Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування»

Виконав: студент 2-го курсу, групи 1ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Кожевніков В.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)
«15» грудня 2023 р.
Опонент: к.т.н., доц. каф. ОТ Богомолов С.В.
(прізвище та ініціали)
«15» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
(прізвище та ініціали)
«15» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.
«19» вересня 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кожевнікову Володимиру Сергійовичу

1. Тема роботи – Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування.

Керівник роботи: Романюк Оксана Володимирівна к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023р. №247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: Методи пошуку файлів – рекурсивний та індексний; методи пошуку дублікатів файлів – за хеш-сумою та за вмістом; середовища розробки – Intelij Idea 2023 та Microsoft SQL Server; мова розробки – Java; операційна система – Windows 11; вихідні дані – результати пошуку файлів.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз питання та постановка задач дослідження; розробка методів та алгоритмів для підвищення ефективності пошуку файлів та їх впорядкування; розробка програмного забезпечення для пошуку файлів та їх впорядкування; тестування застосунку для пошуку файлів та їх впорядкування; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета,

об'єкт та предмет дослідження; задачі дослідження; наукова новизна; практина цінність одержаних результатів; порівняльний аналіз аналогів; метод і блок-схема комплексного алгоритму індексування файлів; метод і блок-схема удосконаленого алгоритму пошуку дублікатів; структура графічного інтерфейсу користувача; тестування програмного забезпечення; результати експерментальних досліджень швидкості пошуку; економічна частина; апробація та публікація результатів роботи; фінальний слайд.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.В., к.т.н., доц. кафедри ПЗ	19.09.2023	01.12.2023
5	Причепа І.В., к.е.н., доц. кафедри ЕПВМ	27.11.2023	01.12.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви робіт	Строк виконання етапів роботи	Примітки
1	Аналіз стану питання, методів пошуку та індексування файлів, порівняння аналогів	20.09.2023 – 29.09.2023	вик
2	Розробка методів та алгоритмів комплексного індексування властивостей файлів та пошуку файлів з використанням міток	02.10.2023 – 13.10.2023	вик
3	Програмна реалізація застосунку	16.10.2023 – 10.11.2023	вик
4	Тестування програмного забезпечення та дослідження ефективності розроблених методів	13.11.2023 – 24.11.2023	вик
5	Економічна частина	27.11.2023 – 01.12.2023	вик

Студент _____ **Кожевніков В.С.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Романюк О.В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4

Кожевніков В.С. Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 94 с.

На укр. мові. Бібліогр.: 30 назв; рис.: 48; табл.: 10.

У магістерській кваліфікаційній роботі розроблено методи та алгоритми для підвищення ефективності пошуку файлів та їх впорядкування.

Удосконалено метод комплексного індексування властивостей та метаданих файлів. Удосконалено метод пошуку дублікатів файлів. Розроблено програмне забезпечення для пошуку файлів та їх впорядкування, проведено його тестування для перевірки працездатності розроблених методів. Досліджено швидкість пошуку файлів розробленим програмним застосунком та виконано порівняння з аналогом.

Для розробки використано мову програмування Java, середовище розробки JetBrains IntelliJ Idea, фреймворк Spring Boot для спрощення процесу розробки, налаштування і розгортання програм, а також JavaFX для розробки графічного інтерфейсу користувача.

. Розроблений застосунок для пошуку файлів та їх впорядкування, можна застосовувати у різних сферах, зокрема в галузях освіти, дослідження та науки, офісного менеджменту та для власних потреб.

Ключові слова: пошук файлів, пошук дублікатів, метод індексного пошуку, індексування метаданих, методи порівняння дублікатів.

ABSTRACT

UDC 004.4

Kozhevnikov V.S. Development of methods and software tools to increase the efficiency of file search and their ordering. Master's qualification work in specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 94 p.

In Ukrainian. Bibliogram.: 30 titles; Figure: 48; Table: 10.

In the master's qualification work, methods and algorithms have been developed to increase the efficiency of file search and their ordering.

The method of complex indexing of properties and metadata of files has been improved. Improved method of finding duplicate files. Software has been developed to search for files and organize them, it has been tested to verify the operability of the developed methods. The speed of searching for files by the developed software application was investigated and a comparison with an analogue was made.

The development uses the Java programming language, the JetBrains IntelliJ Idea development environment, the Spring Boot framework to simplify the development, configuration and deployment of applications, and JavaFX for developing a graphical user interface.

. The developed application for searching files and their ordering can be used in various fields, in particular in the fields of education, research and science, office management and for their own needs.

Keywords: file search, duplicate search, index search method, metadata indexing, duplicate comparison methods.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Покращення ефективності методів пошуку та впорядкування файлів є актуальним завданням у сучасному цифровому середовищі, де об'єм та різноманітність даних стрімко зростають. За статистикою, щоденно створюється 328,77 мільйонів терабайт даних [1], що становить 42 гігабайти на кожного жителя планети. З метою ефективного використання цих об'ємів даних, необхідні надійні засоби для їхнього пошуку та впорядкування.

Хоча всі сучасні операційні системи обладнані інструментами для роботи з файлами, розробники припускають, що користувачі будуть дотримуватись «чистоти» у своєму файловому сховищі. Однак у реальному житті часто зустрічаються непорозумілі назви папок та дублікати файлів. З часом, при рутинній роботі з файлами, може виникнути ситуація, коли важко згадати, де саме були збережені певні дані [2].

Вибір теми дослідження «Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування» можна обґрунтувати кількома важливими аспектами.

Бажання ефективно використовувати ресурси системи стає сильнішим, і оптимізація використання часу процесора та обсягу пам'яті стає необхідністю. В цьому контексті виявляється, що вдосконалені методи пошуку та систематизації файлів можуть виступити важливими союзниками у досягненні цієї мети.

З розвитком технологій виникає новий рівень можливостей у сфері програмування. Швидкість та доступність новацій дозволяють створювати інструменти, спрямовані на розробку більш продуктивних та інтуїтивно зрозумілих засобів для взаємодії з файловою системою.

Практичне застосування цих технологій виявляється в успішних проектах, які здатні спростити щоденні завдання широкого спектру користувачів. Це означає, що впровадження ефективних інструментів для роботи з файлами може мати вагому практичну користь для великої кількості людей.

Однак важливо підкреслити, що в питанні пошуку та систематизації файлів не існує ідеальних рішень. Необхідність подальшого вдосконалення та розвитку інструментів стає очевидною, оскільки існуючі рішення можуть не відповідати всім потребам різних користувачів. Таким чином, постійна прагнення до удосконалення в цьому напрямку залишається актуальним завданням. [3].

Обґрунтування вибору даної теми базується на реальних потребах сучасного суспільства, шляхах оптимізації роботи з інформацією та можливостях вдосконалення існуючих технологій у цій сфері.

Тому розробка нових методів та програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування є актуальною задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності пошуку та впорядкування файлів за рахунок розробки нових методів та програмних засобів.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

- дослідити предметну галузь використання програмного продукту;
- проаналізувати існуючі засоби та методи для управління файлами та їх властивості;
- розробити метод та алгоритм комплексного індексування властивостей файлів;
- розробити удосконалений метод та алгоритм пошуку дублікатів файлів;
- виконати проектування структури програмного забезпечення, налаштувати сервіс управління БД;
- розробити графічний інтерфейс застосунку;
- реалізувати програмне забезпечення згідно запроектованих вимог;
- провести тестування розробленого програмного забезпечення;
- дослідити ефективність розроблених методів.

Об’єктом дослідження є процес пошуку і впорядкування файлів.

Предмет дослідження – методи та засоби підвищення ефективності пошуку та впорядкування файлів.

Методи дослідження. У процесі дослідження використовувались: теорія алгоритмів для розробки алгоритмів і програмних модулів; теорія баз даних для розробки структури бази даних; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна одержаних результатів

1. Удосконалено метод комплексного індексування властивостей та метаданих файлів, який відрізняється від відомих тим, що при індексуванні властивостей файлів створюються пошукові індекси та індекси міток на основі метаданих файлів, що дало можливість зменшити час пошуку та надати користувачу підказку, з якими даними він має справу.

2. Удосконалено метод пошуку дублікатів файлів, який відрізняється від відомих тим, що для пошуку файлів використовується значення хеш-суми, а для порівняння – метод побітового порівняння вмісту, що дало можливість суттєво зменшити час пошуку та підвищити точність знаходження дублікатів файлів.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі, опублікованій у співавторстві, автору належать такі результати: підвищення ефективності методів пошуку файлів та їх упорядкування [4], дослідження ефективності методів пошуку файлів [5].

Практичне значення одержаних результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для підвищення ефективності пошуку та впорядкування файлів.

Апробація результатів роботи. Результати роботи доповідалися та обговорювалися на:

– Міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 20-

21 листопада 2023 р);

– XI Міжнародній науково-практичній конференції «EUROPEAN SCIENTIFIC CONGRESS» (Мадрид, 27-29 листопада 2023р).

Публікації. Основні результати досліджень опубліковано в 2 наукових працях, у матеріалах конференцій.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання

На сьогоднішній день існують різні методи і програмні засобів для підвищення ефективності пошуку файлів та їх впорядкування. Розглянемо деякі ключові аспекти та тенденції в цьому напрямі:

1. Вбудовані засоби операційних систем – більшість сучасних операційних систем, таких як Windows, macOS та різні дистрибутиви Linux, мають вбудовані інструменти для пошуку та впорядкування файлів. Однак ці інструменти можуть бути обмеженими за функціональністю та інтерфейсом.

2. Покращені алгоритми пошуку – розроблено та вдосконалено різні алгоритми пошуку, такі як алгоритми індексації та алгоритми штучного інтелекту для вдосконалення швидкості та точності пошуку файлів.

3. Використання індексації – багато програм використовують техніку індексації для створення швидкого та ефективного каталогу файлів, що полегшує швидкий доступ до них.

4. Методи класифікації та міток – з'явилися програми, які використовують методи класифікації та проставлення міток для кращого впорядкування файлів за категоріями та властивостями.

4. Cloud-based рішення – зростає популярність хмарних рішень для зберігання та управління файлами, які зазвичай мають вдосконалені можливості пошуку та організації.

5. Додатки для автоматизації – деякі програми спрямовані на автоматизацію процесів пошуку та впорядкування, забезпечуючи користувачам зручні та ефективні інструменти.

6. Сучасні тенденції у використанні штучного інтелекту – деякі розробники впроваджують штучний інтелект для аналізу патернів користувацької поведінки та покращення рекомендацій для організації файлів.

В цілому, стан методів і програмних засобів для підвищення ефективності

пошуку файлів постійно розвивається, використовує нові технологій та підходи для вирішення зростаючих вимог користувачів.

Для організації комфортної та ефективної роботи з файлами система має надавати засоби для:

- пошуку файлів за назвою, ключовим словом, розширенням, типом;
- пошук за метаданими (дата, місце створення);
- пошук дублікатів файлів та очищення;
- індексація характеристик файлів для швидкого пошуку та агрегації; підтримка актуальності індексів;
- підтримка використання тегів;
- операції переміщення та зберігання файлів за заданими властивостями;
- робота з хмарними сервісами для зберігання резервних копій.

При розробці методів покращення ефективності пошуку та впорядкування файлів буде вдосконалено швидкість пошуку файлів та покращена ефективність побудови пошукового запиту за рахунок використання індексування метаданих та механізму створення міток.

1.2 Порівняння з аналогами

Існує багато різних програмних засобів для пошуку та впорядкування файлів. Але більшість з них мають вузько-спеціалізовану функціональність або пропонують додаткові функції за оплату. Розглянемо декілька популярних засобів для пошуку та впорядкування файлів.

Windows Explorer – це вбудований файловий менеджер у операційній системі Windows, який має кілька властивостей для пошуку файлів та фільтрації результатів. На рисунку 1.1 зображено інтерфейс файлового менеджеру Windows Explorer. Ось деякі з найбільш корисних пошукових властивостей в Windows Explorer [6]:

- пошук за іменем файлу або папки;
- розширений пошук, можна встановити різні критерії пошуку, такі як тип файлу, дата створення, ключові слова тощо;

– фільтрація результатів – після виконання пошуку, можна використовувати фільтри для подальшої фільтрації результатів. Наприклад, можна відфільтрувати файли за типом або датою;
– індексований пошук (при використанні служби Windows Search);
– пошук за метаданими, можна використовувати ключові слова та інші метадані для пошуку файлів. Наприклад, можна ввести «author:John» або «type:pdf» у поле пошуку;
– пошук у папці та її піддиректоріях;
– пошук за розміром файлу;
– збережені пошукові запити, можна зберегти свої пошукові запити для подальшого використання.

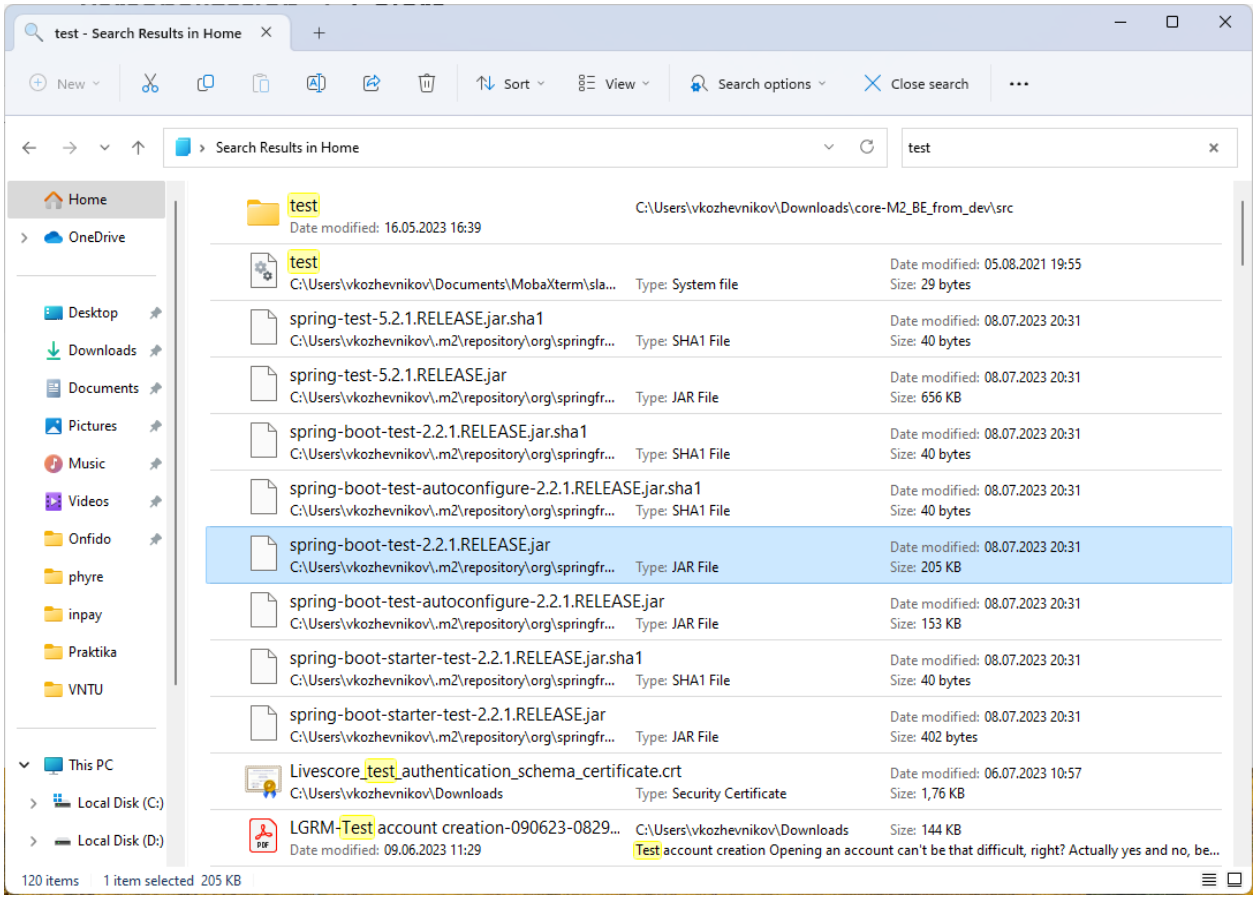

Рисунок 1.1 – Інтерфейс Windows Explorer

Перевагами Windows Explorer є:

1. Інтеграція з операційною системою.

2. Простий та зрозумілий інтерфейс.

До недоліків можна віднести:

1. Може бути повільним при роботі з великою кількістю файлів.

2. Відсутність підтримки тегів.

3. Відсутність автоматизації.

«Everything» – стороння програма для пошуку файлів, яка відома своєю надзвичайною швидкістю та точністю. Основна особливість «Everything» полягає в тому, що вона працює на основі індексу файлів і дисків, що робить пошук надзвичайно швидким. На рисунку 1.2 зображено інтерфейс програми «Everything».

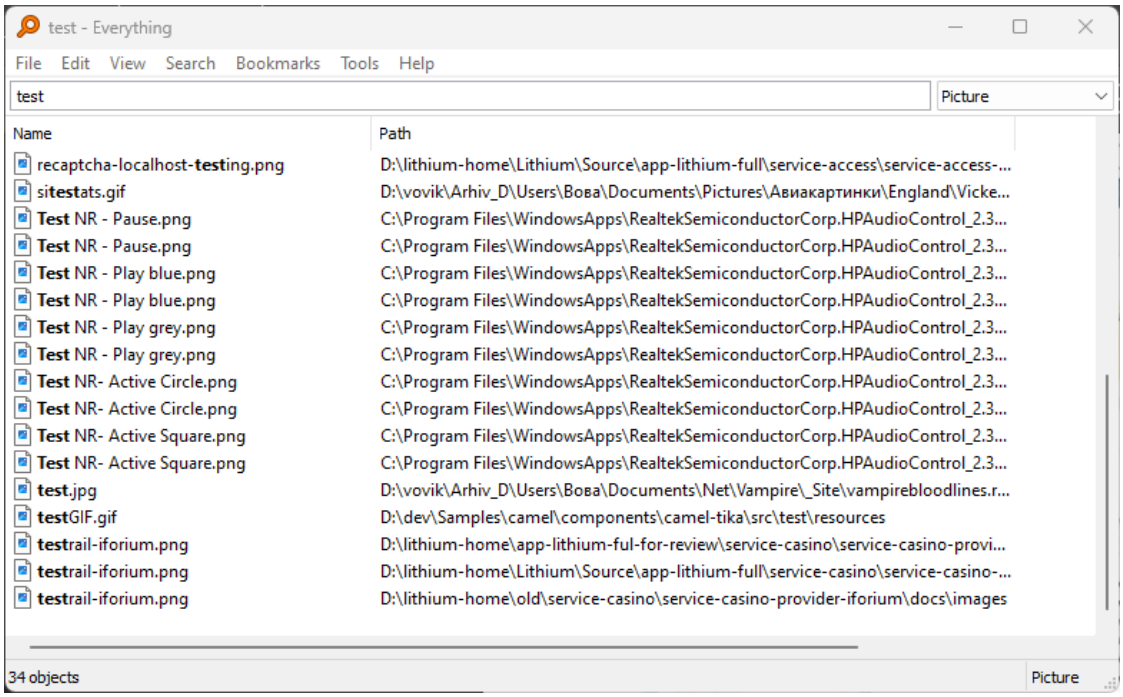

Рисунок 1.2 – Інтерфейс програми «Everything»

Ось деякі з важливих пошукових властивостей програми «Everything»:

- швидкий індексований пошук;
- деталізовані результати;
- регулярні вирази;
- швидкий доступ до папок;
- фільтрація результатів;

– збережені пошукові запити;

– пошук в режимі реального часу.

Перевагами «Everything» є:

1. Надзвичайна швидкість пошуку

2. Простий інтерфейс

До недоліків «Everything» можна віднести:

1. Обмежена функціональність

2. Недоступність на інших платформах

3. Відсутність функцій організації файлів

Застосунок «TagSpaces» – програмне забезпечення для організації і каталогізації файлів на вашому комп'ютері [7]. Основною ідеєю «TagSpaces» є використання тегів для класифікації і організації файлів. Частина функцій активується після покупки Pro версії програми. На рисунку 1.3 наведено інтерфейс застосунку «TagSpaces».

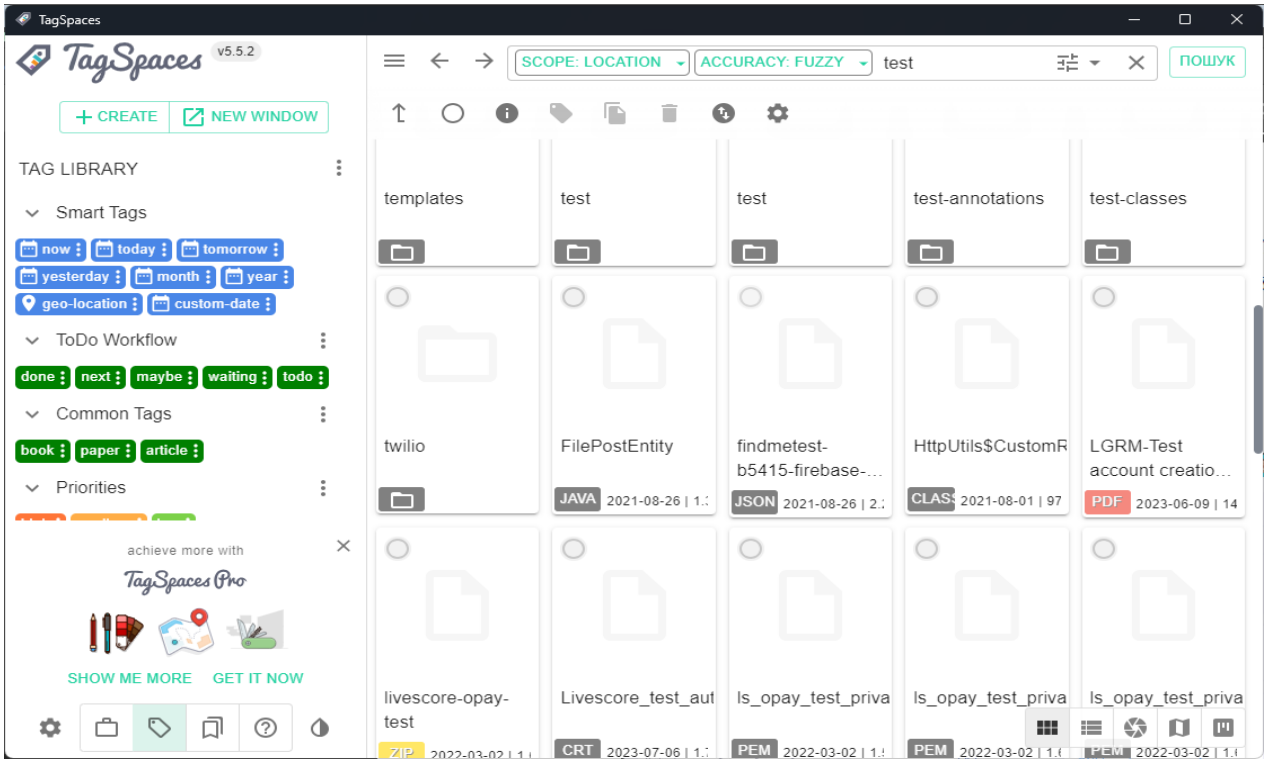

Рисунок 1.3 – Інтерфейс програми «TagSpaces»

Нижче наведено деякі з важливих властивостей «TagSpaces»:

– керування файлами з використанням тегів, можна створювати теги за

своїм власним вибором та призначати їх файлам;

- підтримка різних типів файлів;
- можливість працювати оф лайн;
- шифрування файлів;
- підтримка різних платформ – підтримуються Windows, macOS, Linux і Android;
- пошук та фільтрація файлів;
- редактор тексту;
- робота з хмарними сховищами.

До переваг «TagSpaces» відносять:

1. Можливість використовувати теги для каталогізації і організації файлів
2. Можливість роботи оф лайн
3. Підтримка різних платформ

Недоліками «TagSpaces» є:

1. Багато корисних функцій доступні в платній версії
2. Може вимагати певного часу на вивчення

Проаналізувавши аналоги, врахуємо переваги і недоліки аналогів при розробці власного застосунку. Перелік характеристик зображено в таблиці 1.1.

Таблиця 1.1 – Порівняння характеристик програмних продуктів

Критерій	Windows Explorer	Everything	TagSpaces	Власний застосунок
Зручний інтерфейс	3	3	4	5
Вартість продукту	5	4	2	4
Робота з тегами	1	0	5	4
Швидкість пошуку	3	5	3	5
Підтримка різних платформ	1	1	5	5
Сума	13	13	19	23

З результатів проведеного порівняння можна побачити характеристики, які треба реалізувати в одному продукті. Це оптимальні рішення для пошуку та впорядкування даних; створення інтерфейсів керування зрозумілих для звичайних користувачів; підтримки розширення операцій з даними, які в свою чергу будуть задовольняти поставлені вимоги та враховувати обмеження.

1.3 Аналіз методів пошуку файлів

Структура файлового сховища ПК зазвичай має вигляд незбалансованого дерева [8]. Незбалансовані дерева у контексті файлового сховища можуть виникнути, коли вставка чи видалення файлів не керується алгоритмом балансування, і це може привести до нерівномірного розподілу елементів та погіршення часу пошуку [9]. На рисунку 1.4 зображено простий приклад незбалансованого бінарного дерева. Глибина папок D11, D12, D13 та D14 – різна, що є характерним для незбалансованого дерева. При пошуку елементів у збалансованому дереві – час пошуку наближається до O(log n), де n кількість елементів у сховищі. Чим більше структура розташування файлів буде наближатись до лінійної, тим ймовірніше що час пошуку збільшиться до O(n).

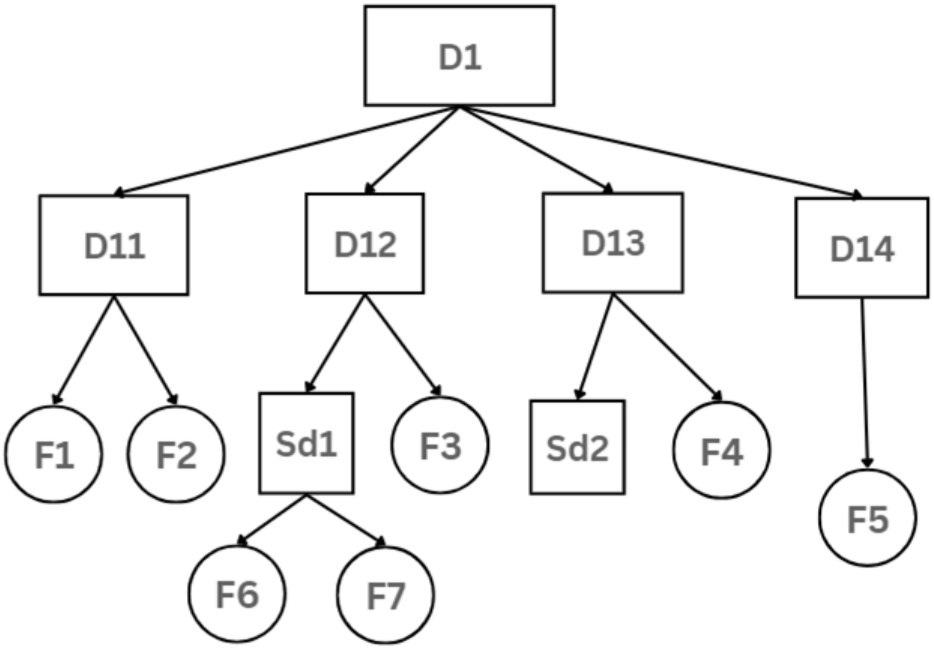

Рисунок 1.4 – Схема структури файлового сховища ПК

Для пошуку файлів зазвичай застосовується рекурсивний метод пошуку.

Рекурсивний метод пошуку – це метод, за яким система або програма починає пошук у заданій директорії, а потім рекурсивно переходить до всіх піддиректорій в цій директорії та їхніх піддиректоріях і так далі, до знаходження всіх файлів, що відповідають певним критеріям пошуку. Цей метод дозволяє знайти всі файли, включаючи ті, які знаходяться у вкладених папках.

Це ефективний метод пошуку, але при великій кількості файлів та вкладених папок може бути повільним. Кожен виклик функції зумовлює витрату часу на виконання та повернення. Для збільшення ефективності пошуку застосовують індексний метод пошуку.

Індексний метод пошуку – це метод, при якому під час пошуку файлів створюється індекс або база даних, яка містить інформацію про місцезнаходження та інші атрибути файлів. Індекс дозволяє значно прискорити процес пошуку, оскільки не потрібно прочитувати або аналізувати кожен файл безпосередньо. Замість цього, можна шукати файли за допомогою індексу, який зберігає важливу інформацію про файли та їх властивості.

Для підтримання індексу в актуальному стані можна застосовувати заплановані задачі для сканування файлів на зміни, а також слухачі подій змін файлів.

1.4 Аналіз методів індексування файлів

Індексування властивостей файлів – це процес створення пошукового індексу, який дозволяє швидко та ефективно знаходити файли на основі їхніх властивостей. Це може бути корисним для пошуку, фільтрації та організації файлів на комп'ютері. На рисунку 1.5 зображено схему застосування індексів для пошуку файлів.

Існує кілька методів індексування властивостей файлів.

Індексування метаданих файлів – цей метод включає збереження метаданих про файли, таких як імена файлів, розміри, дати створення та модифікації, розширення файлів тощо. Ці метадані зазвичай зберігаються в базі

даних або індексі, який дозволяє швидко виконувати пошук за цими властивостями.

Повнотекстовий пошук – метод який використовується для індексування текстового вмісту файлів. Він дозволяє шукати файли за ключовими словами, фразами чи іншими текстовими ознаками вмісту файлів.

Метод індексування міток і тегів – метод коли користувачі можуть призначати файлам мітки, теги або категорії, які вони вважають важливими. Ці мітки можуть бути індексовані і використовуватися для швидкого пошуку та організації файлів за категоріями.

Індексування географічних координат – метод який використовується для індексування географічних координат файлів, таких як місце зйомки фотографій. Це дозволяє користувачам шукати файли за місцем розташування.

Метод індексування категорій та категоризація – файли можуть бути індексовані та поділені на категорії на основі їхнього призначення, наприклад, документи, фотографії, музика, відео тощо. Це допомагає користувачам швидко знаходити файли у визначених категоріях.

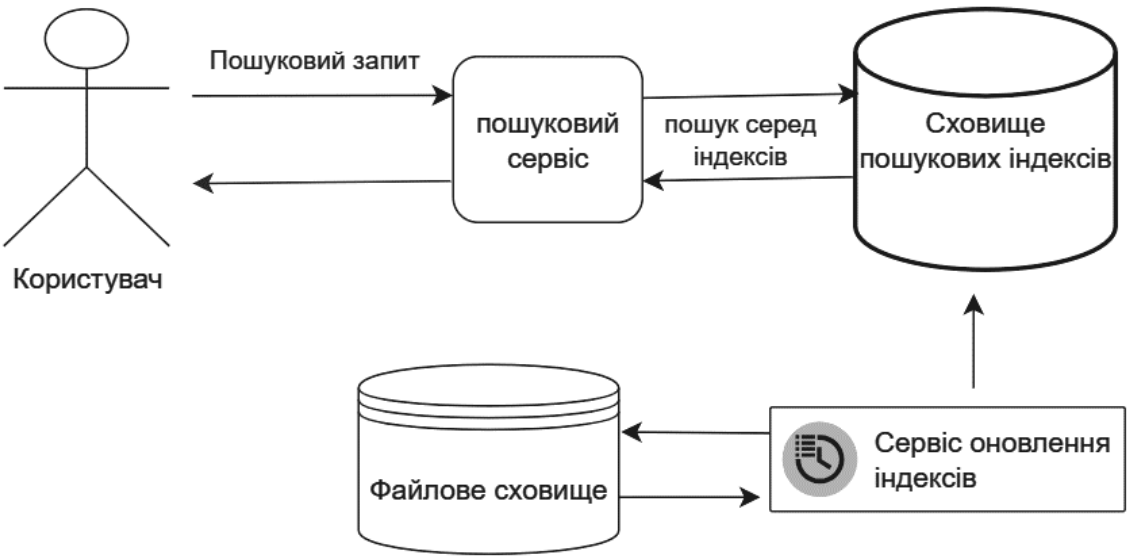

Рисунок 1.5 – Схема застосування індексів для пошуку файлів

Метод індексування за датами – файли можуть бути індексовані за датами створення, модифікації, останнього доступу тощо. Це дозволяє користувачам

шукати файли за конкретними діапазонами часу.

Процес індексованого пошуку файлів складається з декількох субпроцесів. Розглянемо етапи індексованого пошуку файлів:

1. Індексування – першим етапом є створення індексу, який включає в себе інформацію про властивості файлів. Для текстових файлів це може бути список слів і їхніх місць в кожному файлі; для мультимедійних файлів - важливі характеристики, такі як колір, об'єкти, обличчя тощо. Система або програма, яка виконує індексування, переглядає кожен файл у системі та створює структуровану базу даних або індекс, яка прискорює подальший пошук.

2. Пошуковий запит – коли користувач вводить пошуковий запит (ключові слова чи фрази), пошукова система або програма використовує індекс для швидкого визначення файлів, які містять введені критерії. Деякі системи також можуть використовувати додаткові алгоритми або фільтри для поліпшення точності результатів.

3. Представлення результатів – знайдені файли виводяться у вигляді списку або іншого формату користувачу, зазвичай для цього також використовується інформація з індексів. Іноді результати можуть бути відсортовані за релевантністю, датою чи іншими критеріями.

4. Навігація, вибір файлу та операції з файлами – останній етап, в якому для виконання вказаних операцій найчастіше використовуються вбудовані засоби операційних систем, адже це базові функції і вони присутні у кожній операційній системі.

Хоча індексований пошук файлів допомагає покращити швидкість та ефективність пошуку, він також може мати свої недоліки та обмеження. Ось основні з них:

- витрата ресурсів при індексуванні – процес індексування може вимагати значних обчислювальних ресурсів та часу, особливо при великому обсязі даних;
- неактуальність індексу – індекс може стати неактуальним у випадку, якщо дані змінюються часто. Якщо файли додаються, вилучаються чи

змінюються, індекс може не відображати поточний стан системи;

- потреба в обслуговуванні індексу – системи індексованого пошуку потребують обслуговування для підтримки актуальності та ефективності індексу, що може включати в себе регулярні оновлення та перебудови індексу, що може призводити до додаткових навантажень;
- обмеження по обсягу пам'яті – великі обсяги даних можуть призвести до значних обсягів індексу, що може вимагати значних обсягів пам'яті;
- залежність від точних ключових слів – індексований пошук може бути менш ефективним, якщо користувач використовує неточні чи неправильні ключові слова. Система може не знайти файл, якщо він не був індексований відповідним чином.

Індексований пошук дозволяє значно прискорити процес пошуку файлів, особливо в великих обсягах даних. Наразі багато програмного забезпечення використовує цю технологію [10]. Замість того, щоб сканувати кожен файл при кожному запиті, система використовує індекс для швидкого визначення релевантних результатів. Це робить процес пошуку більш ефективним і зручним для користувача, але варто враховувати його обмеження та недоліки при використанні, особливо у контексті конкретних вимог та умов.

1.5 Аналіз методів пошуку дублікатів файлів

Пошук дублікатів файлів може бути корисним для оптимізації використання дискового простору та покращення організації даних. Існують кілька методів для виявлення та управління дублікатами файлів, і їх ефективність залежить від конкретних потреб і умов користувача [11].

Хеш-порівняння – пошук дублікатів за допомогою порівняння хеш-сум є досить швидким та ефективним методом. Основна ідея полягає в тому, щоб порівняти хеш-суми різних файлів, і якщо хеш-суми однакові, це може вказувати на ідентичність файлів або, принаймні, ймовірність того, що файли ідентичні. Хоча порівняння хеш-сум є ефективним методом, слід враховувати, що деякі хеш-функції можуть мати колізії – ситуації, коли різні вхідні дані дають

однаковий хеш. Тому важливо вибрати надійну хеш-функцію для забезпечення точності виявлення дублікатів або виконати додаткову перевірку, порівнюючи їх фактичний вміст.

Пошук дублікатів за допомогою порівняння властивостей файлів використовує інформацію про метадані файлів для визначення їхньої схожості. Цей метод може бути швидким і ефективним, адже порівняння метаданих файлів зазвичай вимагає менше ресурсів, ніж порівняння хеш-сум або фактичного вмісту файлів. Однак важливо враховувати, що схожі властивості не завжди означають ідентичність файлів, тому може бути необхідно використовувати додаткові методи для підтвердження результатів.

Порівняння за вмістом файлів — це метод, при якому фактичний вміст кожного файлу аналізується для визначення його схожості з іншими файлами. Цей метод є більш ресурсозатратним порівняно з порівнянням за метаданими або хеш-сумами, оскільки вимагає читання великої кількості даних. Проте він є дуже ефективним для виявлення навіть найменших різниць у вмісті файлів і дозволяє виявити справжні дублікати, незалежно від їхньої назви, розміру чи інших характеристик.

Вибір конкретного методу залежить від конкретних вимог користувача, обсягу даних та доступних ресурсів. Комбінування різних методів часто використовується для досягнення оптимального виявлення дублікатів файлів.

1.6 Постановка задач розробки

Після аналізу поточного стану питання та порівняння існуючих рішень за визначеними критеріями визначено завдання, які необхідно виконати, для розробки програмного забезпечення:

- дослідити предметну галузь використання програмного продукту;
- проаналізувати існуючі засоби та методи для управління файлами та їх властивості;
- розробити метод та алгоритм комплексного індексування властивостей файлів;

- розробити удосконалений метод та алгоритм пошуку дублікатів файлів;
- виконати проектування структури програмного забезпечення, налаштувати сервіс управління БД;
- розробити графічний інтерфейс застосунку;
- реалізувати програмне забезпечення згідно запроектованих вимог;
- провести тестування розробленого програмного забезпечення;
- дослідити ефективність розроблених методів.

Технічне завдання наведено у Додатку А.

1.7 Висновки

У першому розділі було розглянуто питання пошуку та впорядкування файлів. Проаналізовано популярні існуючі засоби, а саме Windows Explorer, Everything та TagSpaces. Розглянуто переваги та недоліки цих засобів, виконано порівняння характеристик. Визначено аспекти, які можна покращити, зокрема збільшити швидкість пошуку та додати механізм створення міток, які будуть використовуватись як підказки при побудові пошукового запиту, так і для оптимізації швидкості пошуку.

Проаналізовано існуючі методи пошуку та індексування файлів. Досліджено основні методи пошуку дублікатів файлів. У результаті аналізу було обрано напрями удосконалення методів для реалізації необхідної функціональності. Зокрема, було вирішено вдосконалити метод індексування властивостей та метаданих файлів, щоб при індексуванні властивостей файлів створювались пошукові індекси та індекси міток на основі метаданих файлів. Також було вирішено покращити метод пошуку дублікатів, використовуючи кілька методів порівняння – метод порівняння хеш-суми та метод порівняння вмісту.

Проведено постановку задач розробки.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПОШУКУ ФАЙЛІВ ТА ЇХ ВПОРЯДКУВАННЯ

2.1 Розробка комплексного методу індексування властивостей файлів

Розробка методу комплексного індексування властивостей файлів є важливою задачею для організації та управління великими обсягами даних. Цей метод допомагає швидко та ефективно знаходити, фільтрувати та аналізувати файли. На рис. 2.1 наведено схему пошуку файлів за допомогою міток та індексів. Для розробки методу комплексного індексування властивостей файлів необхідно виконати наступні кроки:

- визначити які властивості файлів будуть індексуватися;
- налаштувати БД для зберігання індексів;
- виконати індексування файлів та заповнити відповідні індекси;
- розробити механізм для автоматичної актуалізації індексів.

Визначимо, які властивості файлів нам потрібно індексувати. Це будуть такі дані, як назва файлу, розмір, дата створення, дата останньої зміни, тип файлу, автор, гео-теги. Також необхідно обрахувати і зберігати md5 суму, значення якої буде використовуватись для синхронізації змін. Також може використовуватись для пошуку дублікатів.

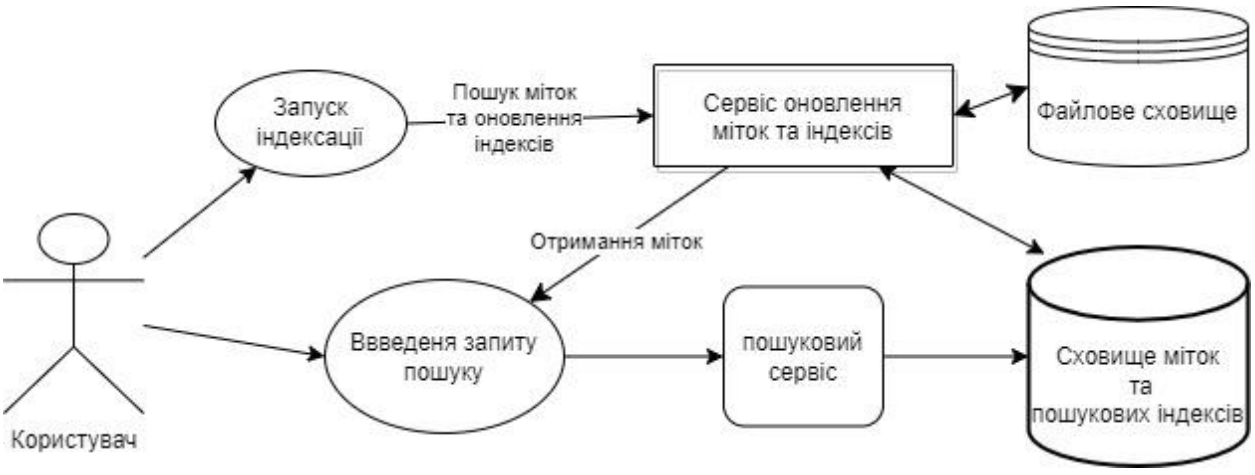

Рисунок 2.1 – Схема методу пошуку файлів за допомогою міток та індексів

Маючи перелік властивостей для індексування необхідно створити структури для зберігання проіндексованих даних. Для цього будемо використовувати реляційну базу даних. На рівні БД налаштуємо структуру таблиць, зв’язки між ними та індекси для швидкого пошуку.

Для виконання індексування необхідно знайти всі вкладені файли та папки для заданого розташування. Використаємо метод рекурсивного пошуку. Властивості знайдених об’єктів опрацюємо і збережемо в БД. Після завершення пошуку файлів та їх індексування – необхідно виконати операції виявлення видалених файлів і відповідно відобразити це в індексах.

Актуальність даних можна підтримувати ручним запуском індексування, налаштуванням запланованої задачі або створенням інтерфейсу для точкової актуалізації зовнішніми слухачами подій.

Розглянемо детально запуск процесу індексування метаданих та пошуку файлів користувачем.

Коли користувач запускає індексування почне виконуватись рекурсивний пошук всіх файлів у файловому сховищі, від початкової заданої папки – вглиб, аналізуючи кожен файл. При аналізі кожного файлу сервіс оновлення міток та індексів вичитує визначені для пошуку властивості і створює в сховищі міток та пошукових індексів, яким виступає БД, відповідний файловий індекс. В якості унікального ключа використовується ім’я та абсолютний шлях його розташування. Якщо запис вже існує у БД, то значення в БД оновлюється поточними. Також визначається перелік міток, які пов’язані з поточним файлом, і для файлу створюється відповідне посилання на кожну мітку. Якщо сховище міток не містить якусь з визначених міток, то спочатку значення мітки записується у БД, а потім створюється для файлу посилання на цю мітку. При індексуванні файлу, сервіс оновлення міток та індексів обов’язково оновлює в БД час індексування останнього індексування. Це значення необхідне щоб легко визначити які файли були видалені у файловому сховищі. Після того, як пошук у файловому сховищі завершився, буде виконуватись алгоритм для визначення видалених файлів. Для цього в БД виконують пошук індексів, час індексування

яких був раніше за час поточного індексування. Знайдені індекси видаляємо.

Пошук файлів починається з створення запиту. Сервіс оновлення міток та індексів надає користувачу список доступних міток. Якщо користувач обирає одну або декілька міток, то пошуковий сервіс звертається до сховища міток та індексів і отримує список файлових індексів, які відносяться до визначених міток. Разом з мітками користувач може ввести фрагмент імені файлу, значення якого буде враховуватись під час пошуку і отримання файлових індексів з сховища міток та індексів.

Операції пошуку будуть виконуватись швидко, завдяки тому, що пошуковий сервіс спілкується тільки з сховищем міток та індексів і не робить жодних запитів до файлового сховища.

2.2 Розробка удосконаленого методу пошуку дублікатів файлів

Розробка удосконаленого методу пошуку дублікатів файлів є важливою задачею для швидкого пошуку дублікатів файлів. Щоб зробити пошук швидким, але зберегти якість – розробимо удосконалений метод пошуку дублікатів, який буде поєднувати декілька методів порівняння файлів – метод хеш-порівняння та метод порівняння за вмістом. Для збільшення швидкості будемо використовувати індексний пошук – створимо індекс для хеш-сум.

Виберемо алгоритм обрахунку хеш-суми. Найпоширенішими алгоритмами хешування є MD5, SHA-1, SHA-256 та SHA-512. MD5: Вважається швидким, але не використовується для криптографічних цілей через вразливості. SHA-1: Також вважається швидким, але вже застарілим для криптографії. SHA-256 і SHA-512: Зазвичай повільніші, але більш безпечні, і використовуються для криптографічних застосувань.

В даному випадку важлива швидкість, а безпека не є критичною, отже MD5 буде хорошим вибором. Розглянемо алгоритм обрахунку хеш-суми MD5.

MD5 – це хеш-алгоритм, розроблений професором Ronald L. Rivest з MIT в 1991 році. Призначений для створення контрольних сум або «відбитків» повідомлення довільної довжини і подальшої перевірки їх достовірності.

Алгоритм MD5 заснований на алгоритмі MD4.

Хеш, отриманий з функції, заснованої на цьому алгоритмі, виробляє рядок з 16 байт. Ця лінія містить 16 шістнадцяткових чисел. У цьому випадку зміна хоча б одного з його символів призведе до подальшої незворотної зміни значень всіх інших бітів рядка. Алгоритм передбачає 5 кроків, а саме: вирівнювання потоку; додавання довжини повідомлення; ініціалізація буфера; обчислення у циклі; результат обрахунку. На рисунку 2.2 зображено схему логіки виконання алгоритму MD5.

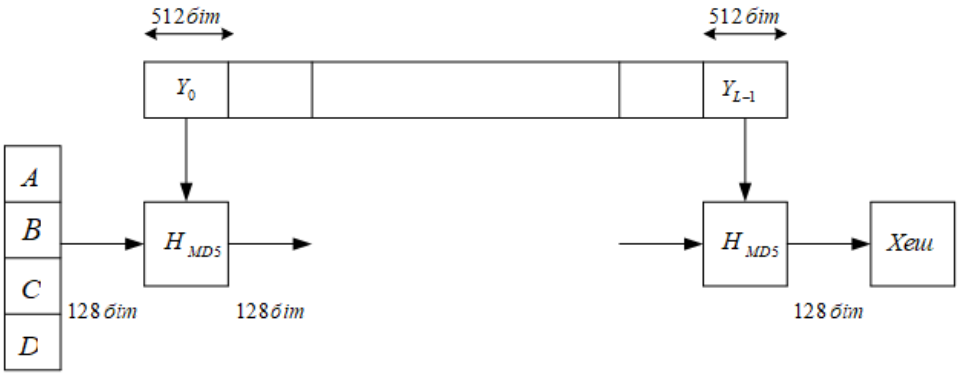

Рисунок 2.2 – Схема логіки виконання алгоритму MD5

На першому кроці «Вирівнювання потоку», одиничний біт спочатку додається до кінця потоку, а потім необхідна кількість нульових бітів. Вхідні дані вирівнюються таким чином, щоб їх новий розмір можна порівняти з 448 за модулем 512. Вирівнювання відбувається навіть якщо довжина вже порівнянна з 448.

На другому кроці, в 64 біти, що залишились, дописують 64-бітне подання довжини даних. Спочатку записують нижні 4 байти. Якщо довжина перевищує 2^{64} – 1, дописуються лише найменші біти. Після цього довжина потоку стає кратною 512. Розрахунки будуть засновані на поданні цього потоку даних як 512-бітного масиву слів.

На третьому кроці використовуються чотири змінних розміром 32 біт і встановлюються початкові значення в 16 біт. Ці змінні будуть зберігати результати проміжних обчислень. Для збереження проміжних і кінцевих

результатів хеш-функції використовується 128-розрядний буфер. Він представляється як чотири 32 – розрядні регістра (*A, B, C, D*). В якості ініціалізуючих значень використовуються наступні шістнадцяткові числа: *A = 01234567, B = 89ABCDEF, C = FEDCBA98, D = 76543210.*

На рисунку 2.3 зображено Схему обробки чергового 512-розрядного блоку.

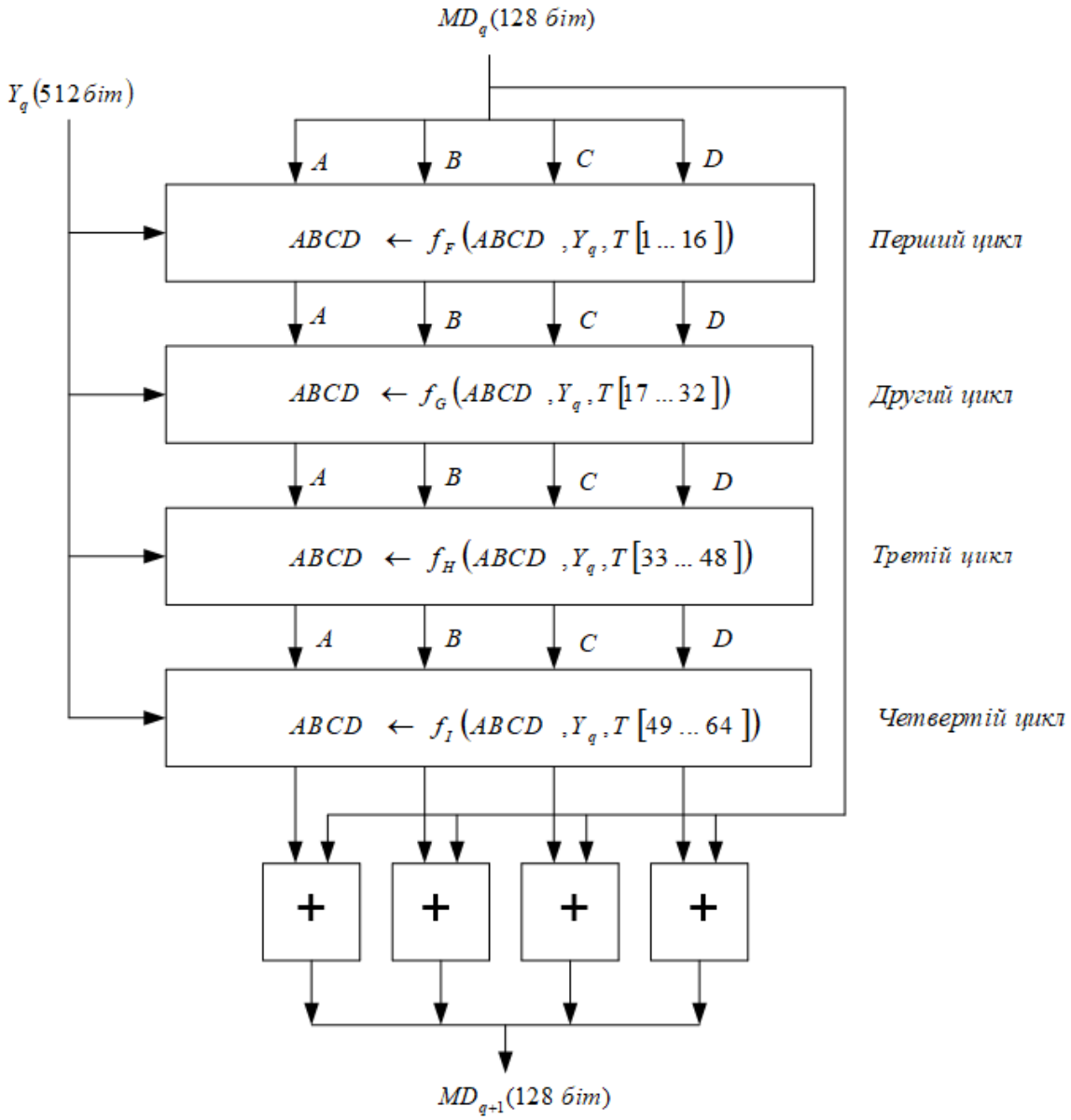

Рисунок 2.3 – Схема обробка чергового 512-розрядного блоку

Під час 4-го кроку «Обчислення в циклі» відбуваються 4 цикли, в яких зберігаються значення, що залишилися після операцій над попередніми блоками. Після всіх операцій підводяться підсумки останніх двох циклів. Циклів в MD5

стали 4 замість 3 в MD4. Було додано нову константу, щоб мінімізувати вплив вхідного повідомлення. Кожен цикл має різну константу на кожному кроці. Вона підсумовується з результатом і блоком даних. Результат кожного кроку додається з результатом попереднього кроку. Через це результат змінюється швидше. Змінився порядок введення слів у циклах 2 і 3.

В результаті на 5-му кроці отримуємо результат обчислень, які знаходяться в буфері - це хеш. Якщо друкувати байт за байтом, починаючи з нижнього байту першої змінної і закінчуючи вищим байтом останньої змінної, то ми отримаємо хеш MD5. Схема логіки виконання окремого кроку зображено на рисунку 2.4.

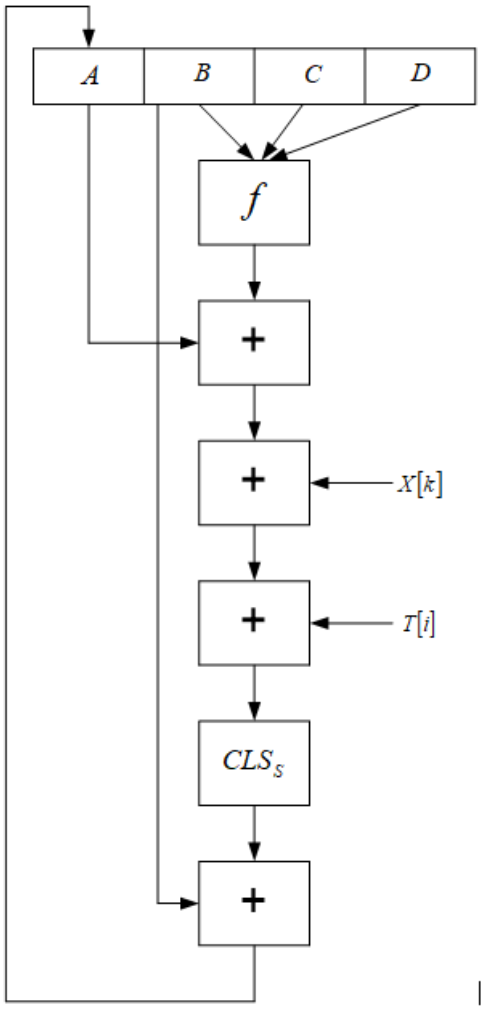

Рисунок 2.4 – Схема логіки виконання окремого кроку обчислення результату

Маючи алгоритм для обрахунку хеш-суми необхідно виконати пошук усіх файлів у заданій папці і для кожного з них обрахувати хеш-суму з використанням

алгоритму md5. Обраховану суму будемо зберігати в БД.

На рис. 2.5 наведено схему удосконаленого пошуку дублікатів файлів.

Процес індексування детально описаний в попередньому розділі. Коли всі данні проіндексовані, можна виконати процес до пошуку дублікатів файлів за хеш-сумою.

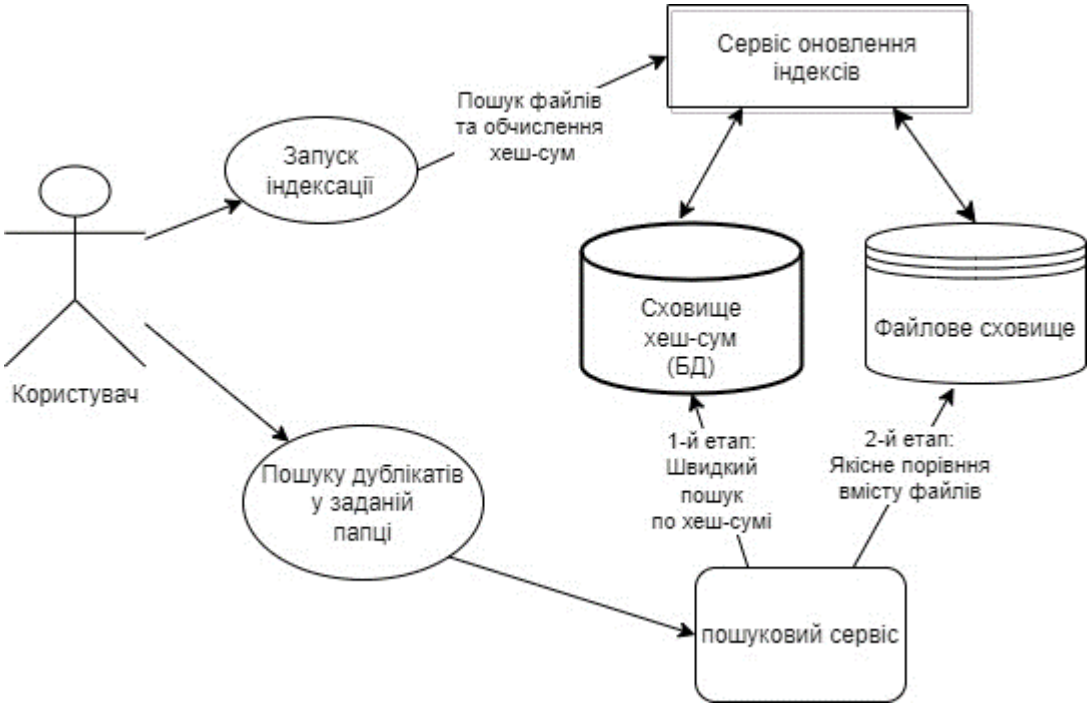

Рисунок 2.5 – Схема удосконаленого методу пошуку дублікатів файлів

Пошук дублікатів виконується запитом пошуку записів індексів для заданої папки з групуванням по значенню хеш-суми та фільтрацією файлових індексів, кількість яких менше двох . Враховуючи, що для значення хеш-суми було створено пошуковий індекс – цей процес не займе багато часу і невдовзі буде отримано списки файлів з однаковими хеш-сумами. Наступним кроком пошуковий сервіс прочитає файли з файлового сховища і порівняє вміст цих файлів за допомогою побітового порівняння. Останній етап порівняння відкине файли, які мають однакові хеш-суми, але мають різний вміст.

Запропонований метод буде ефективним при порівнянні великої кількості файлів та при порівняні великих файлів.

2.3 Проектування бази даних

Для зберігання індексів будемо використовувати реляційну базу даних (РБД) MySQL [12]. Вибір цієї бази даних може бути обґрунтованим з наступних причин:

1. Структурованість даних – РБД надає структурований спосіб зберігання та організації даних у вигляді таблиць і стовпців. Це допомагає ясно визначити структуру індексів файлів і забезпечує легку можливість пошуку та фільтрації даних.

2. Спеціалізовані операції – РБД має вбудовану підтримку операцій, які зручні для зберігання та роботи з індексами. Мова структурованих запитів SQL надає потужні можливості для пошуку, фільтрації та агрегування даних.

3. Індексація – РБД підтримує можливість створення індексів, що значно підвищує швидкість пошуку та фільтрації даних в таблицях. Індекси можуть бути створені на необхідних стовпцях для оптимізації запитів.

4. Нормалізація – реляційна модель підтримує нормалізацію даних, що дозволяє уникнути дублювання даних та забезпечити цілісність інформації. Це особливо корисно для індексів, які мають спільні атрибути.

5. Зв'язки між даними – РБД дозволяє встановлювати зв'язки між різними таблицями за допомогою зовнішніх ключів. Це корисно, коли індекси файлів пов'язані з іншими даними, такими як користувачі, проекти чи теги.

6. Транзакції та безпека – РБД підтримує транзакції, що дозволяють виконувати групу операцій як атомарну одиницю, забезпечуючи цілісність даних. Крім того, РБД може забезпечувати рівень безпеки для даних та контролю доступу до них.

7. Підтримка великих обсягів даних – РБД може обробляти великі обсяги даних та виконувати запити швидко та ефективно, зокрема завдяки можливості індексації.

Зважаючи на ці переваги, реляційна база даних є розумним вибором для зберігання індексів файлів, особливо якщо вони мають складну структуру та пов'язані з іншими даними.

Запроектуємо структуру БД для зберігання індексів файлів. Структура таблиць для зберігання індексів наведена на рисунку 2.6.

Необхідно створити таблицю file_index для зберігання властивостей фалів. Таблиця має містити наступні колонки: id – колонка ідентифікатор таблиці, числового типу з авто-інкрементом; name – стрічкова колонка, зберігатиме ім’я проіндексованого файлу; date_creation та date_update – дати створення та редагування файлу; size – колонка числового типу, зберігає розмір файлу; path – стрічкова колонка, яка містить повний шлях файлу, кожне значення має бути унікальним; type – стрічкова колонка зберігає тип файлу; md5 – стрічкова колонка, містить значення обрахованої за алгоритмом md5 хеш-суми; author – стрічкова колонка, зберігатиме ім’я користувача власника файлу; date_indexed – числова колонка для збереження моменту індексування властивостей файлу, міститиме час в мілісекундах від 01.01.1970. Кожні колонка даної таблиці повинна бути заповнена та мати індекс для швидкого пошуку.

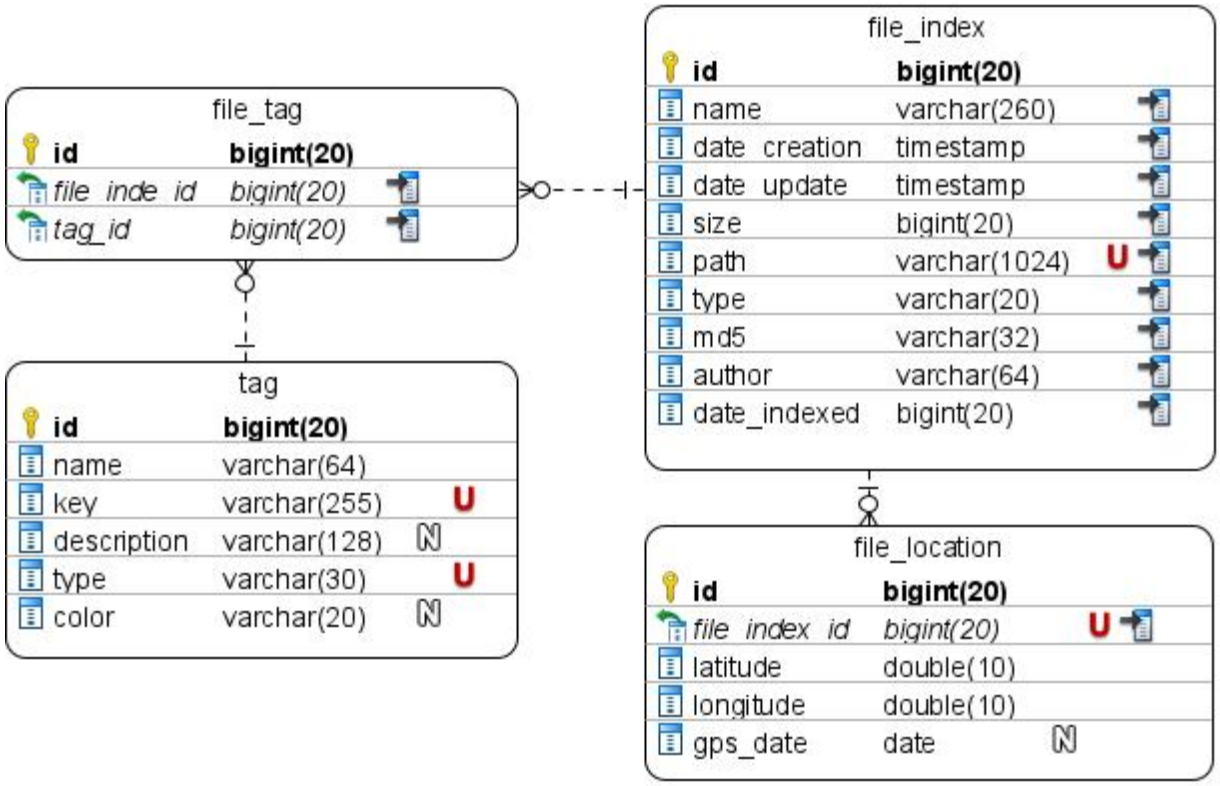

Рисунок 2.6 – Структура таблиць для зберігання індексів

Для зберігання гео-даних необхідно створити окрему таблицю file_location, вона буде містити записи для файлів, які мають гео-інформацію. Таблиця має містити наступні колонки: id – колонка ідентифікатор таблиці, числового типу з авто-інкрементом; file_index_id – ідентифікатор файлового індексу, якому належать гео-дані, містить тільки унікальне значення; latitude – дробовий тип колонки, зберігатиме значення широти; longitude – дробовий тип колонки, зберігатиме значення довготи; gpsDate – колонка, що буде містити дату створення з гео-даних файлу. Таблиця матиме посилання на рядки таблиці file_index використовуючи колонку file_index_id. Рядки таблиці file_location будуть створюватись не для всіх файлів, адже не всі файли містять інформацію про локацію створення.

Для зберігання типів міток необхідно створити таблицю tag. Таблиця буде зберігати інформацію про ім’я, унікальний ключ, опис, тип та колір мітки. Таблиця має містити наступні колонки: id – колонка ідентифікатор таблиці, числового типу з авто-інкрементом; name – стрічкова колонка, зберігатиме ім’я мітки; key – стрічкова колонка, яка буде містити унікальний ключ мітки; description – стрічкова колонка для збереження опису мітки; type – колонка стрічкового типу, міститиме тип мітки; color – стрічкова колонка, зберігатиме колір мітки в html форматі. HTML формат кольору можна вказати кількома способами. Одним із найпоширеніших є використання шістнадцяткового представлення кольору, який складається з шести шістнадцяткових символів, де перші три визначають значення червоного кольору, наступні три - зеленого, а останні три - синього. Також, можна використовувати ключові слова для представлення деяких базових кольорів.

Для зберігання файлів помічених мітками необхідно створити допоміжну таблицю file_tag, яка буде зберігати посилання на ідентифікатор таблиці file_index та ідентифікатор таблиці tag. Таблиця має містити наступні колонки: id – колонка ідентифікатор таблиці, числового типу з авто-інкрементом; file_index_id – числового типу, ідентифікатор файлового індексу, який помічено відповідною міткою; tag_id – числового типу, зберігає ідентифікатор мітки якою

позначено файл. Таблиця має складний унікальний індекс на основі колонок file_index_id та tag_id. Також ці колонки використовуються для створення зв’язків з таблицями file_index та tag відповідно.

2.4 Розробка блок-схем алгоритмів комплексного індексування файлів та удосконаленого методу пошуку дублікатів

Щоб краще уявити загальну структуру проекту побудуємо концептуальну схему проекту. На рисунку 2.7 зображено концептуальну схему застосунку [13].

Кожен з блоків на схемі означає певний модуль проекту. Модулі мають між собою певні зв'язки. Користувач – це основна одиниця, що буде взаємодіяти з застосунком, тому має найбільше зв’язків.

Модуль «Пошук файлів» відповідає за налаштування умов пошуку та безпосередньо за сам пошук, використовується Користувачем. Модуль пов’язаний з модулем «Менеджер БД» – виконує пошук у БД. Також це єдиний модуль, який пов’язаний з модулем «Операції з файлами», адже тільки цей модуль має результати пошуку.

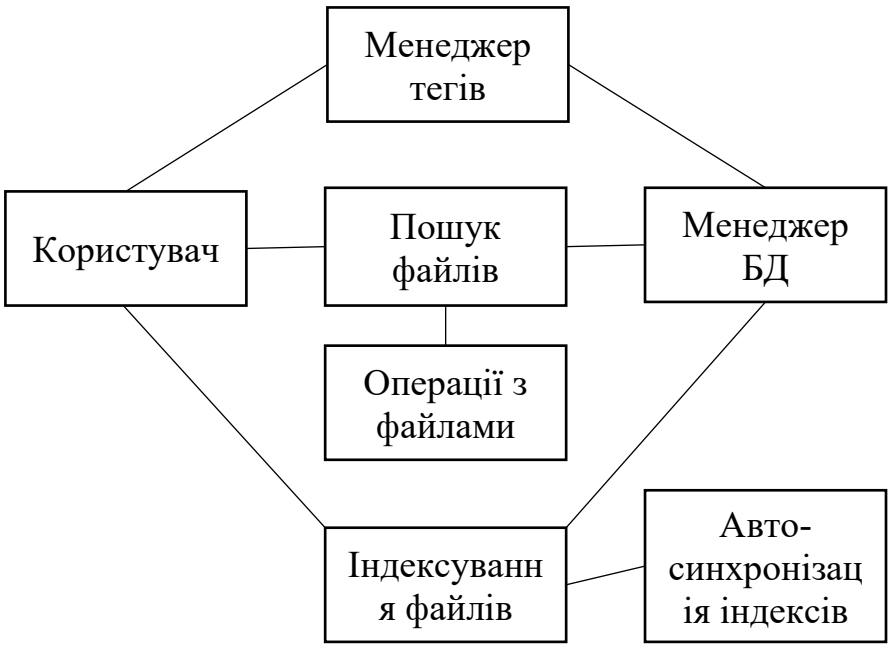

Рисунок 2.7 – Концептуальна схема проекту

Модуль «Операції з файлами» відповідає за створення та виконання операцій з файлами, спираючись на результати пошуку. Модуль пов’язаний з модулем «Пошук файлів».

Модуль «Менеджер БД» – виконує операції читання та запису в БД. Модуль пов’язаний з модулями «Пошук файлів», «Менеджер тегів» та «Індексування файлів».

Модуль «Менеджер тегів» – надає можливість керувати тегами. Модуль пов’язаний з модулями «Менеджер БД» та «Користувач».

Модуль «Індексування файлів» – виконує операції оновлення стану індексів. Модуль пов’язаний з модулями «Менеджер БД», «Авто-синхронізація індексів» та «Користувач».

Модуль «Авто-синхронізація індексів» відповідає за запуск запланованих задач з актуалізації індексів. Модуль пов’язаний з модулем «Індексування файлів».

Розроблено алгоритм індексування файлів. Виконуємо пошук файлів рекурсивним методом в заданій папці. Для кожного файлу спочатку виконується перевірка чи файл був індексований, для цього використовується шлях файлу. Якщо в базі є індекс файлу, то необхідно порівняти дані індексу та властивості файлу. Властивості які відрізняються записуємо в існуючий об’єкт індексу. Якщо індекс не знайдено в БД, то файл є новим то створюємо новий об’єкт індексу на основі властивостей файлу. Вносимо дату індексування файлу, використовуємо дату початку індексування і зберігаємо заповнений індекс в БД. Після того як буде опрацьовано всі файли – необхідно виконати перевірку видалених файлів. Для цього виконуємо пошук в БД по даті індексування та видаляємо індекси файлів, дата індексування яких раніше за поточну. Блок-схему наведено на рисунку 2.8.

Опис алгоритму індексування файлів:

Крок 1.

Виконується пошук файлів рекурсивним методом в заданій та у вкладених папках.

Крок 2.

Перевірка умови чи файл знайдено. Якщо файл знайдено то переходимо до

наступної перевірки, кроку 3. В іншому випадку переходимо до синхронізації видалених файлів, кроку 11.

Крок 3.

Перевірка умови чи файл був раніше індексований, виконується пошук у БД. Якщо індекс знайдено – то переходимо до визначення змін, кроку 4. Якщо запису в БД не знайдено, то переходимо до створення запису, кроку 7.

Крок 4.

Пошук поточних змін властивостей файлу, порівнюючи з деталями з БД. Переходимо до виконання перевірки, кроку 5.

Крок 5.

Перевірка змін у поточному файлі. Якщо зміни є то переходимо до внесення змін в існуючий індекс, крок 6. Якщо змін немає то переходимо до оновлення дати індексування, кроку 9.

Крок 6.

Внесення змін властивостей поточного файлу в об’єкт індексу і перехід до оновлення дати індексування, кроку 9.

Крок 7.

Створення нового об’єкту індексу і перехід до заповнення об’єкту властивостями файлу, кроку 8.

Крок 8.

Об’єкт індексу заповнюється властивостями поточного файлу і переходимо до оновлення дати індексування, кроку 9..

Крок 9.

Оновлення дати індексування, змінна використовується для синхронізації видалених файлів. Далі оновлюємо значення в БД, крок 10.

Крок 10.

Об’єкт індексу зберігаємо у БД, повертаємось до перевірки умови чи знайдено файл, кроку 2.

Крок 11.

Пошук не оновлених об’єктів файлових індексів у БД використовуючи

значення дати поточного індексування файлів. Переходимо до перевірки умови чи запис знайдено, кроку 12.

Крок 12.

Перевірка умови чи знайдено не оновлені об’єкти – це означає, що відповідний файл був видалений і необхідно видалити запис з БД. Переходимо до видалення запису з БД, кроку 13. Якщо записів у БД не знайдено – завершуємо виконання алгоритму.

Крок 13.

Очищення згадки про видалений файл у БД, видаляємо відповідний запис у БД. Завершуємо алгоритм.

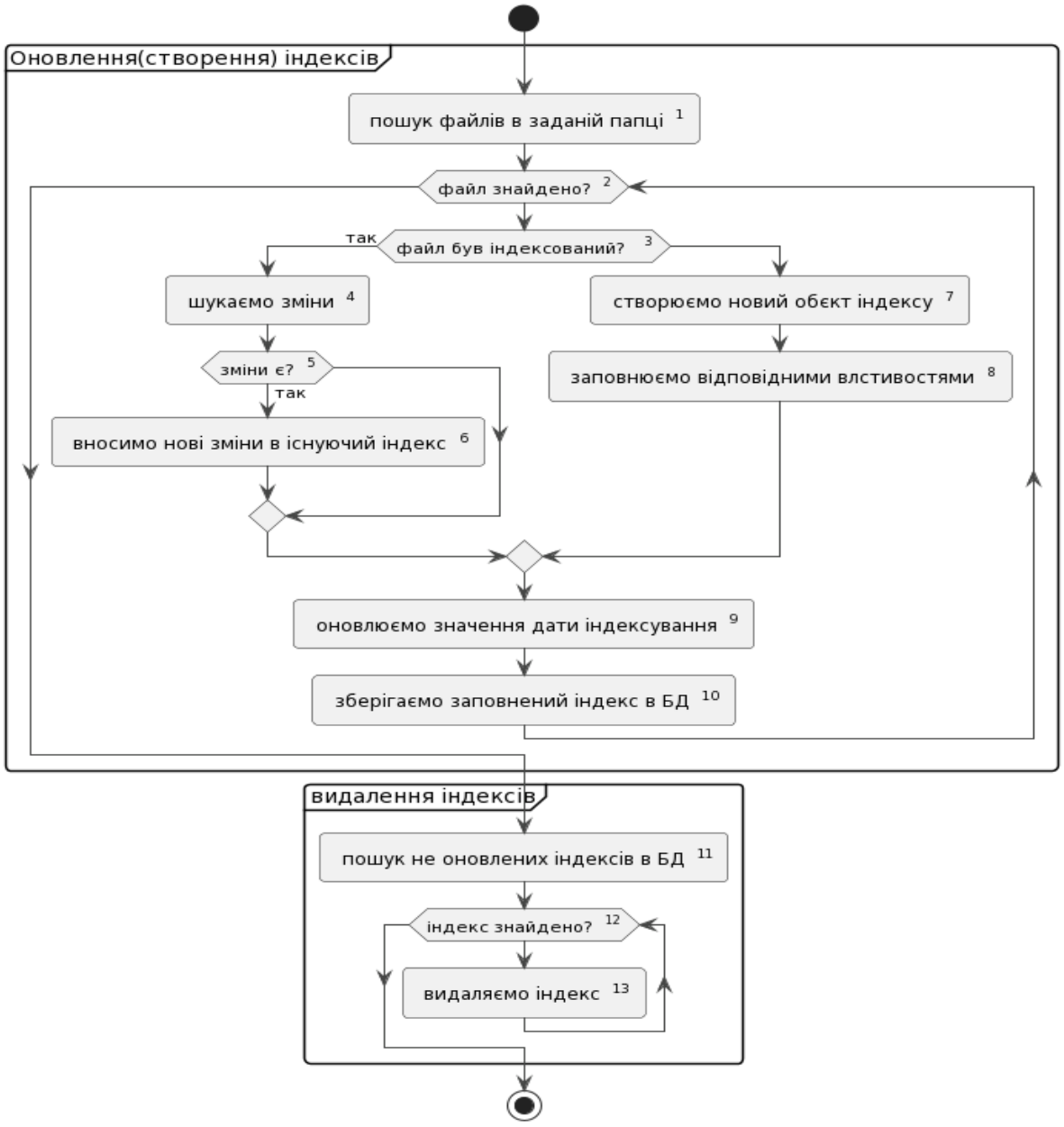

Рисунок 2.8 – Блок-схема алгоритму індексування файлів

Розглянемо алгоритм удосконаленого методу пошуку дублікатів. На рисунку 2.9 зображено блок-схему алгоритму удосконаленого пошуку дублікатів.

Удосконалений алгоритм пошуку дублікатів може використовувати як

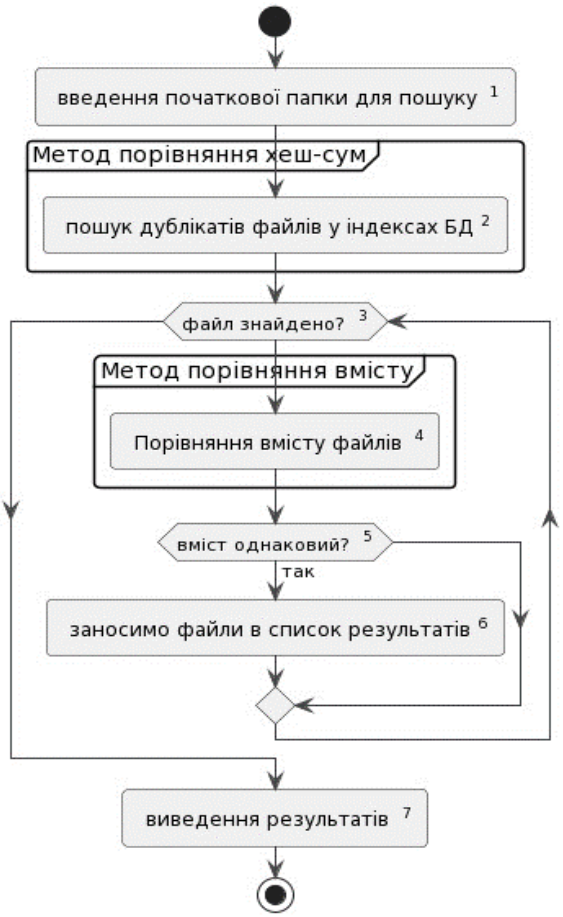

Рисунок 2.9 – Блок-схема алгоритму удосконаленого пошуку дублікатів

метод порівняння хеш-сум, так і порівняння вмісту, для забезпечення більшої точності та ефективності. Основна ідея - використати переваги обох методів, щоб компенсувати їхні недоліки.

Опис алгоритму удосконалений пошуку дублікатів:

Крок 1.

Введення значення початкової папки. Перехід до пошуку дублікатів методом порівняння хеш-сум, крок 2.

Крок 2.

Пошук дублікатів методом порівняння хеш-сум. Виконуємо запит пошуку

індексів файлів в БД по заданій базовій папці, результати групуємо по полю хеш-суми і ігноруємо групи, розмір яких менше 2. Переходимо до перевірки умови чи файл знайдено, кроку 3.

Крок 3.

Перевірка умови чи файли знайдено. Якщо файл знайдено то переходимо до порівняння вмісту файлів потенційних дублікатів, кроку 4. В іншому випадку переходимо до виведення результатів, кроку 7.

Крок 4.

Виконуємо порівняння вмісту знайдених файлів по черзі для кожної групи. Для даної операції використовуються реальні файли. Переходимо до перевірки умови чи вміст однаковий, кроку 5.

Крок 5.

Перевірка умови чи вміст файлів однаковий. Якщо однаковий, то файли є дублікатами, переходимо до оновлення списку результатів, кроку 6. Якщо вміст не однаковий, то беремо наступну групу потенційних дублікатів і переходимо до перевірки умови чи файл задано, кроку 3.

Крок 6.

Занесення інформації про дублікати до списку результатів. Далі переходимо до останнього кроку 7 – виведення результатів.

Крок 7.

Виведення результатів пошуку дублікатів.

Цей удосконалений алгоритм поєднує переваги швидкості та ефективності порівняння хеш-сум з точністю порівняння вмісту файлів, забезпечуючи більш точне виявлення дублікатів.

2.5 Висновки

У другому розділі було розроблено метод комплексного індексування властивостей файлів, що допомагає швидко та ефективно знаходити, фільтрувати та групувати файли за заданими параметрами. Зокрема за назвою файлу, розміром, датою створення, датою останньої зміни, типом файлу,

автором, гео-тегами, md5 сумою.

Розроблено удосконалений методу пошуку дублікатів файлів, в якому для швидкого пошуку буде використовуватись порівняння хеш-суми, а для детального – побітове порівняння вмісту.

Запроектовано структуру індексів та вибрано тип БД для зберігання індексів. Вибір зупинився на реляційній БД MySQL. Властивості вибраної БД вдало задовольняють вимоги задачі. Це структурованість даних, спеціалізовані операції пошуку, фільтрації та агрегування даних, індексація, нормалізація, зв'язки між даними та підтримка великих обсягів даних.

Побудовано алгоритми та блок-схеми алгоритму індексування властивостей файлів та алгоритму пошуку дублікатів файлів. Розглянуто кроки виконання.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОШУКУ ФАЙЛІВ ТА ЇХ ВПОРЯДКУВАННЯ

3.1 Обґрунтування вибору мови програмування

Вибір мови програмування повинен ґрунтуватися на ряді факторів, що включають в себе технічні вимоги, особистий досвід, здатність мови вирішувати специфічні задачі та екосистему мови. Нижче наведено деякі фактори, які можуть вплинути на вибір мови програмування: технічні вимоги проекту, вимоги до продуктивності, особистий досвід розробника, екосистема та бібліотеки, специфічні функціональні вимоги, платформа розгортання, підтримка команди розробників. Розглянемо найпопулярніші мови високого рівня програмування.

Мова програмування Python – це високорівнева, інтерпретована мова програмування, яка набула великої популярності завдяки своїй простоті та гнучкості [14]. Деякі ключові особливості:

- простий та зрозумілий синтаксис, легко читається і зрозуміло навіть людям без значного досвіду в програмуванні;
- високорівнева мова, дозволяє виразно описувати завдання на високому рівні, що допомагає розробникам концентруватися на логіці програми, а не на деталях;
- динамічна типізація – в Python не потрібно явно вказувати типи змінних; типи визначаються автоматично під час виконання програми;
- багато бібліотек та фреймворків – Python має велику кількість бібліотек та фреймворків для різних галузей, таких як веб-розробка (наприклад, Django і Flask), наукові обчислення (наприклад, NumPy і SciPy), штучний інтелект (наприклад, TensorFlow і PyTorch) та багато інших;
- підтримка об'єктно-орієнтованого програмування, що дозволяє розробляти програми, організовані навколо об'єктів та класів;

- широке використання, Python використовується в різних галузях, включаючи веб-розробку, наукові дослідження, аналіз даних, автоматизацію систем, інтеграцію з базами даних, розробку ігор і багато інших галузей;
- велика спільнота розробників, що дозволяє отримувати підтримку, публікувати бібліотеки та знаходити відповіді на запитання;
- кросплатформеність, підтримується на багатьох операційних системах, включаючи Windows, Linux і macOS, що дозволяє розробляти кросплатформені додатки;
- ліцензійна безкоштовність і відкритий код: Python – це вільне та відкрите програмне забезпечення, що означає, що ви можете використовувати його безкоштовно і змінювати вихідний код.

Ці особливості роблять Python потужною та зручною мовою програмування для різних галузей розробки та завдань.

Мова програмування C# є однією з популярних мов програмування, розробленою компанією Microsoft [15]. Вона є частиною технології .NET (платформа розробки програмного забезпечення), і використовується для розробки різноманітних програм, включаючи веб-додатки, десктопні додатки, мобільні додатки, гри та інше. Розглянемо деякі ключові особливості, які роблять її популярною для розробки різних типів додатків:

- підтримує об'єктно-орієнтоване програмування (ООП), що дозволяє розробникам створювати програми, організовані навколо об'єктів та класів. Це полегшує структурування і розвиток додатків;
- мова з багаторівневою безпекою – C# побудована з урахуванням безпеки пам'яті і має ряд вбудованих функцій, що допомагають уникнути типових помилок, пов'язаних з пам'яттю, таких як переповнення буфера;
- платформо незалежність – C# використовує середовище виконання .NET (Common Language Runtime, CLR), що дозволяє розробляти програми, які можуть виконуватися на різних платформах, включаючи

Windows, Linux і macOS;

- багатий набір бібліотек і фреймворків – C# і .NET Framework (та його розширення, такі як .NET Core і .NET 5) надають велику кількість бібліотек і фреймворків для різних завдань, включаючи розробку веб-додатків (ASP.NET), віджетів і інтерфейсів (Windows Presentation Foundation, WPF), наукові обчислення (Math.NET), графіку (OpenGL), роботу з базами даних (Entity Framework) та інше;
- легкість інтеграції з іншими мовами і технологіями – C# має підтримку викликів до коду, написаного на інших мовах, включаючи C і C++, що дозволяє взаємодіяти з існуючим кодом та бібліотеками;
- розширена підтримка від Microsoft – C# розроблено Microsoft, і вона має велику підтримку, включаючи інтегроване середовище розробки (IDE) Visual Studio, що надає засоби для розробки, відлагодження та тестування програм;
- сучасні можливості мови – C# постійно розвивається, додаючи нові можливості до мови, які полегшують розробку;
- ліцензійна безкоштовність – C# і .NET є безкоштовними для використання, що знижує витрати на розробку.

C# – це мова програмування, яка підходить для розробки різноманітних додатків, включаючи веб-додатки, настільні додатки, серверні додатки і багато інших.

Мова програмування Go, також відома як Golang, є досить молодою, але дуже популярною мовою, розробленою в Google в 2007 році і випущено в 2009 році [16]. Основними розробниками Go були Роберт Грісемер, Роб Пайк та Кен Томпсон, які також працювали над іншими важливими проектами, такими як Unix, Plan 9, та інші. Вона була розроблена з метою поєднання простоти та продуктивності. Ось деякі з головних особливостей Go:

- ефективність і продуктивність – Go компілюється в виконуваний код, що робить програми швидкими та ефективними в використанні ресурсів;

- компактний та читабельний синтаксис, що допомагає зменшити кількість коду та спрощує розробку;
- вбудована багато потоковість – мова має вбудовану підтримку горутин (goroutines) та каналів (channels), що дозволяє легко реалізувати паралельні операції та обробку подій;
- система керування пам'яттю, що робить програми більш безпечними від помилок, пов'язаних із витоками пам'яті;
- статична типізація, що дозволяє виявити більшу кількість помилок під час компіляції;
- проста система пакетів, що дозволяє розділити код на легкозрозумілі модулі та бібліотеки;
- відсутність успадкування – Go використовує композицію замість успадкування, що сприяє розробці більш гнучких та зрозумілих програм;
- широкі можливості інтеграції – легко інтегрується з кодом, написаним на інших мовах, що дозволяє використовувати існуючий код та бібліотеки;
- велика та активна спільнота розробників, що видає багато корисних ресурсів та бібліотек;
- ліцензійна безкоштовність і відкритий код, що дозволяє використовувати, модифікувати і розповсюджувати її без обмежень.

Ці особливості роблять Go привабливою мовою для розробки низькорівневих систем, високонавантажених веб-додатків, серверів та інших проектів, де потрібна ефективність, паралельність та низький рівень помилок.

Java – це об'єктно-орієнтована мова програмування, яка була була створена в 1991-1995 роках групою інженерів в компанії Sun Microsystems (зараз власність Oracle Corporation), яку очолював Джеймс Гослинг [17]. Java є однією з найпопулярніших мов програмування у світі, і вона використовується для

розробки різних видів програм, включаючи веб-додатки, мобільні додатки, серверні додатки, ігри та багато інших [18].

Розглянемо деякі з найважливіших особливостей Java:

- платформо незалежність – програми, написані на Java, можуть виконуватися на будь-якій платформі, яка має відповідну віртуальну машину Java (JVM);
- об'єктно-орієнтована мова, що дозволяє створювати програми, які легко розширювати та підтримувати;
- вбудована підтримка багатопоточності, що дозволяє одночасно виконувати багато завдань у великих програмах;
- вбудована система безпеки, яка дозволяє запускати невідомий код у «пісочниці» (sandbox) та обмежує доступ до системних ресурсів;
- збирання сміття – Java автоматично видаляє об'єкти, які більше не використовуються, що робить програми менш вразливими до витоків пам'яті;
- багато бібліотек та фреймворків, які допомагають розробникам швидше реалізувати функціональність та спрощують роботу;
- велика спільнота розробників, що забезпечує підтримку та розвиток мови;
- мова запитів SQL – Java має вбудовану підтримку для взаємодії з базами даних за допомогою мови SQL;
- великий вибір інтегрованих розробних середовищ (IDE) – існують потужні IDE, такі як Eclipse, IntelliJ IDEA, та NetBeans, що допомагають розробникам писати, тестувати та налагоджувати код;
- безкоштовність та відкритий код, що знижує витрати на розробку та ліцензійні обмеження.

Ці особливості роблять Java популярною для різних галузей розробки, включаючи веб-розробку, мобільний розробку, великі корпоративні системи та багато інших застосувань.

Порівняння характеристик мов програмування наведено і таблиці 3.1.

Таблиця 3.1 – Порівняння характеристик мов програмування.

Критерій	Python	C#	Golang	Java
Швидкодія розробленого ПЗ	1	1	2	2
Швидкість розробки	2	1	2	1
Кросплатформеність	1	1	1	1
Широкий вибір бібліотек і фреймворків	2	1	1	2
Розвиток екосистеми мови	1	1	0	1
Особистий досвід розробника	1	0	1	2
Сума балів	8	5	7	9

Розглянуто переваги і недоліки сучасних мов програмування. Вони всі набули високого рівня розвитку і надійності. Усі з розглянутих мов програмування підходять для розробки запроектованого застосунку. В поточному порівняні мова програмування Java набрала найбільше балів тож зупинимо свій вибір на ній.

3.2 Вибір середовища розробки

Наразі, можемо перейти до вибору середовища розробки.

Eclipse – це популярне інтегроване середовище розробки, яке використовується для роботи з різними мовами програмування та технологіями розробки. Підтримує різні мови програмування завдяки плагінам, включаючи Java, C/C++, Python, PHP, Ruby, JavaScript. Ядро Eclipse побудоване на основі системи модульності, це дозволяє легко додавати та видаляти функціональність згідно потреб. Підтримує велику кількість розширень та плагінів, зокрема системи керування версіями, такими як Git, SVN, CVS та іншими. Підтримує роботу з базами даних, включаючи інструменти для запитів SQL. Є потужні інструменти для відладки та профілювання коду. Має інструменти для розробки графічного інтерфейсу користувача. На рисунку 3.1 зображено інтерфейс середовища розробки Eclipse.

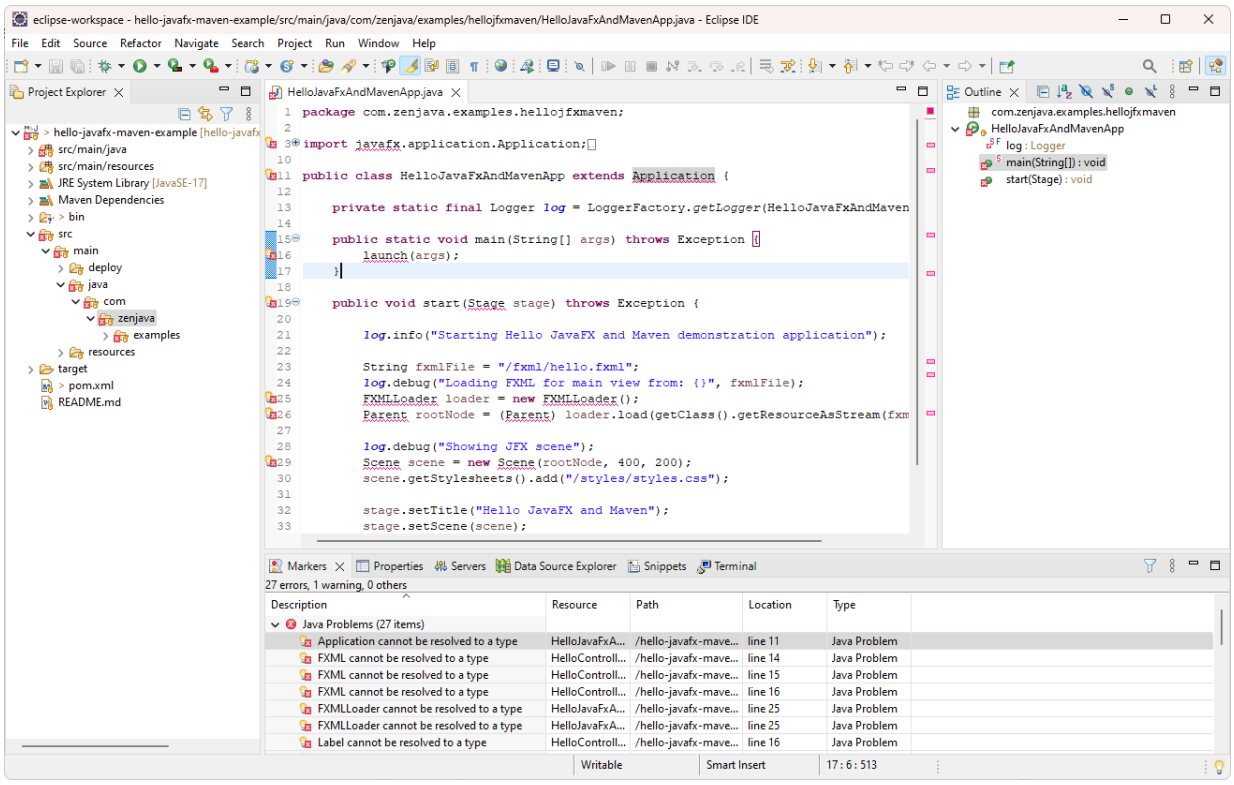

Рисунок 3.1 – Середовище розробки Eclipse

Загалом, Eclipse є потужним та гнучким інструментом для розробки різних видів програм та роботи з різними мовами програмування. Велика кількість розширень і плагінів робить його важливим інструментом для розробників.

IntelliJ IDEA – це одне з найпопулярніших інтегрованих середовищ розробки для роботи з Java [19]. Інтерфейс середовища розробки IntelliJ IDEA зображено на рисунку 3.2. Воно розроблене компанією JetBrains і відоме своєю продуктивністю та численними корисними функціями. IntelliJ IDEA має потужну систему автоматичного завершення коду, яка допомагає розробникам писати код швидше та без помилок. Вона розпізнає контекст і пропонує можливі варіанти завершення методів, змінних, інтерфейсів та інших об'єктів.

Є потужний відладчик, який допомагає виправляти помилки в коді та відстежувати виконання програми. IntelliJ IDEA включає різноманітні інструменти для аналізу коду та виконання рефакторингу, що допомагають поліпшити якість та читабельність коду. IDE легко інтегрується з різними системами керування версіями, такими як Git, SVN та Mercurial. Поза розробкою

Java-додатків, IntelliJ IDEA підтримує інші мови програмування, такі як Kotlin, Groovy, Scala та інші. Є можливість встановлювати різні плагіни та розширення, щоб налаштувати середовище під свої потреби.

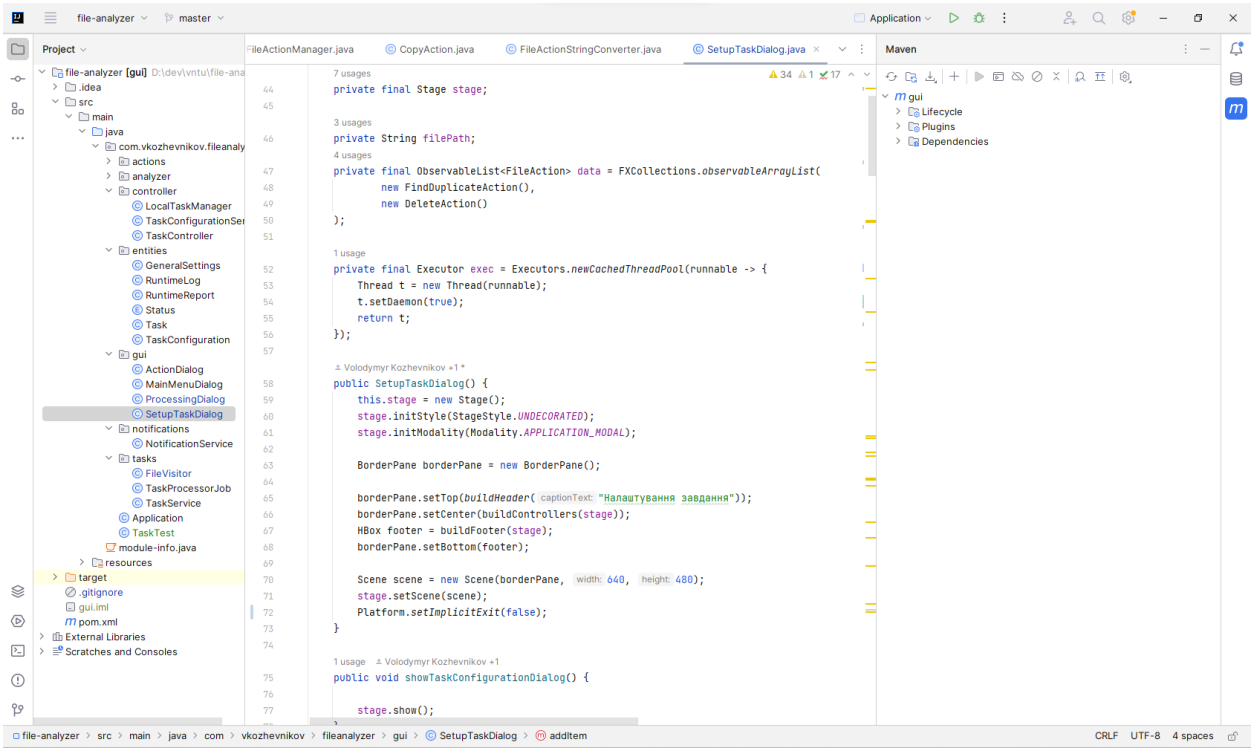

Рисунок 3.2 – Середовище розробки IntelliJ IDEA

IntelliJ IDEA є дуже потужним та продуктивним інструментом для розробки на Java та інших мовах програмування. Воно користується популярністю серед розробників завдяки своїм функціям та зручному інтерфейсу.

NetBeans – це інтегроване середовище розробки, розроблене Apache Software Foundation, яке підтримує різні мови програмування та технології розробки. Інтерфейс середовища розробки NetBeans зображено на рисунку 3.3. IDE підтримує різні мови програмування, включаючи Java, C/C++, PHP, Python, Ruby, та інші. Має інтегровану підтримку для роботи з базами даних, включаючи інструменти для створення та управління базами даних SQL, такі як MySQL, Oracle, та інші. NetBeans інтегрується з різними системами збірки проектів, такими як Maven, Gradle, і Ant. Для розробки на Java NetBeans надає відмінну

підтримку, включаючи автоматичне завершення коду, відлагоджувач, інтеграцію з системами керування версіями, та багато інших інструментів. NetBeans розповсюджується під вільною ліцензією та є відкритим програмним забезпеченням.

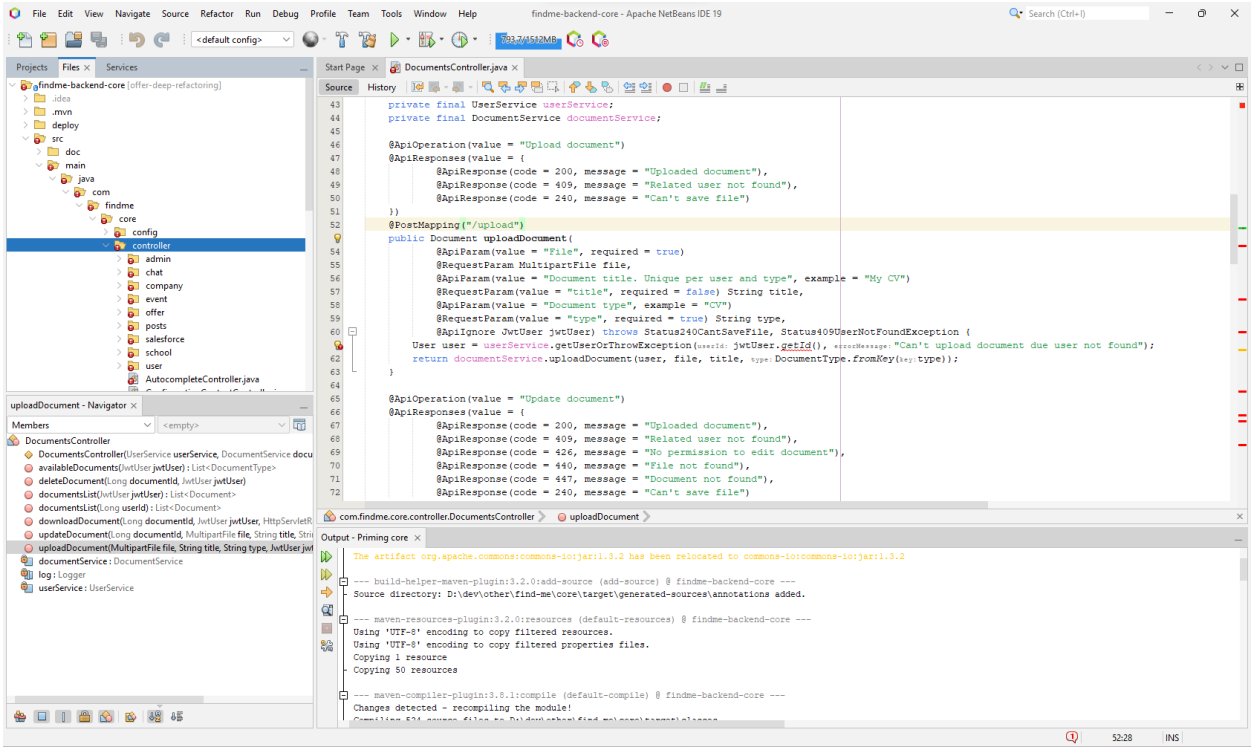

Рисунок 3.3 – Середовище розробки NetBeans

Загалом, NetBeans – це потужне та гнучке середовище розробки, яке відмінно підходить для розробки різних видів програм і підтримує різні мови програмування.

Усі з розглянутих середовищ розробки задовольняють потреби для розробки запроектованого застосунку. Але зручність та ефективність реалізації будь-якої функції в IntelliJ IDEA є вищою за конкурентів. Особливо зручно реалізовані функції системи автоматичного завершення коду та інструменти для аналізу коду та виконання рефакторингу. Також IntelliJ IDEA має найширший вибір доповнень (плагінів), які допомагають розробникам покращити продуктивність та надати додаткову функціональність.

Для розробки програмного забезпечення буде використовуватись IntelliJ IDEA.

3.3 Програмна реалізація застосунку

В процесі програмної реалізації було запроектовано архітектуру застосунку пошуку та впорядкування файлів. Для збірки проекту було обрано Apache Maven [20]. Це популярний інструмент для управління проектами та залежностями в розробці програмного забезпечення. Він використовується для автоматизації збирання, тестування, розгортання та управління проектами на основі конфігураційних файлів, відомих як POM. На рисунку 3.4 зображено файли запроектованого проекту.

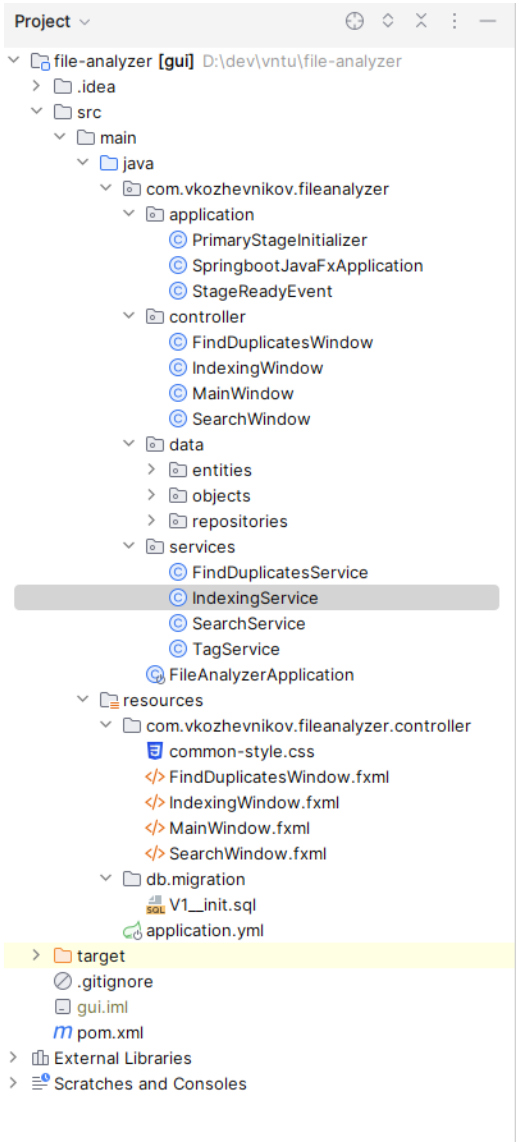

Рисунок 3.4 – Структура проекту

Для зручності та швидкості розробки вирішено використовувати додаткові сторонні бібліотеки та фреймворки. Коротко розглянемо кожен з них.

JavaFX є набором інструментів та бібліотек для розробки графічних користувацьких інтерфейсів (GUI) у мові програмування Java. JavaFX надає засоби для створення багатофункціональних, інтерактивних і естетично здорових додатків з графічним інтерфейсом, включаючи вікна, кнопки, таблиці, графіку та анімацію.

Spring Boot – це проект в рамках екосистеми Spring Framework, який спрощує розробку структурованих, масштабованих та високопродуктивних додатків на основі Spring [21]. Spring Boot надає зручності та зменшує необхідність в конфігурації, що дозволяє розробникам швидше створювати додатки.

JavaFX-Weaver [22] – проект Рене Гілена (Рене Гілен), метою якого є інтеграція Spring і JavaFX. При розробці додатків JavaFX, немає простого способу інтегрувати його з Spring. Інтеграція не працює «з коробки», оскільки додатки JavaFX [23] мають свій життєвий цикл і забезпечують створення екземплярів контролерів. Для вирішення цього питання використовується JavaFX-Weaver.

Project Lombok – це бібліотека для мови програмування Java, яка допомагає спростити розробку, зменшити об’єм коду і зробити код більш читабельним. Lombok надає набір анотацій і інструментів, які автоматично генерують стандартний код, який часто доводиться писати вручну при розробці класів.

Flyway – це інструмент для управління версіями баз даних (Database Version Control) у розробці програмного забезпечення. Flyway дозволяє розробникам зберігати та відстежувати зміни схеми бази даних у вигляді текстових скриптів SQL, і впроваджувати ці зміни у базу даних під час розгортання додатку.

Розробка почалась зі створення структури каталогу та pom файлу проекту. Цей файл є файлом конфігурації Maven (POM - Project Object Model). Він визначає конфігурацію для Maven проекту і включає в себе інформацію про залежності, версії, плагіни, налаштування для збірки проекту та інші параметри.

Розглянемо основні елементи цього POM файлу:

– <project> елемент – основний елемент, який обгортає весь вміст POM файлу;

– <parent> елемент – вказує на те, що проект є дочірнім відносно іншого проекту з артефактом "javafx-weaver-samples" версії "1.3.0" в групі "net.rgielen", використовується для унаслідування та перевизначення конфігурації залежностей та плагінів;

– <modelVersion> елемент – вказує версію POM моделі, у даному випадку, "4.0.0";

– <groupId>, <artifactId>, <version>, <name> елементи – визначають ідентифікатори та версію проекту, використовуються як основа для реалізації ітерацій продукту;

– <properties> елемент – визначає властивості проекту, в даному випадку такі як версія Java та версія Flyway;

– <dependencyManagement> елемент – визначає набір залежностей та їх версій, але не наводить реальних залежностей. Задає стандартні версії для залежностей, які можуть бути використані в дочірніх POM файлах без повторення;

– <dependencies> елемент – визначає залежності для проекту, містить інформацію про бібліотеки та компоненти, які використовуються у проекті, разом із їх версіями та іншими конфігураційними параметрами, такі як Spring Boot, JavaFX, Lombok, Hibernate, Flyway, тощо.

– <build> елемент, який об’єднує елементи <pluginManagement> та <plugins> – визначає конфігурацію для процесу збірки проекту. В цьому розділі необхідно підключити плагін flyway, налаштувати його підключення до БД, а також підключити плагін Spring Boot для збірки та пакування проекту (рис. 3.5).

Наступним кроком створюються пакети для логічної організації класів та ресурсів. Пакет application зберігає системні класи застосунку. У пакеті controller зберігаються класи які реалізують функціональність графічного інтерфейсу. У пакеті data розміщуються класи моделей БД та репозиторії для роботи з ними.

Пакет services об’єднує сервіси, які містять бізнес логіку застосунку.

Також є каталог resources в якому зберігаються усі допоміжні ресурси проекту, зокрема fxml-файли, які є шаблонами розмітки графічного інтерфейсу.

```
<plugins>
    <plugin>
        <groupId>org.flywaydb</groupId>
        <artifactId>flyway-maven-plugin</artifactId>
        <configuration>
            <table>schema_version</table>
            <url>jdbc:mysql://localhost/file_analyzer_db</url>
            <user>root</user>
            <password>mysql</password>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <version>${spring-boot.version}</version>
        <executions>
            <execution>
                <goals>
                    <goal>repackage</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
```

Рисунок 3.5 – Фрагмент налаштування конфігурації для процесу збірки проекту

Після створення пакетів переходять до наповнення їх класами та конфігураційними файлами. Створимо основний файл конфігурації - application.yml, що використовується Spring Boot фреймворком. В ньому необхідно налаштувати параметри з'єднання з базою даних та параметри логування. Інші параметри можна не вказувати, тоді будуть використані значення за замовчанням.

Ще необхідно створити скрипт налаштування структури БД - V1__init.sql і розмістити в пакеті src/main/resources/db/migration. Плагін Flyway перевіряє скрипти міграції на початку кожного запуску програми і якщо скрипт ще не було запущено – виконує його.

Зазвичай коли налаштовано основні конфігураційні файли переходять до

написання класів програми. Для ПЗ, що використовує SpringBoot починають з основного класу FileAnalyzerApplication.java. Цей клас має містити початковий метод запуску «public static void main(String[] args)» з командою запуску додатку та помічений анотацією @SpringBootApplication. Фрагмент класу зображено на рисунку 3.6.

```
@SpringBootApplication
public class FileAnalyzerApplication {
    Volodymyr Kozhevnikov
    public static void main(String[] args) { Application.launch(SpringbootJavaFxApplication.class, args); }
}
```

Рисунок 3.6 – Фрагмент стартового класу SpringBoot додатку

У пакеті com.vkozhevnikov.fileanalyzer.application створимо класи PrimaryStageInitializer.java, SpringbootJavaFxApplication.java та StageReadyEvent.java, які будуть містити код необхідний для ініціалізації графічного інтерфейсу (рис. 3.7).

```
@Override
public void onApplicationEvent(StageReadyEvent event) {
    Stage stage = event.stage;
    stage.initStyle(StageStyle.UNDECORATED);
    stage.setTitle("File Analyzer");
    Scene scene = new Scene(fxWeaver.loadView(MainWindow.class), width: 1024, height: 768);
    stage.setScene(scene);
    stage.show();
}
```

Рисунок 3.7 – Фрагмент налаштування коревої сторінки

У пакеті com.vkozhevnikov.fileanalyzer.controller буде розміщено класи, які описують функціональність виду графічного інтерфейсу. Для кожного виду створюється окремий клас, наприклад IndexingWindow.java. В цьому класі перелічені такі елементи інтерфейсу, як кнопки, написи, панелі розмітки, тощо. Також там налаштовується взаємодію цих елементів між собою та реакція на певні події. На рисунку 3.8 зображено фрагмент налаштування подій при натисканні кнопки «Початок індексації»

```
startButton.setOnAction(actionEvent -> {
    log.info("Обрано <Почати інжексування>: " + filePathString);
    longProperty.set(0L);
    currentProcessingLabel.setText("");
    blockInput();
    tagsContainer.getChildren().clear();
    startIndexing();
});
```

Рисунок 3.8 – Фрагмент налаштування подій при натискані кнопки

Для налаштування розмітки і розміщення елементів графічного інтерфейсу на сторінці застосовуються файли fxml. Ці файли буде розташовано також в пакеті com.vkozhevnikov.fileanalyzer.controller, але у розділі resources замість java. Файли FXML — це файли, які містять опис інтерфейсу користувача в форматі FXML (FXML Markup Language). Ці файли використовуються в JavaFX-додатках для визначення структури та вигляду графічного інтерфейсу програми. Файли FXML забезпечують декларативний спосіб визначення UI, відокремлюючи логіку від подання. Формат FXML подібний до XML, і він використовує теги та атрибути для визначення різних елементів і їх властивостей.

Створимо розмітку елементів інтерфейсу для вікна індексації – IndexingWindow.fxml. Для налаштування розміщення будуть комбінуватись контейнери розмітки BorderPane, HBox, VBox, FlowPane, StackPane — це контейнери для розміщення елементів у JavaFX. Вони дозволяють організовувати та розташовувати елементи користувальницького інтерфейсу відповідно до певної структури.

BorderPane розділяє вікно на п'ять регіонів: верхній (top), нижній (bottom), лівий (left), правий (right) та центральний (center). Один елемент може розташовуватися в кожному регіоні, і вони розтягуються на весь розмір батьківського контейнера. Цей буде контейнер буде використовуватись в якості кореневого та для компоновки панелі налаштування та виведення міток.

HBox контейнер розташовує елементи горизонтально, один за одним в ряд, зліва направо. HBox контейнер використовуватиметься наприклад для групування елементів вибору початкової папки індексування (рис. 3.9), а також для компонування елементів хідера та футера.

```
<HBox styleClass="paneSpacing"  alignment="CENTER_LEFT">
    <Label text="Початкова папка індексування" />
    <TextField fx:id="filePath" HBox.hgrow="ALWAYS" />
    <Button fx:id="btnChoosePath" text="Вибрати" />
</HBox>
```

Рисунок 3.9 – Фрагмент використання контейнеру HBox

Контейнер VBox розташовує елементи вертикально, один під одним в колоні, зверху донизу. Схожий до контейнеру HBox, але групує по вертикалі.

Контейнер FlowPane розташовує елементи в потік, пристосовуючи їх в ряд по горизонталі або вертикалі залежно від налаштувань. Елементи в FlowPane можуть розташовуватися в одному ряду або автоматично переходити на новий ряд, якщо місце закінчується. На рисунку 3.10 зображено фрагмент використання контейнеру FlowPane для для групування налаштування міток.

```
<FlowPane fx:id="labelsContainer" prefHeight="50" alignment="CENTER" styleClass="paneSpacingAndPadding">
    <CheckBox fx:id="tagByAuthor" text="Автор" selected="true"/>
    <CheckBox fx:id="tagByDate" text="Дата створення" selected="true" />
    <CheckBox fx:id="tagByType" text="Тип" selected="true" />
    <CheckBox fx:id="tagByGps" text="Локація" selected="true" />
</FlowPane>
```

Рисунок 3.10 – Фрагмент використання контейнеру FlowPane

StackPane розташовує елементи один поверх одного у вигляді стеку. Останній доданий елемент буде відображатися поверхневим. Зазвичай використовується для розташування елементів так, щоб вони займали одну і ту ж площину.

Для створення мапінгу елементу графічного інтерфейсу у fxml файлі необхідно встановити атрибут «fx:id = "назва_змінної"» (рис. 3.11).

```
<FlowPane fx:id="tagsContainer" prefHeight="148.0" prefWidth="920.0"
          styleClass="indexingFlowPane"/>
```

Рисунок 3.11 – Фрагмент встановлення посилання на змінну

В java-класі, що описує функціональність елементів графічного інтерфейсу, для мапінгу елементу у поле класу необхідно використовувати анотацію @FXML (рис. 3.12).

```
@FXML
private FlowPane tagsContainer;
```

Рисунок 3.12 – Використання анотації @FXML

Для інших сторінок графічних видів створюємо пари java та fxml файлів: MainWindow.fxml та MainWindow.java, FindDuplicatesWindow.fxml та FindDuplicatesWindow.java, SearchWindow.fxml та SearchWindow.java.

Також для оформлення графічного інтерфейсу можна використовувати файли css. Стилізація за допомогою CSS дозволяє змінювати вигляд та відчуття елементів інтерфейсу, таких як кнопки, тексти, контейнери тощо. Використання CSS дозволяє окремо визначати стилі для кожного елементу інтерфейсу. На рисунку 3.13 зображено використання файлів css у fxml файлі та їх підключення за допомогою тегу « <stylesheets> ».

```
<stylesheets>
    <URL value="@common-style.css" />
</stylesheets>
```

Рисунок 3.13 – Підключення файлу стилів

У пакеті « com.vkozhevnikov.fileanalyzer.data » об’єкти-контейнери даних, зокрема об’єкти БД та репозиторії, об’єкти користувацьких графічних елементів тощо.

Створимо та розмістимо об’єкти БД та репозиторії в пакетах

com.vkozhevnikov.fileanalyzer.data.entities та com.vkozhevnikov.fileanalyzer.data.repositories відповідно. Для зручності будемо використовувати функціональність JPA фреймворку [24]. Для роботи з об’єктом БД створимо новий клас з іменем відповідним таблиці БД, наприклад якщо таблиця має назву « file_index » , то клас має називатись FileIndex.java. Клас треба поміти анотацією @Entity. Ця анотація використовується в контексті Java Persistence API (JPA), який визначає стандарт для роботи з реляційними базами даних у Java-додатках. Коли клас позначено анотацією @Entity, він вказує, що цей клас відображає об'єкт, який може бути збережений в базі даних. Кожен екземпляр класу представляє запис в таблиці бази даних. Також необхідно поміти поле класу, яке є ключем у таблиці БД. Для цього використовується анотація @Id. На рисунку 3.14 зображено фрагмент класу для роботи з БД.

```
@Entity
public class FileIndex {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private Date dateCreation;
```

Рисунок 3.14 – Фрагмент класу для роботи з БД

Для об’єднання рядків двох таблиць, наприклад для відображення в об’єктів FileIndex гео-даних з таблиці file_location, треба використати анотацію @OneToOne. Ця анотація в JPA вказує на відношення "один-до-одного" між двома сутностями. Це вказує, що кожен об'єкт одного класу пов'язаний з одним об'єктом іншого класу, і навпаки. На рисунку 3.15 зображено використання анотації @OneToOne. В данному випадку необхідно також вказати параметр « mappedBy = "fileIndex" », що вказує ім'я поля в іншій сутності, яке

```
@OneToOne(mappedBy = "fileIndex")
private FileLocation fileLocation;
```

Рисунок 3.15 – Використання анотації @OneToOne

використовується для зворотного відношення. Параметр «mappedBy» вказує, яке поле в іншій сутності пов'язане з поточним.

Тепер треба об’єднати значення декількох рядків таблиці FileTag з об’єктом FileIndex, тобто відобразити список міток якими позначений файл в об’єкті FileIndex. Для цього використаємо анотацію @OneToMany. Анотація @OneToMany вказує на відношення "один-до-багатьох" між двома сутностями. Це вказує, що один об'єкт одного класу пов'язаний з кількома об'єктами іншого класу. В зв’язку з тим, що таблиця file_index не має посилання на таблицю file_tag, маємо вказати параметр «mappedBy» для зворотнього відношення. Використання анотації @OneToMany зображено на рисунку 3.16.

```
@OneToMany(mappedBy = "fileIndex", fetch = FetchType.EAGER, orphanRemoval = true)
private List<FileTag> tags;
```

Рисунок 3.16 – Використання анотації @OneToMany

Створюємо класи, які будуть відображати об’єкти таблиць БД: FileIndex.java, FileLocation.java, Tag.java та FileTag.java.

Для виконання операцій пошуку, редагування та видалення об’єктів таблиць БД необхідно створити репозиторій. Для цього треба створити новий інтерфейс який наслідує PagingAndSortingRepository< тип_об’єкта_БД, тип_ключа_об’єкта_БД >. Створений репозиторій надає методи findAll(), findById(), save(), delete() для роботи з об'єктом БД. Також в JPA репозиторіях можна використовувати ключове слово « By » в назві методу інтерфейсу репозиторію для автоматичної генерації запитів на основі імен методу. При використанні findAllBy, треба вказати критерії пошуку за полями сутності. Для прикладу, напишемо метод для пошуку об’єкту FileIndexy БД по значенню поля «path». На рисунку 3.17 зображено метод для пошуку FileIndex по параметру «path».

```
@Repository
public interface FileIndexRepository extends PagingAndSortingRepository<FileIndex, Long>
    1 usage   Volodymyr Kozhevnikov
    Optional<FileIndex> findByPath(String path);
```

Рисунок 3.17 – Метод для пошуку FileIndex по параметру «path»

Для створення складнішого запиту пошуку JPA (Java Persistence API) пропонує декілька підходів використовувати запити JPQL або Criteria API.

JPQL (Java Persistence Query Language) – це мова запитів, яка використовується в JPA для виконання запитів до бази даних. JPQL є об'єктно-орієнтованою мовою запитів, яка базується на об'єктах та класах, а не на таблицях та колонках. Створимо запит для пошуку дублікатів у зазначеній папці по значенню хеш-суми. Для цього згрупуємо рядки таблиці file_index по колонці md5 і відкинемо записи, які не відносяться до заданої папки та значення md5, яких зустрічаються тільки один раз. Реалізація запиту з використанням JPQL зображена на рисунку 3.18.

```
@Query("select fi.md5 from FileIndex as fi " +
        "          where length(fi.md5) > 0 " +
        "            and fi.path like :path% " +
        "          group by fi.md5 having count(fi.md5) > 1")
List<String> findMD5Duplicates(@Param("path") String path);
```

Рисунок 3.18 – Метод для пошуку дублікатів у зазначеній папці по значенню хеш-суми

Criteria API в JPA – це програмний інтерфейс для створення динамічних та типобезпечних запитів до бази даних без використання рядкових запитів, таких як ті, що використовують JPQL. Створимо запит для пошуку файлів по масці імені файлу та списку міток, якими помічено файл. Реалізація методу з використанням Criteria API зображено на рисунку 3.19.

```
private static Specification<FileIndex> searchFiles(List<Tag> tags, String query) {
    return (root, criteriaQuery, cb) -> {
        Predicate p = cb.conjunction();
        if (!StringUtils.isEmpty(query)) {
            p = cb.and(p, cb.like(cb.lower(root.get(FileIndex_.NAME)), pattern: query.toLowerCase() + "%"));
        }
        if (!tags.isEmpty()) {
            for (Tag tag : tags) {
                Join<FileIndex, FileTag> joinFileTag = root.join(FileIndex_.TAGS, JoinType.INNER);
                p = cb.and(p, cb.equal(joinFileTag.get(FileTag_.TAG), tag));
            }
        }
        return p;
    };
}
```

Рисунок 3.19 – Використання Criteria API для пошуку файлів

Створюємо інтерфейси, які надаватимуть доступ до операцій з таблицями БД: FileIndexRepository.java, FileLocationRepository.java, TagRepository.java та FileTagRepository.java.

В пакеті com.vkozhevnikov.fileanalyzer.services будуть розміщені класи, які містять основну бізнес логіку. Тобто такі функції, як індексування даних, створення міток, підрахунок хеш-суми, пошук дублікатів, методи порівняння файлів, тощо. Їх буде розподілено по декільком сервісам, в залежності від типу даних які вони обробляють. Створимо наступні класи: FindDuplicatesService.java, IndexingService.java, SearchService.java та TagService.java.

Клас IndexingService.java реалізовує алгоритм індексування файлів та містить методи: updateFileIndexes – виконуватиме рекурсивний пошук в заданій папці за допомогою реалізації пошуку файлів – SimpleFileVisitor; indexFile – метод який аналізуватиме метадані файлу і оновлюватиме індекс файлу у БД; checkAuthorTag, checkDateTag, checkGps, checkTypeTag – методи для автоматичного створення та оновлення міток. Також тут присутні такі допоміжні методи, як resolveExtension – для визначення типу файлу; toRGBCode – для перетворення формату збереження кольору; calculateMd5 – для підрахунку хеш суми.

Клас TagService.java містить методи для управління мітками. Це findOrCreateTag – метод для пошуку або створення мітки; updateFileTag – метод для оновлення мітки; та getAllTags – для отримання усіх міток.

Клас FindDuplicatesService.java реалізовує алгоритм пошуку дублікатів та містить метод для пошуку дублікатів файлів: findMD5Duplicates – виявлення дублікатів хеш-суми для заданої папки; findFilesByMd5 – отримання файлів по значенню хеш-суми; та compareByMemoryMappedFiles – метод для порівняння вмісту двох файлів.

Клас SearchService.java містить метод для виконання пошуку файлів, який реалізовує алгоритм комплексного пошуку файлів.

Отже в поточному розділі було детально розглянуто реалізацію розроблених алгоритмів та основних компонентів проектуємого застосунку.

Лістинг проекту наведено в додатку В.

3.4 Висновки

У третьому розділі було обґрунтовано вибір мови програмування. Для програмної реалізації запроектованого застосунку обрано об'єктно-орієнтовану мову Java. Java є однією з найпоширеніших мов програмування для розробки системних застосунків.

Після вибору мови програмування було розглянуто найпопулярніші середовища розробки для Java. Було вирішено використовувати IntelliJ IDEA, адже це найзручніше та ефективне середовище розробки.

Для розробки графічного інтерфейсу використано Java FX, що є потужним крос-платформним фреймворком для розробки графічних інтерфейсів на мові Java.

Було розглянуто програмну реалізацію застосунку для пошуку та впорядкування файлів. Описано перелік та властивості фреймворків та бібліотек, що будуть застосовуватись у проекті.

Детально пояснюється порядок роботи з основними функціями застосунку.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ ДЛЯ ПОШУКУ ФАЙЛІВ ТА ЇХ ВПОРЯДКУВАННЯ

4.1 Методики тестування

Тестування програмного забезпечення — це процес перевірки програмного продукту з метою виявлення помилок, визначення відповідності вимогам та забезпечення його високої якості. Тестування важливо на всіх етапах розробки програмного продукту, від концепції до етапу підтримки [25].

За підходом проведення тестування програмного забезпечення поділяють на метод білої скриньки, метод чорної скриньки та метод сірої скриньки.

Метод білої скриньки, також відомий як тестування на рівні коду або структурне тестування. У цьому методі тестувальник має повний доступ до внутрішньої структури коду програми. Це означає, що тестувальники аналізують код, виконують його, та перевіряють, чи відповідає він визначеним специфікаціям та вимогам. Перевагами методу білої скриньки є висока точність виявлення помилок у програмному коді та можливість тестування всіх можливих шляхів виконання програми. Однак цей метод вимагає детального знання програмного коду та може вимагати багато часу, особливо для великих проектів.

Метод чорної скриньки — це метод тестування програмного забезпечення, при якому тестування проводиться без знання внутрішньої структури чи деталей реалізації програми. Тестувальники взаємодіють з програмою як із «чорною скринькою», тобто обирають вхідні дані, спостерігають за виведенням та оцінюють реакції системи без знання її внутрішніх механізмів. Основна переваги методу – це можливість тестування програми без знання їхньої внутрішньої структури, що робить його ефективним для тестування на ранніх етапах розробки. Однак цей метод може не виявити деякі типи помилок, які пов'язані із внутрішніми аспектами програмного коду. Тому використання комбінації методів білої та чорної скриньок є поширеною практикою для забезпечення повноцінного тестування програмного продукту.

Метод сірої скриньки представляє собою комбінацію методів білої

скриньки та чорної скриньки. У цьому методі тестування тестувальники мають часткове знання про внутрішню структуру та реалізацію програми. Це може включати обмежене знання коду або внутрішніх процесів системи. Цей метод тестування дозволяє забезпечити більш глибоке та збалансоване покриття тестами, порівняно із чистими методами білої або чорної скриньки. Також він може бути ефективним у випадках, коли повне знання коду недоступне або непрактичне, але є необхідним для виявлення деяких видів помилок.

Також тестування програмного забезпечення можна поділити на типи:

- модульне тестування, перевірка окремих модулів або компонентів програми для переконання у їхній правильності;
- інтеграційне тестування, тестування взаємодії між різними частинами програми;
- системне тестування, проводиться для перевірки повного, інтегрованого програмного продукту, що весь програмний продукт працює як єдина система і відповідає вимогам, визначеним на етапі розробки;
- приймальне тестування, визначає чи задовольняє програмний продукт вимогам користувача;
- функціональне тестування, перевірка функціональності програмного продукту;
- навантажувальне тестування, визначення як програма працює при великому обсязі даних або числі користувачів;
- стрес-тестування, визначення меж продуктивності та стійкості системи за межами звичайного;
- безпекове тестування, перевірка системи на вразливість та виявлення можливих загроз безпеці;
- автоматизоване тестування, використання автоматизованих засобів для виконання тестів, що дозволяє швидше та ефективніше перевіряти функціональність.

Тестування допомагає виявити та виправити помилки ще на ранніх етапах

розробки, зменшити ризик виникнення проблем у продукті, підвищити його якість та впевненість користувачів у його працездатності [26].

4.2 Тестування програмного застосунку

Було проведено тестування програмного застосунку за допомогою юніт-тестів та вручну, це методи «білої скриньки» та «чорної скриньки відповідно».

Юніт-тести було розроблено для перевірки функцій індексування файлів, створення міток та декілька тестів на перевірку різних запитів пошуку [26].

Для розробки юніт-тестів [27] було використано фреймворки JUnit, Mokito та AssertJ. Часто ці бібліотеки використовуються разом у тестовому стеку для створення, виконання та перевірки тестів в Java-проектах. JUnit [28] служить основою для написання тестів, Mockito [29] допомагає управляти залежностями та перевіряти взаємодію об'єктів, а AssertJ робить код тестів більш читабельним та допомагає зробити їхню логіку більш зрозумілою.

Для виконання юніт-тестів було відібрано декілька файлів, різного типу; налаштовано тимчасову БД, окремий основний конфігураційний файл application-integrationtest.yml та розроблено тести для перевірки основних функцій, а саме індексування файлів, автоматичне створення міток, пошук файлів. Загалом процес виконання юніт-тесту виглядає наступним чином: для ізоляції юніт-тестів при пакетному виконанні – очищається контекст за допомогою анотації @DirtiesContext, фрагмент коду з застосування зображено на рисунку 4.1; викликається метод індексування файлів, який створює записи БД; виконуються унікальні для кожного тесту кроки запиту певної інформації, а потім результат порівнюються з очікуваною константою.

```
@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
class IndexingAndSearchServiceTest {
```

Рисунок 4.1 – Використання анотації@DirtiesContext

Метод індексування файлів очікує вхідні дані введені в елементах графічного інтерфейсу. Для того щоб уникнути роботу з елементами графічного

інтерфейсу використано метод spy із бібліотеки Mockito, який дозволяє створити шпигунський об'єкт (spy), тобто об'єкт, який зберігає свою оригінальну реалізацію, але можна вказати, які методи будуть "шпигувати" (слідкувати за їх викликами) і задавати певні поведінкові правила для цих методів. Для методів отримання даних від елементів графічного інтерфейсу було змінено поведінку за допомогою методів «when» та «thenReturn». Якщо метод не налаштований, він виконає свою оригінальну реалізацію. Фрагмент підміни поведінки методів зображено на рисунку 4.2. Основна логіка методу індексування не підмінювалась.

```
IndexingService indexingService = spy(new IndexingService(fileIndexRepository, fileTagRepository,
        fileLocationRepository, tagService));
given(indexingWindow.isTagByAuthor()).willReturn( t: true);
given(indexingWindow.isTagByDate()).willReturn( t: true);
given(indexingWindow.isTagByGps()).willReturn( t: true);
given(indexingWindow.isTagByType()).willReturn( t: true);
doNothing().when(indexingService).updateTagCount(any(), anyMap(), any());
```

Рисунок 4.2 – Підміна поведінки методів за допомогою Mockito

Для перевірки методу індексування файлів, розроблено юніт-тест indexFilesTest(). Його реалізацію зображено на рисунку 4.3. Тест складається з наступних кроків:

– виклик метода індексування даних для тестових файлів, що виконує аналіз файлів, створює відповідні індекси та мітки записи в БД;
– отримання всіх записів FileIndex з репозиторію та порівняння з заготовленим результатом.

```
@Test
void indexFilesTest() throws IOException, ParseException {
    doIndexingWithTags(pathPrefixIndexing, dateIndexed);
    FileIndex fileIndex1 = fileIndex1(pathPrefixIndexing, dateIndexed);
    FileIndex fileIndex2 = fileIndex2(pathPrefixIndexing, dateIndexed);
    FileIndex fileIndex3 = fileIndex3(pathPrefixIndexing, dateIndexed);
    List<FileIndex> fileIndexList = List.of(fileIndex1, fileIndex2, fileIndex3);
    assertThat(fileIndexRepository.findAll()).asString().isEqualTo(fileIndexList.toString());
}
```

Рисунок 4.3 – Юніт-тест для перевірки індексування файлів

Для перевірки методу створення міток при індексуванні файлів, розроблено юніт-тест tagsCreationTest(). Його реалізацію зображено на рисунку 4.4. Тест складається з наступних кроків:

– виклик метода індексування даних для тестових файлів, що виконує аналіз файлів, створює відповідні індекси та мітки записи в БД;
– отримання всіх записів Tag з репозиторію та порівняння з заготовленим результатом.

```
@Test
void tagsCreationTest() throws IOException {
    doIndexingWithTags(pathPrefixIndexing, dateIndexed);

    List<Tag> tags = List.of(tag1, tag2, tag3, tag4, tag5, tag6, tag7);
    assertThat(tagRepository.findAll()).hasSameElementsAs(tags);
}
```

Рисунок 4.4 – Юніт-тест для перевірки створення міток

Для перевірки пошуку файлів та отримання пустого результату, розроблено юніт-тест searchWithTagsOnlyEmptyResponseTest(). Його реалізацію зображено на рисунку 4.5. Тест складається з наступних кроків:

– виклик метода індексування даних для тестових файлів, що виконує аналіз файлів, створює відповідні індекси та мітки записи в БД;
– створення запиту з міток, виконання якого має повернути пустий результат;
– виконання пошуку для створеного запиту і перевірка чи результат пустий.

```
@Test
void searchWithTagsOnlyEmptyResponseTest() throws IOException, ParseException {
    doIndexingWithTags(pathPrefixIndexing, dateIndexed);
    SearchService searchService = new SearchService(fileIndexRepository);

    List<Tag> selectedTags = List.of(tag1, tag3, tag4);

    Page<FileIndex> searchResponse = searchService.search(selectedTags, query: "", PageRequest.of( page: 0,  size: 10));
    assertThat(searchResponse.getContent()).isEmpty();
}
```

Рисунок 4.5 – Юніт-тест searchWithTagsOnlyEmptyResponseTest

Для перевірки пошуку файлів та отримання не пустого результату, розроблено юніт-тест searchByQueryAndTagTest (). Його реалізацію зображено на рисунку 4.6. Тест складається з наступних кроків:

– виклик метода індексування даних для тестових файлів, що виконує аналіз файлів, створює відповідні індекси та мітки записи в БД;
– створення запиту з міток та маски назви файлу, виконання якого має повернути певний не пустий результат;
– виконання пошуку для створеного запиту і перевірка чи результат відповідає очікуваному значенню.

```
@Test
void searchByQueryAndTagTest() throws IOException, ParseException {
    doIndexingWithTags(pathPrefixIndexing, dateIndexed);
    SearchService searchService = new SearchService(fileIndexRepository);

    Page<FileIndex> searchResponse = searchService.search(List.of(tag3),  query: "%l", PageRequest.of( page: 0,  size: 10));
    List<FileIndex> expectedResult = List.of(fileIndex1(pathPrefixIndexing, dateIndexed));

    assertThat(searchResponse.getContent()).isEqualTo(expectedResult);
}
```

Рисунок 4.6 – Юніт-тест searchByQueryAndTagTest

Для виконання юніт-тестів використовується фреймворк JUnit, який налаштовує спеціальне інтеграційне середовище і в ньому запускає юніт-тести. Всі тести успішно виконанні. Це підтверджує, що при тестуванні методом «білої скриньки» помилок не виявлено. На рис. 4.7 показано результати виконання юніт-тестів.

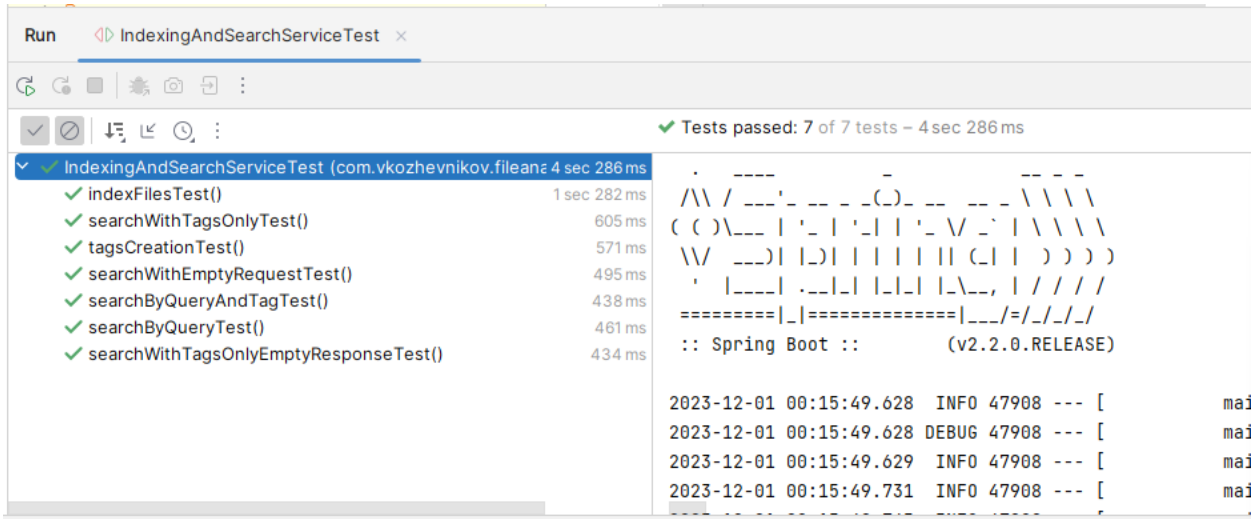

Рисунок 4.7 – Результат виконання юніт-тестів

Тепер розглянемо виконання тестування застосунку за методом «чорної скриньки». Застосуємо ручне тестування за допомогою тест-кейсів. Умову та результати тест-кейсу №1 зображено на рис. 4.8.

Тест кейс №1. Перевірка індексації даних

Кроки виконання:

1. Підготувати папку з фалами (d:\tmp) та запустити застосунок
2. Вибрати розділ «Індексування файлів»
3. Вибрати початкову папку індексування: d:\tmp, відмітити всі види автоматичних міток та натиснути кнопку «Почати»
4. Дочекатися завершення та перевірити результати

Очікувані результати:

Всього файлів оброблено: 15

Мітки «Автор»	Мітки «Дата створення»	Мітки «Тип»	Мітки «Локація»
SEO\VKozhevnikov:13, BUILTIN\Administrators:2	December '23:2, November '23:7, September '22:3, September '23:1, October '23:2	pla:1, log:1, txt:5, mmdb:3, download:1, jpg:1, sql:1, png:2	27.1/34.1:2, December '23:1, November '23:1,

Актуальний результат: Тест пройдено.

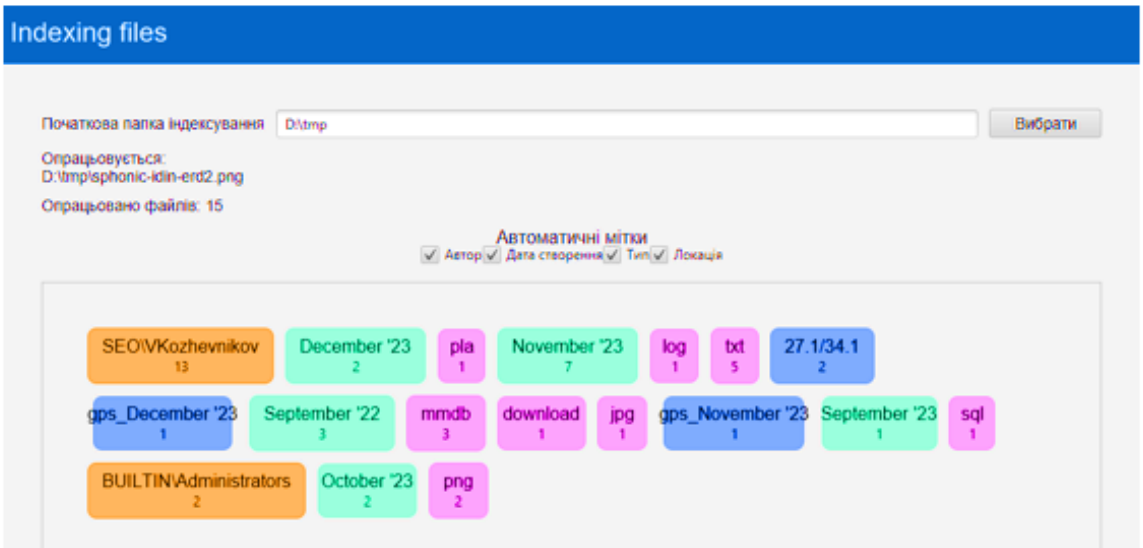

Рисунок 4.8 – Завдання та результат виконання тест-кейсу №1

Тест-кейс — це документ, який містить інструкції та умови для виконання

конкретного тестування програмного продукту. Кожен тест-кейс описує конкретний сценарій або випадок тестування, включаючи вхідні дані, очікувані результати та умови виконання. Тест-кейси є ключовим елементом процесу тестування програмного забезпечення та допомагають забезпечити високу якість продукту.

Було розроблено декілька тест-кейсів щоб перевірити основну функціональність застосунку. Тест-кейс №1 перевіряє функціональність індексування файлів. Мета тесту підготувати папку з файлами, зробити її індексування та перевірити правильність створення міток га основі метаданих файлів.

На рис. 4.9 зображено умову та результати виконання тест-кейсу №2.

Тест кейс №2. Перевірка пошуку дублікатів

Кроки виконання:

1. Має бути виконано індексування файлів в папці d:\tmp
2. Вибрати розділ «Пошук дублікатів»
3. Вибрати початкову папку пошуку: d:\tmp та натиснути кнопку «Почати»
4. Дочекатися завершення та перевірити результати

Очікувані результати:

D:\tmp\folder-with-duplicate\another-file-and-type-name.txt

D:\tmp\gps-test.jpg

Актуальний результат: Тест пройдено.

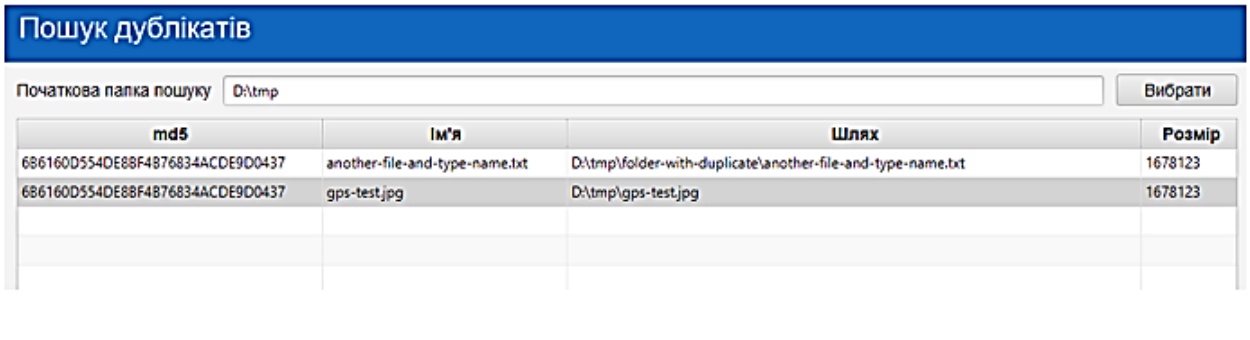

Рисунок 4.9 – Завдання та результат виконання тест-кейсу №2

Завданням цього тест-кейсу є перевірити функцію пошуку дублікатів. Для цього в папці, яка індексувалась в тест-кейсі №1, було розміщено два файли з

різним ім’ям та розташуванням, але з однаковим контентом. Дублікати було успішно знайдено, функціональність пошуку дублікатів файлів працює вірно.

Тест-кейс №3 розроблено, щоб перевірити функціональність пошуку файлів. На рис. 4.10 зображено завдання та результат виконання тест-кейсу №3. Для перевірки функції пошуку використовується набір файлів з попередніх тест-кейсів. Запит пошуку складається з ключового слова та мітки створеної на основі дати. Тест успішно пройдено.

Тест кейс №3. Перевірка пошуку файлів
Кроки виконання:
1. Має бути виконаний попередній тест-кейс 2. Вибрати розділ «Пошук файлів» 3. Вибрати мітку «October ‘23» та у полі пошуку ввести «sphonic» та натиснути кнопку «Почати» 4. Дочекатися завершення та перевірити результати
Очікувані результати:
D:\tmp\sphonic-idin-erd2.png D:\tmp\sphonic-idin-erd.png
Актуальний результат: Тест пройдено

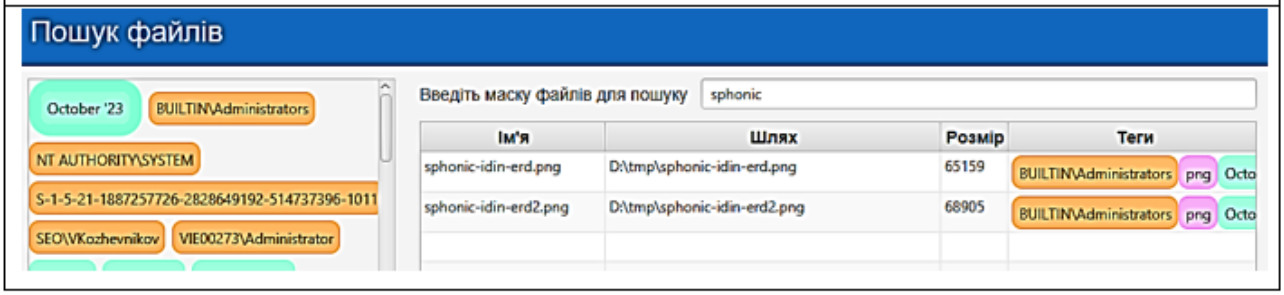

Рисунок 4.10 – Завдання та результат виконання тест-кейсу №3

Завдання та умова тест-кейсу №4 зображено на рис. 4.11. Тест-кейс має важливу мету – перевірити функцію видалення файлу та синхронізації індексів після цього. Для цього необхідно з тестової папки видалити кілька файлів і запустити ще раз індексування файлів. У журналі виконання можна побачити відповідні повідомлення, що індекси видалених файлів очищено. Щоб остаточно впевнитись, необхідно скористатись функцією пошуку файлів і переконатись,

що видалених файлів немає в результаті пошуку.

Тест кейс №4. Перевірка синхронізації при видаленні файлів
Кроки виконання:
1. Має бути виконаний попередній тест-кейс 2. Видалити файли D:\tmp\sphonic-idin-erd2.png та D:\tmp\folder-with-duplicate\another-file-and-type-name.txt 3. Виконати всі кроки з тест-кейсу№1 4. Повернутись в головне меню та вибрати розділ «Пошук файлів» 5. У полі пошуку ввести «sphonic-idin-erd2» та натиснути кнопку «Почати» 6. Дочекатися завершення та перевірити результати 7. Повторити кроки 5 та 6 для another-file-and-type
Очікувані результати:
Жодного файлу не повинно бути знайдено
Актуальний результат: Тест пройдено
: Знадено індекси для видалення: 2 : FileIndex видалено : FileIndex(id=164255, name=another-file-and-type-name.txt, dateCreation=2023-12-01 01:09:01.0 : FileIndex видалено : FileIndex(id=164267, name=sphonic-idin-erd2.png, dateCreation=2023-10-10 15:24:22.0, dateUpd

Рисунок 4.11 – Завдання та результат виконання тест-кейсу №4

Помилок при перевірці реалізованих функцій застосунку методом «чорної скриньки» не виявлено.

4.3 Результати експериментальних досліджень розроблених методів

Було проведено дослідження швидкості пошуку файлів розробленим програмним застосунком в різних умовах та порівняння з використанням команди Get-ChildItem з фреймворку PowerShell. Команда (cmdlet) Get-ChildItemChildItem є однією з базових та потужних команд в оболонці Windows PowerShell. Ця команда використовується для отримання переліку об'єктів (файлів, папок, інших об'єктів) в зазначеній директорії. Було відібрано декілька папок з вкладеними папками та різноманітними файлами, загальна кількість файлів у папках складала приблизно: 200, 1500, 5000, 20000, 60000, 200000,

600000 файлів.

Перед початком пошуку необхідно виконати індексування папок, в яких необхідно виконувати пошук. Під час індексування обробляються метадані файлів, такі як назва, розмір, дата створення, дата останньої зміни, тип файлу, автор, гео-теги. Після процесу індексування, алгоритм пошуку може взаємодіяти з індексами властивостей та міток, уникнувши прямого доступу до файлової системи.

Порівняння швидкості пошуку файлу (групи файлів) виконувалось послідовно, в один і той самий час. Запит пошуку містив частину імені файлу та початкову папку для пошуку, розбиття на сторінки не застосовувалось. Проводилось кілька замірів, обраховувалось середнє значення. Результати вимірювання наведено на рис. 4.12, де вісь «X» відображає кількість файлів, а вісь «Y» – час витрачений на пошук відображає.

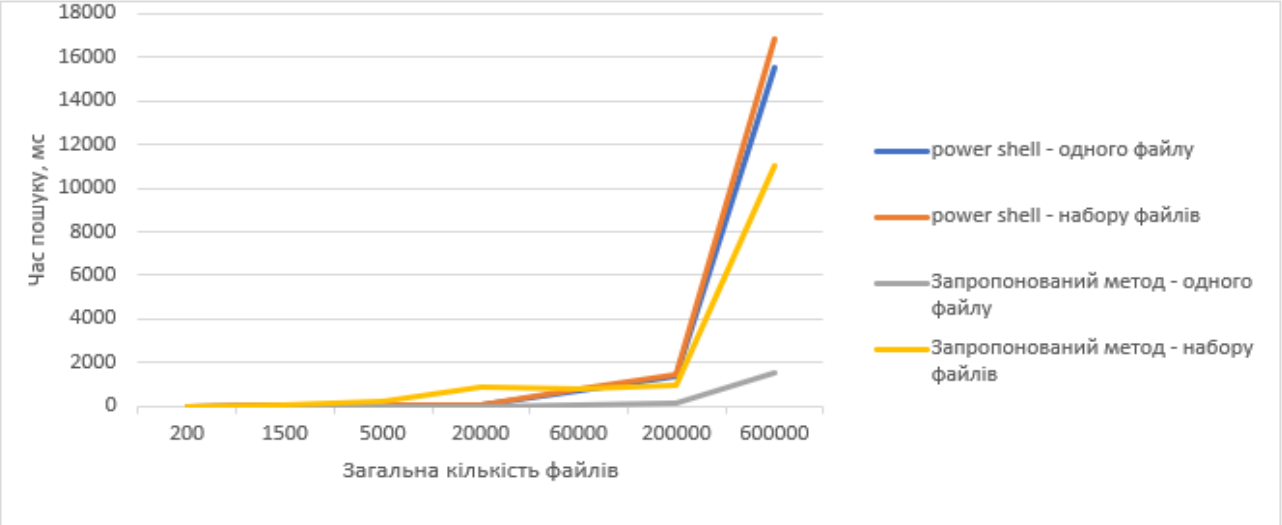

Рисунок 4.12 – Порівняння часу пошуку файлів

З графіків на рисунку 4.6 видно, що розроблений застосунок доцільніше використовувати, коли загальна кількість файлів досить велика. При кількості файлів менше 1000, запропонований метод працює на рівні команди (cmdlet) Get-ChildItem. При виконанні пошуку у папці з загальною кількістю файлів більше 1000, час пошуку запропонованого методу в середньому втричі менший за конкурента. Коли загальна кількість файлів перевищує 50 000 файлів, час

пошуку стає в 10 разів менший за PowerShell. Це стосується випадків, коли результат запиту містить тільки один унікальний файл.

У випадках, коли результат пошуку містить групу файлів, запропонований метод стає ефективнішим за конкурента при кількості файлів у папці більше 60 000, час пошуку у 1,5-2 рази менший за пошук командою Get-ChildItem. До цієї межі пошук запропонованим методом виконується у 3-5 разів повільніше за конкурента.

4.4 Інструкція користувача програмного застосунку

Після запуску застосунку користувач бачить головне меню додатку та має можливість обрати операцію для виконання.

Вибір операції «Індексування файлів» відкриває діалог для вибору початкового каталогу для індексування файлів та інтерфейс для запуску процесу індексування. Скріншот вікна зображено на рис. 4.13.

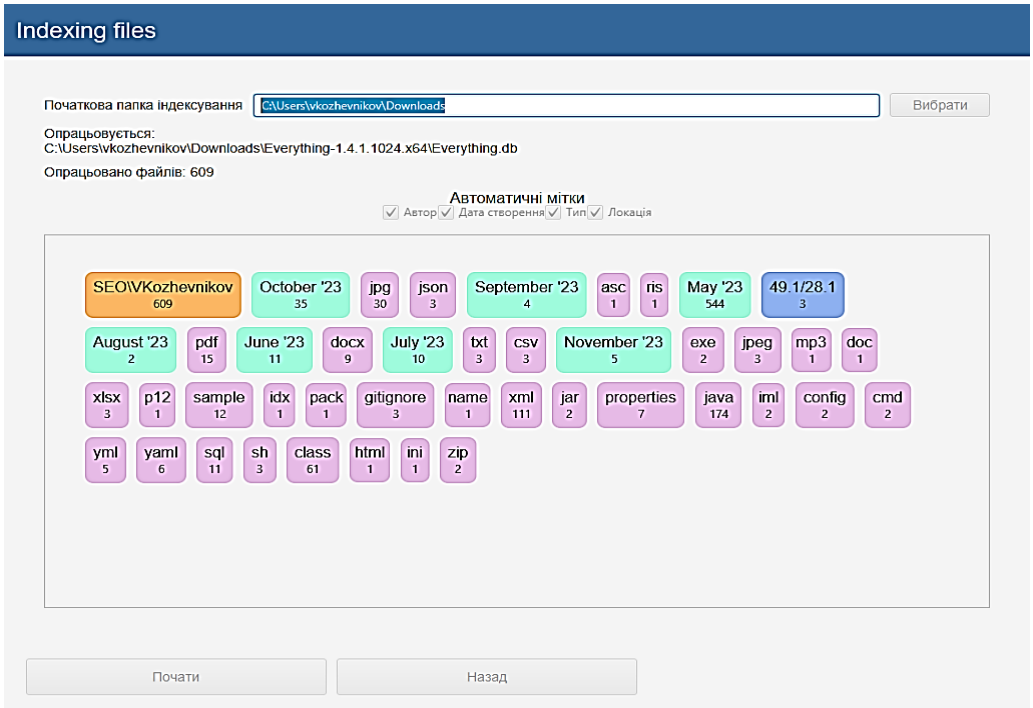

Рисунок 4.13 – Вікно індексування файлів

У вікні є панель з чек-боксами міток, які будуть створюватись під час індексування даних. Є чотири типи міток: автор, дата створення, тип та локація. Мітка типу «автор» створюється на основі атрибуту файлу власник, створюється

індекс з іменами користувачів. Мітка «дата» створюється на основі дати створення файлу, має формат місяць-рік і надає можливість групувати файли за однаковими місяцями створення. Мітка «тип» використовується для групування файлів за типом файлу. Мітка «локація» створюється на основі gps метаданих: довготи і широти. Налаштовано створювати мітки з кроком 0,1˚ довготи та 0.1˚ широти. Це дозволяє згрупувати файли по координатам в квадраті розміром приблизно 10 км на 10 км.

Після запуску процесу індексування, в верхній частині вікна вказується назва та шлях файлу, шо опрацьовується. Також поруч можна побачити кількість опрацьованих файлів. Основну частину вікна займає панель з мітками, які створюються під час індексування. Поруч з назвою мітки вказується кількість файлів, які мають цю мітку. Колір мітки залежить від типу мітки. Якщо під час процесу індексування створено багато міток, то з’являється вертикальна прокрутка для управління панеллю міток. У вікні, поруч з кнопкою «Почати» є кнопка «Назад» для повернення в головне меню.

На рисунку 4.14 зображено вікно пошуку файлів.

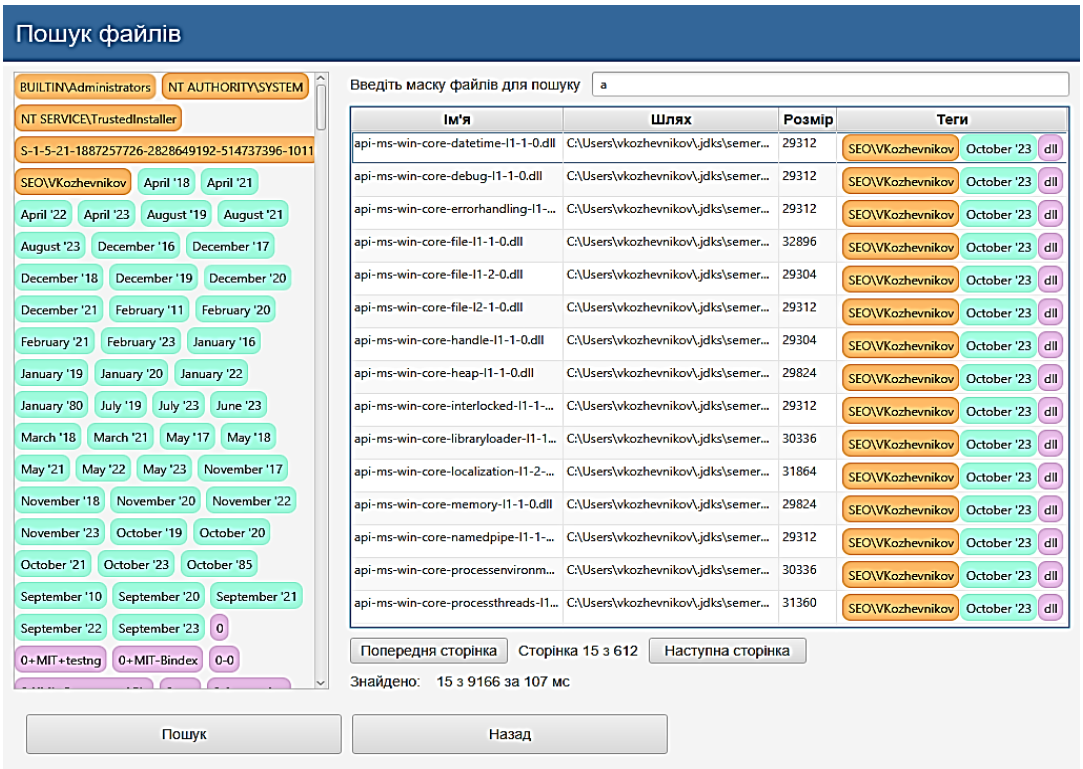

Рисунок 4.14 – Вікно пошуку файлів

Операція «Пошук файлів» відкриває діалог для пошуку файлів за частиною імені та мітками. Вікно розділене на кілька панелей. Зліва розташовано список доступних міток. При виборі мітки або декількох міток, виводяться файли які пов’язані з ними. Фільтр міток працює за умовою «та». Вгорі розташовано поле для введення фрагменту імені файлу за яким буде відбуватися пошук. При введені фрагменту, за замовчанням відбувається порівняння фрагменту з початком строки імені. Є можливість використовувати символ %, який замінятиме один або декілька символів. По центру, основну частину вікна займає таблиця з результатами. Таблиця містить назву файлу, шлях, розмір та список міток файлу. Під таблицею результатів є кнопки керування: «Попередня сторінка» та «Наступна сторінка». Внизу вікна є кнопки «Пошук» та «Назад».

Операція «Пошук дублікатів» відкриває вікно пошуку дублікатів файлів у заданій папці (рис. 4.15).

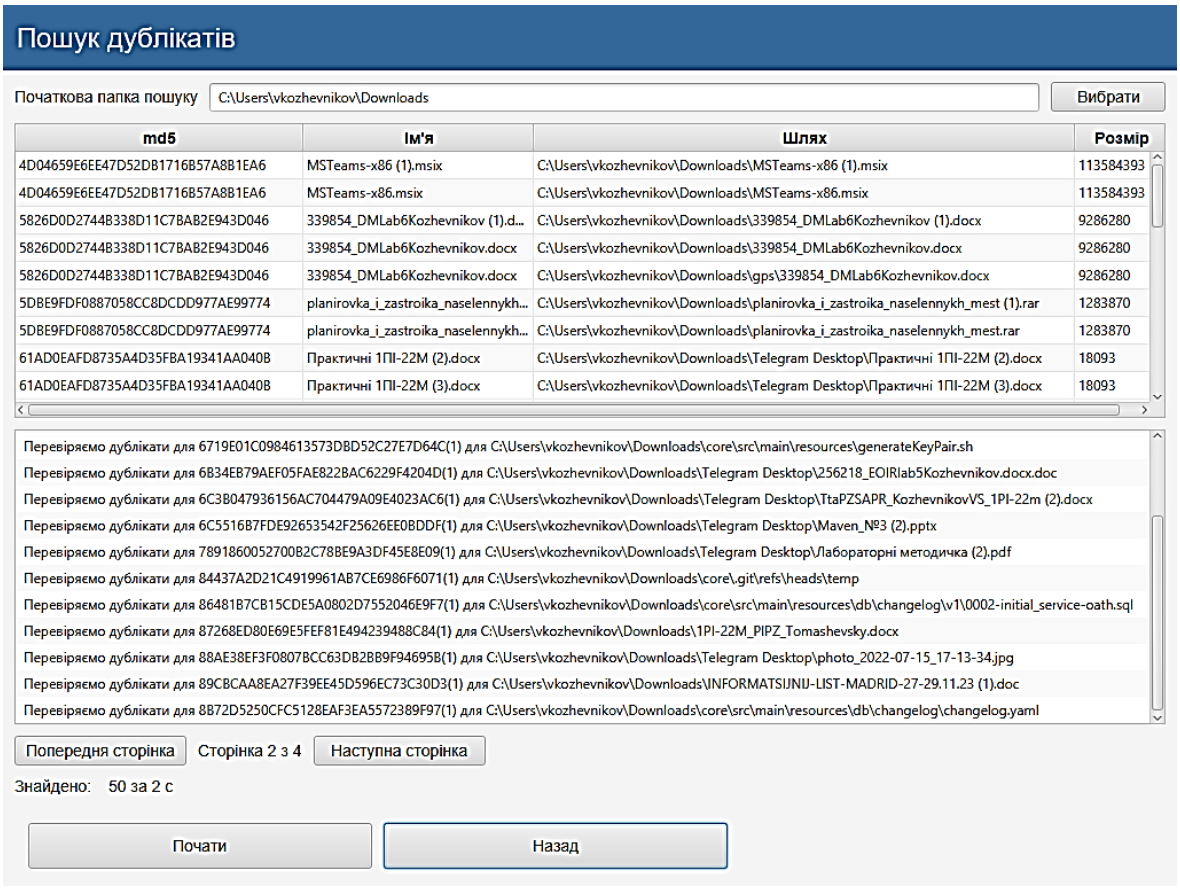

Рисунок 4.15 – Вікно пошуку дублікатів файлів

Вгорі вікна розташоване поле та кнопка для введення початкової папки пошуку. Нижче розташована таблиця результатів пошуку. Таблиця містить хеш-

суму md5, ім’я, шлях та розмір файлу. Під таблицею розташоване текстове поле в якому відображається журнал виконання пошуку та порівняння дублікатів. Нижче розташовані кнопки керування таблицею результатів: «Попередня сторінка» та «Наступна сторінка». Під кнопками є стрічка яка показує загальну кількість знайдених дублікатів хеш-сум. Внизу вікна розташовані кнопки «Почати» та «Назад».

4.5 Висновки

Для перевірки роботи функцій програмного застосунку було проведено тестування з використанням методу «білої скриньки» та методу «чорної скриньки». Було розроблено автоматизовані юніт-тести та складено і виконано набір тест-кейсів основних функцій застосунку.

Для визначення ефективності функції пошуку було проведено дослідження швидкості пошуку файлів в різних умовах і порівняно з аналогом.

Також було описано процес роботи з застосунком та розкрито основні функції реалізованого застосунку.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Під час виконання роботи «Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування» було створено програмний застосунок для пошуку файлів. Реалізація застосунку покликана спростити користувачу роботу з впорядкування файлів. Планується, що програмним продуктом користуватимуться інші споживачі. Перед поширенням програмного продукту необхідно провести аудит науково-технічного рівня та комерційного потенціалу.

Для проведення оцінювання науково-технічного рівня було залучено трьох незалежних експертів – працівників ТОВ «Onseo»: Цибульського Максима, Руслана Миколайчука та Миколу Півня. Оцінювання здійснювалось за 5-ти бальною шкалою за 12-ма критеріями, які наведено в табл. 5.1 [30].

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу

Критерії оцінювання за 5-ти бальною шкалою					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Критерії оцінювання за 5-ти бальною шкалою					
Кри-терій	0	1	2	3	4
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів, отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати аудиту комерційного потенціалу розробки експертами вказано в таблиці 5.2.

Таблиця 5.2 – Результати аудиту комерційного потенціалу розробки

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали		
1. Технічна здійсненність концепції	3	3	3
2. Ринкові переваги (наявність аналогів)	2	2	3
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	3	3	4
5. Ринкові переваги (експлуатаційні витрати)	3	4	4
6. Ринкові перспективи (розмір ринку)	1	2	2
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	38	40	42
Середньоарифметична сума балів *СБс*	40		

Обрахувавши середньоарифметичну суму балів результатів аудиту, визначимо рівень комерційного потенціалу розробки, скориставшись значеннями таблиці 5.3.

Маємо суму балів – 40, що відповідає рівню «вище середнього».

Порівняємо новий продукт з доступними аналогами. Для порівняння обрано «TagSpaces» - це програма для організації та впорядкування файлів за допомогою тегів. Головними недоліками аналога є висока ціна, швидкість

пошуку при великій кількості файлів, відсутність пошуку дублікатів.

Таблиця 5.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 - 10	Низький
11 - 20	Нижче середнього
21 - 30	Середній
31 - 40	Вище середнього
41 - 48	Високий

У новому продукті ці недоліки вирішуються додаванням нового функціоналу, покращенням швидкості пошуку за рахунок використання індексованого пошуку та доступнішою ціною.

Виконаємо порівняння технічних показників нового продукту та аналога, дані занесемо в таблицю 5.4.

Як бачимо з таблиці 5.4 у аналога відсутній достатній функціонал. Новий продукт має суттєво удосконалений пошук файлів та дублікатів, має зручніший інтерфейс. Важливою перевагою також є доступніша ціна.

Виконаємо аналіз якості продукту, оскільки це є найбільш ефективним способом задоволення вимог споживачів, і порівняємо його з аналогом. Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.4.

$$q_i = \frac{P_{Hi}}{P_{Bi}} \tag{5.1}$$

або

$$q_i = \frac{P_{Bi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, P_{Bi} – числові значення і-го параметру відповідно нового і базового виробів.

Таблиця 5.4 – Основні технічні показники нового продукту та аналога

Показники	Новий продукт	Аналог	Відносний показник якості	Коефіцієнт вагомості параметра
Зручний інтерфейс, бали	5	4	1,25	25%
Вартість продукту, грн	600	1 482	2,47	25%
Робота з тегами, бали	4	5	0,8	20%
Швидкість пошуку, бали	5	3	1,67	15%
Пошук дублікатів, бали	5	2	2,5	15%

$$q_1 = \frac{5}{4} = 1{,}25;$$

$$q_2 = \frac{1482}{600} = 2{,}47;$$

$$q_3 = \frac{4}{5} = 0{,}8;$$

$$q_4 = \frac{5}{3} = 1{,}67;$$

$$q_5 = \frac{5}{2} = 2{,}5.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \propto_i \qquad (5.3)$$

$$К_{я.в.} = 1{,}25 \cdot 0{,}25 + 2{,}47 \cdot 0{,}25 + 0{,}8 \cdot 0{,}2 + 1{,}67 \cdot 0{,}15 + 2{,}5 \cdot 0{,}15 = 1{,}72$$

Визначивши відносний коефіцієнт показника якості, який перевищує одиницю, можна зробити висновок, що нова розробка виявляється якіснішою в порівнянні з базовим товаром-конкурентом.

5.2 Прогнозування витрат на науково дослідні роботи

Витрати, пов'язані із здійсненням науково-дослідної роботи, розподіляються за наступними категоріями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт,

інші витрати та накладні витрати.

Виконаємо обрахунок витрат на заробітну плату виконавців проекту.

Обчислимо основну заробітну *Зо*, за формулою:

$$З_О = \frac{М}{Т_р} \times t \text{ (грн)} \quad (5.4)$$

де М – місячний посадовий оклад конкретного розробника, грн;

Тр – число робочих днів в місяці - 22;

t – число днів роботи розробника - 66.

Обрахуємо заробітну плату керівника з місячним окладом 55 000 грн і кількістю робочих днів – 55; заробітну плату розробника з місячним окладом 49 500 грн і кількістю робочих днів – 44; та заробітну плату тестувальника з місячним окладом – 37400 грн і кількістю робочих днів – 22. Число місяців роботи над проектом - 3. Обраховані значення занесемо в таблицю 5.5.

Обрахуємо додаткову заробітну плату розробників $З_д$. Розрахуємо її як 11% від основної заробітної плати:

$$З_д = З_о \cdot 0{,}11 \text{ (грн)} \quad (5.5)$$

$$З_д = 273\,900{,}00 \cdot 0{,}11 = 30\,129{,}00 \text{ (грн)}$$

Таблиця 5.5 - Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник роботи	55 000,00	2 500,00	55	137 500,00
Розробник	49 500,00	2 250,00	44	99 000,00
Тестувальник	37 400,00	1 700,00	22	37 400,00
Всього				273 900,00

Згідно із чинним законодавством, відрахування на заробітну плату, а саме Єдиний соціальний внесок, складають 22% від суми основної та додаткової заробітної плати:

$$H_{зп} = \left(З_о + З_д\right) \cdot \frac{\beta}{100} \text{ (грн)} \quad (5.6)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %.

Оскільки ця діяльність належить до бюджетної сфери, ставка єдиного внеску на загальнообов'язкове державне соціальне страхування становитиме 22%, отже:

$$H_{зп} = (273\,900{,}00 + 30\,129{,}00) \cdot \frac{22}{100} = 66\,886{,}38 \text{ (грн)}$$

Розрахунок витрат на матеріали (М) та комплектуючі (К), що були використані на даному етапі роботи, виконується окремо для кожного типу матеріалів за допомогою формули:

$$М = \sum_1^n Н_і \cdot Ц_і \cdot К_і - \sum_1^n В_і \cdot Ц_в \text{ (грн)} \quad (5.7)$$

де $Н_і$ – витрати матеріалу *і*-го найменування, кг;

$Ц_і$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_і$ – коефіцієнт транспортних витрат, К*і* = (1,1…1,15);

$В_і$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Інформацію про використані матеріали зведемо до таблиці 5.6.

Таблиця 5.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Кількість	Вартість витраченого матеріалу, грн.
Блокнот YES A5 80 аркушів спіраль з гумкою Minions Banana	199,90	3	599,70
Папір офісний А4, Maestro Standart+, клас В, 80 г/м2, 500 л	179,90	2	359,80
ДОШКА МАРК. 65х100 UKRBOARDS	1599,00	1	1 599,00
Набір маркерів для дошок 4шт+губка 8800-84 Buromax	129,90	1	129,90
Ручка масляна Round Stic BIC	11,90	6	71,40
Накопичувач USB 16Gb GOODRAM UUN2 Unity	179,00	2	358,00
Всього			3 117,80
З врахуванням коефіцієнта транспортування			3 429,58

Витрати, пов'язані з програмним забезпеченням для наукової роботи, охоплюють витрати на створення та придбання спеціалізованих програмних засобів і необхідного програмного забезпечення для проведення дослідження.

Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{пргі} \cdot К_і, \quad (5.8)$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{пргі}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо ($К_і$ = 1,10…1,12).

k – кількість найменувань програмних засобів.

Отримані результати занесемо в таблицю 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування ПЗ	Кількість, ліцензія	Ціна за одиницю, грн	Вартість
IntelliJ IDEA Ultimate	1	6 422,00	6 422,00
Microsoft Office 2021 Pro Plus	1	2 190.00	2 190,00
Всього			8 612,00

Розрахунок амортизації для обладнання та приміщення, використовуваного у процесі розробки, виконується за формулою:

$$A = \frac{Ц}{T_{кор}} \cdot \frac{T}{12} \text{ [грн]} \quad (5.9)$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$T_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Розрахуємо амортизацію за формулою 5.9 на основні засоби вартістю понад 20 000 грн і занесемо в таблицю 5.8.

Таблиця 5.8 – Амортизація обладнання

Найменування обладнання	Ціна, грн	Амортизація, грн
Ноутбук Lenovo IdeaPad 5 16ABR8 (82XG005ARA) Cloud Grey	26 999,00	2 249,92
БФП кольорового друку Epson L3201	8 625,00	718,75
Маршрутизатор інтернет WiFi5 Xiaomi Router AC1200	1 399,00	116,58
Всього		3 085,25

$$А_Н = \frac{26999}{3} \cdot \frac{3}{12} = 2249{,}92;$$

$$А_Б = \frac{8625}{3} \cdot \frac{3}{12} = 718{,}75;$$

$$А_М = \frac{1399}{3} \cdot \frac{3}{12} = 116{,}58.$$

В рамках статті «Паливо та енергія для науково-виробничих цілей» враховуються витрати на усі види палива та енергії, які використовуються безпосередньо з технологічною метою у проведенні наукових досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_і}, \quad (5.10)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}2 \cdot 360 \cdot 7{,}65 \cdot 0{,}5}{0{,}8} + \frac{0{,}04 \cdot 60 \cdot 7{,}65 \cdot 0{,}5}{0{,}8} + \frac{0{,}02 \cdot 360 \cdot 7{,}65 \cdot 0{,}5}{0{,}8} =$$

$$= 390{,}15 \text{ грн}$$

Загальновиробничі витрати ($В_{нзв}$) охоплюють витрати на управління організацією, відрядження, утримання, ремонт та експлуатацію основних засобів, а також витрати на опалення, освітлення, водопостачання, охорону праці та інші витрати. Загальновиробничі витрати (Внзв) можуть бути оцінені як (100...150)% від суми основної заробітної плати розробників та робітників, які брали участь у даному проекті.

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%} \quad (5.11)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 273\,900 \cdot \frac{100}{100\%} = 273\,900{,}00 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 273\,900{,}00 + 30\,129{,}00 + 66\,886{,}38 + 3\,429{,}58 + 8\,612{,}00 +$$
$$+3\,085{,}25 + 390{,}15 + 40363{,}63 + 273\,900{,}00 = 700\,695{,}99 \text{ грн}$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta'} \qquad (5.12)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,9. Звідси:

$$ЗВ = \frac{700\,695{,}99}{0{,}9} = 778\,551{,}1 \text{ грн}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

В даному розділі проведено кількісний прогноз, щодо очікуваного корисного ефекту або прибутку, який може бути здобутий у майбутньому від впровадження результатів проведеної наукової роботи. Для кожного з років, коли передбачається досягнення позитивних результатів від впровадження розробки, буде розраховано збільшення чистого прибутку підприємства ($\Delta П_і$) за відповідною формулою

$$\Delta П_і = \sum_1^n (\Delta Ц_о \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \qquad (5.13)$$

де ΔЦ0 – покращення основного оціночного показника від впровадження

результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_о$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки;

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ – коефіцієнт, який враховує рентабельність продукту. ρ = 0,25;

ν – ставка податку на прибуток. У 2023 році – 18%.

Припустимо, що при впроваджені результатів наукової розробки покращується якість, що дозволяє підвищити ціну його реалізації на 100 грн. Кількість користувачів також збільшиться: протягом першого року на 5000, протягом другого року – на 7000, протягом третього року на 10000. Реалізація продукції до впровадження розробки складала 20000 користувачів, а її ціна 600 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [100 \cdot 20\,000 + (600 + 100) \cdot 5\,000] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right) = \\ = 939\,207{,}5 \text{ грн}$$

$$\Delta П_2 = [100 \cdot 20\,000 + (600 + 100) \cdot (5\,000 + 7\,000)] \times \\ \times 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right) = 1\,775\,956 \text{ грн}$$

$$\Delta П_3 = [100 \cdot 20\,000 + (600 + 100) \cdot (5\,000 + 7\,000 + 10\,000)] \times \\ \times 0{,}833 \cdot 0{,}25 \cdot \left(1 - \frac{18}{100}\right) = 2\,971\,311 \text{ грн}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ} \tag{5.14}$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}}$ = 2…5).

$$PV = 2 \cdot 778\,551{,}1 = 1\,557\,102{,}2$$

Розрахуємо абсолютну ефективність вкладених інвестицій $\text{Е}_{\text{абс}}$ згідно наступної формули:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV) \tag{5.15}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$\text{ПП} = \sum_1^{\text{Т}} \frac{\Delta\text{П}_\text{і}}{(1+\tau)^\text{і}} \tag{5.16}$$

де $\Delta\text{П}_\text{і}$ - збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т - період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

$\tau -$ ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2; t – період часу (в роках).

$$ПП = \frac{939\ 207,5}{(1+0,2)^1} + \frac{1\ 775\ 956}{(1+0,2)^2} + \frac{2\ 971\ 311}{(1+0,2)^3} = 3\ 735\ 484,38 \text{ грн}$$

$$Е_{абс} = (3\ 735\ 484,38 - 1\ 557\ 102,2) = 2\ 178\ 382,18 \text{ грн}$$

Оскільки $Е_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій. Для цього користуються формулою:

$$Е_В = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 \qquad (5.17)$$

$Т_ж$ - життєвий цикл наукової розробки, роки.

$$Е_В = \sqrt[з]{1 + \frac{2\ 178\ 382,18}{1\ 557\ 102,2}} - 1 = 0.339 = 34\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f \qquad (5.18)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2018 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0{,}18 + 0{,}5 = 0{,}23$$

Так як $E_в > \tau_{min}$ то інвестор має бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0{,}34} = 2{,}95 \text{ роки}$$

Так як $T_{ок}$ < 3-х років, то фінансування даної наукової розробки є доцільним.

5.5 Висновки

Була проведена оцінка комерційного потенціалу розробки методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування. Порівняння з аналогічними розробками показало, що новий підхід вищої якості та конкурентоспроможний, як з технічної, так і з економічної точки зору.

Вартість науково-дослідної роботи на кожному етапі оцінюється у 700 695,99 грн. Загальні витрати на виконання та впровадження результатів дослідження складають 778 551,1 грн. Прогнозується, що інвестиції у проект повністю окупляться за період до 3-х років при очікуваному прибутку у розмірі 3 735 484,38 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено методи та алгоритми для підвищення ефективності пошуку файлів та їх впорядкування, працездатність яких продемонстрована розробленим програмним застосунком для пошуку та впорядкування файлів.

Проведено аналіз сучасного стану питання та порівняння існуючих аналогів програмного забезпечення для пошуку та впорядкування файлів. Розглянуто переваги та недоліки цих засобів, виконано порівняння характеристик. Визначено аспекти, які можна покращити, зокрема збільшити швидкість пошуку та додати механізм створення міток, які будуть використовуватись як підказки при побудові пошукового запиту, так і для оптимізації швидкості пошуку.

Проаналізовано існуючі методи пошуку та індексування файлів. Досліджено основні методи пошуку дублікатів файлів. У результаті аналізу було обрано напрями удосконалення методів для реалізації необхідної функціональності.

Було розроблено метод комплексного індексування властивостей файлів, що допомагає швидко та ефективно знаходити, фільтрувати та групувати файли за заданими параметрами. Зокрема, за назвою файлу, розміром, датою створення, датою останньої зміни, типом файлу, автором, гео-тегами, md5 сумою.

Удосконалено метод пошуку дублікатів файлів, в якому для швидкого пошуку використовується порівняння хеш-суми, а для детального – побітове порівняння вмісту.

Запроектовано структуру індексів та обрано БД MySQL для зберігання індексів. Властивості вибраної БД вдало задовольняють вимоги задачі: структурованість даних, спеціалізовані операції пошуку, фільтрації та агрегування даних, індексація, нормалізація, зв'язки між даними та підтримка великих обсягів даних.

Розроблено алгоритми та блок-схеми алгоритмів індексування

властивостей файлів та пошуку дублікатів файлів.

Виконано варіантний аналіз і обґрунтування вибору засобів та середовища розробки для реалізації програмного застосунку. За результатами аналізу було обрано мову програмування Java та середовище розробки JetBrains IntelliJ Idea. Виконано програмну реалізацію застосунку для пошуку та впорядкування файлів.

Для перевірки роботи функцій програмного застосунку було проведено тестування з використанням методу «білої скриньки» та методу «чорної скриньки». Було розроблено автоматизовані юніт-тести та складено і виконано набір тест-кейсів основних функцій застосунку.

Для визначення ефективності функції пошуку було проведено дослідження швидкості пошуку файлів в різних умовах і виконано порівняння аналогом.

Була проведена оцінка комерційного потенціалу розробки, яка показала, що новий підхід вищої якості та конкурентоспроможний, як з технічної, так і з економічної точки зору.

Задачі магістерської кваліфікаційної роботи виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Duarte F. Amount of Data Created Daily [Електронний ресурс] / Fabio Duarte // Exploding topics. – Режим доступу: https://explodingtopics.com/blog/data-generated-per-day (дата звернення: 30.12.2023). – Назва з екрана.

2. Chin M. File not found [Електронний ресурс] / Monica Chin // The Verge. – Режим доступу: https://www.theverge.com/22684730/students-file-folder-directory-structure-education-gen-z (дата звернення: 30.12.2023). – Назва з екрана.

3. Senior K. Why Is File Management Important? Here's What To Tell Your Team [Електронний ресурс] / Kevin Senior // Glasscubes. – Режим доступу: https://www.glasscubes.com/why-is-file-management-important/ (дата звернення: 30.12.2023). – Назва з екрана.

4. Кожевніков В.С. Підвищення ефективності методів пошуку файлів та їх упорядкування / Кожевніков В.С., Романюк О.В. // Електронні інформаційні ресурси: створення, використання, доступ : Міжнар. науково-практ. Інтернет-конф., Суми/Вінниця, 20–21 листоп. 2023 р. – Суми/Вінниця, 2023. – С. 121–122.

5. Кожевніков В.С. Дослідження ефективності методів пошуку файлів / Кожевніков В.С., Романюк О.В. // XI міжнародної науково-практичної конференція EUROPEAN SCIENTIFIC CONGRESS – 2023, Мадрид, 27-29 листоп. 2023р. – Мадрид, 2023. – Режим доступу: https://sci-conf.com.ua/wp-content/uploads/2023/11/EUROPEAN-SCIENTIFIC-CONGRESS-27-29.11.23.pdf (дата звернення: 30.12.2023). – Назва з екрана.

6. Hachman M. Can't find that file? These advanced Windows Search tips can help [Електронний ресурс] / Mark Hachman // PCWorld. – Режим доступу: https://www.pcworld.com/article/1341497/how-to-find-what-youre-looking-for-with-windows-search.html (дата звернення: 30.12.2023). – Назва з екрана.

7. Orosz A. Why I'm still using TagSpaces and so should you [Електронний ресурс] / Attila Orosz // Medium.com. – Режим доступу: https://attilaorosz.medium.com/why-im-still-using-tagspaces-and-so-should-you-b384bc2d6f9b (дата звернення: 30.12.2023). – Назва з екрана.

8. Aprilliant A. How to Find a File System Efficiently Using Breadth-First Search and Depth-First Search [Електронний ресурс] / Audhi Aprilliant // Medium.com. – Режим доступу: https://audhiaprilliant.medium.com/how-to-find-a-file-system-efficiently-using-breadth-first-search-and-depth-first-search-3881ec26320a (дата звернення: 30.12.2023). – Назва з екрана.

9. Бхаргава А. Грокаємо алгоритми: Ілюстрований посібник для програмістів і допитливих / Адітья Бхаргава. Пер. із англ. Олександра Медведа. - Київ : ArtHuss, 2023. - 288 с.

10. Tariq N. A Comparison Of Popular File Indexing Software: Which One Is Right For You? [Електронний ресурс] / Nayab Tariq // Medium.com. – Режим доступу: https://nayab-tariq.medium.com/a-comparison-of-popular-file-indexing-software-which-one-is-right-for-you-122f5e88dede (дата звернення: 30.12.2023). – Назва з екрана.

11. Klein C. Find duplicate photos and other files [Електронний ресурс] / Carsten Klein // Medium.com. – Режим доступу: https://towardsdatascience.com/find-duplicate-photos-and-other-files-88b0d07ef020 (дата звернення: 30.12.2023). – Назва з екрана.

12. Nichter D. Efficient MySQL Performance: Best Practices and Techniques. / Daniel Nichter. – 1-ше вид. – Sebastopol : O'Reilly Media, 2021. – 335 p.

13. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. / Martin Fowler. – 3-те вид. – Boston : Addison-Wesley Professional, 2003. – 208 p.

14. Ramalho L. Fluent Python / Luciano Ramalho. – 2-ге вид. – Sebastopol : O'Reilly Media, 2022. – 1012 p.

15. Troelsen A, Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming / Philip Japikse. – 11-те вид. – New York : Apress, 2022. – 1705 p.

16. Doxsey C. Introducing Go: Build Reliable, Scalable Programs. / Caleb Doxsey. – 1-ше вид. – Sebastopol : O'Reilly Media, 2016. – 122 p.

17. Horstmann C. Core Java Volume I--Fundamentals (Core Series). / Cay Horstmann. – 12-те вид. – Florida : Oracle Press, 2021. – 944 p.

18. Horstmann C. Core Java Volume II--Fundamentals (Core Series). / Cay Horstmann. – 12-те вид. – Florida : Oracle Press, 2022. – 944 p.

19. Hagos T. Beginning IntelliJ IDEA: Integrated Development Environment for Java Programming. / Ted Hagos. – 1-ше вид. – New York : Apress, 2021. – 292 p.

20. Varanasi B. Introducing Maven: A Build Tool for Today's Java Developers / Balaji Varanasi. – 2-ге вид. – New York : Apress, 2019. – 157 p.

21. Walls C. Spring Boot in Action. / Craig Walls. – 1-ше вид. – New York : Manning, 2016. – 264 p.

22. Creating a Spring Boot JavaFX Application with FxWeaver [Електронний ресурс] / RGIELEN.NET. – Режим доступу: https://rgielen.net/posts/2019/creating-a-spring-boot-javafx-application-with-fxweaver/ (дата звернення: 30.12.2023). – Назва з екрана.

23. Sharan K. Learn JavaFX 17: Building User Experience and Interfaces with Java / Kishori Sharan, Peter Späth. – 2-ге вид. – New York : Apress, 2022. - 924 p

24. Tudose C. Java Persistence with Spring Data and Hibernate / Cătălin Tudose. – 1-ше вид. – New York : Manning, 2023. – 616 p.

25. Mohan G. Full Stack Testing. A Practical Guide for Delivering High Quality Software. / Gayathri Mohan. – 1-ше вид. – Sebastopol : O'Reilly Media, 2022 – 406 p.

26. Myers G. The Art of Software Testing / Glenford J. Myers, Corey Sandler, Tom Badgett. - 3-тє вид. – New York : Wiley, 2011. – 256 p.

27. Meszaros G. xUnit Test Patterns: Refactoring Test Code. / Gerard Meszaros. – 1-те вид. – Boston : Addison-Wesley Professional, 2007. – 833 p.

28. Tudose C. JUnit in Action / Cătălin Tudose. – 3-тє вид. – New York : Manning, 2020. – 560 p.

29. Acharya S. Mockito Essentials / Sujoy Acharya. – 1-ше вид. – Sebastopol : O'Reilly Media, 2014. – 214 p.

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач каф. ПЗ
д.т.н., проф. О. Н. Романюк
«19» вересня 2023 р.

Технічне завдання
на магістерську кваліфікаційну роботу « Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. каф. ПЗ Романюк О.В.
"19" 09 2023 р.

Виконав:
_____студент гр.1ПІ-22м Кожевніков В.С.
"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування».

Галузь застосування – офісний менеджмент, освіта, дослідження та наука, власне застосування.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності пошуку та впорядкування файлів за рахунок розробки нових методів та програмних засобів.

Призначення роботи – розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Aprilliant A. How to Find a File System Efficiently Using Breadth-First Search and Depth-First Search [Електронний ресурс] / Audhi Aprilliant // Medium.com. – Режим доступу: https://audhiaprilliant.medium.com/how-to-find-a-file-system-efficiently-using-breadth-first-search-and-depth-first-search-3881ec26320a (дата звернення: 30.12.2023). – Назва з екрана.

2. Creating a Spring Boot JavaFX Application with FxWeaver [Електронний ресурс] / RGIELEN.NET. – Режим доступу: https://rgielen.net/posts/2019/creating-a-spring-boot-javafx-application-with-fxweaver/ (дата звернення: 30.12.2023). – Назва з екрана.

3. Sharan K. Learn JavaFX 17: Building User Experience and Interfaces with Java / Kishori Sharan, Peter Späth. – 2-ге вид. – New York : Apress, 2022. - 924 p

4. Mohan G. Full Stack Testing. A Practical Guide for Delivering High Quality Software. / Gayathri Mohan. – 1-ше вид. – Sebastopol : O'Reilly Media, 2022 – 406 p.

5. Технічні вимоги

Методи пошуку файлів – рекурсивний та індексний; методи пошуку дублікатів файлів – за хеш-сумою та за вмістом; середовища розробки – Intelij Idea 2023 та Microsoft SQL Server; мова розробки – Java; операційна система – Windows 11; вихідні дані – результати пошуку файлів.

6. Конструктивні вимоги

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання, методів пошуку та індексування файлів, порівняння аналогів	20.09.2023 – 29.09.2023
2	Розробка методів та алгоритмів комплексного індексування властивостей файлів та пошуку файлів з використанням міток	02.10.2023 – 13.10.2023
3	Програмна реалізація застосунку	16.10.2023 – 10.11.2023
4	Тестування програмного забезпечення та дослідження ефективності розроблених методів	13.11.2023 – 24.11.2023
5	Економічна частина	27.11.2023 – 01.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

Протокол перевірки МКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування.**

Тип роботи: кваліфікаційна робота
Підрозділ: кафедра програмного забезпечення, ФІТКІ, 1ПІ-22м
Науковий керівник: к.т.н. доц. Романюк О. В.

Unicheck	
Оригінальність	**95%**
Схожість	5%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck.

Автор роботи _____ Кожевніков В.С.

Керівник роботи _____ Романюк О.В.

ДОДАТОК В

Лістинг коду

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>javafx-weaver-samples</artifactId>
        <groupId>net.rgielen</groupId>
        <version>1.3.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.vkozhevnikov.fileanalyzer</groupId>
    <artifactId>gui</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>gui</name>

    <properties>
        <java.level>11</java.level>
        <flyway.version>7.15.0</flyway.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <!-- Import dependency management from Spring Boot -->
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
```

```
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.199</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.vintage</groupId>
    <artifactId>junit-vintage-engine</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.15.3</version>
</dependency>
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
</dependency>
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
</dependency>

<dependency>
    <groupId>net.rgielen</groupId>
    <artifactId>javafx-weaver-spring-boot-starter</artifactId>
    <version>1.3.0</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-jpamodelgen</artifactId>
</dependency>
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
</dependency>
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
<dependency>
    <groupId>com.drewnoakes</groupId>
    <artifactId>metadata-extractor</artifactId>
    <version>2.18.0</version>
</dependency>
```

```
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.flywaydb</groupId>
                    <artifactId>flyway-maven-plugin</artifactId>
                    <version>${flyway.version}</version>
                    <configuration>
                        <table>schema_version</table>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.flywaydb</groupId>
                <artifactId>flyway-maven-plugin</artifactId>
                <configuration>
                    <table>schema_version</table>
                    <url>jdbc:mysql://localhost/file_analyzer_db</url>
                    <user>root</user>
                    <password>mysql</password>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring-boot.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

src/main/resources/application.yml

```
spring:
  application:
    name: file-analyzer
  datasource:
    url: jdbc:mysql://localhost/file_analyzer_db
    username: root
    password: mysql
    driver-class-name: org.mariadb.jdbc.Driver
  jpa:
    show-sql: false
  flyway:
    enabled: true
logging:
  level:
    root: info
    com.vkozhevnikov: debug
```

src/main/resources/db/migration/V1__init.sql

```
CREATE TABLE `file_index`
(
    `id`            bigint(20)    NOT NULL AUTO_INCREMENT,
    `name`          varchar(260)  NOT NULL,
    `date_creation` datetime      NOT NULL,
    `date_update`   datetime      NOT NULL,
    `path`          varchar(1024) NOT NULL,
    `size`          bigint(20)    NOT NULL,
    `type`          varchar(20)   NOT NULL,
    `md5`           varchar(32)   NOT NULL,
    `author`        varchar(64)   NOT NULL,
    `date_indexed`  bigint(20) default 0     NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY `path_idx` (`path`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;


CREATE TABLE `file_location`
(
    `id`            bigint(20) NOT NULL AUTO_INCREMENT,
    `file_index_id` bigint(20) NOT NULL,
    `latitude`      double     NOT NULL,
    `longitude`     double     NOT NULL,
    `gps_date`      datetime DEFAULT NULL,
    PRIMARY KEY (`id`),
    CONSTRAINT `fk_file_index_location_idx` FOREIGN KEY (`file_index_id`)
REFERENCES `file_index` (`id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;


CREATE TABLE `tag`
(
    `id`          bigint(20)  NOT NULL AUTO_INCREMENT,
    `name`        varchar(64) NOT NULL,
    `key`        varchar(64) NOT NULL,
    `description` varchar(128) DEFAULT NULL,
    `type`        varchar(20) NOT NULL,
    `option`      varchar(20)  DEFAULT NULL,
    `color`       varchar(20) DEFAULT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY type_key_idx (`key`, `type`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;

CREATE TABLE `file_tag`
(
    `id`            bigint(20) NOT NULL AUTO_INCREMENT,
    `file_index_id` bigint(20) NOT NULL,
    `tag_id`        bigint(20) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY file_index_tag_idx (`file_index_id`, `tag_id`),
    CONSTRAINT `fk_file_index_file_tag_idx` FOREIGN KEY (`file_index_id`)
REFERENCES `file_index` (`id`),
    CONSTRAINT `fk_tag_file_tag_idx` FOREIGN KEY (`tag_id`) REFERENCES `tag`
(`id`)
) ENGINE = InnoDB
  DEFAULT CHARSET = utf8;
```

src/main/resources/com/vkozhevnikov/fileanalyzer/controller/IndexingWindow.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.CheckBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.FlowPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>
<?import java.net.URL?>
<BorderPane fx:id="indexingWindow" xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.vkozhevnikov.fileanalyzer.controller.IndexingWindow">
    <stylesheets>
        <URL value="@common-style.css" />
    </stylesheets>

    <top>
        <HBox styleClass="header">
            <Text fill="white" style="-fx-font: 24 arial;" text="Індексування
файлів" />
        </HBox>
    </top>


    <center>
        <VBox styleClass="indexingFlowPane">
            <HBox styleClass="paneSpacing"  alignment="CENTER_LEFT">
                <Label text="Початкова папка індексування" />
                <TextField fx:id="filePath" HBox.hgrow="ALWAYS" />
                <Button fx:id="btnChoosePath" text="Вибрати" />
            </HBox>
            <VBox>
                <Label text="Опрацьовується: " />
                <Label fx:id="currentProcessingLabel" />
            </VBox>
            <HBox>
                <Label text="Опрацьовано файлів: " />
                <Label fx:id="filesProcessedLabel" />
            </HBox>

            <BorderPane VBox.vgrow="ALWAYS">
                <top>
                    <StackPane alignment="TOP_CENTER" styleClass="paneSpacing">
                        <Label style="-fx-font: 16 arial;" text="Автоматичні
мітки"/>
                        <FlowPane fx:id="labelsContainer" prefHeight="50"
alignment="CENTER" styleClass="paneSpacingAndPadding">
                            <CheckBox fx:id="tagByAuthor" text="Автор"
selected="true"/>
                            <CheckBox fx:id="tagByDate" text="Дата створення"
selected="true" />
                            <CheckBox fx:id="tagByType" text="Тип"
selected="true" />
                            <CheckBox fx:id="tagByGps" text="Локація"
selected="true" />
                        </FlowPane>
                    </StackPane>
```

```
                </top>
                <center>
                    <ScrollPane prefHeight="300">
                        <FlowPane fx:id="tagsContainer" prefHeight="148.0"
prefWidth="920.0" styleClass="indexingFlowPane">

                        </FlowPane>
                    </ScrollPane>
                </center>
            </BorderPane>
        </VBox>
    </center>

    <bottom>
        <HBox styleClass="footer">
            <Text fill="white" style="-fx-font: 24 arial" />
            <Button fx:id="startButton" prefHeight="40" prefWidth="150"
text="Почати" />
            <Button fx:id="backButton" prefHeight="40" prefWidth="150"
text="Назад" />
        </HBox>
    </bottom>

</BorderPane>
```

src/main/resources/com/vkozhevnikov/fileanalyzer/controller/SearchWindow.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ScrollPane?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.FlowPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.StackPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Text?>
<?import java.net.URL?>
<BorderPane fx:id="searchWindow" xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1"


      fx:controller="com.vkozhevnikov.fileanalyzer.controller.SearchWindow">
    <stylesheets>
        <URL value="@common-style.css"/>
    </stylesheets>

    <top>
        <HBox styleClass="header">
            <Text fill="white" style="-fx-font: 24 arial;" text="Пошук файлів"/>
        </HBox>
    </top>
    <left>
        <StackPane styleClass="paneSpacingAndPadding">
            <ScrollPane fx:id="scrollTags" maxHeight="700.0" maxWidth="300.0"
prefHeight="700" prefWidth="300.0">
                <FlowPane fx:id="tagsPane" maxWidth="280.0" prefHeight="695.0"
prefWidth="280.0"
```

```
                        styleClass="searchTagsPane"/>
            </ScrollPane>
        </StackPane>
    </left>
    <center>
        <VBox styleClass="paneSpacingAndPadding">
            <HBox alignment="CENTER_LEFT" styleClass="paneSpacing">
                <Label text="Введіть маску файлів для пошуку"/>
                <TextField fx:id="searchName" HBox.hgrow="ALWAYS"/>
            </HBox>
            <TableView fx:id="resultTable" VBox.Vgrow="ALWAYS"/>
            <HBox alignment="CENTER_LEFT" styleClass="paneSpacing">
                <Button fx:id="previousPage" text="Попередня сторінка" />
                <Label fx:id="currentPage" HBox.hgrow="ALWAYS"
alignment="CENTER"/>
                <Button fx:id="nextPage" text="Наступна сторінка"/>
            </HBox>
            <HBox alignment="CENTER_LEFT" styleClass="paneSpacing">
                <Label text="Знайдено: "/>
                <Label fx:id="foundCount" HBox.hgrow="ALWAYS"/>
            </HBox>
        </VBox>
    </center>
    <bottom>
        <HBox styleClass="footer">
            <Text fill="white" style="-fx-font: 24 arial"/>
            <Button fx:id="startButton" prefHeight="40" prefWidth="150"
text="Пошук"/>
            <Button fx:id="backButton" prefHeight="40" prefWidth="150"
text="Назад"/>
        </HBox>
    </bottom>

</BorderPane>
```

src/main/java/com/vkozhevnikov/fileanalyzer/controller/IndexingWindow.java

```
package com.vkozhevnikov.fileanalyzer.controller;

import com.vkozhevnikov.fileanalyzer.services.IndexingService;
import javafx.beans.property.SimpleLongProperty;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.stage.DirectoryChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import lombok.extern.slf4j.Slf4j;
import net.rgielen.fxweaver.core.FxmlView;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.io.File;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

import static java.util.Objects.nonNull;
```

```
@Component
@FxmlView
@Slf4j
public class IndexingWindow {

    private Stage stage;

    @FXML
    private Button startButton;

    @FXML
    private Button backButton;
    @FXML
    private Button btnChoosePath;
    @FXML
    private Label filesProcessedLabel;
    @FXML
    private Label currentProcessingLabel;

    @FXML
    private BorderPane indexingWindow;
    @FXML
    private TextField filePath;
    @FXML
    private FlowPane labelsContainer;
    @FXML
    private CheckBox tagByAuthor;
    @FXML
    private CheckBox tagByDate;
    @FXML
    private CheckBox tagByType;
    @FXML
    private CheckBox tagByGps;
    @FXML
    private FlowPane tagsContainer;


    @Autowired
    private IndexingService indexingService;
    private String filePathString;
    private SimpleLongProperty longProperty = new SimpleLongProperty(0L);
    private DirectoryChooser directoryChooser = new DirectoryChooser();

    private final Executor executor = Executors.newCachedThreadPool(runnable ->
{
        Thread t = new Thread(runnable);
        t.setDaemon(true);
        return t;
    });

    @FXML
    public void initialize() {
        this.stage = new Stage();
        stage.setScene(new Scene(indexingWindow, 1024, 768));
        stage.initStyle(StageStyle.UNDECORATED);
        stage.initModality(Modality.APPLICATION_MODAL);

        startButton.setOnAction(actionEvent -> {
            log.info("Обрано <Почати інжексування>: " + filePathString);
            longProperty.set(0L);
            currentProcessingLabel.setText("");
            blockInput();
            tagsContainer.getChildren().clear();
```

```
            startIndexing();
        });

        filesProcessedLabel.textProperty().bind(longProperty.asString());

        backButton.setOnAction(
                actionEvent -> stage.close()
        );

        filePath.textProperty().addListener(
                (observable, oldValue, newValue) -> {
                    this.filePathString = newValue;
                    if (nonNull(this.filePathString)) {
                        startButton.setDisable(false);
                    }
                });

        startButton.setDisable(true);

        btnChoosePath.setOnAction(e -> {
            File selectedDirectory = directoryChooser.showDialog(stage);
            if (nonNull(selectedDirectory)) {

filePath.textProperty().set(selectedDirectory.getAbsolutePath());
            }
        });
    }

    public void startIndexing() {
        Runnable task = indexingService.getRunnable(filePathString,
longProperty, currentProcessingLabel, this);
        executor.execute(task);
    }

    public boolean isTagByAuthor() {
        return tagByAuthor.isSelected();
    }

    public boolean isTagByDate() {
        return tagByDate.isSelected();
    }

    public boolean isTagByType() {
        return tagByType.isSelected();
    }

    public boolean isTagByGps() {
        return tagByGps.isSelected();
    }

    public FlowPane getTagsContainer() {
        return tagsContainer;
    }

    public void show() {
        stage.show();
    }


    private void blockInput() {
        startButton.setDisable(true);
        backButton.setDisable(true);
        btnChoosePath.setDisable(true);
        labelsContainer.setDisable(true);
```

```
        log.info("Вікно індексування заблоковано");
    }
    public void unblockInput() {
        startButton.setDisable(false);
        backButton.setDisable(false);
        btnChoosePath.setDisable(false);
        labelsContainer.setDisable(false);
        log.info("Вікно індексування розблоковано");
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/controller/SearchWindow.java

```
package com.vkozhevnikov.fileanalyzer.controller;

import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import com.vkozhevnikov.fileanalyzer.data.entities.FileTag;
import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import com.vkozhevnikov.fileanalyzer.data.objects.SearchContext;
import com.vkozhevnikov.fileanalyzer.data.objects.TagPaneSearchable;
import com.vkozhevnikov.fileanalyzer.services.SearchService;
import com.vkozhevnikov.fileanalyzer.services.TagService;
import javafx.application.Platform;
import javafx.beans.binding.Bindings;
import javafx.beans.property.ReadOnlyObjectWrapper;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import lombok.extern.slf4j.Slf4j;
import net.rgielen.fxweaver.core.FxmlView;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Component;

import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

@Component
@FxmlView
@Slf4j
public class SearchWindow {
```

```
    @Autowired
    private TagService tagService;
    @Autowired
    private SearchService searchService;
    private Stage stage;
    @FXML
    private BorderPane searchWindow;
    @FXML
    public ScrollPane scrollTags;
    @FXML
    private Button startButton;
    @FXML
    private Button backButton;
    @FXML
    private FlowPane tagsPane;
    @FXML
    private TextField searchName;
    @FXML
    public TableView<FileIndex> resultTable;
    private final ObservableList<FileIndex> resultTableObservable =
FXCollections.observableArrayList();
    @FXML
    public Button previousPage;
    @FXML
    public Label currentPage;
    @FXML
    public Button nextPage;

    @FXML
    private Label foundCount;

    private SearchContext context;

    private final Comparator<TagPaneSearchable> comparator =
Comparator.comparing(TagPaneSearchable::isSelected).reversed()
            .thenComparing(t -> t.getTag().getType())
            .thenComparing(Node::getId);

    @FXML
    public void initialize() {
        this.stage = new Stage();
        stage.setScene(new Scene(searchWindow, 1024, 768));
        stage.initStyle(StageStyle.UNDECORATED);
        stage.initModality(Modality.APPLICATION_MODAL);

        this.context = new SearchContext(this);
        List<TagPaneSearchable> tagPanes = tagService.getAllTags().stream()
                .map(tag -> TagPaneSearchable.of(tag, context))
                .collect(Collectors.toList());
        tagPanes.forEach(tagPane -> tagPane.setOnMouseClicked(event -> {
            tagPane.toggleSelected();
            tagPanes.sort(comparator);
            tagsPane.getChildren().setAll(tagPanes);
        }));
        tagPanes.sort(comparator);
        tagsPane.getChildren().setAll(tagPanes);
        startButton.setOnAction(event -> context.search());
        backButton.setOnAction(
                actionEvent -> stage.close()
        );

        scrollTags.setHbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);

        nextPage.setOnAction(event -> context.next());
```

```
        previousPage.setOnAction(event -> context.previous());
        nextPage.setDisable(true);
        previousPage.setDisable(true);
        initTable();
    }

    private void initTable() {
        resultTable.setItems(resultTableObservable);

        TableColumn<FileIndex, String> nameColumn = new TableColumn<FileIndex,
String>("Ім'я");
        nameColumn.setCellValueFactory(new PropertyValueFactory<FileIndex,
String>("name"));
        nameColumn.setSortable(false);

nameColumn.prefWidthProperty().bind(resultTable.widthProperty().multiply(0.22));
        resultTable.getColumns().add(nameColumn);

        TableColumn<FileIndex, String> pathColumn = new TableColumn<FileIndex,
String>("Шлях");
        pathColumn.setCellValueFactory(new PropertyValueFactory<FileIndex,
String>("path"));
        pathColumn.setSortable(false);

pathColumn.prefWidthProperty().bind(resultTable.widthProperty().multiply(0.4));
        resultTable.getColumns().add(pathColumn);

        TableColumn<FileIndex, String> sizeColumn = new TableColumn<FileIndex,
String>("Розмір");
        sizeColumn.setCellValueFactory(new PropertyValueFactory<FileIndex,
String>("size"));
        sizeColumn.setSortable(false);

sizeColumn.prefWidthProperty().bind(resultTable.widthProperty().multiply(0.08));
        resultTable.getColumns().add(sizeColumn);

        TableColumn<FileIndex, List<TagPaneSearchable>> keyColumn = new
TableColumn<FileIndex, List<TagPaneSearchable>>("Теги");
        keyColumn.setCellValueFactory(rowData -> {
            List<TagPaneSearchable> list = rowData.getValue().getTags().stream()
                    .map(FileTag::getTag)
                    .map(tag -> TagPaneSearchable.of(tag, null))
                    .collect(Collectors.toList());
            return new ReadOnlyObjectWrapper<>(list);
        });
        keyColumn.setCellFactory(column -> {
            TableCell<FileIndex, List<TagPaneSearchable>> cell = new
TableCell<>();
            cell.itemProperty().addListener((observableValue, o, newValue) -> {
                if (newValue != null) {
                    HBox graphicContainer = new HBox();
                    graphicContainer.setAlignment(Pos.CENTER);
                    graphicContainer.getChildren().addAll(newValue);


cell.graphicProperty().bind(Bindings.when(cell.emptyProperty()).then((Node)
null).otherwise(graphicContainer));
                }
            });
            return cell;
        });
```

```
        keyColumn.setSortable(false);

keyColumn.prefWidthProperty().bind(resultTable.widthProperty().multiply(0.3));
        resultTable.getColumns().add(keyColumn);


    }

    public Runnable createSearchTask(Pageable pageable) {
        return () -> {
            Platform.runLater(this::blockInput);
            Platform.runLater(resultTableObservable::clear);

            List<Tag> tags = tagsPane.getChildren().stream()
                    .map(TagPaneSearchable.class::cast)
                    .filter(TagPaneSearchable::isSelected)
                    .map(TagPaneSearchable::getTag)
                    .collect(Collectors.toList());
            String query = searchName.getText();

            log.info("Шукаємо  " + tags.stream().map(tag -> tag.getId() + " - "
+ tag.getName()).collect(Collectors.toList()));
            Long start = System.currentTimeMillis();

            Page<FileIndex> result = searchService.search(tags, query,
pageable);

            long searchRuntime = System.currentTimeMillis() - start;
            log.info("Пошук завершено за " + searchRuntime + " мс (" +
result.getContent().size() + " з " + result.getTotalElements() + ")");

            Platform.runLater(() -> {
                foundCount.setText(result.getContent().size() + " з " +
result.getTotalElements() + " за " + searchRuntime + " мс");
                resultTableObservable.setAll(result.getContent());
                this.context.setHasNext(result.hasNext());
                currentPage.setText("Сторінка " + (result.getNumber() + 1) + " з
" + result.getTotalPages());
                nextPage.setDisable(!result.hasNext());
                previousPage.setDisable(result.getNumber() < 1);
                this.unblockInput();
            });
        };
    }

    private void blockInput() {
        searchWindow.setDisable(true);
        log.info("Вікно пошуку заблоковано");
    }

    private void unblockInput() {
        searchWindow.setDisable(false);
        log.info("Вікно пошуку розблоковано");
    }

    public void show() {
        stage.show();
    }

}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/entities/FileIndex.java

```
package com.vkozhevnikov.fileanalyzer.data.entities;

import com.fasterxml.jackson.annotation.JsonBackReference;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Index;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import java.util.Date;
import java.util.List;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@ToString(exclude = "tags")
@Table(indexes = {
        @Index(name = "path_idx", columnList = "path", unique = true),
})
public class FileIndex {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private Date dateCreation;
    private Date dateUpdate;
    private String path;
    private Long size;
    private String type;
    private String md5;
    private String author;
    private Long dateIndexed;

    @OneToOne(mappedBy = "fileIndex")
    @JsonBackReference
    @JoinColumn(nullable = true)
    private FileLocation fileLocation;

    @OneToMany(mappedBy = "fileIndex", fetch = FetchType.EAGER, orphanRemoval =
true)
    private List<FileTag> tags;

}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/entities/FileLocation.java

```
package com.vkozhevnikov.fileanalyzer.data.entities;

import com.fasterxml.jackson.annotation.JsonManagedReference;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import java.util.Date;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Table
@ToString(exclude = "fileIndex")
public class FileLocation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @OneToOne
    @JoinColumn(nullable = false, name = "file_index_id")
    @JsonManagedReference
    private FileIndex fileIndex;
    private Double latitude;
    private Double longitude;
    @Temporal(TemporalType.TIMESTAMP)
    private Date gpsDate;

}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/entities/Tag.java

```
package com.vkozhevnikov.fileanalyzer.data.entities;

import com.vkozhevnikov.fileanalyzer.data.objects.TagType;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Table
public class Tag {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(nullable = false)
    private String name;
    private String description;
    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private TagType type;
    @Column(name = "key", nullable = false)
    private String key;
    @Column(name = "option")
    private String option;
    private String color;
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/objects/SearchContext.java

```
package com.vkozhevnikov.fileanalyzer.data.objects;

import com.vkozhevnikov.fileanalyzer.controller.SearchWindow;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.PageRequest;

import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

@Slf4j
public class SearchContext {
    private final SearchWindow window;
    private int pageSize;
    private int pageNumber;
    @Setter
    private boolean hasNext;
    private final Executor executor = Executors.newCachedThreadPool(runnable ->
{
        Thread t = new Thread(runnable);
        t.setDaemon(true);
        return t;
    });

    public SearchContext(SearchWindow window) {
        this.window = window;
        this.pageSize = 15;
        this.pageNumber = 0;
        this.hasNext = false;
    }

    public void search() {
        PageRequest request = PageRequest.of(pageNumber, pageSize);
        Runnable task = window.createSearchTask(request);
        executor.execute(task);
    }
```

```
    public void next() {
        if (hasNext) {
            this.pageNumber++;
            search();
        }
    }
    public void previous() {
        if (pageNumber > 0) {
            this.pageNumber--;
            search();
        }
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/objects/TagPaneCountable.java

```
package com.vkozhevnikov.fileanalyzer.data.objects;

import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import javafx.geometry.Pos;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import lombok.Getter;

public class TagPaneCountable extends StackPane {
    private final Text text;
    private final Text textCount;
    private final VBox vBox;
    @Getter
    private final Tag tag;

    private final TagRectangle tagRectangle;

    public static TagPaneCountable of(Tag tag) {
        return new TagPaneCountable(tag);
    }

    protected TagPaneCountable(Tag tag) {
        this.tag = tag;
        this.setId(tag.getKey());
        this.text = new Text(tag.getKey());
        this.text.setStyle("-fx-font: 16 arial;");
        this.textCount = new Text("0");
        this.tagRectangle = new TagRectangle(Color.web(tag.getColor()));
        this.vBox = new VBox(text, textCount);
        this.vBox.setAlignment(Pos.CENTER);
        this.getChildren().addAll(tagRectangle, vBox);
        resize();
    }

    public TagPaneCountable setCount(Integer count) {
        this.textCount.setText(count > 0 ? String.valueOf(count) : "");
        resize();
        return this;
    }

    private void resize() {
        double width = Math.max(this.text.getBoundsInLocal().getWidth(),
this.textCount.getBoundsInLocal().getWidth());
        double height = this.text.getBoundsInLocal().getHeight() +
```

```
this.textCount.getBoundsInLocal().getHeight();
        this.tagRectangle.setWidth(width + 25);
        this.tagRectangle.setHeight(height + 15);
    }


}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/objects/TagPaneSearchable.java

```
package com.vkozhevnikov.fileanalyzer.data.objects;

import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import lombok.Getter;

public class TagPaneSearchable extends StackPane {
    private final SearchContext searchContext;
    private final Text text;
    @Getter
    private final Tag tag;
    private final TagRectangle tagRectangle;
    @Getter
    private boolean selected;

    private TagPaneSearchable(Tag tag, SearchContext context) {
        this.tag = tag;
        this.setId(tag.getKey());
        this.text = new Text(tag.getKey());
        this.tagRectangle = new TagRectangle(Color.web(tag.getColor()));
        this.getChildren().addAll(tagRectangle, text);
        resize();
        this.searchContext = context;
    }
    public static TagPaneSearchable of(Tag tag, SearchContext context) {
        return new TagPaneSearchable(tag, context);
    }
    public void toggleSelected() {
        this.selected = !this.selected;
        this.tagRectangle.setSelected(this.selected);
        this.resize();
        searchContext.search();
    }
    private void resize() {
        double width = this.text.getBoundsInLocal().getWidth();
        double height = this.text.getBoundsInLocal().getHeight();
        this.tagRectangle.setWidth(width + 10);
        this.tagRectangle.setHeight(height + 10);
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/objects/TagRectangle.java

```
package com.vkozhevnikov.fileanalyzer.data.objects;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.StrokeType;
```

```
public class TagRectangle extends Rectangle {
    private boolean selected;
    private Color color;
    public TagRectangle(Color color) {
        this.color = color;
        setWidth(25);
        setHeight(25);
        setArcWidth(15);
        setArcHeight(15);
        setFill(color.deriveColor(0, 1.2, 1, 0.6));
        setStroke(color);
    }

    public void setSelected(boolean selected) {
        this.selected = selected;
        if (this.selected) {
            setStrokeWidth(10);
            setStrokeType(StrokeType.OUTSIDE);
        } else {
            setStrokeWidth(0);
            setStrokeType(null);
        }
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/objects/TagType.java

```
package com.vkozhevnikov.fileanalyzer.data.objects;

import lombok.AllArgsConstructor;
import lombok.Getter;

@AllArgsConstructor
public enum TagType {
    AUTHOR(true),
    DATE(true),
    FILE_TYPE(true),
    GPS(true),
    GPS_DATE(true),
    CUSTOM(false);

    @Getter
    private boolean auto;
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/data/repositories/FileIndexRepository.java

```
package com.vkozhevnikov.fileanalyzer.data.repositories;

import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.Date;
import java.util.List;
import java.util.Optional;
```

```
@Repository
public interface FileIndexRepository extends
PagingAndSortingRepository<FileIndex, Long>, JpaSpecificationExecutor<FileIndex>
{
    Optional<FileIndex> findByPath(String path);
    @Query("select fi.md5 from FileIndex as fi " +
            "          where length(fi.md5) > 0 " +
            "            and fi.path like :path% " +
            "          group by fi.md5 having count(fi.md5) > 1")
    List<String> findMD5Duplicates(@Param("path") String path);
    List<FileIndex> findAllByMd5(String md5);
    List<FileIndex> findAllByDateIndexedLessThanAndPathLike(Long dateIndexed,
String path);
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/services/FindDuplicatesService.java

```
package com.vkozhevnikov.fileanalyzer.services;

import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileIndexRepository;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Path;
import java.util.List;

@Service
@Slf4j
@AllArgsConstructor
public class FindDuplicatesService {

    private final FileIndexRepository fileIndexRepository;
    public List<String> findMD5Duplicates(String path) {
        return fileIndexRepository.findMD5Duplicates(path.replace("\\","\\\\"));
    }


    public List<FileIndex> findFilesByMd5(String md5) {
        return fileIndexRepository.findAllByMd5(md5);
    }

    public static boolean compareByMemoryMappedFiles(Path path1, Path path2) {
        try (RandomAccessFile randomAccessFile1 = new
RandomAccessFile(path1.toFile(), "r");
             RandomAccessFile randomAccessFile2 = new
RandomAccessFile(path2.toFile(), "r")) {

            FileChannel ch1 = randomAccessFile1.getChannel();
            FileChannel ch2 = randomAccessFile2.getChannel();
            if (ch1.size() != ch2.size()) {
                return false;
            }
            long size = ch1.size();
            MappedByteBuffer m1 = ch1.map(FileChannel.MapMode.READ_ONLY, 0L,
size);
            MappedByteBuffer m2 = ch2.map(FileChannel.MapMode.READ_ONLY, 0L,
```

```
size);

            return m1.equals(m2);
        } catch (FileNotFoundException e) {
            log.warn("Файл не знайдено: " + e.getMessage());
            return false;
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/services/IndexingService.java

```
package com.vkozhevnikov.fileanalyzer.services;

import com.drew.imaging.ImageMetadataReader;
import com.drew.imaging.ImageProcessingException;
import com.drew.lang.GeoLocation;
import com.drew.metadata.Metadata;
import com.drew.metadata.exif.GpsDirectory;
import com.vkozhevnikov.fileanalyzer.controller.IndexingWindow;
import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import com.vkozhevnikov.fileanalyzer.data.entities.FileLocation;
import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import com.vkozhevnikov.fileanalyzer.data.objects.TagPaneCountable;
import com.vkozhevnikov.fileanalyzer.data.objects.TagType;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileIndexRepository;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileLocationRepository;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileTagRepository;
import javafx.application.Platform;
import javafx.beans.property.SimpleLongProperty;
import javafx.collections.ObservableList;
import javafx.scene.Node;
import javafx.scene.control.Label;
import javafx.scene.paint.Color;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import javax.xml.bind.DatatypeConverter;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.FileVisitResult;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.SimpleFileVisitor;
import java.nio.file.attribute.BasicFileAttributeView;
import java.nio.file.attribute.BasicFileAttributes;
import java.nio.file.attribute.FileOwnerAttributeView;
import java.nio.file.attribute.UserPrincipal;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;
import java.util.Optional;
import java.util.concurrent.atomic.AtomicInteger;

import static java.util.Objects.isNull;
import static java.util.Objects.nonNull;
import static java.util.Optional.of;
import static java.util.Optional.ofNullable;
import static java.util.function.Predicate.not;

@Service
@Slf4j
@AllArgsConstructor
public class IndexingService {
    private final FileIndexRepository fileIndexRepository;
    private final FileTagRepository fileTagRepository;
    private final FileLocationRepository fileLocationRepository;
    private final TagService tagService;

    private void updateFileIndexes(String path, SimpleLongProperty
filesProcessedProperty, Label currentProcessingLabel, IndexingWindow
indexingWindow) {
        List<FileIndex> indexedFiles = new ArrayList<>();
        Long indexedDate = System.currentTimeMillis();
        Map<String, AtomicInteger> smartTags = new HashMap<>();
        try {
            Files.walkFileTree(Paths.get(path), new SimpleFileVisitor<Path>() {
                @Override
                public FileVisitResult preVisitDirectory(Path dir,
BasicFileAttributes attrs) throws IOException {
                    log.debug("Знайдено папку: " + dir.toString());
                    return FileVisitResult.CONTINUE;
                }

                @Override
                public FileVisitResult visitFile(Path filePath,
BasicFileAttributes attrs) {
                    try {
                        log.debug("Знадено файл: " + filePath.toString());
                        Platform.runLater(() ->
currentProcessingLabel.setText(filePath.toString()));
                        FileIndex fileIndex = indexFile(filePath, indexedDate);
                        indexedFiles.add(fileIndex);
                        Platform.runLater(() ->
filesProcessedProperty.set(filesProcessedProperty.getValue() + 1));
                        checkAuthorTag(fileIndex, smartTags, indexingWindow);
                        checkDateTag(fileIndex, smartTags, indexingWindow);
                        checkTypeTag(fileIndex, smartTags, indexingWindow);
                        checkGps(fileIndex, smartTags, indexingWindow);

                    } catch (IOException e) {
                        log.error("Помилка під час індексування файлів " +
e.getMessage(), e);
                    }
                    return FileVisitResult.CONTINUE;
                }

                @Override
                public FileVisitResult visitFileFailed(Path file, IOException
exc) throws IOException {
                    log.debug("Помилка під час пошуку : " + file.toString(),
exc);
                    return FileVisitResult.CONTINUE;
                }
```

```
            });
        } catch (IOException e) {
            log.error("Помилка при індексуванні файлів " + e.getMessage(), e);
        }
        log.info("Знайдено індекси: " + indexedFiles.size());
        deleteIndexes(path, indexedDate);
        Platform.runLater(indexingWindow::unblockInput);
    }

    private void deleteIndexes(String path, Long indexedDate) {
        List<FileIndex> fileIndexList =
fileIndexRepository.findAllByDateIndexedLessThanAndPathLike(indexedDate,
path.replace("\\", "\\\\") + "%");
        log.info("Знадено індекси для видалення: " + fileIndexList.size());
        for (FileIndex fileIndex : fileIndexList) {
            ofNullable(fileIndex.getFileLocation())
                    .ifPresent(fileLocationRepository::delete);
            fileIndexRepository.delete(fileIndex);
            log.debug("FileIndex видалено : " + fileIndex);
        }
    }

    public Runnable getRunnable(String path, SimpleLongProperty
filesProcessedProperty, Label currentProcessingLabel, IndexingWindow
indexingWindow) {
        return () -> {
            log.info("Початок індексування файлів, початкова папка " + path);
            Long start = System.currentTimeMillis();
            updateFileIndexes(path, filesProcessedProperty,
currentProcessingLabel, indexingWindow);
            log.info("Індексування завершено за " + (System.currentTimeMillis()
- start) + " мс");
        };
    }

    public FileIndex indexFile(Path filePath, Long dateIndexed) throws
IOException {
        log.info("Індексування файлу: " + filePath);
        BasicFileAttributeView basicView =
                Files.getFileAttributeView(filePath,
BasicFileAttributeView.class);
        BasicFileAttributes basicFileAttributes = basicView.readAttributes();
        FileOwnerAttributeView ownerAttributeView =
Files.getFileAttributeView(filePath, FileOwnerAttributeView.class);
        UserPrincipal owner = ownerAttributeView.getOwner();
        String path = filePath.toAbsolutePath().toString();

        Metadata metadata = null;
        try {
            metadata =
ImageMetadataReader.readMetadata(Files.newInputStream(filePath));
        } catch (ImageProcessingException e) {
            log.debug("Помилка ImageProcessingException (" + filePath + "): " +
e.getMessage());
        }

        FileIndex fileIndex = fileIndexRepository.findByPath(path)
                .orElseGet(() -> {
                    FileIndex newIndex = FileIndex.builder()
                            .name(filePath.getFileName().toString())
                            .dateCreation(new Date())
                            .path(path)

.type(resolveExtension(filePath.getFileName().toString())
```

```
                                    .orElse(""))
                            .build();
                    log.debug("Знайдено новий індекс: " + path);
                    return newIndex;
                });
        fileIndex.setDateCreation(new
Date(basicFileAttributes.creationTime().toMillis()));
        fileIndex.setDateUpdate(new
Date(basicFileAttributes.lastModifiedTime().toMillis()));
        fileIndex.setSize(basicFileAttributes.size());
        fileIndex.setMd5(calculateMd5(filePath));
        fileIndex.setAuthor(owner.getName());
        fileIndex.setDateIndexed(dateIndexed);
        FileIndex finalFileIndex = fileIndexRepository.save(fileIndex);
        if (nonNull(metadata)) {
            metadata.getDirectoriesOfType(GpsDirectory.class).stream()
                    .findFirst()
                    .ifPresent(gpsDirectory -> {
                        FileLocation fileLocation =
fileLocationRepository.findByFileIndex(finalFileIndex)
                                .orElseGet(() -> FileLocation.builder()
                                        .fileIndex(finalFileIndex)
                                        .build());
                        GeoLocation geoLocation = gpsDirectory.getGeoLocation();
                        if (nonNull(geoLocation)) {
                            fileLocation.setLatitude(geoLocation.getLatitude());

fileLocation.setLongitude(geoLocation.getLongitude());
                            fileLocation.setGpsDate(gpsDirectory.getGpsDate());
                            fileLocation =
fileLocationRepository.save(fileLocation);
                            finalFileIndex.setFileLocation(fileLocation);
                        }
                    });
        }
        log.debug("Оновлено індекс: " + finalFileIndex);
        return finalFileIndex;
    }

    public void checkAuthorTag(FileIndex fileIndex, Map<String, AtomicInteger>
smartTags, IndexingWindow indexingWindow) {
        if (!indexingWindow.isTagByAuthor()) {
            return;
        }
        Optional<String> author = ofNullable(fileIndex.getAuthor())
                .filter(not(String::isEmpty));
        if (author.isPresent()) {
            String key = author.get();
            Tag tag = tagService.findOrCreateTag(TagType.AUTHOR, key, key,
"Smart tag by author", toRGBCode(Color.DARKORANGE));
            updateTagCount(tag, smartTags, indexingWindow);
            tagService.updateFileTag(fileIndex, tag);
        }
    }

    public void checkTypeTag(FileIndex fileIndex, Map<String, AtomicInteger>
smartTags, IndexingWindow indexingWindow) {
        if (!indexingWindow.isTagByType()) {
            return;
        }
        Optional<String> type = ofNullable(fileIndex.getType())
                .filter(not(String::isEmpty));
        if (type.isPresent()) {
            String key = type.get();
```

```
            Tag tag = tagService.findOrCreateTag(TagType.FILE_TYPE, key, key,
"Smart tag by type", toRGBCode(Color.PLUM));
            updateTagCount(tag, smartTags, indexingWindow);
            tagService.updateFileTag(fileIndex, tag);
        }
    }

    public void checkGps(FileIndex fileIndex, Map<String, AtomicInteger>
smartTags, IndexingWindow indexingWindow) {
        if (!indexingWindow.isTagByGps()) {
            return;
        }
        FileLocation fileLocation = fileIndex.getFileLocation();
        if (isNull(fileLocation)) {
            return;
        }

        DecimalFormat dec = new DecimalFormat("#0.1");

        String latitude = dec.format(fileLocation.getLatitude());
        String longitude = dec.format(fileLocation.getLongitude());
        String key = latitude + "/" + longitude;
        Tag tag = tagService.findOrCreateTag(TagType.GPS, key, key, "Smart tag
by gps", toRGBCode(Color.CORNFLOWERBLUE));
        updateTagCount(tag, smartTags, indexingWindow);
        tagService.updateFileTag(fileIndex, tag);

        if (nonNull(fileLocation.getGpsDate())) {
            SimpleDateFormat sdf = new SimpleDateFormat("MMMMM ''yy");
            Optional<String> date = of(sdf.format(fileIndex.getDateCreation()))
                    .filter(not(String::isEmpty));
            if (date.isPresent()) {
                key = "gps_"+date.get();
                tag = tagService.findOrCreateTag(TagType.GPS_DATE, key, key,
"Smart tag by gps date", toRGBCode(Color.CORNFLOWERBLUE));
                updateTagCount(tag, smartTags, indexingWindow);
                tagService.updateFileTag(fileIndex, tag);
            }
        }
    }

    public void checkDateTag(FileIndex fileIndex, Map<String, AtomicInteger>
smartTags, IndexingWindow indexingWindow) {
        if (!indexingWindow.isTagByDate()) {
            return;
        }

        SimpleDateFormat sdf = new SimpleDateFormat("MMMMM ''yy");
        Optional<String> date = of(sdf.format(fileIndex.getDateCreation()))
                .filter(not(String::isEmpty));

        if (date.isPresent()) {
            String key = date.get();
            Tag tag = tagService.findOrCreateTag(TagType.DATE, key, key, "Smart
tag by date per month", toRGBCode(Color.AQUAMARINE));
            updateTagCount(tag, smartTags, indexingWindow);
            tagService.updateFileTag(fileIndex, tag);
        }
    }

    public void updateTagCount(Tag tag, Map<String, AtomicInteger> smartTags,
IndexingWindow indexingWindow) {
        Platform.runLater(() -> {
            smartTags.putIfAbsent(tag.getKey(), new AtomicInteger(0));
```

```
            int count = smartTags.get(tag.getKey()).incrementAndGet();
            ObservableList<Node> children =
indexingWindow.getTagsContainer().getChildren();
            children.stream()
                    .filter(node -> tag.getKey().equalsIgnoreCase(node.getId()))
                    .findFirst()
                    .map(TagPaneCountable.class::cast)
                    .orElseGet(() -> {
                        TagPaneCountable newTag = TagPaneCountable.of(tag);
                        children.add(newTag);
                        return newTag;
                    })
                    .setCount(count);
        });
    }

    private Optional<String> resolveExtension(String filename) {
        return ofNullable(filename)
                .filter(f -> f.contains("."))
                .map(f -> f.substring(filename.lastIndexOf(".") + 1))
                .filter(extension -> extension.length() < 20);
    }

    public static String toRGBCode(Color color) {
        return String.format("#%02X%02X%02X",
                (int) (color.getRed() * 255),
                (int) (color.getGreen() * 255),
                (int) (color.getBlue() * 255));
    }

    private String calculateMd5(Path path) {
        try (InputStream is = new
FileInputStream(path.toAbsolutePath().toFile())) {
            MessageDigest md5 = MessageDigest.getInstance("MD5");
            byte[] buffer = new byte[1024]; // or any other size
            int len;
            while ((len = is.read(buffer)) != -1) {
                md5.update(buffer, 0, len); // only update with the just read
bytes
            }
            return DatatypeConverter.printHexBinary(md5.digest());
        } catch (IOException | NoSuchAlgorithmException e) {
            log.error("Не можу обрахувати md5, " + e.getMessage(), e);
        }
        return "";
    }
}
```

src/main/java/com/vkozhevnikov/fileanalyzer/services/SearchService.java

```
package com.vkozhevnikov.fileanalyzer.services;

import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex_;
import com.vkozhevnikov.fileanalyzer.data.entities.FileTag;
import com.vkozhevnikov.fileanalyzer.data.entities.FileTag_;
import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileIndexRepository;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
```

```
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;

import javax.persistence.criteria.Join;
import javax.persistence.criteria.JoinType;
import javax.persistence.criteria.Predicate;
import java.util.List;

@Service
@Slf4j
@AllArgsConstructor
public class SearchService {
    private final FileIndexRepository fileIndexRepository;

    public Page<FileIndex> search(List<Tag> selectedTags, String query, Pageable
pageable) {
        if (selectedTags.isEmpty() && StringUtils.isEmpty(query)) {
            log.warn("Запит пустий?");
            return Page.empty();
        }
        return fileIndexRepository.findAll(searchFiles(selectedTags, query),
pageable);
    }

    private static Specification<FileIndex> searchFiles(List<Tag> tags, String
query) {
        return (root, criteriaQuery, cb) -> {
            Predicate p = cb.conjunction();

            if (!StringUtils.isEmpty(query)) {
                p = cb.and(p, cb.like(cb.lower(root.get(FileIndex_.NAME)),
query.toLowerCase() + "%"));
            }

            if (!tags.isEmpty()) {
                for (Tag tag : tags) {
                    Join<FileIndex, FileTag> joinFileTag =
root.join(FileIndex_.TAGS, JoinType.INNER);
                    p = cb.and(p, cb.equal(joinFileTag.get(FileTag_.TAG), tag));
                }
            }
            return p;
        };
    }

}
```

src/test/java/com/vkozhevnikov/fileanalyzer/services/IndexingAndSearchServiceTest.java

```
package com.vkozhevnikov.fileanalyzer.services;

import com.vkozhevnikov.fileanalyzer.controller.IndexingWindow;
import com.vkozhevnikov.fileanalyzer.data.entities.FileIndex;
import com.vkozhevnikov.fileanalyzer.data.entities.FileLocation;
import com.vkozhevnikov.fileanalyzer.data.entities.Tag;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileIndexRepository;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileLocationRepository;
import com.vkozhevnikov.fileanalyzer.data.repositories.FileTagRepository;
import com.vkozhevnikov.fileanalyzer.data.repositories.TagRepository;
import org.junit.jupiter.api.Test;
import org.junit.runner.RunWith;
import org.mockito.Mock;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.junit4.SpringRunner;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.text.ParseException;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import static com.vkozhevnikov.fileanalyzer.data.objects.TagType.AUTHOR;
import static com.vkozhevnikov.fileanalyzer.data.objects.TagType.DATE;
import static com.vkozhevnikov.fileanalyzer.data.objects.TagType.FILE_TYPE;
import static com.vkozhevnikov.fileanalyzer.data.objects.TagType.GPS;
import static com.vkozhevnikov.fileanalyzer.data.objects.TagType.GPS_DATE;
import static org.assertj.core.api.AssertionsForInterfaceTypes.assertThat;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyMap;
import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.doNothing;
import static org.mockito.Mockito.spy;


@RunWith(SpringRunner.class)
@DataJpaTest
@TestPropertySource(
        locations = "classpath:application-integrationtest.yml")
@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
class IndexingAndSearchServiceTest {

    private static final Tag tag1 = Tag.builder()
            .id(1L)
            .name("SEO\\VKozhevnikov")
            .description("Smart tag by author")
            .type(AUTHOR)
            .key("SEO\\VKozhevnikov")
            .color("#FF8C00")
            .build();
    private static final Tag tag2 = Tag.builder()
            .id(2L)
            .name("November '23")
            .description("Smart tag by date per month")
            .type(DATE)
            .key("November '23")
            .color("#7FFFD4")
            .build();
    private static final Tag tag3 = Tag.builder()
            .id(3L)
            .name("gif")
            .description("Smart tag by type")
            .type(FILE_TYPE)
```

```
                .key("gif")
                .color("#DDA0DD")
                .build();
    private static final Tag tag4 = Tag.builder()
                .id(4L)
                .name("txt")
                .description("Smart tag by type")
                .type(FILE_TYPE)
                .key("txt")
                .color("#DDA0DD")
                .build();
    private static final Tag tag5 = Tag.builder()
                .id(5L)
                .name("jpg")
                .description("Smart tag by type")
                .type(FILE_TYPE)
                .key("jpg")
                .color("#DDA0DD")
                .build();
    private static final Tag tag6 = Tag.builder()
                .id(6L)
                .name("27.1/34.1")
                .description("Smart tag by gps")
                .type(GPS)
                .key("27.1/34.1")
                .color("#6495ED")
                .build();
    private static final Tag tag7 = Tag.builder()
                .id(7L)
                .name("November '23")
                .description("Smart tag by gps date")
                .type(GPS_DATE)
                .key("November '23")
                .color("#6495ED")
                .build();

    @Autowired
    private FileIndexRepository fileIndexRepository;
    @Autowired
    private FileLocationRepository fileLocationRepository;
    @Autowired
    private TagRepository tagRepository;
    @Autowired
    private FileTagRepository fileTagRepository;
    @Mock
    private IndexingWindow indexingWindow;
    private Long dateIndexed = System.currentTimeMillis();
    private Path pathPrefixIndexing = Paths.get("src", "test", "resources",
"indexing");
    @Test
    void indexFilesTest() throws IOException, ParseException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        List<FileIndex> fileIndexList = List.of(fileIndex1(pathPrefixIndexing,
dateIndexed), fileIndex2(pathPrefixIndexing, dateIndexed),
fileIndex3(pathPrefixIndexing, dateIndexed));

assertThat(fileIndexRepository.findAll()).asString().isEqualTo(fileIndexList.toS
tring());
    }

    @Test
    void tagsCreationTest() throws IOException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);
```

```
        List<Tag> tags = List.of(tag1, tag2, tag3, tag4, tag5, tag6, tag7);
        assertThat(tagRepository.findAll()).hasSameElementsAs(tags);
    }

    @Test
    void searchWithEmptyRequestTest() throws IOException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        SearchService searchService = new SearchService(fileIndexRepository);
        Page<FileIndex> searchReponse =
searchService.search(Collections.emptyList(), "", PageRequest.of(0, 10));

        assertThat(searchReponse.getContent()).asList().isEmpty();
    }

    @Test
    void searchWithTagsOnlyTest() throws IOException, ParseException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        SearchService searchService = new SearchService(fileIndexRepository);
        Page<FileIndex> searchResponse = searchService.search(List.of(tag1,
tag4), "", PageRequest.of(0, 10));


assertThat(searchResponse.getContent().get(0)).isEqualTo(fileIndex2(pathPrefixIn
dexing, dateIndexed));
    }

    @Test
    void searchWithTagsOnlyEmptyResponseTest() throws IOException,
ParseException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        SearchService searchService = new SearchService(fileIndexRepository);
        Page<FileIndex> searchResponse = searchService.search(List.of(tag1,
tag3, tag4), "", PageRequest.of(0, 10));

        assertThat(searchResponse.getContent()).isEmpty();
    }

    @Test
    void searchByQueryTest() throws IOException, ParseException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        SearchService searchService = new SearchService(fileIndexRepository);
        Page<FileIndex> searchResponse = searchService.search(List.of(), "%l",
PageRequest.of(0, 10));


assertThat(searchResponse.getContent()).isEqualTo(List.of(fileIndex1(pathPrefixI
ndexing, dateIndexed), fileIndex2(pathPrefixIndexing, dateIndexed)));
    }

    @Test
    void searchByQueryAndTagTest() throws IOException, ParseException {
        doIndexingWithTags(pathPrefixIndexing, dateIndexed);

        SearchService searchService = new SearchService(fileIndexRepository);
        Page<FileIndex> searchResponse = searchService.search(List.of(tag3),
"%l", PageRequest.of(0, 10));


assertThat(searchResponse.getContent()).isEqualTo(List.of(fileIndex1(pathPrefixI
```

```
ndexing, dateIndexed)));
    }

    private void doIndexingWithTags(Path pathPrefix, Long dateIndexed) throws
IOException {
        TagService tagService = new TagService(tagRepository,
fileTagRepository);
        IndexingService indexingService = spy(new
IndexingService(fileIndexRepository, fileTagRepository, fileLocationRepository,
tagService));

        given(indexingWindow.isTagByAuthor()).willReturn(true);
        given(indexingWindow.isTagByDate()).willReturn(true);
        given(indexingWindow.isTagByGps()).willReturn(true);
        given(indexingWindow.isTagByType()).willReturn(true);
        doNothing().when(indexingService).updateTagCount(any(), anyMap(),
any());

        Map<String, AtomicInteger> smartTags = new HashMap<>();

        try (Stream<Path> stream = Files.list(pathPrefix)) {
            Set<Path> files = stream
                    .filter(file -> !Files.isDirectory(file))
                    .collect(Collectors.toSet());
            for (Path path : files) {
                FileIndex fileIndex = indexingService.indexFile(path,
dateIndexed);
                indexingService.checkAuthorTag(fileIndex, smartTags,
indexingWindow);
                indexingService.checkDateTag(fileIndex, smartTags,
indexingWindow);
                indexingService.checkTypeTag(fileIndex, smartTags,
indexingWindow);
                indexingService.checkGps(fileIndex, smartTags, indexingWindow);
            }
        }
    }

    private static FileIndex fileIndex1(Path pathPrefix, Long dateIndexed)
throws ParseException {
        return FileIndex.builder()
                .id(1L)
                .name("bl.gif")
                .dateCreation(new Date(1701297571680L))
                .dateUpdate(new Date(1612205366000L))
                .path(pathPrefix.toAbsolutePath() + "\\bl.gif")
                .size(83L)
                .type("gif")
                .md5("9EB68B7AB8E7666465B63B4B0021DCCB")
                .author("SEO\\VKozhevnikov")
                .dateIndexed(dateIndexed)
                .build();
    }

    private static FileIndex fileIndex2(Path pathPrefix, Long dateIndexed)
throws ParseException {
        return FileIndex.builder()
                .id(2L)
                .name("empty-file.txt")
                .dateCreation(new Date(1701297468343L))
                .dateUpdate(new Date(1701297468343L))
                .path(pathPrefix.toAbsolutePath() + "\\empty-file.txt")
                .size(0L)
                .type("txt")
```

```
                .md5("D41D8CD98F00B204E9800998ECF8427E")
                .author("SEO\\VKozhevnikov")
                .dateIndexed(dateIndexed)
                .build();
    }

    private static FileIndex fileIndex3(Path pathPrefix, Long dateIndexed)
throws ParseException {
        return FileIndex.builder()
                .id(3L)
                .name("gps-test.jpg")
                .dateCreation(new Date(1701297286758L))
                .dateUpdate(new Date(1617793875323L))
                .path(pathPrefix.toAbsolutePath() + "\\gps-test.jpg")
                .size(1678123L)
                .type("jpg")
                .md5("6B6160D554DE8BF4B76834ACDE9D0437")
                .author("SEO\\VKozhevnikov")
                .dateIndexed(dateIndexed)
                .fileLocation(FileLocation.builder()
                        .id(1L)
                        .latitude(27.070786972222223)
                        .longitude(33.890045)
                        .gpsDate(new Date(1616326651000L))
                        .build())
                .build();
    }
}
```

ДОДАТОК Г

Ілюстративнна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПОШУКУ ФАЙЛІВ ТА ЇХ ВПОРЯДКУВАННЯ

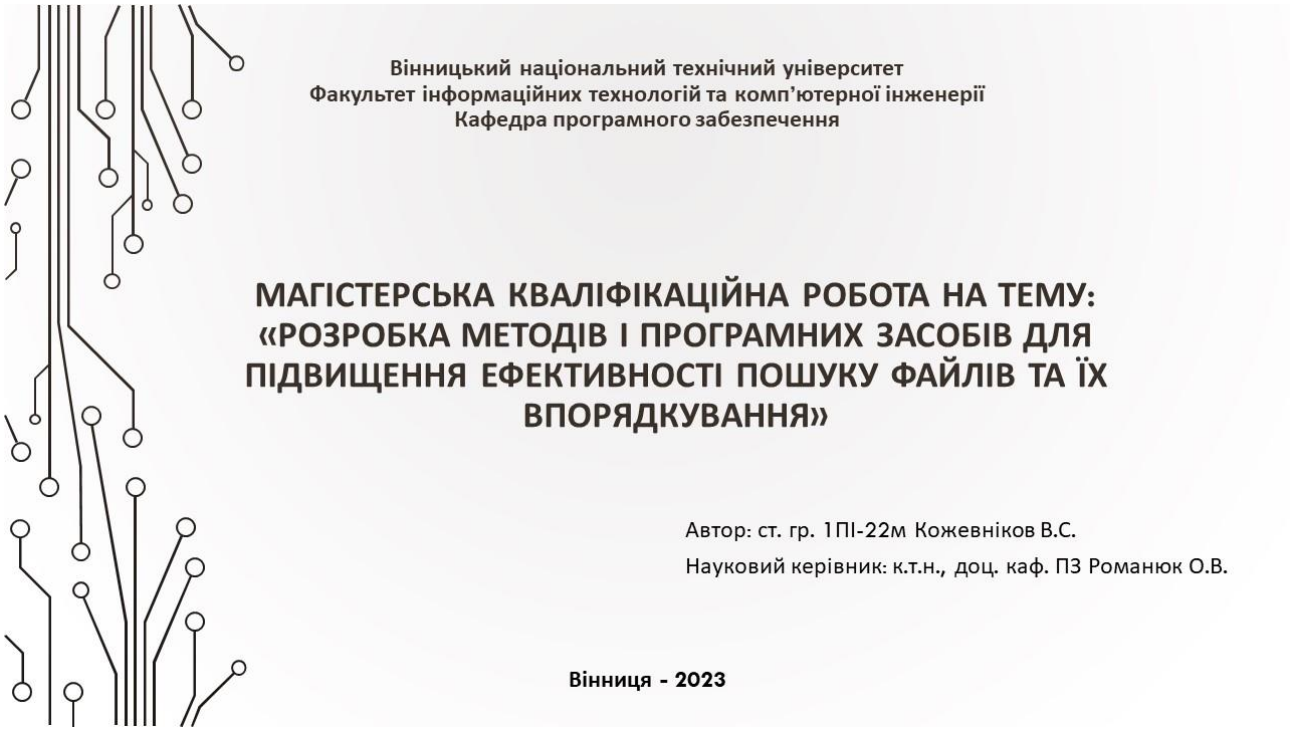

Рисунок Г.1 – Титульний слайд

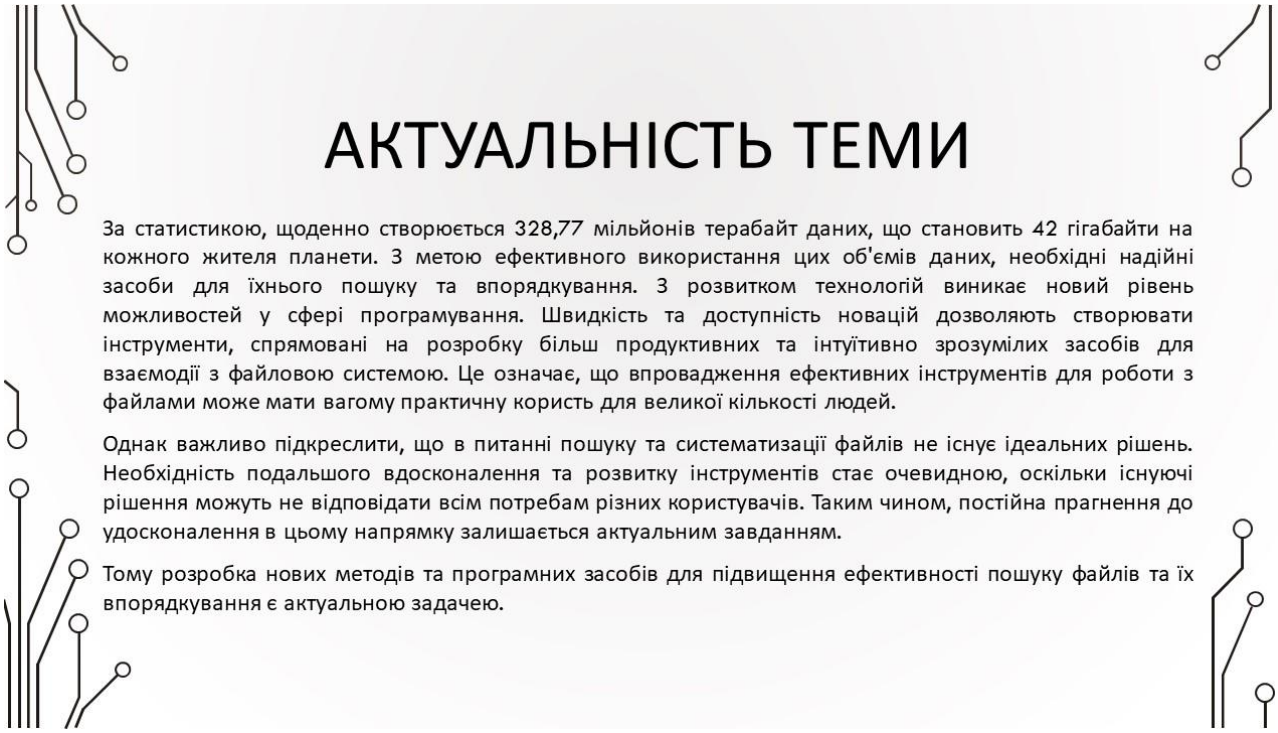

Рисунок Г.2 – Актуальність теми

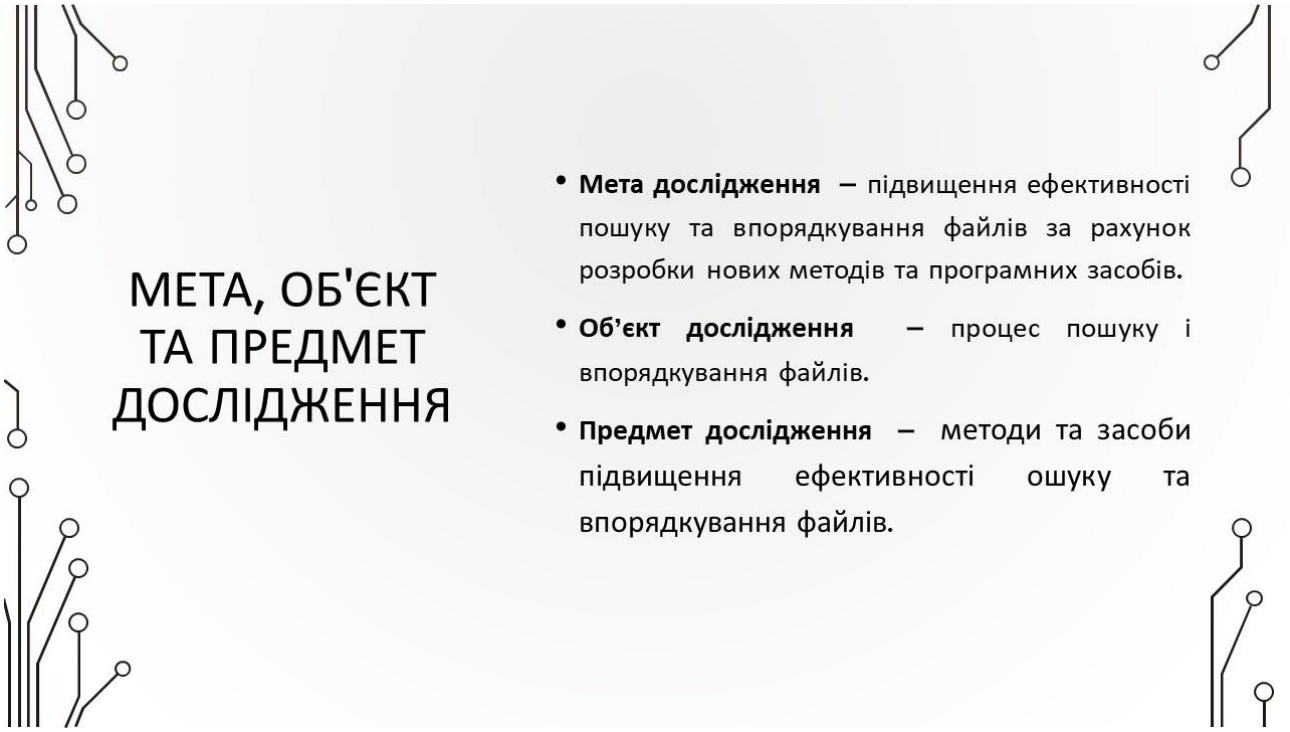

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

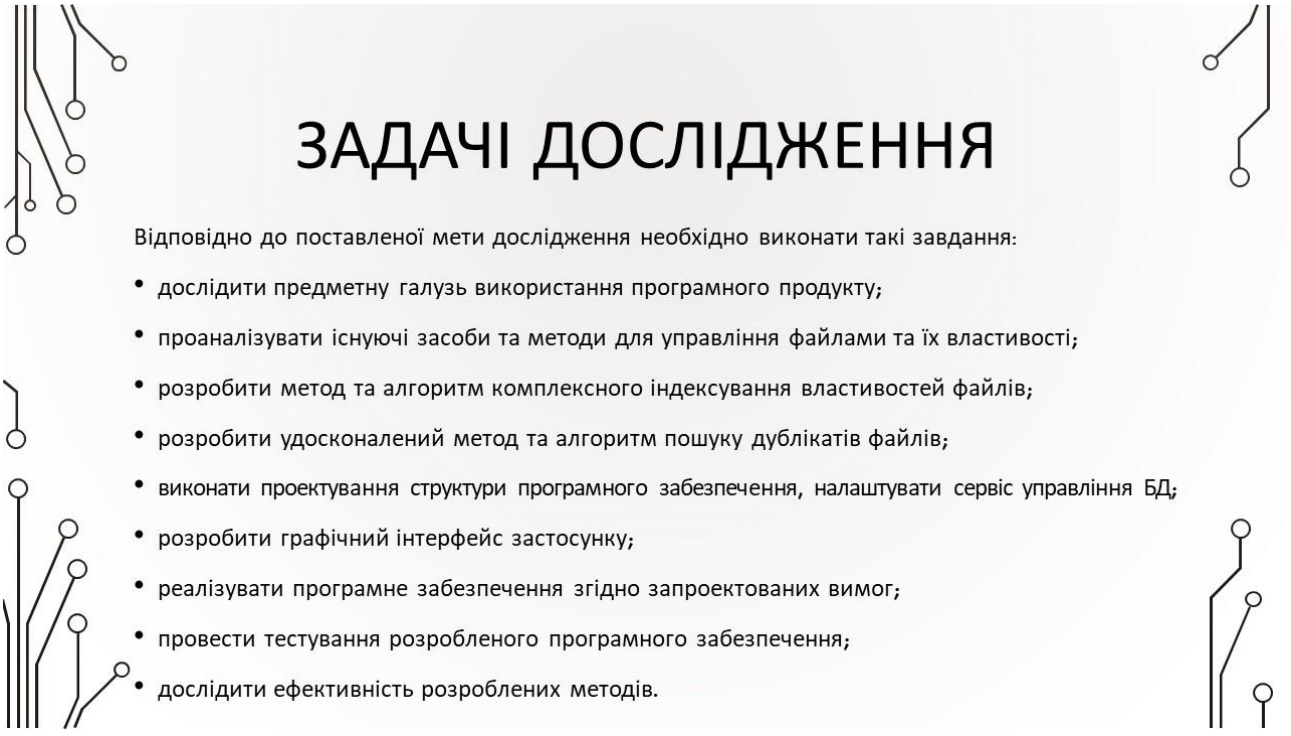

Рисунок Г.4 – Задачі дослідження

Рисунок Г.5 – Наукова новизна

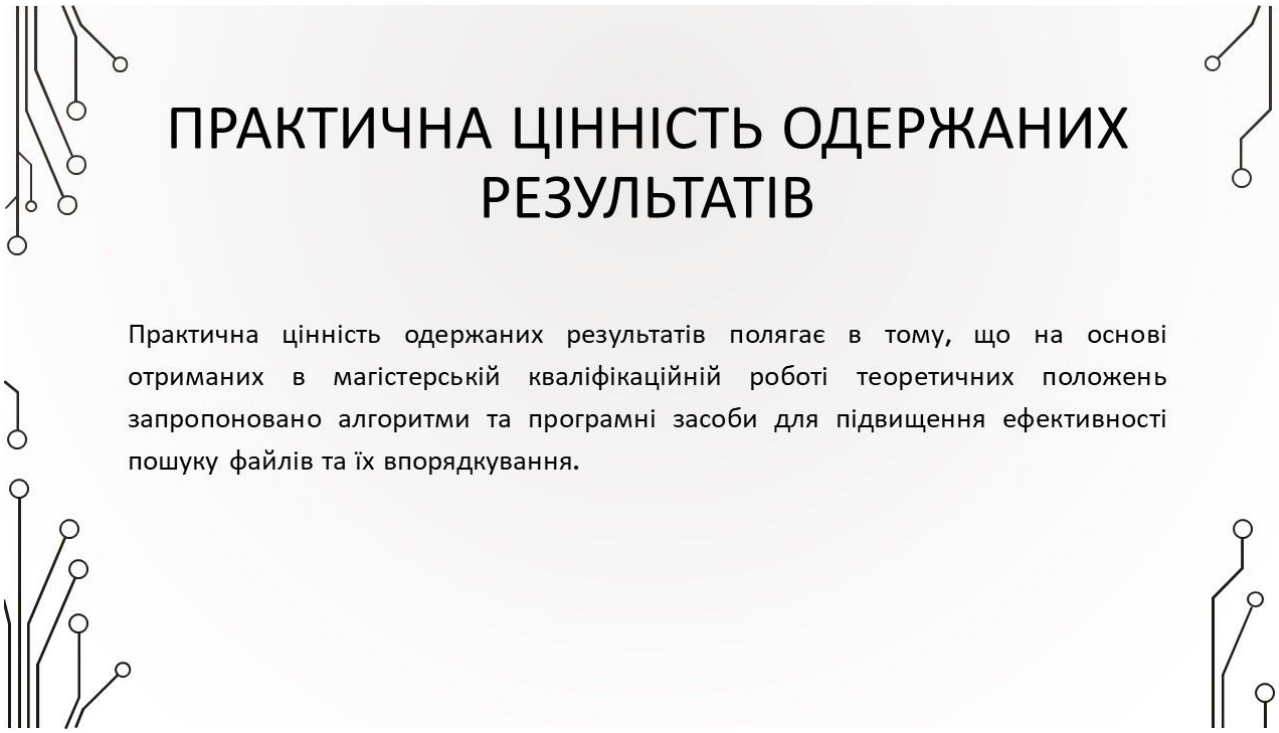

Рисунок Г.6 – Практична цінність одержаних результатів

ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

Критерій	Windows Explorer	Everything	TagSpaces	Власний застосунок
Зручний інтерфейс	3	3	4	5
Вартість продукту	5	4	2	4
Робота з тегами	1	0	5	4
Швидкість пошуку	3	5	3	5
Підтримка різних платформ	1	1	5	5
Сума	13	13	19	23

Рисунок Г.7 – Порівняльний аналіз аналогів

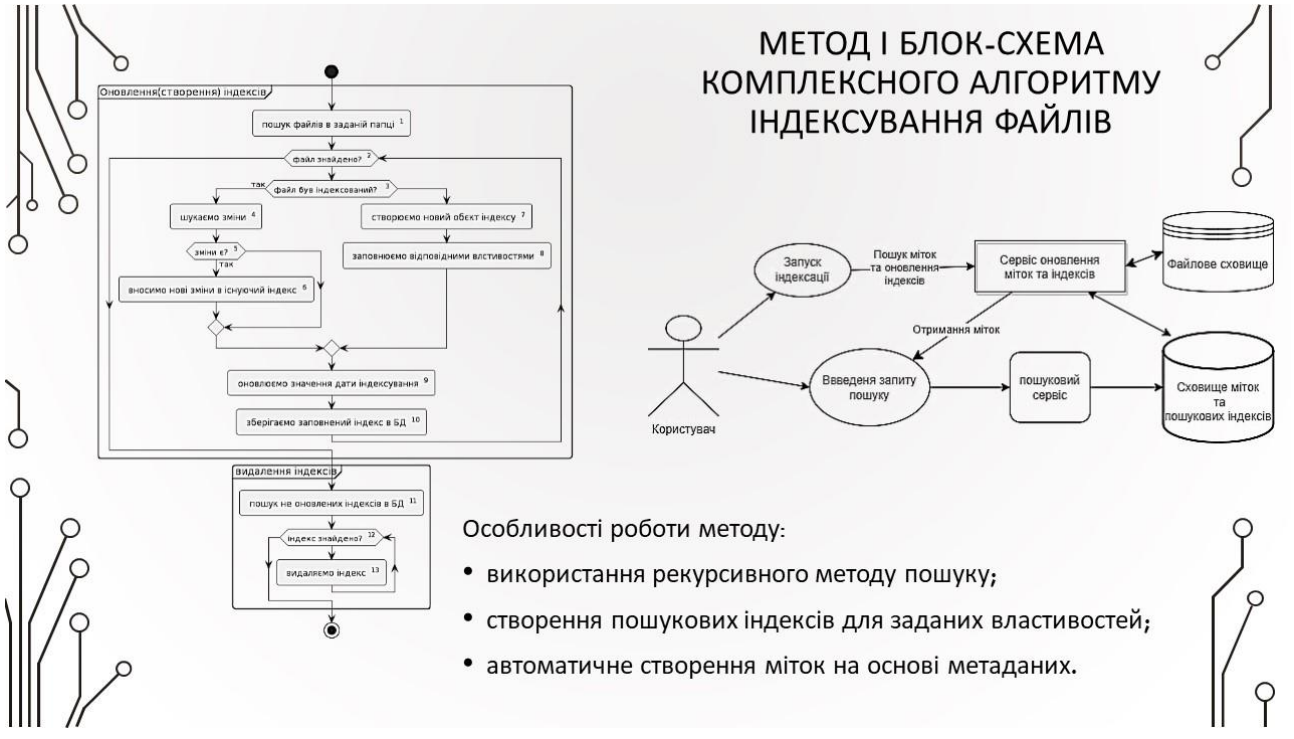

Рисунок Г.8 – Метод і блок-схема комплексного алгоритму індексування файлів

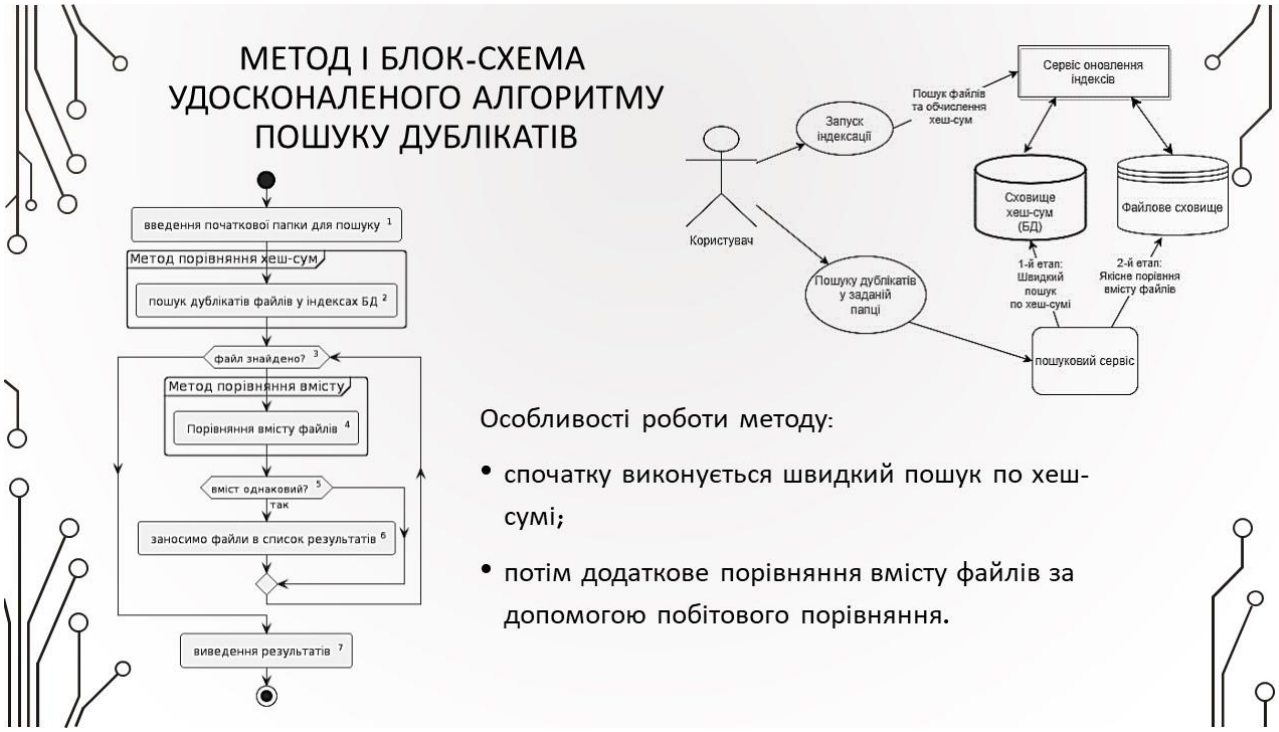

Рисунок Г.9 – Метод і блок-схема удосконалого алгоритму пошуку дублікатів

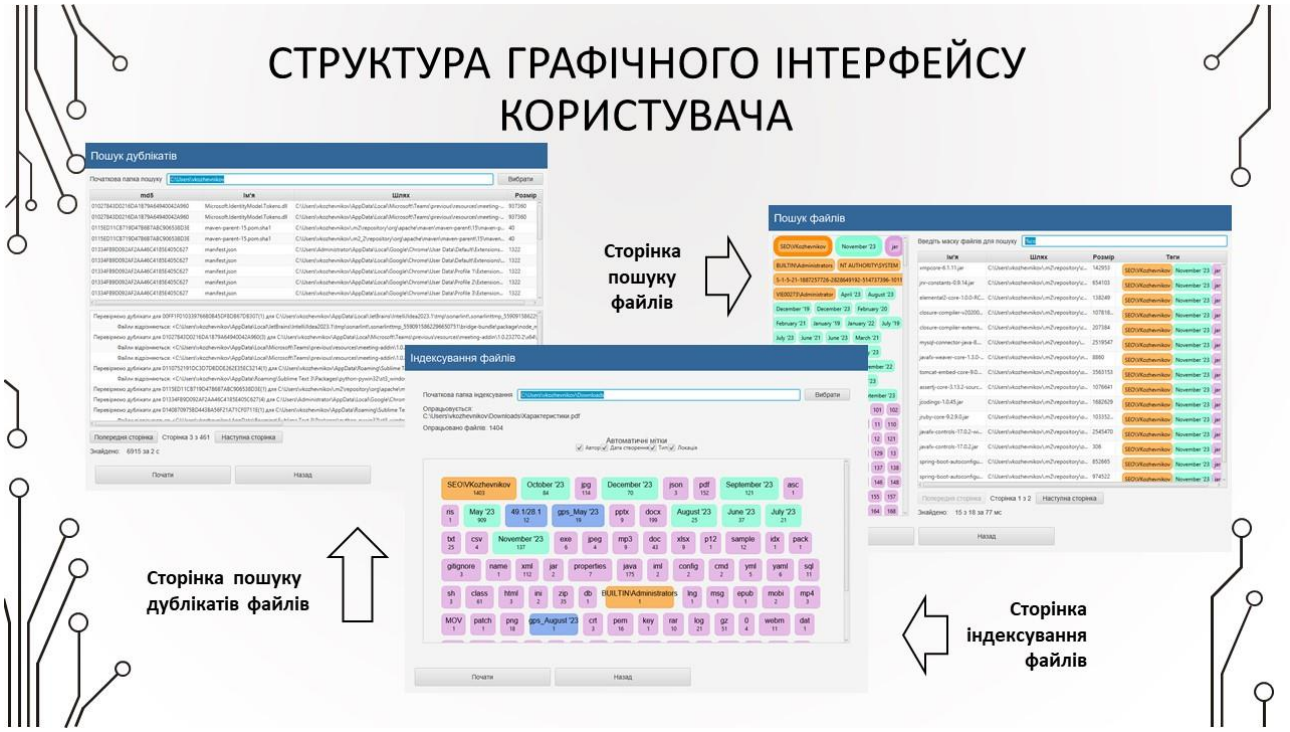

Рисунок Г.10 – Структура графічного інтерфейсу користувача.

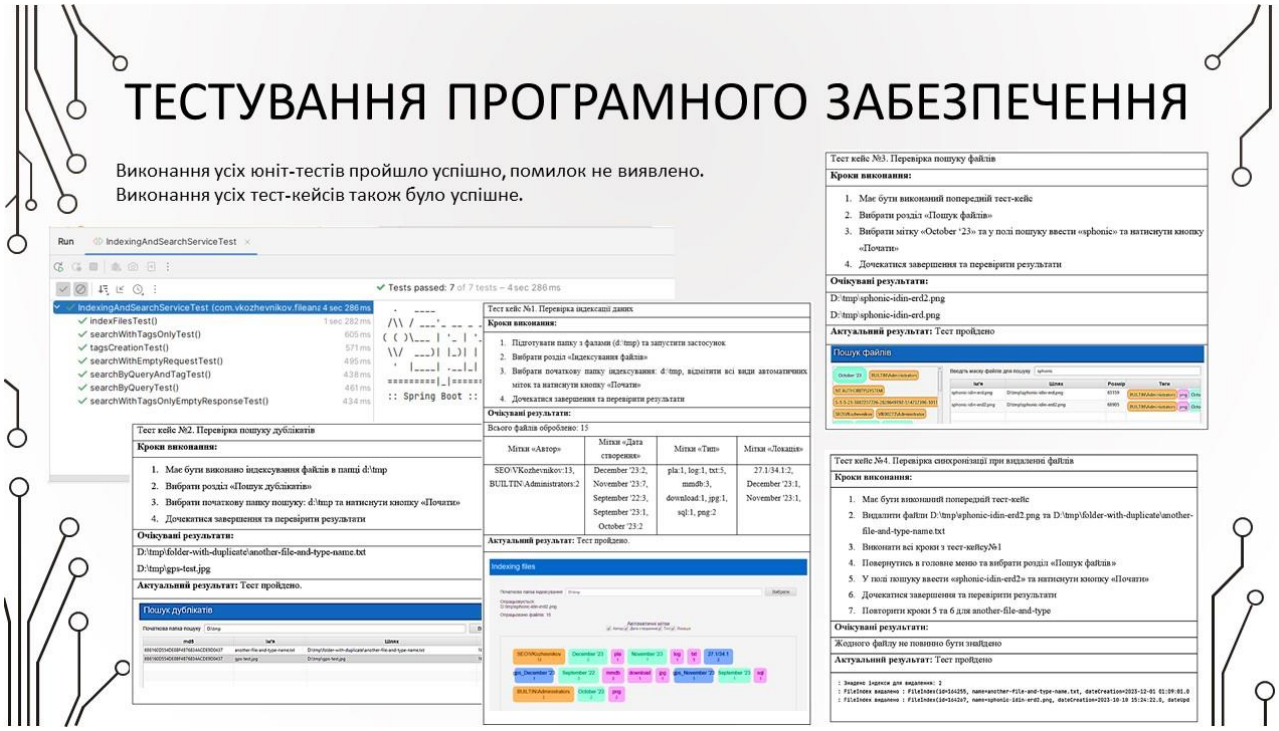

Рисунок Г.11 – Тестування програмного забезпечення

Рисунок Г.12 – Результати експериментальних досліджень шкидкості пошуку

ЕКОНОМІЧНА ЧАСТИНА

- Згідно проведених досліджень рівень комерційного потенціалу розробки становить 40 балів, що свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).
- Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів складає *ЗВ = 778 551,1 грн*
- Період окупності інвестицій, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки складає $T_{ок}$ *= 2,95 року*
- Можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів і програмних засобів для підвищення ефективності пошуку файлів та їх впорядкування».

Рисунок Г.13 – Економічна частина

АПРОБАЦІЯ ТА ПУБЛІКАЦІЇ РЕЗУЛЬТАТІВ РОБОТИ

Результати роботи доповідалися та обговорювалися на:

- Міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 20-21 листопада 2023 р);
- XI Міжнародній науково-практичній конференції «EUROPEAN SCIENTIFIC CONGRESS» (Мадрид, 27-29 листопада 2023р).

Основні результати досліджень опубліковано в 2 наукових працях, у матеріалах конференцій.

Рисунок Г.14 – Апробація та публікація результатів роботи

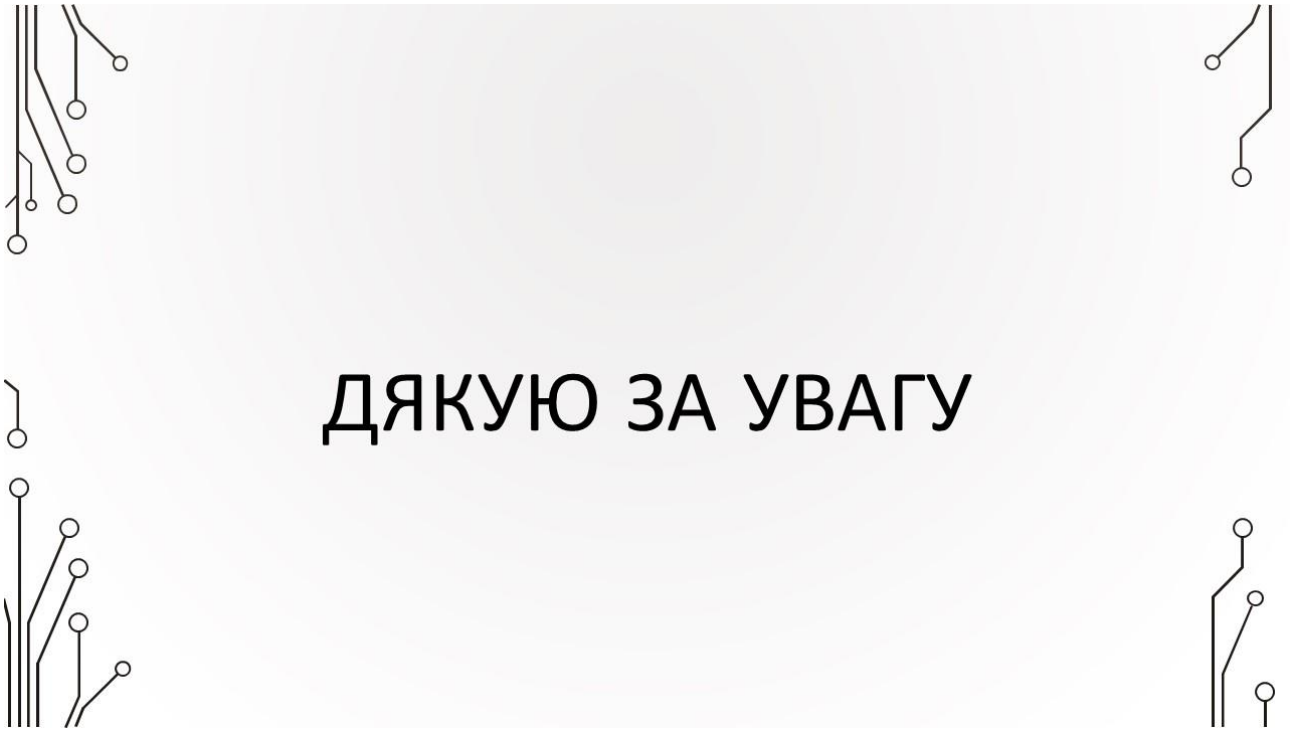

Рисунок Г.15 – Фінальний слайд