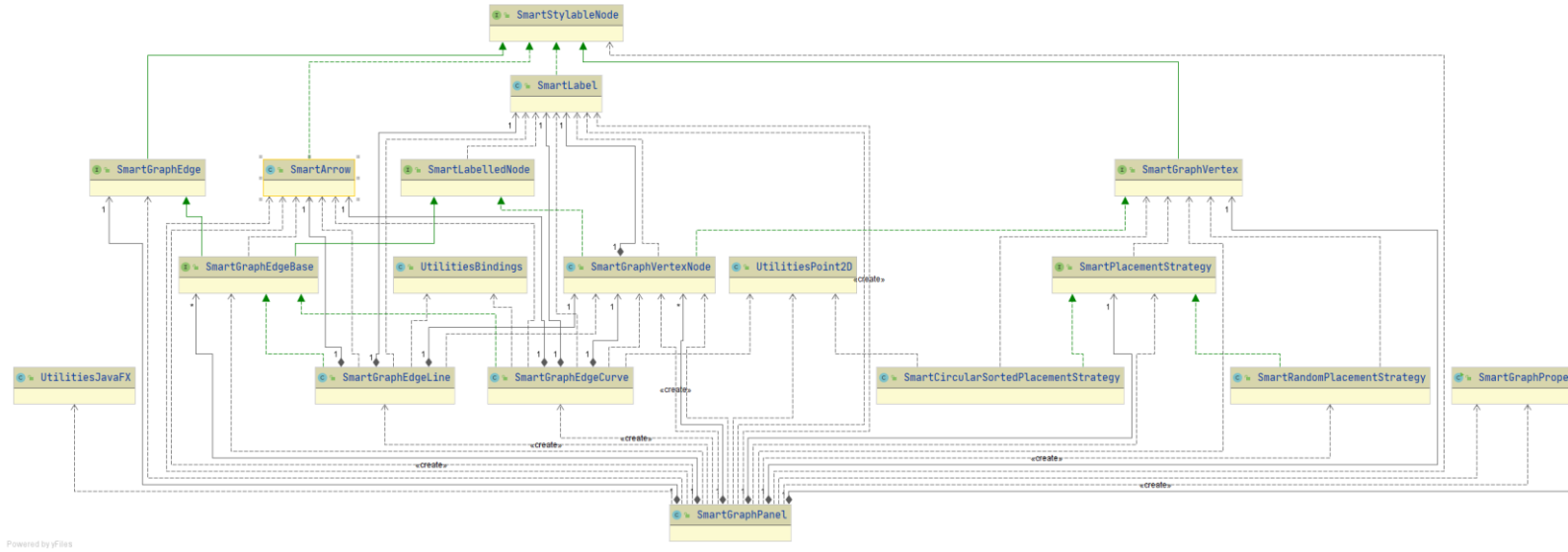

Рисунок 4.21 – структура пакету GraphView

Переглядаючи діаграму можемо зробити висновок про те, що єдиною точкою входу до цього програмного пакету є сутність SmartGraphPanel – вона агрегує усі елементи керування ті відображення які стосуються графу. Детальніше розглянемо цей класс для більш повного розуміння процесу відображення на рисунку 4.22

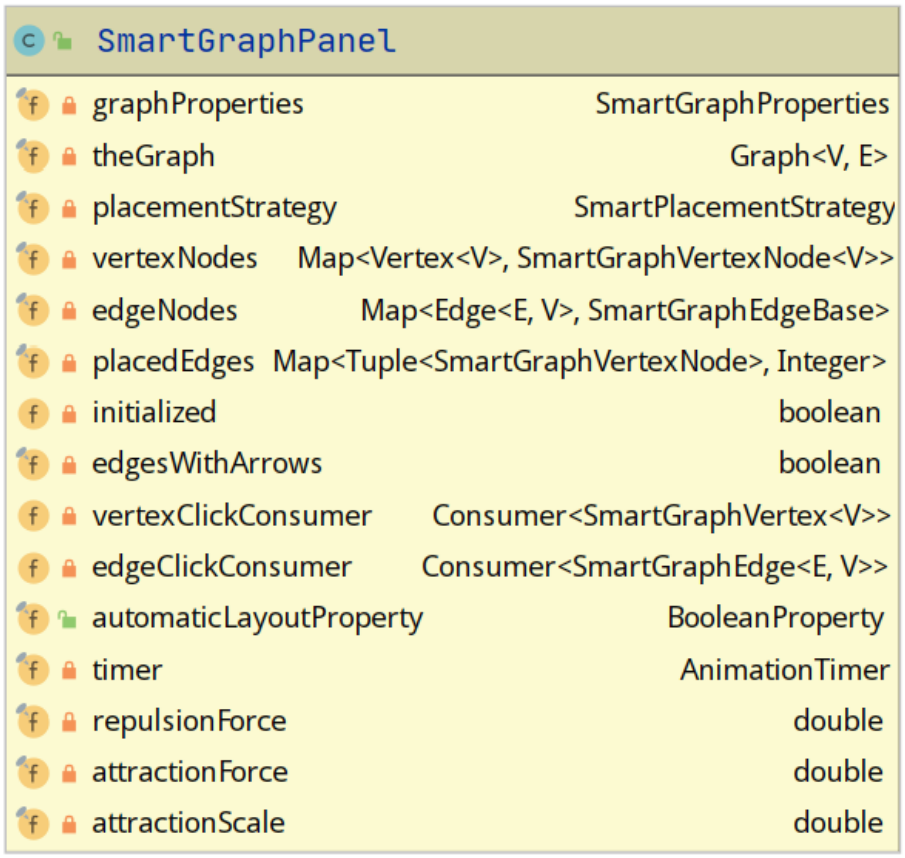

Рисунок 4.22 – структура SmartGraphPanel

Клас складається безпосередньо із графу над яким виконуються всі операції по відображенню та модифікації, стратегії розміщення вершин, позиції графів, змінних, які контролюють швидкість анімації, силу притягування та відштовхування вершин.

Розглянемо методи, які описують поведінку панелі із графом, на рисунку 4.23.

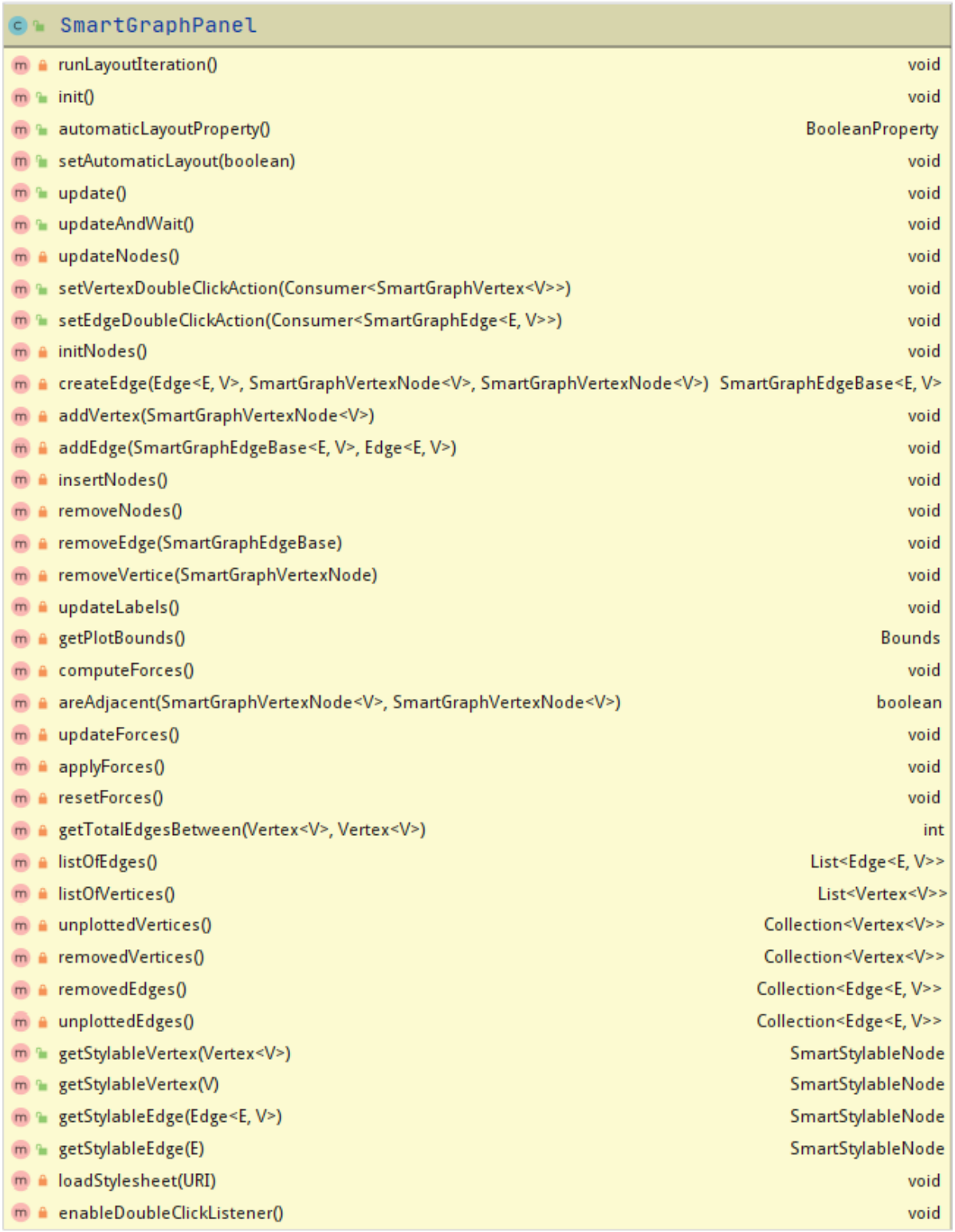

Рисунок 4.23 – методи SmartGraphPanel

Панель містить методи для завантаження довільних стилів(кольорів вершин, ребер, стилю ребер, кольору заднього фону), оновлення стану графу з урахуванням зміни положення будь-якої вершини та констант які модифікують сили пртягування та відштовхування по правилам стратегії розміщення вершин. Також присутні методи додавання, модифікації чи видалення вершин із графу під час виконання, встановлення функцій-обробників подій пов'язаних із взаємодією користувача з вершинами чи ребрами.

Стратегія розміщення описується інтерфейсом SmartPlacementStrategy та класами, які імплементують цей інтерфейс, їх зображено на рисунку 4.24.

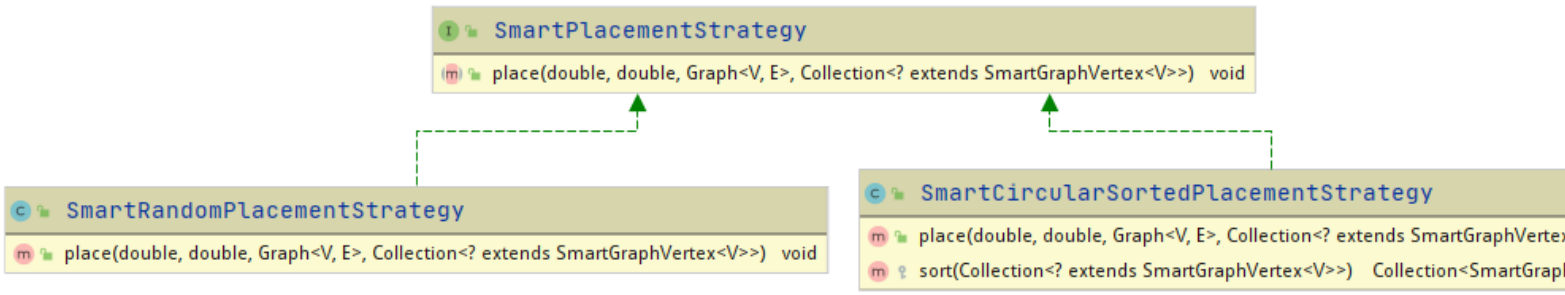

Рисунок 4.24 – ієрархія стратегій розміщень

Інтерфейс описує єдиний метод place, який розміщує вершини по матриці розміру N на M, розмірності якої передаються разом із вершинами які необхідно розміщувати. Наразі в системі присутні 2 імплементації:

1. SmartRandomPlacementStrategy – стратегія за якої всі вершини графу розміщуються в довільних місцях матриці
2. SmartCircularSortedPlacementStrategy – стратегія за якої всі вершини графу розміщуються відносно їх впливу, який визначається кількістю і напрямком ребер до інших вершин

4.6 Висновки до розділу

У цьому розділі було продемонстровано алгоритми, що використовуються при вирішенні задач роботи, показано структуру програмного комплексу, варіанти використання у вигляді схеми взаємодії користувача із системою та зображено високорівневу компонентну модель застосунку.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Вимоги до ЕВМ користувача:

1. Операційна система, для якої є реалізація Oracle JDK8.
2. Встановлений Oracle JRE 8
3. Мінімальні вимоги до апаратного забезпечення:
   3.1. Процесор: 1,0 ГГц
   3.2. Оперативна пам'ять: 128 Мбайт
   3.3. Відеокарта: ATI Radeon 9400 128 Мбайт
   3.4. Жорсткий диск: 100МБ вільного простору
   3.5. Аудіокарта: DirectX-сумісна

Інсталяція та запуск:

Програма не потребує інсталяції, розповсюджується у вигляді єдиного файлу в форматі JAR. Переконайтесь, що до системних шляхів додано шлях до виконавчого файлу java.exe, або ж до java(якщо ваша операційна система не на базі Microsoft Windows). Якщо ви не бажаєте додавати до системних шляхів даний файл, то в такому випадку для запуску програми із CLI необхідно буде використовувати java додаючи шлях до цього файлу на вашій системі. Наприклад: /usr/bin/java -jar AcademicsSearch.jar

Інакше для запуску програми необхідно викликати наступну команду у CLI:

java -jar AcademicsSearch.jar

Якщо середовище вірно налаштовано, та виконані всі необхідні для програмного забезпечення умови то у віконному режимі відкриється інтерфейс програми, як зображено на рисунку 5.1.

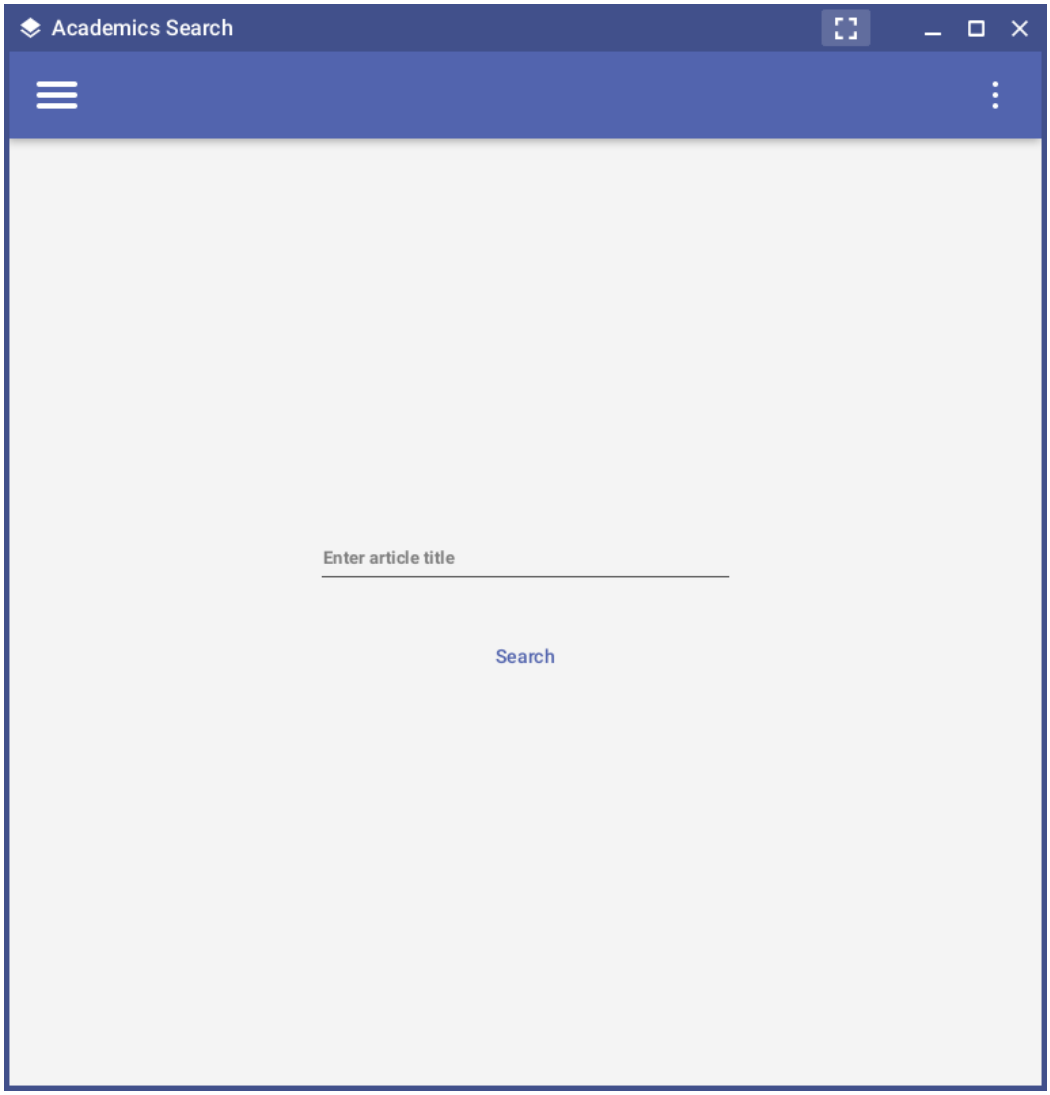

Рисунок 5.1 – основне вікно програми

Розглянемо кожен елемент дизайну та його призначення окремо.

В правому верхньому кутку є 3 кнопки, які відповідають за закриття вікна програми, згортання вікна та зміни його розміру. Дещо лівіше знаходиться кнопка яка дозволяє перейти в повноекранний режим роботи. В лівому кутку програми зображено логотип застосунку та його назву. Під ними знаходиться кнопка що відкриває альтернативне бокове меню, воно буде використовуватись для налаштування графічного інтерфейсу програми, розширення за допомогою плагінів, збереження результатів та відновлення проміжних результатів аналізу. Приклад вигляду програми в повноекранному безрамочному режимі зображено на рисунку 5.2

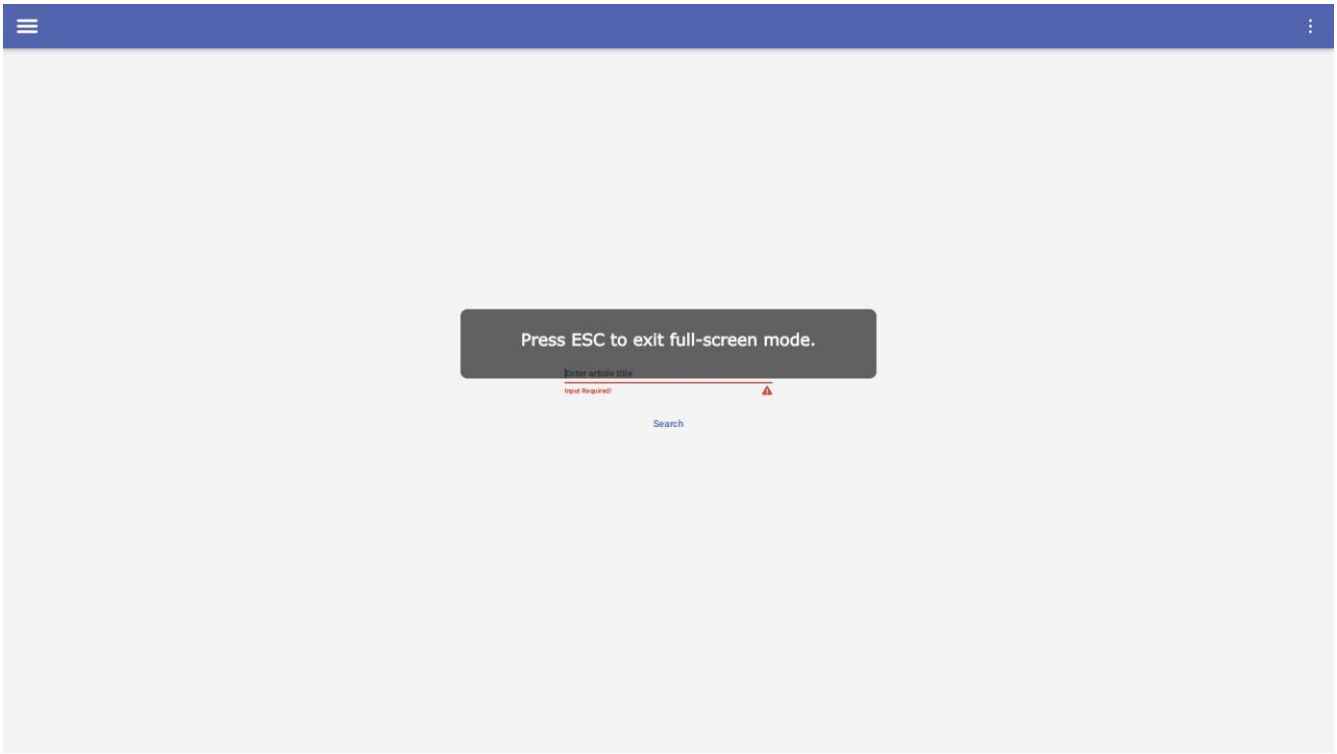

Рисунок 5.2 – повноекранний режим роботи

Для виходу з повноекранного режиму необхідно натиснути кнопку Escape.

На синій полосі в правій частині знаходиться кнопка яка відкриває контекстне меню. В контекстному меню 2 кнопки: About та Exit. Перша відкриває модальне вікно з інформацією про програмне забезпечення, а друга закриває програму.

На рисунку 5.3 зображено модальне вікно з інформацією про програму

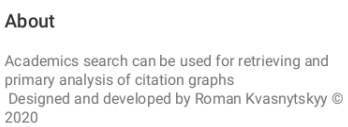

Рисунок 5.3 – інформаційне вікно

В центрі робочого вікна є 2 керучі елементи – текстове поле та кнопка що ініціює пошук публікації. Зазначимо, що кнопка заблокована до тих пір, допоки користувач не введе щось до текстового поля. У разі якщо

користувач натиснув на кнопку без жодного вводу, він отримає підказку, яка вкаже на необхідність заповнення текстового поля. Підказка зображена на рисунку 5.4

Рисунок 5.4 – підказка про необхідність вводу

Після введення необхідного пошукового запиту під час очікування відповіді від серверу користувач бачить на екрані анімацію яка описує прогресс, під час очікування відповіді від серверу інтерфейс не блокується та залишається інтерактивним. На рисунку 5.6 зображено інтерфейс програми під час очікування обробки пошукового запиту

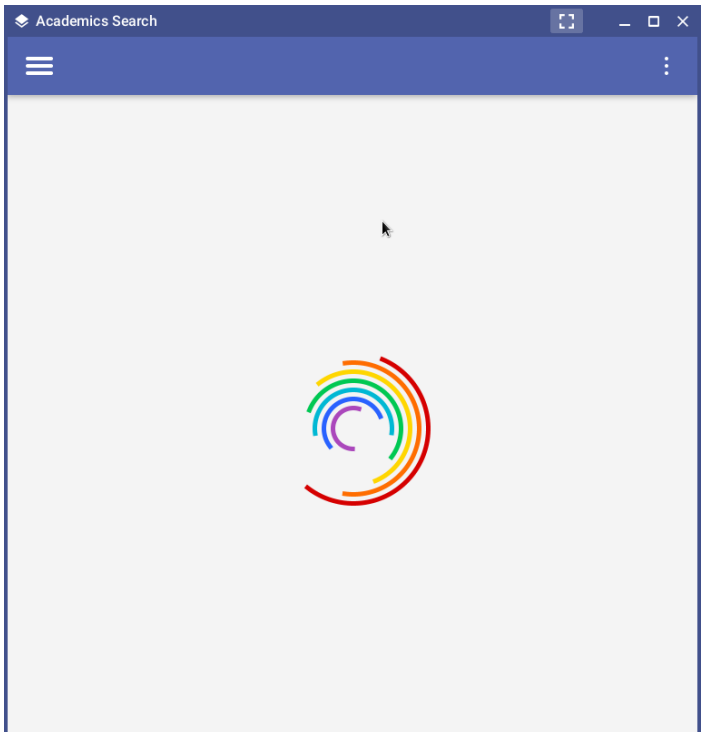

Рисунок 5.6 – очікування відповіді від серверу

Екран очікування відповіді від серверу необхідний, оскільки запити які надходять від клієнту обробляються на сервері значний час, і для того аби користувачу не здавалось, що система «зависла» відображається анімація із семи різнокольорових ліній, що рухають по колу до тих пір

допоки ПК користувача не отримає по мережі відповідь від серверу, та застосунок не обробить відповідь від серверу. Коли відповідь буде оброблено – користувача буде перенаправлено на наступний екран із результатами його пошукового запиту, де він зможе переглянути деталі публікацій, обрахувати індекс цитованості та зберегти проміжний результат, для того аби мати можливість продовжити роботу з цього ж місця. Також у користувача буде можливість обрати необхідний йому вид аналізу для кожної публікації.

При натисканні на кнопку із зображенням трьох горизонтальних паралельно зображених відрізків відчиняється побічне меню із доступними вікнами, як описано на рисунку 5.7.

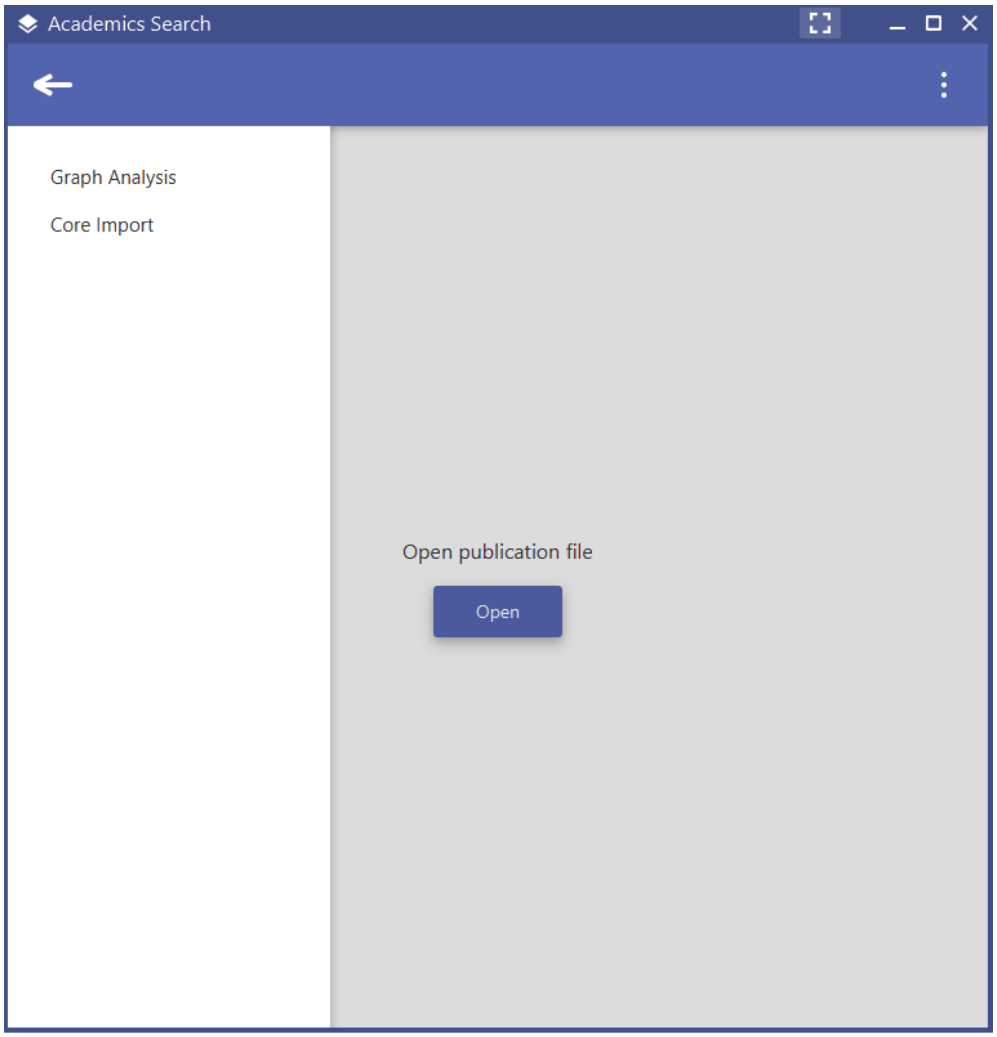

Рисунок 5.7 – бокове меню

Оскільки застосунок є здебільшого каркасом, та платформою для різноманітних аналітичних засобів, в ньому є елемент користувацького інтерфейсу, який дозволяє обрати необхідний користувачу. Цим елементов якраз і виступає бокове меню. Наразі, для демонстрації роботи в ньому присутні 2 вкладки:

1. Graph Analysis – аналітичний модуль, який здебільшого аналізує графи цитованності. Натискання на цей текст у меню, відчиняє головне цього модуля.
2. Core Import – продемонстрований декількаами сторінками вище модуль для імпорту данних з агрегатору НТЛ Core.

Розглянемо головне меню аналітичного модулю “Graph Analysis”. В центрі екрану розміщено кнопку, яка дозволяє обрати попередньо імпортований з агрегатору перелік публікацій із взаємозв’язками, як на рисунку 5.8.

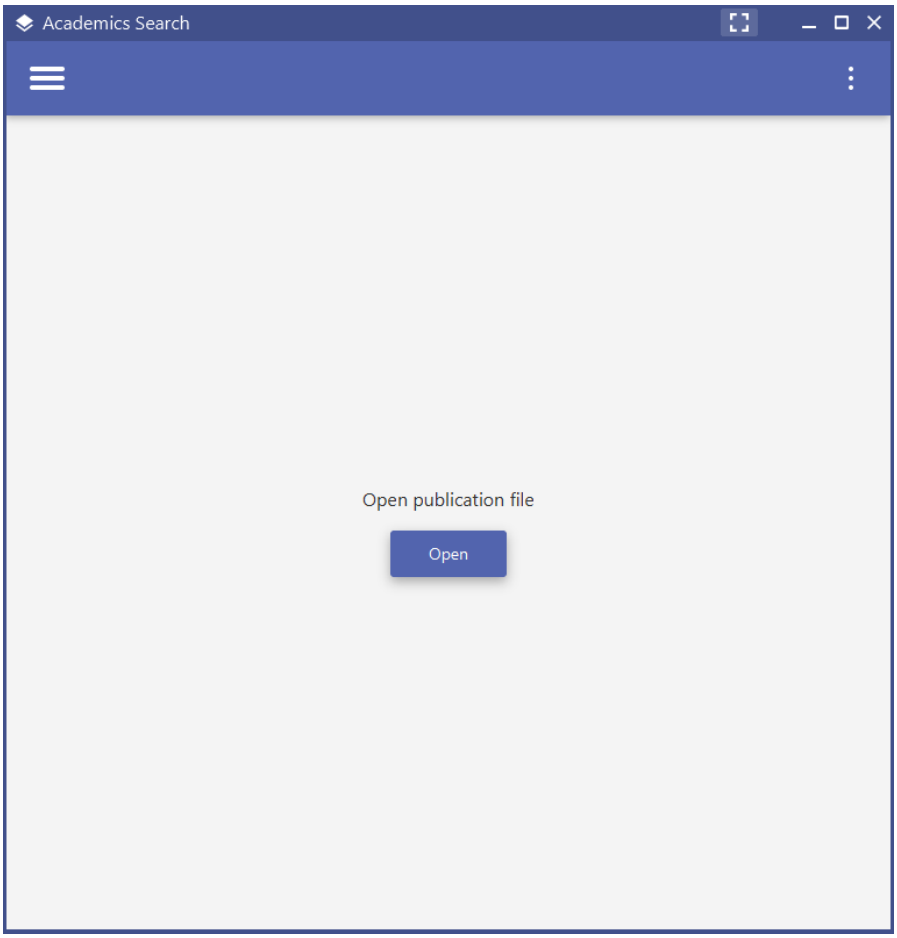

Рисунок 5.8 – головний екран модулю Graph Analysis

При натисканні на кнопку “Open” відчиняється діалогове вікно у стилістиці системи користувача із заголовком “Open publications file”, як на рисунку 5.9

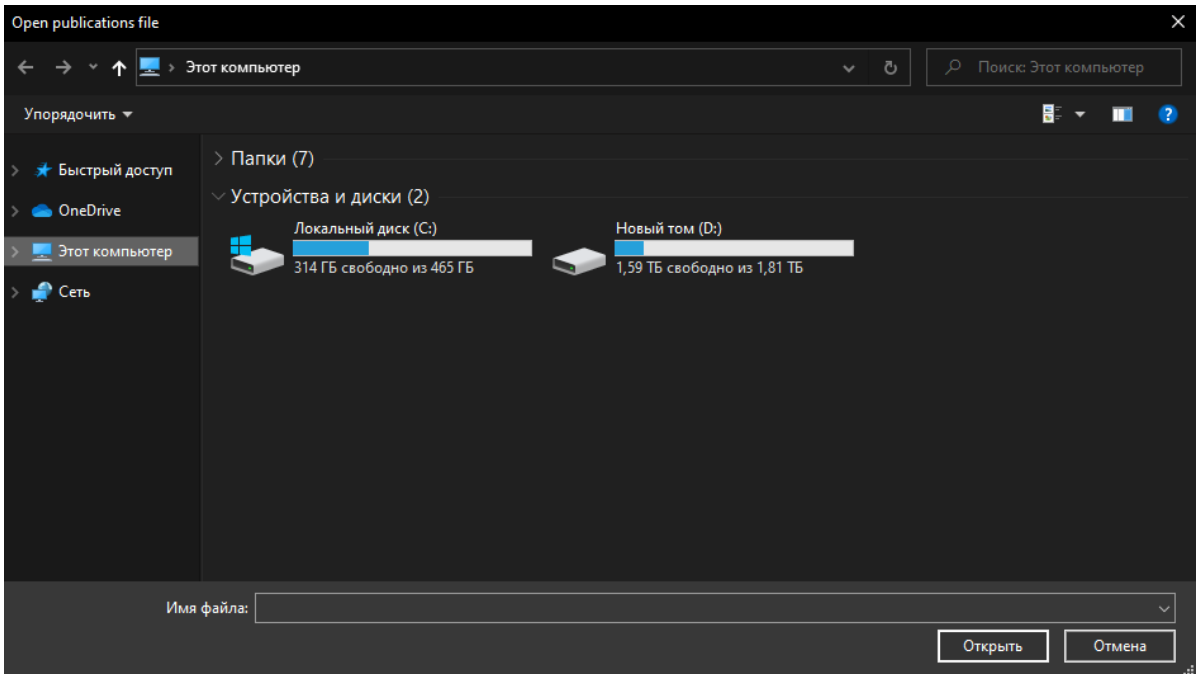

Рисунок 5.9 – діалогове вікно запиту файлу

Якщо користувач обирає файл, який неможливо ідентифікувати як файл імпортованих публікацій він отримає про це сповіщення, та зможе повторити спробу, як зображено на рисунку 5.10

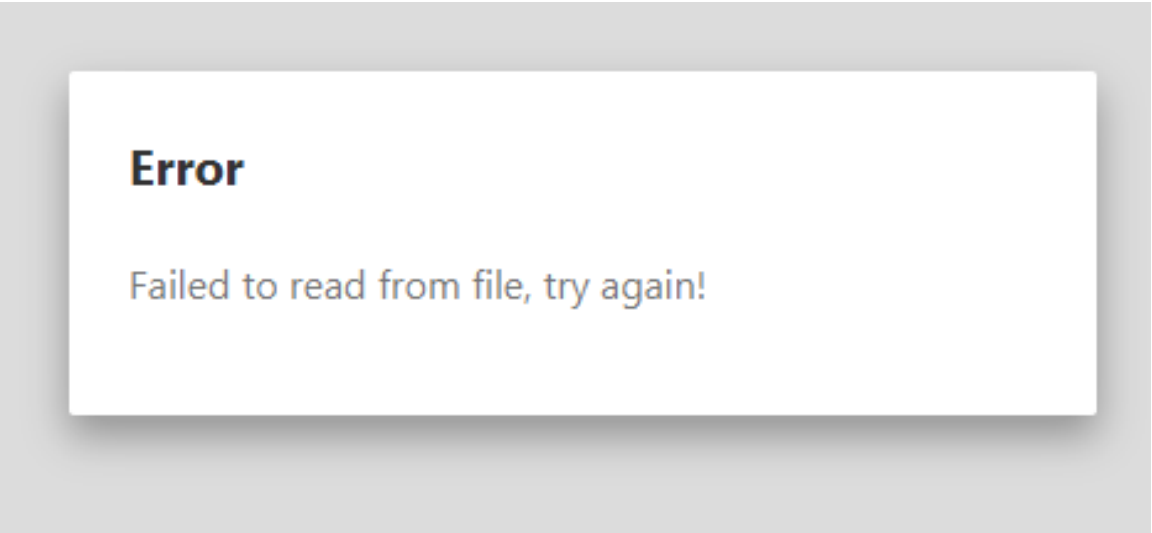

Рисунок 5.10 – помилка читання файлу

Якщо користувач обрав корректний файл, то після недовготривалого очікування, пов’язанного з первинним аналізом користувач опиняється на сторінці із таблицею імпортованих статей з інформацією про публікації. В первинному варіанті із назвою роботи, роком її випуску та DOI, як показано на рисунку 5.11

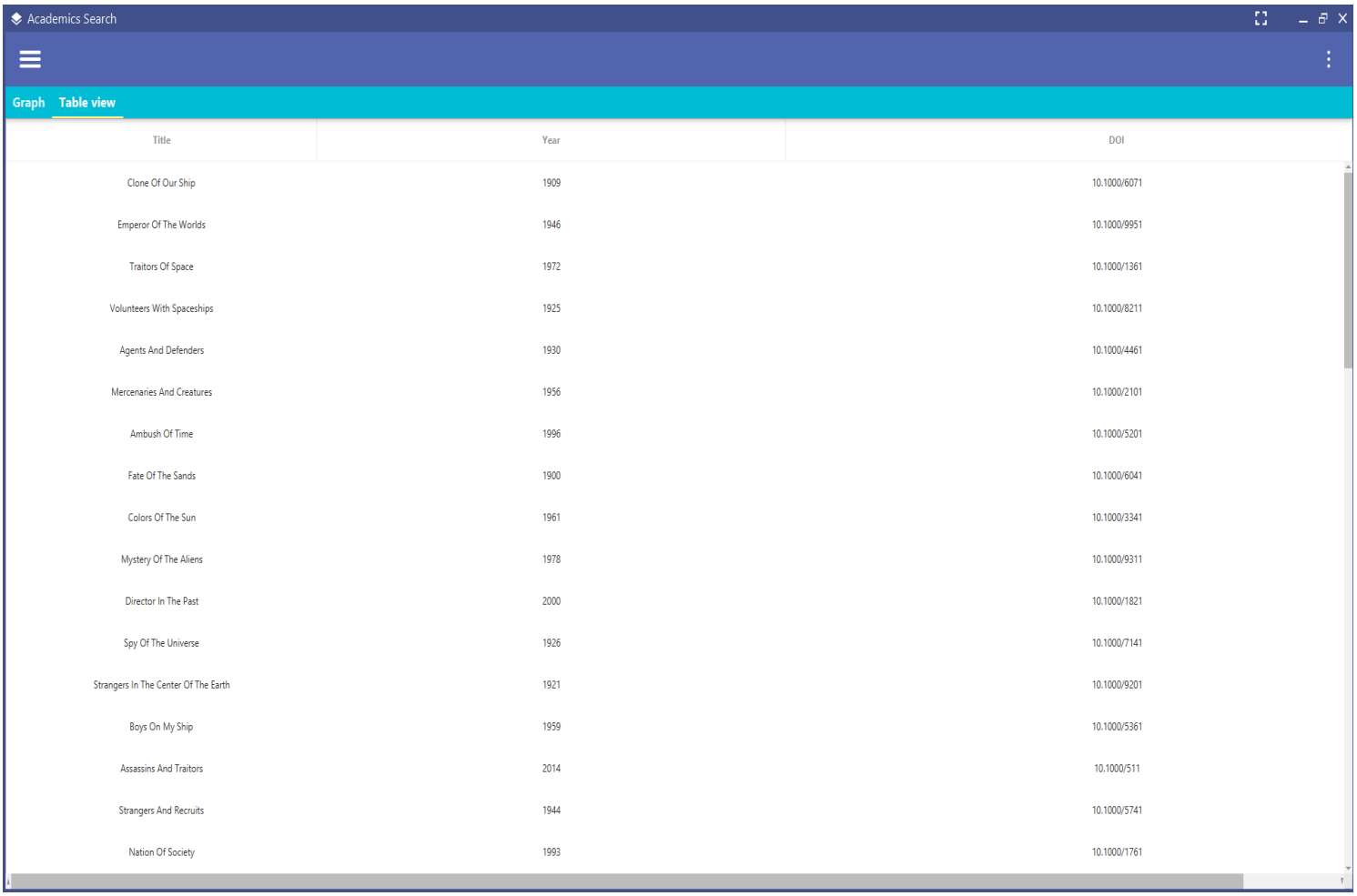

Рисунок 5.11 – Таблиця імпортованих публікаційї

Користувач отримує можливість навігації по екранам графічно-аналітичного модуля за допомогою вкладок, що виділені на яскраво-синьому фоні у верхній частині інтерфейсу. Для переключення між вкладками достатньо на них натиснути. Якщо відкрити вікно Graph користувач побачить публікації у вигляді орієнтованного ациклічного графу. Кожна із вершин графу в цьому елементі користувацького інтерфейсу є рухомою, і важливо зауважити, що елементи графу будуть зміщуватись відповідно до того, як користувач перетягує вершини, для того аби зберегти загальну структуру. Адже під час побудови графу усі поєднані

вершини притягуються одна до одної, а всі незв’язані намаються віддалитись, таким чином природньо розміщуюсь в стані рівноваги, коли усі потенціали поля будуть врівноважені. Користувачу це може нагадати схему взаємодії електронів, і недарма – принцип роботи той самий. Це показано на рисунку 5.12.

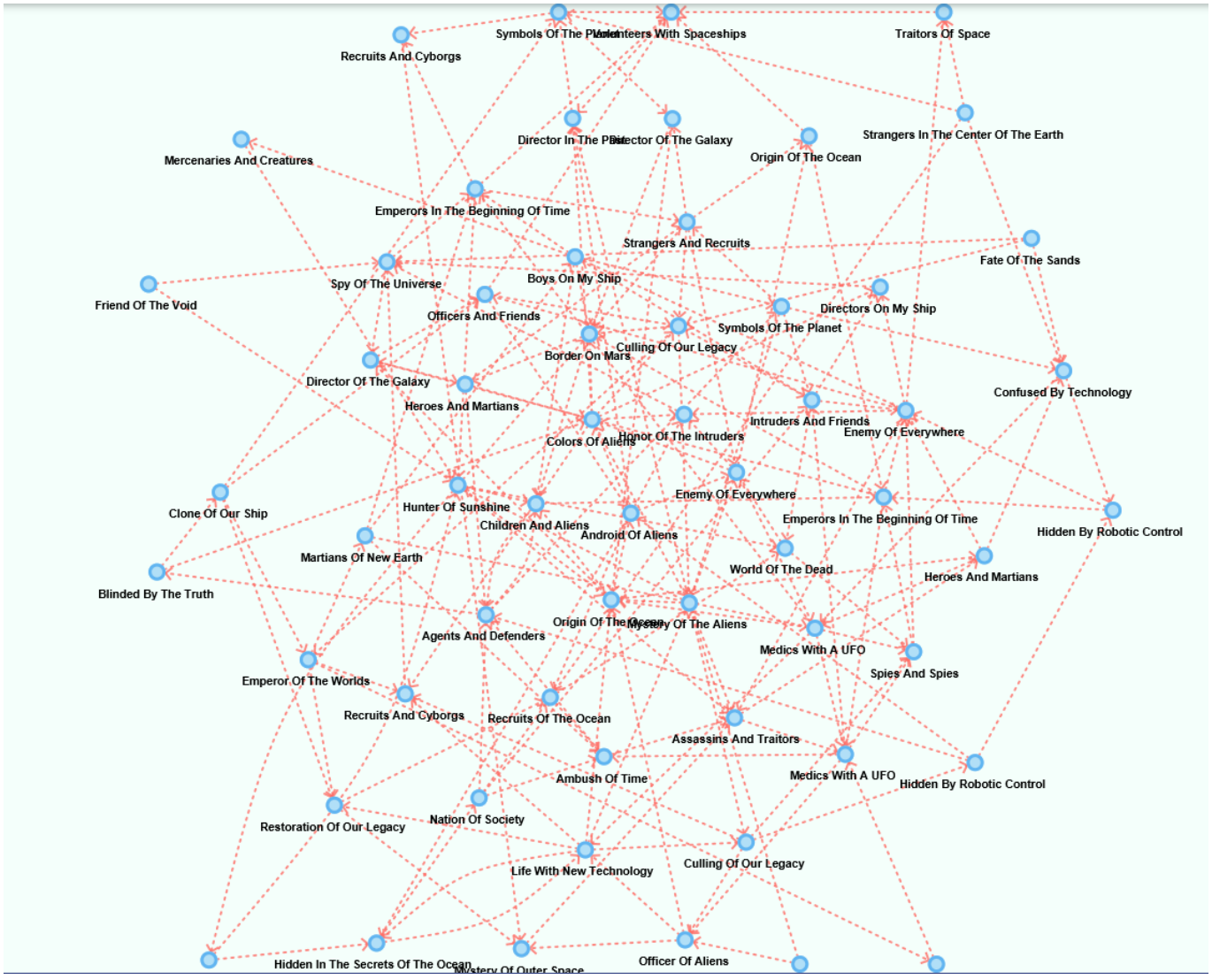

Рисунок 5.12 – розміщення елементів графу

Легко побачити, що ті вершини(публікації) які мають найменшу кількість зв’язків(цитат), неначе «витісняються» на край графу, а всередині залишаються, ті які мають найбільшу кількість зв’язків. Цей ефект, можна назвати ефектом кластеризації – ми бачимо, що найбільш цитовані елементи знаходяться в «середині» графу, оскільки тільки так всі сили будуть врівноважені. З цього ми ще й можемо зробити висновок відносно того, які вершини є найбільш зв’язними, або в нашому випадку, які публікації є основними в цьому наборі публікацій.

При спробі зміщення вершин графу, весь граф почне перебудовуватись внаслідок порушення рівноваги в силах притягування та відштовхування. Той же граф з рисунку 5.12 після перебудови прийме вигляд як на рисунку 5.13.

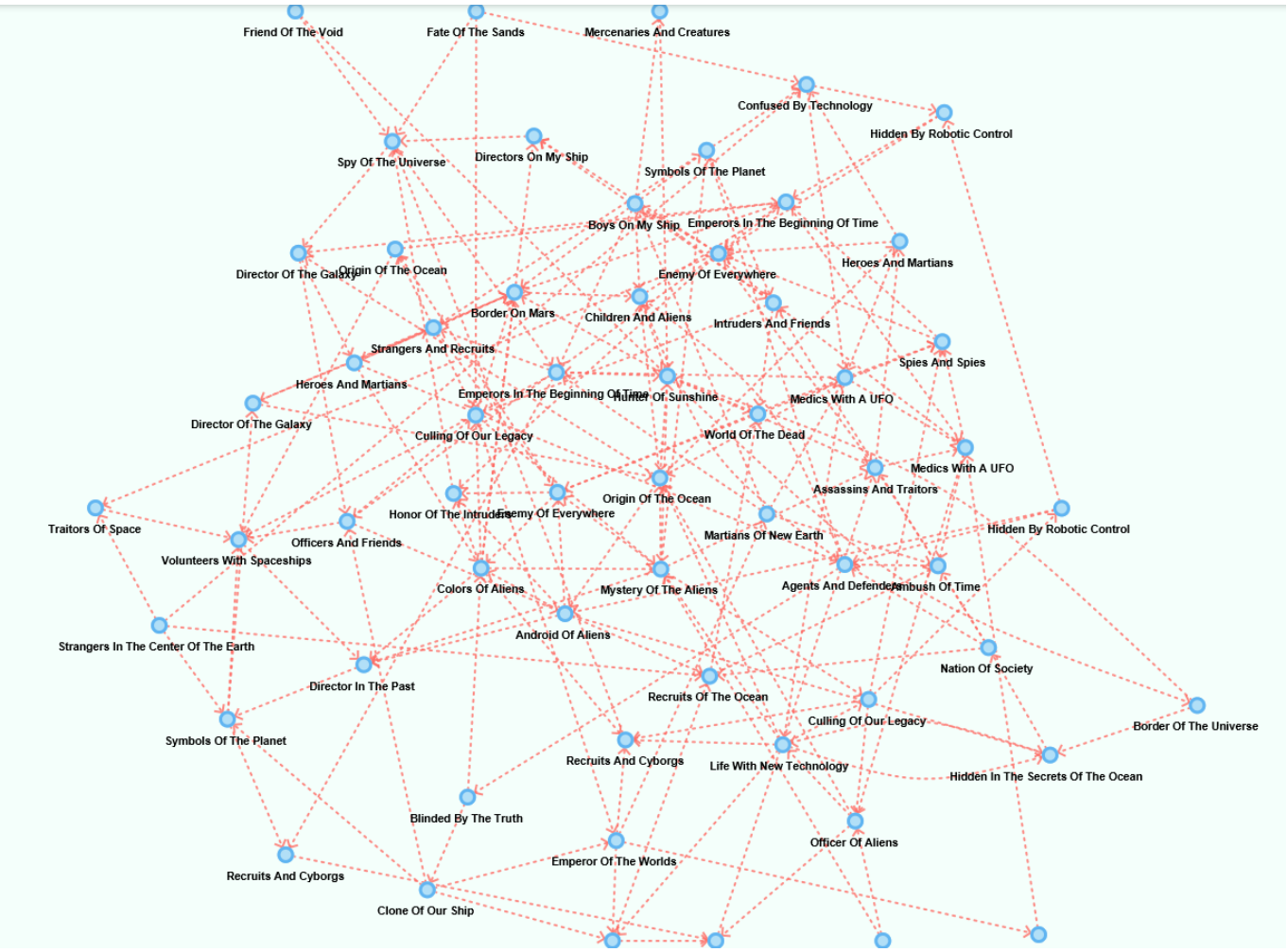

Рисунок 5.13 – зміщенний граф

Якщо уважно придивитись до нового розміщення, можемо побачити, що суттєво нічого не змінилось, при спробі зміщення одного із центральних елементів за межі основного накопичення елементів – сили притягування повертають його до центру і повертають точно в те саме місце де він був попереду. Єдине чого можно досягти – це поміняти місцями елементи на периферії скупчення, є велика ймовірність того, що вони приймуть нове стабільше положення.

Варто зазначити, що не так просто розгледіти назву кожної окремої вершини, особливо, якщо ця вершина знаходиться в середині скупчення та її назва зливається

із зв’язками між вершинами, самими іншими вершинами, або ж їхніми назвами, і саме тому при подвійному натисканні на будь який елемент графу користувач може детальніше переглянути інформацію про вершину, як зображено на рисунку 5.14

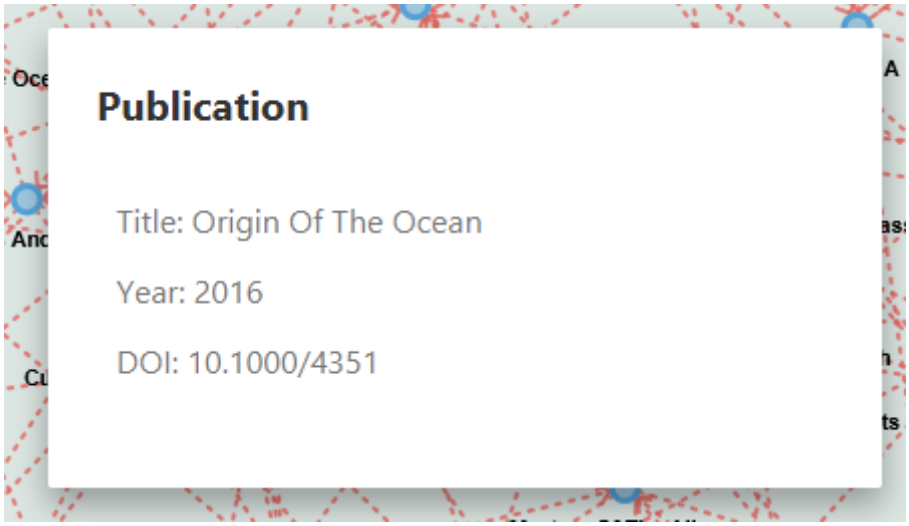

Рисунок 5.14 – інформативне вікно публікації

Також, на правій стороні вікна графу розміщено інформативний блок із загальною інформацією про мережу, як на рисунку 5.15

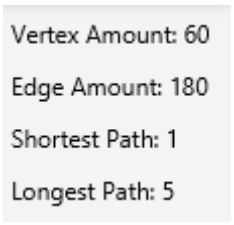

Рисунок 5.15 – статистика графу

Розглянемо за пунктами:

1. Vertex Amount – кількість вершин(публікацій), які було імпортовано.
2. Edge Amount – кількість зв’язків(цитувань), які встановлені між публікаціями.
3. Shortest Path – мінімальна кількість вершин, які можна пройти із однієї вершини до іншої.
4. Longest Path – максимальна кількість вершин, які можна пройти із однії вершини до іншої.

ВИСНОВКИ

Було проведено дослідження на тему аналізу науково-технічної літератури, в результаті якого було описано методологію аналізу впливу наукових публікацій та їх взаємозв’язків на базі графу цитувань між науковими працями. Спроектовано та розроблено програмне забезпечення яке реалізує вищевказану методологію, спрощує та автоматизує левову частку роботи необхідно для аналізу літератури. Автор ознайомився із сучасними пошуковими системами, бібліографіями та найбільшими агрегаторами наукових публікацій. Ознайомився з областю наукометричних та бібліографічних задач, та поглибив свої знання в області теорії графів , застосувавши її при розробці методології аналізу публікацій наукового характеру.

ДЖЕРЕЛА

1. Citation counts for research evaluation: standards of good practice for analyzing bibliometric data and presenting and interpreting results. Lutz Bornmann[1,]*, Rüdiger Mutz[1], Christoph Neuhaus[1], Hans-Dieter Daniel[1,2]

2. Спекторський І. Я. Дискретна математика. — К.: «Політехніка», . 220 с. ISBN966-622-136-5

3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 22.5, pp. 552–557.

Додаток А

Аналіз характеристик даних, що збираються з відібраних джерел науково – технічної інформації

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6286_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6286_20Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6286_20Б 12-1	AcademicsSearch.jar	Застосунок

Додаток Б

Аналіз характеристик даних, що збираються з відібраних джерел науково – технічної інформації

Лістинг програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6286_20Б

Аркушів 10

Київ 2020

Частина класу NetworkCalculator із функціями аналізу:

```
public CitationScoreParameters getCitationScoreParameters() {
 return (CitationScoreParameters) this.citScoreParams.clone();
}

public void setCitationScoreParameters(CitationScoreParameters paramCitationScoreParameters) {
 if (paramCitationScoreParameters.equals(this.citScoreParams))
  return;
 int i = this.pub.length;
 this.citScoreParams = (CitationScoreParameters) paramCitationScoreParameters.clone();
 if (paramCitationScoreParameters.citScoreMethod == 0) {
  double[] arrayOfDouble = this.citNetwork.getImportanceScores(paramCitationScoreParameters.citWindowLength - 1,
paramCitationScoreParameters.fracCounting);
   for (byte b = 0; b < i; b++)
    (this.pub[b]).citScore = arrayOfDouble[b];
  this.citScoresInteger = !paramCitationScoreParameters.fracCounting;
 } else if (paramCitationScoreParameters.citScoreMethod == 1) {
  for (Publication publication : this.pub) publication.citScore = publication.externalCitScore;
  this.citScoresInteger = this.externalCitScoresInteger;
 }
}

public boolean canGoBackward() {
 return (this.currentSubnetworkIndex > 0);
}

public boolean goBackward() {
 if (!canGoBackward())
  return false;
 this.currentSubnetworkIndex--;
 this.currentSubnetwork = this.subnetwork.get(this.currentSubnetworkIndex);
 updateSelectedSubnetwork();
 return true;
}

public boolean canGoForward() {
 return (this.currentSubnetworkIndex < this.subnetwork.size() - 1);
}

public boolean goForward() {
 if (!canGoForward())
  return false;
 this.currentSubnetworkIndex++;
 this.currentSubnetwork = this.subnetwork.get(this.currentSubnetworkIndex);
 updateSelectedSubnetwork();
 return true;
}

public void goToFullNetwork() {
 goToNewSubnetwork(this.citNetwork.getFullGraph());
}

public boolean canClearSelectedSubnetwork() {
 return (this.selectedSubnetworkStats != null);
}

public boolean clearSelectedSubnetwork() {
 if (!canClearSelectedSubnetwork())
  return false;
 clearSelectedSubnetwork2();
 return true;
}

public boolean canDrillDown() {
 return (this.selectedSubnetworkStats != null && this.selectedSubnetworkStats.nPubs > 0);
}


public boolean getCitationScoresInteger() {
 return this.citScoresInteger;
}

public boolean getRecordsIncomplete() {
 return this.recordsIncomplete;
```

```
}

public int[][] getCitingCitedPublicationIndices() {
  return this.citNetwork.getEdgeSourceDestNodeIDs(this.currentSubnetwork.pubIndex);
}

public int[][] getCitingCitedPublicationIndicesFullNetwork() {
  return this.citNetwork.getEdgeSourceDestNodeIDs(this.citNetwork.getFullGraph());
}

public int[] getIndicesNonEmptyGroups() {
  return getIndicesNonEmptyGroups(getPublicationsGroupIndex());
}

public int[] getIndicesNonEmptyGroupsFullNetwork() {
  return getIndicesNonEmptyGroups(this.groupIndexHistory[this.groupIndexHistoryCurrentState]);
}

public boolean canUndoGroups() {
  return (this.groupIndexHistoryCurrentState != this.groupIndexHistoryOldestState);
}

public boolean undoGroups() {
  if (!canUndoGroups())
    return false;
  this.groupIndexHistoryCurrentState = (this.groupIndexHistoryCurrentState - 1 + 11) % 11;
  if (this.currentSubnetwork.selectionParams.selectionMethod == 2)
    updateSelectedSubnetwork();
  return true;
}

public boolean canRedoGroups() {
  return (this.groupIndexHistoryCurrentState != this.groupIndexHistoryNewestState);
}

public boolean redoGroups() {
  if (!canRedoGroups())
    return false;
  this.groupIndexHistoryCurrentState = (this.groupIndexHistoryCurrentState + 1) % 11;
  if (this.currentSubnetwork.selectionParams.selectionMethod == 2)
    updateSelectedSubnetwork();
  return true;
}

public void clearAllGroups() {
  int i = this.currentSubnetwork.pubIndex.length;
  int[] arrayOfInt = this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
  for (byte b = 0; b < i; b++)
    arrayOfInt[this.currentSubnetwork.pubIndex[b]] = 0;
  updateGroupIndices(arrayOfInt);
}

public void clearAllGroupsFullNetwork() {
  int i = this.pub.length;
  updateGroupIndices(new int[i]);
}

public int[] getNPublicationsPerGroup() {
  int j = this.currentSubnetwork.pubIndex.length;
  int i = 0;
  for (byte b = 0; b < j; b++) {
    if (this.groupIndexHistory[this.groupIndexHistoryCurrentState][this.currentSubnetwork.pubIndex[b]] > i)
      i = this.groupIndexHistory[this.groupIndexHistoryCurrentState][this.currentSubnetwork.pubIndex[b]];

public void movePublicationsFromOneGroupToAnotherFullNetwork(int paramInt1, int paramInt2) {
  int i = this.pub.length;
  int[] arrayOfInt = this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
  for (byte b = 0; b < i; b++) {
    if (arrayOfInt[b] == paramInt1)
      arrayOfInt[b] = paramInt2;
  }
  updateGroupIndices(arrayOfInt);
}

public boolean[] getPublicationsMarked() {
  return this.currentSubnetwork.pubMarked.clone();
```

```
}

public void setPublicationMarked(int paramInt, boolean paramBoolean) {
  if (this.currentSubnetwork.pubMarked[paramInt] == paramBoolean)
    return;
  this.currentSubnetwork.pubMarked[paramInt] = paramBoolean;
  if (this.currentSubnetwork.selectionParams.selectionMethod == 0)
    updateSelectedSubnetwork();
}

public void setPublicationsMarked(boolean[] paramArrayOfboolean) {
  this.currentSubnetwork.pubMarked = paramArrayOfboolean.clone();
  if (this.currentSubnetwork.selectionParams.selectionMethod == 0)
    updateSelectedSubnetwork();
}

public void setPublicationsMarked(int[] paramArrayOfint, boolean paramBoolean) {
  for (int i : paramArrayOfint) this.currentSubnetwork.pubMarked[i] = paramBoolean;
  if (this.currentSubnetwork.selectionParams.selectionMethod == 0)
    updateSelectedSubnetwork();
}

public boolean[] getPublicationsSelected() {
  return this.pubSelected.clone();
}

public boolean[] getPublicationsVisible() {
  return this.currentSubnetwork.pubVisible.clone();
}

public void setPublicationVisible(int paramInt, boolean paramBoolean) {
  if (this.currentSubnetwork.pubVisible[paramInt] == paramBoolean)
    return;
  this.currentSubnetwork.pubVisible[paramInt] = paramBoolean;
  clearVisiblePublicationsData();
}

public void setPublicationsVisible(boolean[] paramArrayOfboolean) {
  this.currentSubnetwork.pubVisible = paramArrayOfboolean.clone();
  clearVisiblePublicationsData();
}

public void setPublicationsVisible(int[] paramArrayOfint, boolean paramBoolean) {
  for (int i : paramArrayOfint) this.currentSubnetwork.pubVisible[i] = paramBoolean;
  clearVisiblePublicationsData();
}

public VisiblePublicationsParameters getVisiblePublicationsParameters() {
  return (VisiblePublicationsParameters) this.visiblePubsParams.clone();
}

public void setVisiblePublicationsParameters(VisiblePublicationsParameters paramVisiblePublicationsParameters) {
  this.visiblePubsParams = (VisiblePublicationsParameters) paramVisiblePublicationsParameters.clone();
  initVisiblePublications();
}

public LayoutParameters getLayoutParameters() {
  return (LayoutParameters) this.layoutParams.clone();
}

public void setLayoutParameters(LayoutParameters paramLayoutParameters) {
  this.layoutParams = (LayoutParameters) paramLayoutParameters.clone();
  clearVisiblePublicationsData();
}

public boolean[][] getCitationRelationsVisiblePublicationsInTransitiveReduction(int paramInt) {
  int i = this.currentSubnetwork.visiblePub.length;
  boolean[][] arrayOfBoolean2 = new boolean[i][];
  boolean[][] arrayOfBoolean1 = new boolean[i][];
  for (byte b = 0; b < i; b++) {
    arrayOfBoolean2[b] = new boolean[b];
    arrayOfBoolean1[b] = new boolean[b];
    for (int j = b - 1; j >= 0; j--) {
      if (this.currentSubnetwork.citRelationVisiblePubs[b][j] != null && (this.currentSubnetwork.citRelationVisiblePubs[b][j]).minNSteps <= paramInt &&
!arrayOfBoolean1[b][j]) {
        arrayOfBoolean2[b][j] = true;
```

```
        arrayOfBoolean1[b][j]  = true;
        for (byte b1 = 0; b1 < j; b1++) {
          if (arrayOfBoolean1[j][b1])
            arrayOfBoolean1[b][b1]  = true;
        }
      }
    }
  }
  return arrayOfBoolean2;
}

public int[] filterPublications(FilterParameters paramFilterParameters)  {
  int j = this.currentSubnetwork.pubIndex.length;
  int i = this.author.length;
  int k = this.source.length;
  if (!paramFilterParameters.equals(this.currentSubnetwork.filterParams))  {
    this.currentSubnetwork.filterParams = (FilterParameters) paramFilterParameters.clone();
    String str1 = paramFilterParameters.author.trim();
    String str3 = paramFilterParameters.title.trim();
    String str2 = paramFilterParameters.source.trim();
    this.currentSubnetwork.filteredPubIndex  = new int[j];
    int[] arrayOfInt1  = new int[i];
    int[] arrayOfInt2  = new int[k];
    byte b1 = 0;
    for (byte b2 = 0; b2 < j; b2++) {
      Publication publication = this.pub[this.currentSubnetwork.pubIndex[b2]];
      if ((paramFilterParameters.beginYear  >= 0 && publication.year < paramFilterParameters.beginYear)  || (paramFilterParameters.endYear  >= 0 &&
publication.year > paramFilterParameters.endYear)  || (!Double.isNaN(paramFilterParameters.minCitScore) && publication.citScore <
paramFilterParameters.minCitScore) || (!Double.isNaN(paramFilterParameters.maxCitScore)  && publication.citScore > paramFilterParameters.maxCitScore)
|| (paramFilterParameters.groupIndex  >= 0 && this.groupIndexHistory[this.groupIndexHistoryCurrentState][this.currentSubnetwork.pubIndex[b2]]  !=
paramFilterParameters.groupIndex))
        continue;
      if (!str2.isEmpty()) {
        int m = publication.sourceIndex;
        if (arrayOfInt2[m]  == 0)
          arrayOfInt2[m]  = this.source[m].matches(str2)  ? 1 : -1;
        if (arrayOfInt2[m]  == -1)
          continue;
      }
      if (!str1.isEmpty()) {
        byte b;
        for (b = 0; b < publication.authorIndex.length; b++) {
          int m = publication.authorIndex[b];
          if (arrayOfInt1[m]  == 0)
            arrayOfInt1[m]  = this.author[m].matches(str1)  ? 1 : -1;
          if (arrayOfInt1[m]  == 1)
            break;
        }
        if (b == publication.authorIndex.length)
          continue;
      }
      if (str3.isEmpty() || publication.title.matches(str3)) {
        this.currentSubnetwork.filteredPubIndex[b1] = b2;
        b1++;
      }
    }
    this.currentSubnetwork.filteredPubIndex  = Arrays.copyOfRange(this.currentSubnetwork.filteredPubIndex,  0, b1);
  }
  return this.currentSubnetwork.filteredPubIndex;
}

public boolean  askClearAllGroupsFullNetwork()  {
  int i = this.pub.length;
  int j = this.currentSubnetwork.pubIndex.length;
  byte b1 = 0;
  byte b2;
  for (b2 = 0; b2 < i; b2++) {
    if (this.groupIndexHistory[this.groupIndexHistoryCurrentState][b2] > 0)
      b1++;
  }
  byte b3 = 0;
  for (b2 = 0; b2 < j; b2++) {
    if (this.groupIndexHistory[this.groupIndexHistoryCurrentState][this.currentSubnetwork.pubIndex[b2]]  > 0)
      b3++;
  }
  return (b1 > b3);
```

```
}

public ConnectedComponentsResults identifyConnectedComponents(ConnectedComponentsParameters paramConnectedComponentsParameters, boolean
paramBoolean) {
 int i = this.currentSubnetwork.pubIndex.length;
 ConnectedComponentsResults connectedComponentsResults = new ConnectedComponentsResults();
 connectedComponentsResults.largestComponentOnly = paramConnectedComponentsParameters.largestComponentOnly;
 int[] arrayOfInt1 = this.citNetwork.identifyConnectedComponents(this.currentSubnetwork.pubIndex,
paramConnectedComponentsParameters.largestComponentOnly, paramConnectedComponentsParameters.minComponentSize);
 int[] arrayOfInt3 = paramBoolean ? new int[this.pub.length] : this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
 connectedComponentsResults.nComponents = 0;
 connectedComponentsResults.nNonAssignedPubs = 0;
 connectedComponentsResults.nPubsLargestComponent = 0;
 byte b1;
 for (b1 = 0; b1 < i; b1++) {
  if (arrayOfInt1[b1] == 0) {
   connectedComponentsResults.nNonAssignedPubs++;
  } else if (arrayOfInt1[b1] == 1) {
   connectedComponentsResults.nPubsLargestComponent++;
  }
  if (arrayOfInt1[b1] > connectedComponentsResults.nComponents)
   connectedComponentsResults.nComponents = arrayOfInt1[b1];
  arrayOfInt3[this.currentSubnetwork.pubIndex[b1]] = 0;
 }
 int[] arrayOfInt2 = new int[connectedComponentsResults.nComponents + 1];
 int[] arrayOfInt4 = getIndicesNonEmptyGroups(arrayOfInt3);
 b1 = 1;
 byte b2 = 0;
 for (byte b3 = 1; b1 <= connectedComponentsResults.nComponents; b3++) {
  if (b2 < arrayOfInt4.length && arrayOfInt4[b2] == b3) {
   b2++;
  } else {
   arrayOfInt2[b1] = b3;
   b1++;
  }
 }
 connectedComponentsResults.groupIndexLargestComponent = arrayOfInt2[1];
 for (b1 = 0; b1 < i; b1++)
  arrayOfInt3[this.currentSubnetwork.pubIndex[b1]] = arrayOfInt2[arrayOfInt1[b1]];
 updateGroupIndices(arrayOfInt3);
 return connectedComponentsResults;
}

public ClusteringResults clusterPublications(ClusteringParameters paramClusteringParameters, boolean paramBoolean) {
 int i = this.currentSubnetwork.pubIndex.length;
 ClusteringResults clusteringResults = new ClusteringResults();
 int[] arrayOfInt1 = this.citNetwork.clusterNodes(this.currentSubnetwork.pubIndex, paramClusteringParameters.resolution,
paramClusteringParameters.minClusterSize, paramClusteringParameters.mergeSmallClusters, paramClusteringParameters.nRandomStarts,
paramClusteringParameters.nIterations, paramClusteringParameters.randomSeed);
 int[] arrayOfInt3 = paramBoolean ? new int[this.pub.length] : this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
 clusteringResults.nClusters = 0;
 clusteringResults.nNonAssignedPubs = 0;
 byte b1;
 for (b1 = 0; b1 < i; b1++) {
  if (arrayOfInt1[b1] == 0)
   clusteringResults.nNonAssignedPubs++;
  if (arrayOfInt1[b1] > clusteringResults.nClusters)
   clusteringResults.nClusters = arrayOfInt1[b1];
  arrayOfInt3[this.currentSubnetwork.pubIndex[b1]] = 0;
 }
 int[] arrayOfInt2 = new int[clusteringResults.nClusters + 1];
 int[] arrayOfInt4 = getIndicesNonEmptyGroups(arrayOfInt3);
 b1 = 1;
 byte b2 = 0;
 for (byte b3 = 1; b1 <= clusteringResults.nClusters; b3++) {
  if (b2 < arrayOfInt4.length && arrayOfInt4[b2] == b3) {
   b2++;
  } else {
   arrayOfInt2[b1] = b3;
   b1++;
  }
 }
 for (b1 = 0; b1 < i; b1++)
  arrayOfInt3[this.currentSubnetwork.pubIndex[b1]] = arrayOfInt2[arrayOfInt1[b1]];
 updateGroupIndices(arrayOfInt3);
 return clusteringResults;
```

```
}

public CoreResults identifyCorePublications(CoreParameters paramCoreParameters, int paramInt) {
 int i = this.currentSubnetwork.pubIndex.length;
 CoreResults coreResults = new CoreResults();
 boolean[] arrayOfBoolean = this.citNetwork.identifyCoreNodes(this.currentSubnetwork.pubIndex, paramCoreParameters.minNCits);
 int[] arrayOfInt = this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
 coreResults.nCorePubs = 0;
 for (byte b = 0; b < i; b++) {
  if (arrayOfBoolean[b]) {
   arrayOfInt[this.currentSubnetwork.pubIndex[b]] = paramInt;
   coreResults.nCorePubs++;
  }
 }
 updateGroupIndices(arrayOfInt);
 return coreResults;
}

public ShortestPathResults identifyShortestPath(int paramInt) {
 int i = this.currentSubnetwork.pubIndex.length;
 ShortestPathResults shortestPathResults = new ShortestPathResults();
 shortestPathResults.pathFound = false;
 byte b1;
 for (b1 = 0; b1 < i && !this.currentSubnetwork.pubMarked[b1]; b1++);
 if (b1 == i)
  return shortestPathResults;
 byte b2 = b1;
 while (++b1 < i && !this.currentSubnetwork.pubMarked[b1])
  b1++;
 if (b1 == i)
  return shortestPathResults;
 byte b3 = b1;
 while (++b1 < i && !this.currentSubnetwork.pubMarked[b1])
  b1++;
 if (b1 < i)
  return shortestPathResults;
 Path path = this.citNetwork.identifyShortestPath(this.currentSubnetwork.pubIndex, b3, b2);
 if (!path.pathFound)
  return shortestPathResults;
 int[] arrayOfInt = this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
 shortestPathResults.pathFound = true;
 shortestPathResults.length = path.length;
 shortestPathResults.nShortestPathPubs = 0;
 for (b1 = b2; b1 <= b3; b1++) {
  if (path.subgraphNodePath[b1]) {
   arrayOfInt[this.currentSubnetwork.pubIndex[b1]] = paramInt;
   shortestPathResults.nShortestPathPubs++;
  }
 }
 updateGroupIndices(arrayOfInt);
 return shortestPathResults;
}

public LongestPathResults identifyLongestPath(int paramInt) {
 int i = this.currentSubnetwork.pubIndex.length;
 LongestPathResults longestPathResults = new LongestPathResults();
 longestPathResults.pathFound = false;
 byte b1;
 for (b1 = 0; b1 < i && !this.currentSubnetwork.pubMarked[b1]; b1++);
 if (b1 == i)
  return longestPathResults;
 byte b2 = b1;
 while (++b1 < i && !this.currentSubnetwork.pubMarked[b1])
  b1++;
 if (b1 == i)
  return longestPathResults;
 byte b3 = b1;
 while (++b1 < i && !this.currentSubnetwork.pubMarked[b1])
  b1++;
 if (b1 < i)
  return longestPathResults;
 Path path = this.citNetwork.identifyLongestPath(this.currentSubnetwork.pubIndex, b3, b2);
 if (!path.pathFound)
  return longestPathResults;
 int[] arrayOfInt = this.groupIndexHistory[this.groupIndexHistoryCurrentState].clone();
 longestPathResults.pathFound = true;
```

```
  longestPathResults.length = path.length;
  longestPathResults.nLongestPathPubs = 0;
  for (b1 = b2; b1 <= b3; b1++) {
    if (path.subgraphNodePath[b1]) {
      arrayOfInt[this.currentSubnetwork.pubIndex[b1]]  = paramInt;
      longestPathResults.nLongestPathPubs++;
    }
  }
  updateGroupIndices(arrayOfInt);
  return longestPathResults;
}


private SubnetworkStatistics getSubnetworkStatistics(int[] paramArrayOfint)  {
  SubgraphStatistics subgraphStatistics = this.citNetwork.getSubgraphStatistics(paramArrayOfint);
  SubnetworkStatistics subnetworkStatistics = new  SubnetworkStatistics();
  subnetworkStatistics.nPubs = subgraphStatistics.nNodes;
  subnetworkStatistics.nCits = subgraphStatistics.nEdges;
  subnetworkStatistics.beginYear = subgraphStatistics.beginTime;
  subnetworkStatistics.endYear  = subgraphStatistics.endTime;
  subnetworkStatistics.nPubsPerYear = subgraphStatistics.nNodesPerTimeUnit;
  return subnetworkStatistics;
}

private int[] getIndicesNonEmptyGroups(int[] paramArrayOfint)  {
  int j = paramArrayOfint.length;
  int i = 0;
  byte b;
  for (b = 0; b < j; b++) {
    if (paramArrayOfint[b]  > i)
      i = paramArrayOfint[b];
  }
  boolean[] arrayOfBoolean  = new boolean[i + 1];
  i = 0;
  for (b = 0; b < j; b++) {
    if (paramArrayOfint[b]  > 0 && !arrayOfBoolean[paramArrayOfint[b]])  {
      arrayOfBoolean[paramArrayOfint[b]]  = true;
      i++;
    }
  }
  int[] arrayOfInt  = new int[i];
  i = 0;
  for (b = 0; b < arrayOfBoolean.length;  b++) {
    if (arrayOfBoolean[b])  {
      arrayOfInt[i]  = b;
      i++;
    }
  }
  return arrayOfInt;
}

private void updateGroupIndices(int[]  paramArrayOfint)  {
  this.groupIndexHistoryCurrentState = (this.groupIndexHistoryCurrentState  + 1) % 11;
  if (this.groupIndexHistoryOldestState == this.groupIndexHistoryCurrentState)
    this.groupIndexHistoryOldestState = (this.groupIndexHistoryOldestState + 1) % 11;
  this.groupIndexHistoryNewestState  = this.groupIndexHistoryCurrentState;
  this.groupIndexHistory[this.groupIndexHistoryCurrentState] = paramArrayOfint;
  if (this.currentSubnetwork.selectionParams.selectionMethod  == 2)
    updateSelectedSubnetwork();
}

private void initVisiblePublications() {
  int i = this.currentSubnetwork.pubIndex.length;
  int j = Math.min(this.visiblePubsParams.nVisiblePubs, i);
  this.currentSubnetwork.pubVisible = new boolean[i];
  if (!this.visiblePubsParams.proportionalGroupRepresentation) {
    double[] arrayOfDouble  = new double[i];
    byte b1;
    for (b1 = 0; b1 < i; b1++)
      arrayOfDouble[b1]  = (this.pub[this.currentSubnetwork.pubIndex[b1]]).citScore;
    Arrays.sort(arrayOfDouble);
    double d = arrayOfDouble[i  - j];
    for (b1 = 1; b1 < j && arrayOfDouble[i  - j  + b1] == d; b1++) ;
    byte b2 = 0;
    for (byte b3 = 0; b2 < j; b3++) {
      if ((this.pub[this.currentSubnetwork.pubIndex[b3]]).citScore > d) {
```

```
        this.currentSubnetwork.pubVisible[b3] = true;
        b2++;
       } else if ((this.pub[this.currentSubnetwork.pubIndex[b3]]).citScore  == d && b1 > 0) {
        this.currentSubnetwork.pubVisible[b3] = true;
        b1--;
        b2++;
       }
      }
     } else {
      int[] sigma = getNPublicationsPerGroup();
      int m = sigma.length;
      int[][] defaultCompulsion = new int[m][];
      int k;
      for (k = 0; k < m; k++) {
       defaultCompulsion[k] = new int[sigma[k]];
       sigma[k] = 0;
      }
      for (k = 0; k < i; k++) {
       int n = this.groupIndexHistory[this.groupIndexHistoryCurrentState][this.currentSubnetwork.pubIndex[k]];
       defaultCompulsion[n][sigma[n]] = k;
       sigma[n] = sigma[n] + 1;
      }
      int[] compulsionArray = new int[m];
      double[] arrayOfDouble  = new double[m];
      double d = i / j;
      k = 0;
      byte b;
      for (b = 0; b < m; b++) {
       compulsionArray[b] = (int) (sigma[b] / d);
       arrayOfDouble[b]  = sigma[b] - compulsionArray[b] * d;
       k += compulsionArray[b];
      }
      if (k < j) {
       double[] arrayOfDouble1  = arrayOfDouble.clone();
       Arrays.sort(arrayOfDouble1);
       double d1 = arrayOfDouble1[m  - j + k];
       for (b = 1; b < j - k && arrayOfDouble1[m  - j + k + b] == d1; b++) ;
       for (byte b1 = 0; k < j; b1++) {
        if (arrayOfDouble[b1]  > d1) {
         compulsionArray[b1] = compulsionArray[b1] + 1;
         k++;
        } else if (arrayOfDouble[b1]  == d1 && b > 0) {
         compulsionArray[b1] = compulsionArray[b1] + 1;
         b--;
         k++;
        }
       }
      }
      for (k = 0; k < m; k++) {
       if (compulsionArray[k] > 0) {
        double[] tempCompulsion = new double[sigma[k]];
        for (b = 0; b < sigma[k]; b++)
         tempCompulsion[b] = (this.pub[this.currentSubnetwork.pubIndex[defaultCompulsion[k][b]]]).citScore;
        Arrays.sort(tempCompulsion);
        double d1 = tempCompulsion[sigma[k] - compulsionArray[k]];
        for (b = 1; b < compulsionArray[k] && tempCompulsion[sigma[k] - compulsionArray[k] + b] == d1; b++) ;
        byte b1 = 0;
        for (byte b2 = 0; b1 < compulsionArray[k]; b2++) {
         if ((this.pub[this.currentSubnetwork.pubIndex[defaultCompulsion[k][b2]]]).citScore  > d1) {
          this.currentSubnetwork.pubVisible[defaultCompulsion[k][b2]] = true;
          b1++;
         } else if ((this.pub[this.currentSubnetwork.pubIndex[defaultCompulsion[k][b2]]]).citScore  == d1 && b > 0) {
          this.currentSubnetwork.pubVisible[defaultCompulsion[k][b2]] = true;
          b--;
          b1++;
         }
        }
       }
      }
     }
     clearVisiblePublicationsData();
    }

    private void clearVisiblePublicationsData() {
     this.currentSubnetwork.nLayersPerYear  = null;
     this.currentSubnetwork.visiblePub = null;
```

```
  this.currentSubnetwork.citRelationVisiblePubs = null;
  this.currentSubnetwork.citRelationVisiblePubsWeight = null;
 }

 private void updateVisiblePublicationsData() {
  int i = this.currentSubnetwork.pubIndex.length;
  if (this.currentSubnetwork.visiblePub != null)
   return;
  byte b1 = 0;
  byte b2;
  for (b2 = 0; b2 < i; b2++) {
   if (this.currentSubnetwork.pubVisible[b2])
    b1++;
  }
  byte b3 = b1;
  this.currentSubnetwork.visiblePub = new VisiblePublication[b3];
  b1 = 0;
  for (b2 = 0; b2 < i; b2++) {
   if (this.currentSubnetwork.pubVisible[b2]) {
    this.currentSubnetwork.visiblePub[b1] = new VisiblePublication();
    (this.currentSubnetwork.visiblePub[b1]).index = b2;
    b1++;
   }
  }
  this.currentSubnetwork.citRelationVisiblePubs = new CitationRelation[b3][];
  Connection[][] arrayOfConnection = this.citNetwork.getConnections(this.currentSubnetwork.pubIndex, this.currentSubnetwork.pubVisible);
  for (b1 = 0; b1 < b3; b1++) {
   this.currentSubnetwork.citRelationVisiblePubs[b1] = new CitationRelation[b1];
   for (b2 = 0; b2 < b1; b2++) {
    if (arrayOfConnection[b1][b2] != null) {
     this.currentSubnetwork.citRelationVisiblePubs[b1][b2] = new CitationRelation();
     (this.currentSubnetwork.citRelationVisiblePubs[b1][b2]).minNSteps = (arrayOfConnection[b1][b2]).minNSteps;
     (this.currentSubnetwork.citRelationVisiblePubs[b1][b2]).nIntermediatePubs = (arrayOfConnection[b1][b2]).nIntermediateNodes;
    }
   }
  }
  this.currentSubnetwork.citRelationVisiblePubsWeight = this.citNetwork.getConnectionWeights(this.currentSubnetwork.pubIndex,
this.currentSubnetwork.pubVisible);
  assignVisiblePublicationsToLayer();
  assignVisiblePublicationsToLocationWithinLayer();
 }

 private void assignVisiblePublicationsToLayer() {
  int n = this.currentSubnetwork.visiblePub.length;
  int i = this.currentSubnetwork.stats.beginYear;
  int j = this.currentSubnetwork.stats.endYear;
  this.currentSubnetwork.nLayersPerYear = new int[j - i + 1];
  byte b1 = 0;
  byte b2 = 0;
  int k = 0;
  for (int m = i; m <= j; m++) {
   while (b1 < n && (this.pub[this.currentSubnetwork.pubIndex[(this.currentSubnetwork.visiblePub[b1]).index]]).year == m)
    b1++;
   if (b1 == b2) {
    this.currentSubnetwork.nLayersPerYear[m - i] = 1;
    k++;
   } else {
    boolean[][] arrayOfBoolean = new boolean[b1 - b2][];
    int i1;
    for (i1 = 0; i1 < b1 - b2; i1++) {
     arrayOfBoolean[i1] = new boolean[i1];
     for (byte b3 = 0; b3 < i1; b3++)
      arrayOfBoolean[i1][b3] = (this.currentSubnetwork.citRelationVisiblePubs[b2 + i1][b2 + b3] != null);
    }
    int[] arrayOfInt = assignVisiblePublicationsToLayer(arrayOfBoolean);
    i1 = 0;
    for (byte b = 0; b < b1 - b2; b++) {
     (this.currentSubnetwork.visiblePub[b2 + b]).layer = arrayOfInt[b] + k;
     if (arrayOfInt[b] > i1)
      i1 = arrayOfInt[b];
    }
    this.currentSubnetwork.nLayersPerYear[m - i] = i1 + 1;
    k += i1 + 1;
    b2 = b1;
   }
  }
```

```
}

private int[] assignVisiblePublicationsToLayer(boolean[][] paramArrayOfboolean) {
  int j = paramArrayOfboolean.length;
  int[] arrayOfInt2 = new int[j];
  int i;
  for (i = j - 1; i >= 0; i--) {
    for (int k = i + 1; k < j; k++) {
      if (paramArrayOfboolean[k][i] && arrayOfInt2[k] + 1 > arrayOfInt2[i])
        arrayOfInt2[i] = arrayOfInt2[k] + 1;
    }
  }
  int[] arrayOfInt4 = new int[j];
  for (i = 0; i < j; i++) {
    for (byte b = 0; b < i; b++) {
      if (paramArrayOfboolean[i][b])
        arrayOfInt4[i] = arrayOfInt4[i] + 1;
    }
  }
  int[] arrayOfInt1 = new int[j];
  int[] arrayOfInt3 = new int[j];
  for (i = 0; i < j; i++) {
    byte b = -1;
    int k = -1;
    for (byte b1 = 0; b1 < j; b1++) {
      if (arrayOfInt4[b1] == 0 && arrayOfInt2[b1] > k) {
        b = b1;
        k = arrayOfInt2[b1];
      }
    }
    while (arrayOfInt3[arrayOfInt1[b]] >= this.layoutParams.maxNPubsPerLayer)
      arrayOfInt1[b] = arrayOfInt1[b] + 1;
    arrayOfInt3[arrayOfInt1[b]] = arrayOfInt3[arrayOfInt1[b]] + 1;
    arrayOfInt2[b] = -1;
    for (k = b + 1; k < j; k++) {
      if (paramArrayOfboolean[k][b]) {
        if (arrayOfInt1[b] + 1 > arrayOfInt1[k])
          arrayOfInt1[k] = arrayOfInt1[b] + 1;
        arrayOfInt4[k] = arrayOfInt4[k] - 1;
      }
    }
  }
  return arrayOfInt1;
}
```

Додаток В

Аналіз характеристик даних, що збираються з відібраних джерел науково – технічної інформації

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ6286_20Б

Аркушів 9

Київ 2020

АНОТАЦІЯ

Додаток містить опис застосунку для аналізу науково-технічної літератури. Додаток розроблено за допомогою мови програмування Java у Intellij IDEA за допомогою JDK та сторонніх бібліотек.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис застосунку для аналізу науково-технічної літератури, що відповідає вимогам поставлени в 1 розділі цієї роботи. Додаток Б містить фрагменти сирцевого коду одного з класів . Застосунок працює на базі Java Virtual Machine на будь-якому Java Runtime Environemnt. Систему розроблено в середовищі Intellij IDEA мовою Java з використанням JDK та інших сторонніх бібліотек.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Додаток надає можливості для імпортування на локальну машину та подальший аналіз мережі публікацій.

- Імпортування публікацій із стороннього агрегатору Core
- Збереження на локальну машину
- Завантаження файлу публікацій для аналізу
- Аналіз залежностей між публікаціями
- Побудова графу залежностей публікацій

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Застосунок є однокомпонентною системою, проте містить інтеграцію із зовнішним сервісом – постачальником публікацій Core. Основними сутностями якими оперує система це публікації, автори, цитати та графи. Користувач через застосунок співпрацює із API Core для імпорту публікацій та цитат. Застосунок за допомогою аналітичного модуля конвертує список цих публікацій та цитат до орієнтованого ациклічного графу та відображає користувачу цей граф разом із додатковою інформацією про кожну публікацію.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для повноцінного використання застосунку користувач повинен мати ПК із активним підключенням до мережі інтернет на базі будь якої оперативної системи, яка підтримує JVM. Можлива часткова робота застосунку в оффлайн режимі – при роботі із модулем аналізу залежностей цитат. Він не потребує комунікації із зовнішніми системами, а тому може експлуатуватись і без активного підключення.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для того аби встановити застосунок необхідно скопіювати JAR файл у будь-яке місце на диску, та запустити його через командну стрічку.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідними даними для застосунку є файл із публікаціями й цитатами для модулю аналізу графів. Для модулю імпорту із системи Core вхідними даними є користувацькі запити. Вихідними даними програми є візуальні інтерпретації графів залежностей наданої мережі публікацій.