МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КІСЕЛЬОВ МИХАЙЛО ДМИТРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«_____» _____ 2025р.

ДОДАТОК ДЛЯ ПАРАМЕТРИЗОВАНОГО ПОШУКУ ЕЛЕКТРОННИХ ДОКУМЕНТІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Бабаков Р.М., докт. техн. наук, доцент,
професор кафедри інформаційних технологій
_____

Оцінка: _____ / _____ / _____
(бали/ за шкалою ЄКТС/ за національною шкалою)
Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Кісельов М.Д. Додаток для параметризованого пошуку електронних документів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2025.

У роботі розглянуто проблеми ефективного пошуку інформації у великих локальних файлових системах. Розроблено настільний застосунок, що дозволяє здійснювати пошук документів за кількома параметрами, зокрема за ключовими словами, типом файлу, датою створення та кількістю сторінок. Програму реалізовано з використанням мови Python та бібліотек tkinter, PyMuPDF і docx. Застосунок протестовано в умовах обробки великої кількості файлів, що підтвердило його стабільність та зручність у користуванні.

Ключові слова: пошук документів, параметри, десктопний застосунок, Python, GUI.

ABSTRACT

Kiselov M. D. Application for Parameterized Search of Electronic Documents. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia, 2025.

The thesis addresses the issue of efficient information retrieval in large local file systems. A desktop application was developed to search documents by multiple parameters, including keywords, file type, creation date, and page count. The program was implemented in Python using tkinter, PyMuPDF, and docx. It was tested under conditions of large-scale file processing, demonstrating stability and ease of use.

Keywords: document search, parameters, desktop application, Python, GUI.

ЗМІСТ

ВСТУП

На сьогоднішній день досить значну кількість інформації зберігають в електронних документах різних форматів. З розвитком цифрових технологій зростає потреба у швидкому та ефективному пошуку необхідних документів на комп'ютерах та в корпоративних архівах. Параметризований пошук, який враховує не лише ключові слова, а й додаткові параметри, такі як формат файлу, дата створення або кількість сторінок, дозволяє підвищити точність і швидкість пошуку.

Об'єктом дослідження в цій роботі є процес пошуку документів у файлових системах, а предметом — методи та інструменти параметризованого пошуку текстових документів. Метою роботи є розробка програмного забезпечення для параметризованого пошуку документів на комп'ютері з графічним інтерфейсом користувача.

Для досягнення поставленої мети було виконано такі завдання: проаналізовано існуючі методи пошуку, вивчено особливості роботи з основними форматами документів (PDF, DOCX, TXT), розроблено алгоритми пошуку за параметрами, створено програмний додаток із зручним інтерфейсом та проведено його тестування.

У роботі використано методи аналізу, програмування на мові Python, застосування бібліотек для роботи з різними форматами документів, а також розробка інтерфейсу на основі бібліотеки tkinter.

Практична цінність роботи полягає у створенні зручного інструменту, який може бути використаний для автоматизованого пошуку документів у різних сферах діяльності — від навчання до бізнесу.

Структура роботи складається з вступу, трьох розділів, висновків та списку використаних джерел. У першому розділі розглянуто теоретичні основи пошуку документів, описано формати файлів і методи пошуку. Другий розділ присвячений розробці алгоритмів і програмного забезпечення. У третьому розділі наведено результати тестування та оцінку ефективності розробленого додатку.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ПАРАМЕТРИЗОВАНОГО ПОШУКУ ДОКУМЕНТІВ

1.1 Актуальність

У сучасному цифровому середовищі значна частина людської діяльності пов’язана з роботою з електронними документами. Документи, що створюються, редагуються, зберігаються і використовуються у повсякденній роботі, охоплюють найрізноманітніші сфери: від особистих нотаток, навчальних матеріалів і ділової кореспонденції — до технічної документації, фінансової звітності, юридичних актів і адміністративних форм. Кожна одиниця інформації — окремий файл — у більшості випадків зберігається у певному форматі, має свої метадані, структуру, дату створення та модифікації, і нерідко — великі обсяги текстового вмісту.

У зв’язку зі зростанням обсягів зберігання даних на особистих комп’ютерах, мережевих сховищах та у хмарних платформах виникає гостра проблема швидкого і точного доступу до потрібної інформації. Користувачі щодня працюють із десятками або сотнями файлів, які зберігаються у різних папках, на різних пристроях, у різних форматах. Якщо ще кілька десятиліть тому документ вважався фізичним носієм знань, то сьогодні він є цифровим об’єктом, який легко дублюється, змінюється і поширюється. Це сприяє зростанню кількості копій одного й того ж документа, зменшенню структурованості збереження файлів і, відповідно, ускладненню процесу їх виявлення.

Традиційні інструменти пошуку файлів, доступні в операційних системах, часто працюють на рівні файлових назв або простого індексування вмісту. Наприклад, у типовому файловому менеджері користувач може знайти документ за його частковою назвою або за деякими метаданими, якщо вони індексовані системою. Однак така модель пошуку є надто загальною і недостатньо гнучкою у випадках, коли потрібно здійснити відбір за більш

точними або комбінованими критеріями. Особливо це стає помітно тоді, коли документи зберігаються в нерозпізнаних форматах або містять специфічні ознаки, які не враховуються типовими механізмами пошуку.

Сучасні завдання, які постають перед користувачами, потребують більш глибокої системи пошуку — здатної враховувати не лише ключові слова, але й такі параметри, як тип документа, його довжина, кількість сторінок, дата створення, структура, наявність вкладених об’єктів, мова тексту, а іноді й місце фізичного збереження. Наприклад, працівник державної установи, шукаючи копію договору, може пам’ятати лише те, що документ був у форматі PDF, мав понад 10 сторінок і був створений приблизно в серпні минулого року. У такій ситуації пошук лише за ключовими словами не дасть результату, якщо не вказати додаткові фільтри. І саме тут виникає потреба у системах параметризованого пошуку, які дозволяють задавати складні багаторівневі умови фільтрації.

З огляду на ці обставини, розробка програмного забезпечення, здатного виконувати параметризований пошук, набуває особливої актуальності. Така система може стати незамінним інструментом у повсякденній роботі — як для пересічного користувача, який зберігає персональні документи на домашньому комп’ютері, так і для співробітника великої компанії, що оперує гігабайтами внутрішньої документації. Наприклад, в архівах бібліотек, де зберігаються скановані копії старих книг і актів, або в юридичних відділах, де необхідно відшукати документ із конкретним номером справи, параметризований підхід дозволяє значно скоротити час на пошук та уникнути людських помилок.

Не менш важливим є аспект точності результатів. Пошукова система, що використовує лише один параметр, як-от ключове слово, може видати сотні або тисячі результатів, серед яких більшість не є релевантними. Коли ж система дозволяє комбінувати кілька критеріїв, це дає змогу значно обмежити область пошуку і сфокусуватися на тих документах, які дійсно

відповідають вимогам користувача. У багатьох випадках саме така можливість робить процес обробки інформації ефективним і керованим.

Актуальність розробки подібних систем також обумовлюється тим, що навіть спеціалізовані інструменти, які існують на ринку, не завжди задовольняють конкретні потреби користувача. Деякі з них орієнтовані лише на корпоративний сектор, інші вимагають встановлення серверних компонентів, а деякі не підтримують повноцінної роботи з усіма поширеними форматами файлів. Крім того, існують обмеження у безкоштовних версіях програмного забезпечення, які зменшують кількість оброблюваних файлів або обмежують доступ до параметрів фільтрації. У таких умовах створення власного адаптованого інструменту параметризованого пошуку може бути доцільним кроком як з технічної, так і з практичної точки зору.

Таким чином, у контексті загального зростання обсягу цифрової інформації, ускладнення структури документів і підвищення вимог до точності обробки даних, розробка системи параметризованого пошуку електронних документів постає як своєчасне, технологічно обґрунтоване і практично необхідне завдання. Вона дозволяє не лише вирішити проблему пошуку в умовах надлишку інформації, але й забезпечити ефективне управління електронними ресурсами в умовах сучасного інформаційного середовища.

1.2 Класифікація та особливості форматів електронних документів

Розуміння форматів електронних документів є ключовим аспектом при побудові ефективної системи пошуку, оскільки саме ці формати визначають спосіб зберігання, доступу до вмісту, можливість обробки метаданих і виконання параметризації результатів. У сучасному цифровому середовищі існує велика кількість форматів, але найпоширенішими у повсякденному використанні залишаються PDF, DOCX та TXT — кожен із них має власну

специфіку, яка безпосередньо впливає на процес аналізу даних і реалізацію пошукових алгоритмів.

Формат PDF (Portable Document Format) був створений як універсальний засіб збереження структури документа незалежно від програмного забезпечення, операційної системи чи апаратної платформи. Він забезпечує візуальну сталість — вигляд документа на екрані точно відповідає друкованому варіанту. Завдяки цьому PDF став стандартом у діловому і юридичному документообігу, офіційній кореспонденції, архівах і публікаціях. Водночас, структура PDF-файлів є складною і багаторівневою. Окрім основного тексту, вони можуть містити векторну графіку, растрові зображення, гіперпосилання, вкладені файли, форми, кнопки, скрипти і навіть мультимедійні об’єкти. Документ у форматі PDF не завжди зберігає текст у простому вигляді — у випадках сканованих сторінок вміст може бути представлено лише як зображення, що унеможливлює класичний текстовий аналіз без застосування технологій оптичного розпізнавання символів (OCR). У свою чергу, навіть у текстових PDF-файлах текст може бути розташований фрагментами у випадковому порядку, без логічної послідовності, що суттєво ускладнює його повноцінне вилучення.

Ще однією особливістю PDF є наявність метаданих, які зберігаються у спеціальних структурах документа. Ці метадані можуть містити відомості про автора, назву, дату створення, тип шрифтів, мову, ключові слова тощо. Однак достовірність цих полів не гарантована, оскільки користувач або стороннє програмне забезпечення можуть змінити їх у будь-який момент. Існує також можливість зашифрувати PDF-документ або обмежити його функціональність (наприклад, заборонити копіювання, друк або відкриття без пароля), що створює додаткові виклики для пошукових систем, які взаємодіють із такими файлами.

Формат DOCX, який прийшов на зміну старішому формату DOC, є представником так званих контейнерних структур. Він реалізований у вигляді архіву, який містить набір XML-документів, зокрема основний вміст

документа, стилі, структуру, медіа-файли, відомості про шрифти, схеми, макети тощо. Завдяки цій структурі DOCX поєднує гнучкість форматування з можливістю програмної обробки на рівні окремих елементів. Цей формат став стандартом де-факто у корпоративному діловодстві, наукових звітах, дисертаціях, навчальних роботах, юридичних документах тощо. DOCX дозволяє зберігати багату структуру документа — з таблицями, виносками, стилями абзаців, графічними вставками, гіперпосиланнями, формулами та іншими елементами.

З точки зору пошуку, формат DOCX є перспективним, оскільки основний вміст представлено в XML, що дозволяє точно визначати межі текстових блоків, їхні властивості та ієрархію. Це відкриває можливості для фільтрації за змістовими категоріями. Водночас складність структури означає, що для повноцінного вилучення тексту необхідно враховувати велику кількість службової інформації, вміло відокремлювати корисний вміст від структурних елементів і уникати втрати даних через некоректне парсування. У DOCX також містяться вбудовані метадані, зокрема дані про автора, редактора, кількість редагувань, дату останнього збереження, програму, якою створено документ. Ці поля можуть бути використані в пошуку, проте потребують додаткової валідації та очищення від шуму. Ще одним викликом при обробці DOCX є наявність версій документа, прихованих фрагментів або вмісту, захищеного паролем, який недоступний без спеціальної обробки.

Формат TXT вважається найпростішим і універсальним, оскільки представляє собою звичайний послідовний запис символів без форматування, розмітки або структурних об'єктів. Саме завдяки своїй лаконічності TXT-файли широко використовуються для зберігання журналів подій, конфігураційних параметрів, чорнових нотаток, логів і службової інформації. Перевагою цього формату є його легка оброблюваність: прочитати TXT-файл можна практично будь-яким програмним засобом, незалежно від платформи чи мови програмування. Проте ця ж особливість створює певні обмеження:

відсутність будь-яких метаданих унеможливлює пошук за додатковими критеріями, окрім текстового вмісту або назви файлу. Крім того, оскільки TXT не має стандартизованого способу зберігання кодування, інтерпретація вмісту може призводити до некоректного відображення символів, особливо у випадку використання кирилиці чи нестандартних шрифтів. У такому разі пошук втрачає точність або стає повністю непридатним.

Окремо слід відзначити відмінності у контекстах використання різних форматів. Так, PDF найчастіше застосовується для розповсюдження готових, завершених документів, які не потребують редагування і мають зберігати фіксований вигляд. DOCX — навпаки, призначений для живого документообігу, коли текст постійно змінюється, додаються нові розділи, ведеться історія змін. TXT же використовується у системному та інженерному середовищі, де головним є простота збереження і доступність. Саме ці особливості зумовлюють необхідність врахування контексту під час пошуку: для PDF — можливість часткового шифрування і візуального шуму, для DOCX — складність внутрішньої структури, а для TXT — відсутність допоміжної інформації.

Крім зазначених аспектів, важливим фактором при класифікації форматів є наявність підтримки з боку бібліотек для програмного опрацювання. У контексті реалізації пошукової системи це означає необхідність використання спеціалізованих інструментів, здатних коректно зчитувати і трактувати структуру файлів. Для PDF-файлів потрібні засоби, які можуть працювати з шарами, шрифтами і кодуванням. Для DOCX — компоненти, що здатні парсити XML-структуру і вилучати потрібні вузли. Для TXT — прості механізми обробки тексту з урахуванням можливого різноманіття кодувань. Усе це робить формат документа не лише носієм інформації, але й технічною змінною, яка визначає спосіб реалізації пошукової логіки та її обмеження.

1.3 Методи повнотекстового пошуку в документах

Сучасні інформаційні системи мають справу з величезними обсягами текстових даних, що зберігаються у вигляді документів різних форматів. В умовах цифрової революції та постійного зростання обсягів інформації ефективний повнотекстовий пошук стає одним із найважливіших інструментів для швидкого доступу до необхідних даних. Повнотекстовий пошук дозволяє знаходити не лише документи, а й конкретні фрагменти тексту всередині них, що істотно підвищує функціональність інформаційних систем.

Ідея повнотекстового пошуку базується на побудові індексу — спеціальної структури даних, яка зберігає інформацію про появу слів у документах. Індекс служить для швидкого пошуку відповідностей ключовим словам чи фразам без необхідності послідовного перегляду всього вмісту документів. Завдяки індексації система може миттєво визначити, в яких документах зустрічаються задані слова, і повернути релевантні результати.

Індексація може бути побудована на різних рівнях: від простого словника слів до складних структур, що враховують позицію слова в документі, частоту використання, синонімічні зв’язки тощо. Це дозволяє не лише швидко знаходити документи, але й здійснювати ранжування результатів за релевантністю.

Найпоширеніший тип повнотекстового пошуку — пошук за ключовими словами. Користувач вводить один або кілька слів, і система шукає документи, що містять ці слова. Такий пошук може бути:

1. Точним — шукаються документи, де зустрічається точна форма слова. Наприклад, запит «пошук» знайде документи з цим словом, але не з формою «пошуку» чи «шукати».

2. Частковим — враховує частини слова або варіації. Наприклад, запит «пошук*» знайде всі слова, що починаються на «пошук».

3. З урахуванням регістру — пошук може бути чутливим або нечутливим до великих і малих літер.

Проте пошук за ключовими словами має суттєві обмеження. Він не враховує синтаксичні чи семантичні відношення між словами, не враховує контексту і може пропускати релевантні документи через морфологічні відмінності слів. Для покращення якості результатів застосовують різні методи обробки тексту.

У багатьох мовах слова змінюються за відмінками, числами, часами, що ускладнює простий пошук. Щоб подолати це, у пошукових системах застосовують стемінг — метод, який зводить слово до основи (стеми), відкидаючи закінчення [1-3]. Наприклад, слова «пошук», «пошуку», «шукати» можуть бути зведені до спільної основи «шук-». Також можна використовувати лематизацію — більш складний процес, який визначає лемму (початкову форму) слова з урахуванням контексту. Це дозволяє точніше ідентифікувати зв’язки між словами.

Застосування цих методів значно покращує якість пошуку, особливо для мов з багатою морфологією, таких як українська чи російська.

Регулярні вирази — потужний інструмент для побудови складних пошукових запитів. Вони дозволяють задавати шаблони, які враховують варіації слів, позиції символів, довжину та інші характеристики. Наприклад, регулярний вираз пошук.* знайде всі слова, що починаються на «пошук», включно з «пошук», «пошуку», «пошуковий» тощо.

Однак робота з регулярними виразами вимагає від користувача знань їхньої синтаксичної структури і може бути складною для непідготовлених осіб. Тому в інтерфейсі програм часто передбачають спрощені форми пошуку або готові шаблони.

Параметризований пошук є розширенням повнотекстового, що дозволяє додатково фільтрувати результати за різними параметрами, окрім ключових слів. Основні параметри включають:

1. Тип файлу — наприклад, PDF, DOCX, TXT. Це дозволяє користувачу швидко виключити непотрібні формати.

2. Дата створення або зміни файлу — корисно для обмеження пошуку документами, які були створені або відредаговані у певний проміжок часу.

3. Розмір файлу або кількість сторінок — наприклад, можна шукати тільки великі документи або документи певної мінімальної довжини.

Параметризований пошук дає змогу зменшити кількість нерелевантних результатів, що особливо важливо при роботі з великими колекціями файлів.

Для забезпечення високої продуктивності пошуку великі системи створюють індекси, які зберігають попередньо оброблену інформацію про вміст документів. Індексація дозволяє знаходити документи за ключовими словами за долі секунди навіть у мільйонних колекціях.

Однак індексація — це ресурсомісткий процес, який вимагає часу і потужності для створення та оновлення індексів при зміні чи додаванні нових документів. У десктопних застосунках часто використовують поліндексований пошук або пошук у реальному часі без індексації, що позначається на швидкості, але спрощує архітектуру.

Метадані — це інформація про документи, яка не входить у їх текстовий вміст, але може бути корисною для пошуку. Прикладами метаданих є дата створення, автор, розмір, ключові слова, опис тощо. Застосування метаданих у параметризованому пошуку дозволяє ефективно фільтрувати документи без необхідності аналізувати повний текст.

Наприклад, пошук документів, створених певним користувачем у заданий період, або пошук файлів, розмір яких перевищує визначений поріг.

Семантичний пошук та штучний інтелект

Сучасні технології активно впроваджують семантичний пошук, що враховує значення слів у контексті, а не лише їх текстове збігання. Для цього використовують методи машинного навчання, нейронні мережі та інші алгоритми штучного інтелекту.

Семантичний пошук дозволяє знаходити документи, які відповідають за змістом, навіть якщо в них немає точних ключових слів. Це особливо важливо для складних інформаційних запитів і великих баз даних.

Однак через високу складність і ресурсоємність такі технології поки що не є стандартом у простих програмних рішеннях для персонального використання.

1.4 Безпека при обробці електронних документів

У контексті систем параметризованого пошуку електронних документів питання безпеки обробки файлів набуває особливої актуальності. З огляду на те, що програмне забезпечення, орієнтоване на аналіз локального вмісту, взаємодіє з численними об’єктами файлової системи, у теоретичному аспекті доцільним є розгляд потенційних загроз і особливостей, пов’язаних із форматами документів, рівнем доступу до файлів та наявністю елементів захисту [4, 5].

Обробка електронних документів може супроводжуватись низкою ризиків, пов’язаних зі структурною складністю деяких форматів. Зокрема, у документах формату PDF допускається наявність вбудованих скриптів, які теоретично здатні впливати на поведінку програмного середовища під час обробки. Аналогічні особливості спостерігаються і в документах формату DOCX, де можлива присутність активного вмісту, зокрема елементів автоматичного оновлення, вбудованих полів та макросів. Теоретично ці механізми можуть бути використані для зміни вмісту документа або виклику зовнішніх ресурсів, хоча їх обробка зазвичай не підтримується бібліотеками, орієнтованими лише на текстовий вміст.

З іншого боку, прості формати, як-от TXT, не передбачають структури, здатної нести небезпеку у вигляді вбудованих об’єктів чи сценаріїв. Водночас навіть у таких документах можливе розміщення фрагментів, які за певних умов можуть бути сприйняті як потенційно небажані. Наприклад, URL-посилання або спеціально сформовані рядки можуть становити інтерес із погляду інформаційної гігієни, особливо в середовищах, де реалізовано автоматичний перехід за посиланням.

Питання прав доступу також відіграє важливу роль у контексті безпечної обробки документів. У середовищі операційних систем із багаторівневою структурою дозволів доступ до файлів може бути обмежено на рівні облікових записів, груп користувачів або системних налаштувань. Документи, що зберігаються у захищених директоріях, можуть бути недоступними для програм, що виконуються від імені користувача з обмеженими правами. У теоретичному плані це означає, що системи, які здійснюють сканування великих обсягів файлів, можуть стикатись із ситуаціями, коли окремі об’єкти не підлягають відкриттю або зчитуванню.

Формати електронних документів, що передбачають можливість встановлення обмежень доступу, додають ще один рівень складності до питання безпеки. Наприклад, PDF-документи можуть бути зашифровані або захищені паролем, що робить неможливим доступ до їх вмісту без відповідного ключа. Існують також варіанти, коли встановлюються обмеження на копіювання, друк або перегляд документа, що потребує наявності відповідної підтримки на рівні програмного забезпечення. Документи такого типу можуть залишатись недоступними для повного аналізу у випадках, коли обробка вмісту здійснюється без інтерактивної участі користувача.

Окрему категорію ризиків становлять нетипові або нестандартні представлення документів. Зокрема, у PDF-файлах текстовий вміст може бути поданий у вигляді зображень, що унеможливлює його традиційне зчитування без використання додаткових механізмів розпізнавання. У таких випадках звичайна текстова обробка не дає очікуваного результату, хоча сам файл формально задовольняє критеріям пошуку за форматом. Подібна ситуація може виникати і в DOCX-документах із вмістом, який розміщено у вкладених об’єктах або представлено не у вигляді основного тексту, а через допоміжні структури.

У межах аналізу безпечної обробки документів теоретичний інтерес становить також питання достовірності метаданих. Багато форматів

документів містять у своїй структурі поля, що зберігають інформацію про автора, дату створення, час останньої модифікації та інші характеристики. Проте такі дані не є обов’язковими для збереження у достовірному вигляді та можуть бути змінені на будь-якому етапі життєвого циклу документа. Це породжує можливість маніпуляцій із метаданими, які при використанні як параметрів пошуку здатні вплинути на точність та надійність результатів. У теоретичній площині це означає, що метадані можуть виконувати допоміжну роль, проте їх інтерпретація потребує обережності.

Безпека обробки документів також передбачає врахування архітектурних особливостей програмного середовища, у якому реалізується функціонал пошуку. З позицій теорії бажаною є організація логіки обробки таким чином, щоб взаємодія з файловою системою та документами здійснювалася у контрольованому середовищі, з передбачуваними механізмами обробки помилок та виключенням виконання стороннього коду. Практика використання спеціалізованих бібліотек, орієнтованих виключно на обробку текстового вмісту, є поширеним рішенням, що дозволяє мінімізувати ризики, пов’язані з інтерпретацією потенційно небезпечних структур.

Таким чином, у рамках побудови системи параметризованого пошуку документів виникає потреба в теоретичному осмисленні аспектів безпеки, пов’язаних із обробкою різних форматів, взаємодією з файловою системою, роботою з метаданими та можливими формами захисту вмісту. Такий аналіз є необхідним для формування надійного та стійкого до помилок підходу, що враховує як структурні, так і поведінкові характеристики електронних документів.

1.5 Огляд існуючих програм і бібліотек (аналоги)

Сучасний ринок програмного забезпечення для пошуку документів представлений широким спектром різноманітних продуктів, від простих утиліт до складних систем, які використовуються у великих організаціях і

корпораціях. Ці рішення суттєво відрізняються за функціональністю, швидкодією, зручністю використання і підтримкою різних форматів документів. Для розробника важливо розуміти, які інструменти вже існують, які їхні сильні та слабкі сторони, щоб створити конкурентоспроможний і корисний продукт.

Одна з найпопулярніших програм для пошуку файлів на локальному комп'ютері — це Everything (рис. 1.1). Вона відома своєю неймовірною швидкістю пошуку за іменами файлів, що досягається завдяки індексації всієї файлової системи. Проте Everything має суттєве обмеження — вона не підтримує пошук за вмістом документів, що значно зменшує її застосування у випадках, коли необхідно знайти документи за текстовим вмістом. Тим не менш, для швидкого пошуку за назвами файлів цей інструмент залишається лідером.

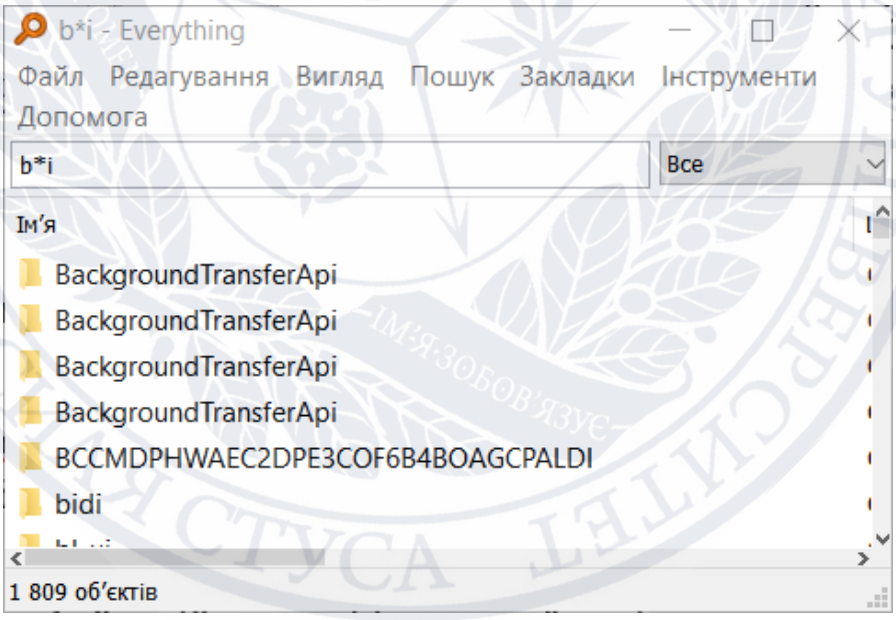

Рисунок 1.1 – Фрагмент інтерфейсу продукту Everything

Іншим популярним і більш функціональним є DocFetcher — безкоштовна програма з відкритим кодом, яка підтримує повнотекстовий пошук у багатьох форматах файлів, включаючи PDF, DOCX, TXT, HTML та

інші. DocFetcher створює індекси вмісту документів, що дозволяє проводити швидкий пошук за ключовими словами. Крім того, програма надає можливість фільтрації за типом файлу та іншими параметрами. Незважаючи на потужність, DocFetcher має деякі недоліки: інтерфейс користувача може бути незручним для початківців, а також потребує попередньої індексації, що займає час і ресурси.

Вбудовані системні інструменти, такі як Windows Search або Spotlight в macOS, також забезпечують базову підтримку повнотекстового пошуку та індексації документів. Вони мають глибоку інтеграцію з операційною системою, що робить їх дуже зручними для повсякденного використання. Проте, ці інструменти мають обмеження щодо гнучкості параметрів пошуку і не завжди можуть задовольнити потреби користувачів, яким потрібен більш тонкий і контрольований пошук.

Як приклад можна взяти стандартну функціональність операційної системи Windows (рис. 1.2). Вбудована система пошуку дозволяє знаходити файли за іменем, частково за вмістом та деякими метаданими. Вона використовує індексацію обраних користувачем каталогів, що забезпечує певну швидкість при повторному зверненні до вже проаналізованих даних. Разом з тим, така система орієнтована переважно на базові потреби та не передбачає широкої гнучкості при комбінуванні умов пошуку. Застосування додаткових фільтрів, таких як часові діапазони, типи документів або вміст тексту, реалізується лише частково, а для складніших запитів вимагає знання спеціального синтаксису. У випадках, коли документи зберігаються поза межами індексованих областей, стандартний пошук також не гарантує повноти результатів.

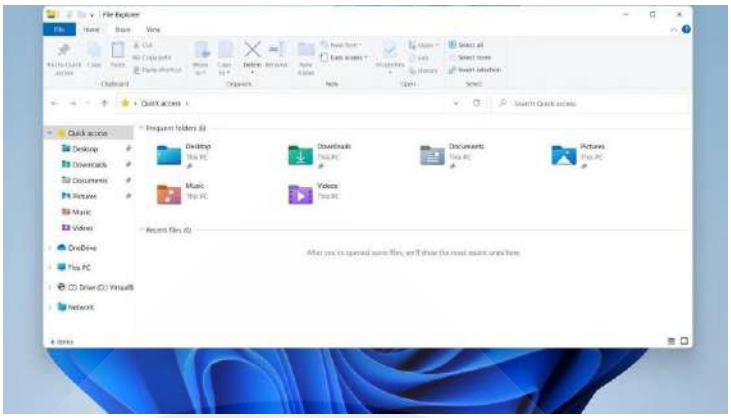

Рисунок 1.2 – Вікно пошуку фалів через File Explorer у Windows

Copernic Desktop Search — це комерційний продукт з широким набором можливостей, який підтримує не лише пошук по файлах і документах, а й по електронній пошті, мультимедіа та інших типах даних. Цей пошуковик має інтуїтивний інтерфейс і розвинені можливості фільтрації, включно з параметричним пошуком. Однак, платна ліцензія може бути перешкодою для деяких користувачів, особливо для приватних осіб чи невеликих компаній.

Для розробників відкриті численні бібліотеки, які полегшують роботу з різними форматами документів і реалізацію пошукових механізмів. PyMuPDF (відомий також як fitz) дозволяє ефективно працювати з PDF-файлами, витягуючи текст і метадані, проте не підтримує оптичне розпізнавання, що ускладнює роботу з відсканованими документами. Бібліотека python-docx надає можливість читати та обробляти DOCX-файли, розпаковуючи структуру документа і отримуючи текстовий вміст. Комбінація модулів os і re дозволяє обходити файлову систему і виконувати пошук із застосуванням регулярних виразів для складних запитів. Для створення інтерфейсу користувача в Python широко використовується tkinter, що забезпечує простий у використанні графічний інтерфейс.

Проведений аналіз показує, що існуючі рішення або не мають достатньої гнучкості у параметрах пошуку, або мають складний інтерфейс, який ускладнює їхнє використання для непрофесіоналів. Саме це створює

потребу у розробці програмного забезпечення, яке поєднуватиме зручність, простоту і розширений функціонал параметризованого пошуку документів.

1.6 Аналіз ринку програм пошуку документів

Ринок програмного забезпечення для пошуку документів активно розвивається, оскільки зростає потреба організацій та приватних користувачів у швидкому і точному доступі до інформації. Програми пошуку є важливою складовою систем управління документами, електронних архівів, корпоративних сховищ та індивідуальних робочих місць.

Сучасні ринки характеризуються наявністю як великих гравців, що пропонують комплексні рішення для бізнесу, так і безлічі простих десктопних утиліт, орієнтованих на побутове використання. Високий рівень конкуренції змушує виробників прагнути до удосконалення функціоналу, покращення швидкодії та зручності користувацького інтерфейсу.

Великий сегмент ринку займають комерційні продукти, такі як Copernic Desktop Search (рис. 1.3) , X1 Search, dtSearch, які пропонують глибоку інтеграцію з корпоративними системами, розвинені можливості індексації, підтримку великої кількості форматів файлів, а також складні функції фільтрації і аналізу. Вони орієнтовані на потреби бізнесу, що вимагає обробки великих обсягів даних і підтримки командної роботи. Вартість таких продуктів часто є значною, що обмежує їх використання у приватних осіб та малих підприємств.

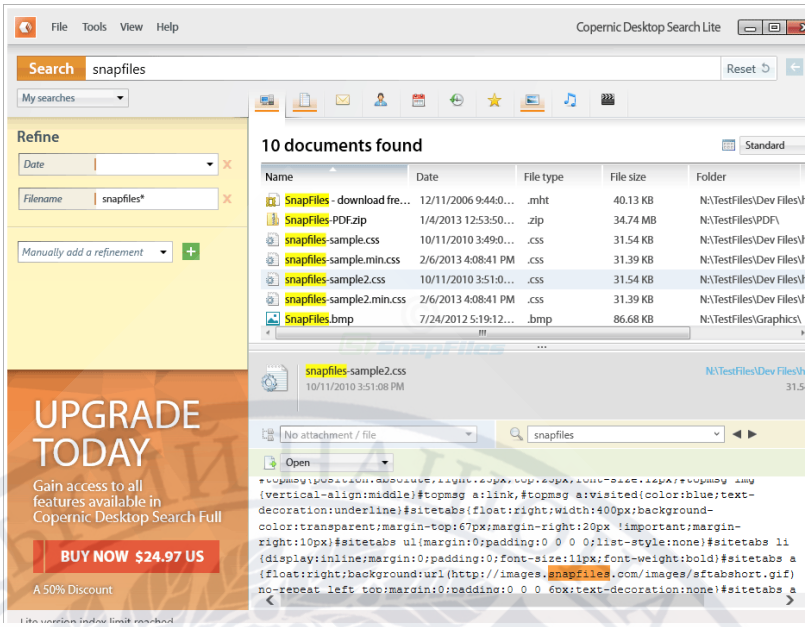

Рисунок 1.3 – Фрагмент інтерфейсу продукту Copernic Desktop Search

Безкоштовні та open-source рішення, такі як DocFetcher (рис. 1.4), Recoll, Searchmonkey, забезпечують базову функціональність повнотекстового пошуку і підтримують різноманітні формати файлів. Вони привабливі для окремих користувачів і невеликих організацій, але часто поступаються комерційним продуктам у зручності інтерфейсу та швидкодії.

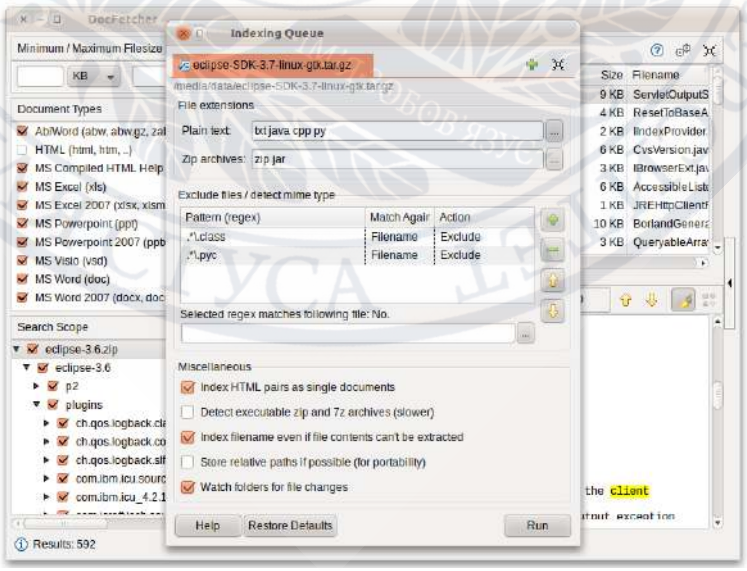

Рисунок 1.4 – Фрагмент інтерфейсу продукту DocFetcher

У сегменті мобільних додатків також спостерігається активний розвиток, з появою утиліт, що дозволяють здійснювати пошук документів на смартфонах і планшетах, що особливо актуально в умовах росту мобільності користувачів.

Загальна тенденція розвитку ринку полягає у підвищенні ролі параметризованого та семантичного пошуку, інтеграції з хмарними сервісами, а також впровадженні штучного інтелекту для покращення якості пошуку.

Проте у багатьох існуючих рішеннях відсутня достатня простота і доступність для звичайних користувачів, що створює можливість для розробки інтуїтивно зрозумілих і гнучких додатків, які поєднують у собі потужний функціонал і зручність.

Таким чином, аналіз ринку підтверджує актуальність розробки програмного забезпечення, що підтримує параметризований пошук документів із простим інтерфейсом і широкими можливостями, яке може бути використане як у професійній, так і у повсякденній діяльності.

1.7 Постановка задачі

Основним завданням даної дипломної роботи є розробка програмного забезпечення для параметризованого пошуку текстових документів на персональному комп’ютері з графічним інтерфейсом користувача. Програма повинна забезпечувати пошук за ключовими словами із можливістю фільтрації результатів за форматом файлу, датою створення або зміни, кількістю сторінок та іншими параметрами.

Процес формування функціоналу системи базується на аналізі потреб користувачів та сучасних методів пошуку документів. При виборі технологій і підходів до реалізації враховано ефективність, зручність використання, а також можливість подальшого масштабування та розвитку застосунку.

У межах роботи поставлено такі завдання:

1. розробити механізми зчитування та обробки документів різних форматів;

2. реалізувати алгоритми параметризованого пошуку за ключовими словами та фільтрації за додатковими параметрами;

3. створити інтуїтивно зрозумілий графічний інтерфейс для зручної взаємодії користувача з програмою;

4. провести тестування функціоналу і оцінити ефективність розробленого рішення.

Впровадження розробленої системи дозволить значно підвищити ефективність пошуку документів, скоротити час на пошук необхідної інформації та забезпечити гнучкі можливості пошуку в різних сферах діяльності.

РОЗДІЛ 2

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАМЕТРИЗОВАНОГО ПОШУКУ ДОКУМЕНТІВ

2.1 Загальна архітектура системи

Розробка програмного забезпечення для параметризованого пошуку документів є складним завданням, що вимагає продуманого підходу до побудови архітектури системи. Під час створення програмного продукту основною метою було поєднати функціональність, продуктивність та зручність користування, що уможливить ефективну роботу із великими обсягами документів різних форматів.

Для розробки програмного забезпечення використовувалось інтегроване середовище Visual Studio Code (рис. 2.1) [6-8]. Це сучасне середовище розробки, створене компанією Microsoft, яке набуло широкої популярності завдяки своїй легкості, гнучкості та розширюваності. Visual Studio Code забезпечує підтримку Python «з коробки», має вбудований термінал, потужні засоби автодоповнення, відлагодження, а також велику кількість розширень, що значно спрощують розробку та налагодження програмного коду. Його застосування дозволило ефективно організувати процес програмування, тестування та керування проєктом.

Visual Studio Code

Рисунок 2.1 – Логотип Visual Studio Code

Першим кроком на шляху розробки стало визначення загальної структури програми. Розглядалося декілька варіантів.

Монолітна архітектура — де всі компоненти (інтерфейс, логіка пошуку, обробка файлів) реалізуються в одному модулі. Такий підхід спрощує початкову реалізацію і дозволяє швидко отримати робочий прототип. Проте у міру ускладнення програми монолітний код стає важким для підтримки, тестування і розширення. Зокрема, при масштабуванні пошуку або додаванні нових форматів файлів, код стає заплутаним і уразливим до помилок.

Модульна (компонентна) архітектура — із чітким поділом на окремі логічні блоки, які відповідають за різні функції: користувацький інтерфейс, обробку параметрів пошуку, безпосередній пошук, роботу з файлами і відображення результатів. Такий підхід значно підвищує гнучкість і дозволяє розробляти, тестувати та вдосконалювати кожен модуль незалежно. Крім того, він дає можливість легко додавати підтримку нових функцій і форматів. Проте потребує значно більшого обсягу часу, а також ретельного планування взаємодії між модулями.

Клієнт-серверна архітектура — передбачає розподіл системи на клієнтську частину (інтерфейс) і сервер (логіка пошуку). Це більш складний варіант, який дозволяє розподіляти навантаження і масштабувати систему, але потребує налаштування мережевої взаємодії і серверного оточення. Для десктопної програми, що працює локально на одному комп’ютері, цей варіант був надмірним.

З огляду на цілі проєкту, обмежені строки та обсяг, було обрано монолітну реалізацію з внутрішнім логічним поділом на функціональні блоки. Такий підхід поєднує простоту впровадження з можливістю майбутнього розширення або рефакторингу у модульну структуру.

На основі обраної архітектурної моделі система організована як єдиний застосунок із наступними внутрішніми логічними компонентами:

1. Графічний інтерфейс користувача (GUI) — створений з використанням бібліотеки tkinter. Цей компонент відповідає за збір параметрів пошуку від користувача, запуск і зупинку пошуку, відображення прогресу та результатів. Основне завдання — забезпечити інтуїтивно зрозуміле і комфортне середовище для користувача, де він може легко задавати умови пошуку і переглядати знайдені файли.

2. Модуль обробки параметрів — отримує введені користувачем дані (ключові слова, дати, кількість сторінок, формати файлів), проводить їх валідацію і підготовку для передачі в пошуковий модуль. Цей блок гарантує коректність введених значень і унеможливлює запуск пошуку з некоректними параметрами.

3. Пошуковий компонент — реалізує алгоритм обходу файлової системи, відбору файлів за форматом і метаданими, вилучення тексту з документів і пошук ключових слів у тексті. Використовує спеціалізовані бібліотеки (PyMuPDF для PDF, python-docx для DOCX, стандартне читання для TXT) для максимально коректної роботи з різними форматами.

4. Механізм керування багатопоточністю — забезпечує запуск пошуку в окремому потоці, що дозволяє не блокувати інтерфейс користувача і відображати актуальний прогрес виконання. Дозволяє реалізувати функцію зупинки пошуку за запитом користувача.

5. Компонент виводу результатів — управляє додаванням знайдених файлів у скролювальний текстовий блок, обробкою кліків на результатах і відкриттям вибраних файлів за допомогою системних засобів.

Вибір бібліотек для реалізації функціональності здійснювався на основі балансу між простотою використання, надійністю та швидкістю розробки. Для побудови графічного інтерфейсу було обрано tkinter — стандартну бібліотеку Python, яка не потребує додаткового встановлення і забезпечує базові можливості для створення форм, полів введення, кнопок та прогресбару. Попри існування потужніших інструментів, таких як PyQt або wxPython, tkinter виявився оптимальним варіантом з огляду на строки

виконання проєкту та потребу в кросплатформеності без складного розгортання. Для роботи з PDF-документами було використано PyMuPDF (fitz) — швидку й стабільну бібліотеку, яка забезпечує просте отримання тексту зі сторінок документів. Інші бібліотеки, такі як pdfminer, хоча й мають глибші аналітичні можливості, виявилися менш зручними в інтеграції та повільнішими у виконанні. Обробку DOCX-файлів реалізовано за допомогою python-docx, яка дозволяє витягувати текстову інформацію та підтримує структуру абзаців без необхідності взаємодії з внутрішньою XML-структурою файлів. Для реалізації багатопоточності застосовано стандартний модуль threading, який дозволяє запускати пошук у фоновому режимі без блокування інтерфейсу, що забезпечує плавну роботу GUI навіть при обробці великої кількості файлів [9-12].

Обрана структура програми забезпечує логічну організованість, спрощує підтримку коду та дозволяє при потребі перенести частини логіки у зовнішні модулі. За необхідності програму можна масштабувати, винісши пошукову логіку в окремий клас або модуль, або розширити підтримку форматів без суттєвої перебудови системи.

2.2 Реалізація графічного інтерфейсу користувача

Одним із ключових аспектів зручності користування програмою є реалізація графічного інтерфейсу. Навіть найфункціональніша система буде незручною й малопридатною до практичного використання, якщо вона не забезпечує інтуїтивно зрозумілий і наочний інтерфейс. Саме тому на етапі проєктування GUI особливу увагу було приділено ергономіці, послідовності взаємодії з користувачем, мінімізації кількості кроків до отримання результату пошуку та забезпеченню зворотного зв’язку про стан системи [13-15].

Для реалізації графічного інтерфейсу було проаналізовано кілька популярних бібліотек для Python:

1. PyQt/PySide — надзвичайно потужна бібліотека для створення складних інтерфейсів із сучасним виглядом. Має багатий набір віджетів, підтримує стилі оформлення, анімації, вкладки та багато іншого. Основний недолік — складність у вивченні для новачка, об'ємність коду, а також необхідність встановлення додаткових бібліотек, що не входять до стандартної поставки Python.

2. Kivy — сучасна бібліотека з підтримкою кросплатформенності (у тому числі для мобільних додатків). Має нестандартний підхід до побудови інтерфейсів, потребує часу на освоєння. Переваги: гнучкість, адаптивність, підтримка жестів.

3. Tkinter — стандартна бібліотека, яка входить до складу Python «з коробки». Має все необхідне для створення базового інтерфейсу: поля вводу, кнопки, фрейми, списки, текстові області, чекбокси тощо. Працює на всіх основних платформах без необхідності інсталяції сторонніх компонентів [16-19].

З огляду на те, що метою було створення настільного інструменту з простим та ефективним інтерфейсом, і зважаючи на обмежений час на реалізацію, було вирішено використовувати саме tkinter. Ця бібліотека дозволила швидко створити робочий прототип і надалі розширювати функціональність без суттєвих технічних труднощів.

Інтерфейс програми побудований вертикально, за логікою «згори вниз», де кожен наступний елемент є продовженням попереднього. Це відповідає природному порядку дій користувача — від налаштувань параметрів до запуску пошуку й перегляду результатів.

Основні елементи інтерфейсу:

1. Кнопка перемикання мови (UA/EN) — розташована у верхньому правому куті. Дає змогу змінити мову інтерфейсу на українську або англійську. Це рішення забезпечує доступність програми для ширшого кола користувачів. Для реалізації мови створено словник з перекладами, який підключається динамічно.

2. Поле введення ключових слів — основне текстове поле, в яке користувач вводить слова для пошуку. Воно одразу видно після запуску програми, що акцентує увагу на головній функції.

3. Поля для задання кількості сторінок — розміщені у горизонтальному фреймі. Два окремих поля для мінімальної і максимальної кількості сторінок дозволяють задавати діапазон. Якщо поля порожні — обмеження не застосовується.

4. Поля для введення дат — аналогічно реалізовані у наступному фреймі. Формат введення РРРР-ММ-ДД є універсальним і легко піддається парсингу за допомогою datetime.

5. Чекбокси форматів файлів (PDF, DOCX, TXT) — увімкнені за замовчуванням. Дають змогу фільтрувати типи файлів при пошуку. Усі три типи активні одразу після запуску, що зручно для більшості користувачів.

6. Кнопки запуску та зупинки пошуку — чітко відокремлені і розміщені по центру. При запуску пошуку кнопка зупинки активується, дозволяючи перервати процес у будь-який момент.

7. Прогресбар і текстова індикація прогресу — з’являються лише під час пошуку, не захаращують інтерфейс у спокійному стані. Цей підхід дозволяє зробити інтерфейс чистішим і менш відволікаючим.

8. Результати пошуку — виводяться у великий ScrolledText, розташований у нижній частині вікна. Натискання на шлях до файлу відкриває документ. Над результатами є підпис-нагадування про цю функцію.

Одним із важливих завдань було зробити інтерфейс максимально очевидним і дружнім до користувача. Це досягалося через зрозумілі написи біля кожного елементу керування, використання за замовчуванням активованих чекбоксів форматів файлів, а також додавання підказки над полем результатів, що інформує користувача про можливість відкривати файли за кліком. Крім того, було враховано типові помилки користувача, наприклад, ситуації, коли не задано жодного параметра пошуку — у таких

випадках з’являється відповідне попередження. Окрему увагу приділено логіці активності елементів: кнопка зупинки, наприклад, залишається неактивною до моменту початку пошуку, що запобігає помилковим діям.

Такий підхід до побудови інтерфейсу був розроблений на основі спостережень за реальними сценаріями використання — коли користувач хоче якомога швидше знайти файл, не заглиблюючись у складні налаштування. Саме тому, наприклад, пошук запускається однією кнопкою, всі формати активовані одразу, а прогрес показується у відсотках.

Інтерфейс підтримує дві мови (українську і англійську), що реалізовано за допомогою словника зі статичними мітками. У подальшому легко додати інші мови, не змінюючи логіку роботи програми. Такий підхід до локалізації був обраний як найпростіший і найефективніший для десктопного додатку з обмеженим обсягом тексту в інтерфейсі.

2.3 Обробка параметрів пошуку та валідація введених даних

Одним із важливих компонентів системи є обробка введених користувачем параметрів пошуку. Їхня правильна інтерпретація, перевірка на коректність і подальша передача до пошукового модуля є критичними для стабільної роботи програми. Якщо цього не зробити — система може або виконати пошук за хибними критеріями, або завершитися з помилкою ще до початку пошуку. Тому до реалізації цього етапу було застосовано підхід з чітким розділенням: введення, перевірка, обробка, передача.

Користувач вводить одне або кілька ключових слів у текстовому полі. Спочатку вводилося припущення, що варто дозволити пошук лише за цілим виразом, однак це обмежувало б можливості. Було обрано гнучкіший підхід: кожне слово, розділене пробілом, інтерпретується як окремий пошуковий запит. Це дозволяє знайти більше релевантних документів, де можуть міститися не всі слова водночас, а хоча б одне з них.

Щоб пошук був нечутливим до регістру, слова, введені користувачем, порівнюються із вмістом документа у приведеній до однакового формату

текстовій формі. Це гарантує, що "Договір" і "договір" будуть трактуватись однаково.

Також було передбачено перевірку: якщо не введено жодного слова та інші параметри також не задані, система виводить попередження — користувач має вказати хоча б одну умову пошуку, інакше запуск не має сенсу.

Користувач може задати мінімальну і максимальну кількість сторінок документа. Значення вводяться у два окремі поля. Спочатку вони трактуються як текст, тому система перевіряє, чи є ці значення цілими числами. Якщо хоча б одне з них задано — воно буде враховане як обмеження під час фільтрації.

Додатково реалізовано перевірку на логічність діапазону: якщо введено і мінімальне, і максимальне значення, але мінімальне більше за максимальне — пошук не запускається, а користувач отримує відповідне повідомлення.

Водночас було враховано, що точну кількість сторінок можливо дізнатись лише для PDF-документів. Для DOCX і TXT це неможливо визначити без складного аналізу. Тому при спробі одночасно задати обмеження за сторінками і вибрати лише TXT або DOCX — програма видає повідомлення про неможливість такої перевірки.

Для підвищення точності пошуку користувач може вказати діапазон дат створення документів. Обидва поля (дата «від» і дата «до») вводяться вручну у форматі «рік-місяць-день». Такий формат є міжнародно прийнятим, зручним для обробки і зрозумілим користувачеві.

Під час введення перевіряється правильність формату. Якщо користувач випадково вводить дати у довільному вигляді (наприклад, із крапками, косими рисками або у текстовій формі), програма зупиняє пошук і виводить відповідне повідомлення про помилку.

Крім цього, перевіряється логіка діапазону: наприклад, дата «від» не може бути пізнішою за дату «до». Якщо одне з полів залишити порожнім —

воно не буде враховуватись як обмеження. Таким чином, користувач може задати лише одну межу (наприклад, тільки «не пізніше» певної дати).

У процесі пошуку для кожного файла система зчитує дату його створення та порівнює з заданим діапазоном. Якщо файл не входить у діапазон — він виключається з результатів.

Користувач може обрати, з якими форматами документів проводити пошук: PDF, DOCX або TXT. Для цього в інтерфейсі передбачені три незалежні перемикачі. За замовчуванням усі формати активовані — це зроблено для зручності більшості користувачів, які не хочуть вручну вмикати кожен тип.

Перед запуском пошуку програма перевіряє стан усіх трьох перемикачів. Якщо користувач знімає всі галочки, система не виводить помилку, а навпаки — трактує це як бажання шукати по всіх форматах. Це рішення було прийняте для підвищення зручності і запобігання помилковим зупинкам пошуку.

Після завершення всіх перевірок і обробки параметрів ці дані передаються у пошуковий модуль у вигляді готових структур — чисел, дат і списків. Це дає змогу ізолювати процес збору параметрів від безпосереднього виконання пошуку, що спрощує і структуру коду, і можливість тестування кожної частини системи окремо.

Таким чином, обробка параметрів пошуку у програмі реалізована із врахуванням практичних сценаріїв, типових помилок користувачів і технічних обмежень. Це дозволяє зробити процес взаємодії з системою не лише функціональним, а й надійним і зручним у реальному використанні.

2.4 Робота з різними форматами документів

Однією з найважливіших вимог до системи параметризованого пошуку є підтримка декількох типів текстових документів. Враховуючи різноманітність форматів, які можуть використовуватись користувачами у

повсякденній роботі, доцільно було зосередитись на трьох найбільш поширених**:** PDF**,** DOCX **та** TXT**.**

Кожен із цих форматів має свою специфіку структури, способу зберігання тексту, метаданих і технічних засобів для обробки. Це унеможливлює створення універсального методу вилучення інформації та потребує індивідуального підходу до кожного типу файлу. У цьому підрозділі розглядається логіка обробки кожного з трьох форматів документів, а також пояснюється, чому були обрані саме ті бібліотеки, які реалізовано у програмі.

Формат PDF (Portable Document Format) є одним з найпоширеніших у світі для зберігання фінальної версії документів. Його основна перевага — фіксоване відображення на всіх пристроях. Однак, з точки зору розробника, PDF є одним із найскладніших форматів для вилучення тексту. Це пов’язано з тим, що PDF може містити як «живий» текст (який можна скопіювати), так і зображення (наприклад, відсканові сторінки), а також має складну внутрішню структуру.

Для обробки PDF у програмі було обрано бібліотеку PyMuPDF (відомою також як fitz). Це рішення ґрунтується на порівнянні з іншими популярними варіантами, такими як pdfminer.six та PyPDF2, і зумовлене перевагами, які надає PyMuPDF [20-22]. По-перше, ця бібліотека має високу швидкодію — вона демонструє значно кращу продуктивність при обробці великих PDF-файлів, ніж pdfminer. По-друге, її синтаксис є дуже простим, що дозволяє зчитувати текст зі сторінок буквально у кілька рядків коду, спрощуючи розробку. Крім того, PyMuPDF повністю підтримує Unicode, що забезпечує коректну роботу з документами українською мовою. Ще однією перевагою є можливість визначати кількість сторінок у документі, що є критично важливим для реалізації фільтрації за цим параметром.

Процес обробки PDF-файлів у програмі реалізовано наступним чином: документ відкривається за допомогою PyMuPDF, зчитується загальна кількість сторінок, після чого кожна сторінка послідовно опрацьовується з

метою вилучення тексту. Увесь отриманий текст конкатенується в один рядок, який використовується для подальшого пошуку за ключовими словами.

Якщо документ не містить текстового шару (наприклад, відсканований PDF), вилучення буде неможливим, але PyMuPDF у такому випадку просто повертає порожній результат — без аварійного завершення програми.

Формат DOCX — це сучасний формат текстових документів, який використовується в Microsoft Word. Він побудований на основі XML і містить структуру документа (абзаци, таблиці, стилі, шрифти, розділи тощо). Однак для цілей повнотекстового пошуку не потрібно глибоко аналізувати структуру — достатньо витягти текст абзаців.

Для обробки DOCX було використано бібліотеку python-docx, яка дозволяє відкривати документ, проходити по абзацах і збирати текстовий вміст. Альтернативні рішення на кшталт docx2txt або textract були відкинуті через обмежену гнучкість або додаткові залежності (наприклад, потреба в зовнішніх утилітах).

Важливим викликом стало питання оцінки кількості сторінок, оскільки DOCX не містить прямої інформації про сторінкову розмітку. Для розв’язання цього було реалізовано наближену оцінку: програма підраховує кількість слів у документі, і далі ділить це число на умовну норму в 500 слів на сторінку. Це грубий, але візуально досить точний спосіб, який дозволяє користувачеві хоча б приблизно фільтрувати за обсягом документа.

Формат TXT — найпростіший і найстаріший формат зберігання текстової інформації. Він не має структурованості, шрифтів, стилів чи метаданих — це просто послідовність символів.

Перевагою є те, що його можна читати без будь-яких додаткових бібліотек — стандартними засобами Python. Водночас, як і у випадку з DOCX, у TXT-файлах неможливо точно визначити кількість сторінок. Навіть наближена оцінка є менш надійною, оскільки вміст може бути дуже

нерівномірно розподілений — один рядок у файлі може бути цілим абзацом, а може бути порожнім.

Для читання тексту з TXT використовується звичайне відкриття файлу у режимі читання з кодуванням UTF-8. Була врахована можливість помилок при зчитуванні файлів з некоректним кодуванням (наприклад, Windows-1251). У таких випадках використовуються параметри errors='ignore', щоби не переривати пошук повністю — а лише пропустити пошкоджені символи.

Незалежно від формату, після вилучення тексту, він зводиться до єдиного вигляду — суцільного рядка у нижньому регістрі. Це дозволяє забезпечити єдиний підхід до пошуку ключових слів і виключити проблеми, пов’язані з регістром чи форматуванням.

Усі етапи зчитування файлів — від відкриття до аналізу вмісту — були обгорнуті в конструкції обробки винятків. Це дозволяє програмі бути стійкою до пошкоджених або нестандартних файлів, не переривати пошук, а просто пропускати проблемні документи з виведенням відповідного повідомлення в консоль розробника.

Реалізація підтримки трьох основних форматів — PDF, DOCX та TXT — дозволила охопити найбільш поширені сценарії використання програми. Обрані бібліотеки виявилися оптимальними за співвідношенням гнучкості, простоти та надійності. Кожен формат обробляється окремим блоком, що полегшує майбутню підтримку та можливе розширення (наприклад, додавання підтримки RTF, ODT чи HTML у наступних версіях програми).

2.5 Основний алгоритм параметризованого пошуку

Ключовим компонентом у роботі програми є реалізація механізму пошуку документів, що відповідають заданим параметрам. Цей модуль забезпечує реальну функціональність системи, тобто перевіряє кожен файл у вибраній користувачем директорії та визначає, чи відповідає він критеріям пошуку. Для реалізації такого функціоналу в Python доступна велика кількість готових алгоритмів, що дозволяють гнучко підходити до побудови

пошукової логіки [23-26]. Побудова цього алгоритму потребувала зваженого підходу — потрібно було знайти компроміс між точністю пошуку, швидкодією, стійкістю до помилок і зручністю інтеграції з графічним інтерфейсом.

Першим етапом у процесі пошуку є отримання повного списку потенційно релевантних файлів для перевірки. Для цього у програмі реалізовано обхід дерева директорій з проходженням усіх вкладених папок у межах обраної користувачем директорії. Під час розробки були розглянуті два підходи: використання рекурсивної функції з ручними викликами для кожної вкладеної папки та застосування методу os.walk() — вбудованого інструменту мови Python [27-29]. Останній було обрано як основний варіант, оскільки він виявився більш стабільним, не потребує ручного контролю за рекурсією та одразу повертає повні шляхи до файлів у кожному каталозі. Завдяки цьому реалізація обходу вийшла як ефективною, так і безпечною з точки зору навантаження на систему.

Після того як зібрано список усіх файлів у структурі теки, виконується попередня фільтрація. З нього виключаються всі об’єкти, які не відповідають обраним форматам — залишаються лише ті файли, розширення яких були дозволені користувачем у налаштуваннях пошуку, наприклад, тільки .pdf і .docx.

Після формування списку файлів алгоритм послідовно перевіряє кожен із них. Цей процес включає кілька кроків.

Першим із них є перевірка дати створення файлу. Для цього використовується дата, яку повертає системна функція отримання часу створення (в операційних системах Windows — це саме дата створення, у Linux/macOS — час останньої модифікації, що є важливою технічною відмінністю). Якщо задано обмеження по датах, програма порівнює ці значення з діапазоном. У разі, якщо файл створено поза межами вказаного діапазону, він одразу виключається з пошуку, що дозволяє зменшити навантаження на процес вилучення тексту.

Наступним етапом є обробка вмісту файлу. Як описано в попередньому підрозділі, кожен формат обробляється окремо. Отриманий текст приводиться до єдиного вигляду — тобто формується як суцільний рядок у нижньому регістрі — для уніфікованої роботи алгоритму пошуку.

Далі виконується перевірка кількості сторінок. Якщо користувач вказав обмеження по кількості сторінок, програма застосовує відповідні перевірки. У PDF-документах кількість сторінок визначається безпосередньо, у DOCX — приблизно через кількість слів, а у TXT такі перевірки не здійснюються, тому ці файли автоматично виключаються з обробки, якщо встановлено обмеження за кількістю сторінок. Цей крок був важливий для досягнення функціональності параметризованого пошуку.

Останній і найбільш важливий етап — пошук ключових слів. Розглядалися варіанти точного або часткового збігу, врахування морфології та порядку слів. Для першої версії програми обрано простий та ефективний підхід: якщо хоча б одне ключове слово зі списку знайдене у тексті файлу, файл вважається релевантним. Це дозволяє знайти документи, які містять одне або кілька заданих слів, і не вимагає повного збігу чи порядку, що підвищує зручність. Пошук реалізовано через просте порівняння з текстом, без використання регулярних виразів чи лематизації, що дозволяє суттєво зекономити час обробки. У наступних версіях можна розширити цей алгоритм, додавши більш складну фільтрацію або навіть підтримку логічних операторів (AND, OR, NOT).

Якщо файл проходить усі перевірки, його шлях додається до результатів пошуку. Одразу після цього він виводиться у текстове поле результатів, область прокручується так, щоб користувач одразу бачив останній запис, а також оновлюється індикатор прогресу — у відсотках, згідно з кількістю перевірених файлів. Цей механізм дає користувачу постійний зворотний зв’язок і відчуття контролю.

Під час роботи алгоритму можливі ситуації, коли файл пошкоджено, немає прав доступу або виникає помилка в бібліотеці при зчитуванні

(наприклад, порожній PDF чи нестандартний DOCX). Усі ці випадки перехоплюються конструкціями обробки винятків (try-except). Помилка не зупиняє роботу програми — файл просто пропускається, а повідомлення виводиться в консоль для розробника. Це дозволяє виконати пошук повністю, навіть якщо серед тисяч файлів є кілька проблемних.

Прогрес обчислюється на основі індексу поточного файлу у списку всіх перевірених. Після завершення пошуку індикатор прогресу досягає 100%, у вікні результатів з’являється повідомлення про завершення пошуку, а кнопка зупинки стає неактивною. У разі, якщо пошук було зупинено користувачем вручну, програма акуратно припиняє обробку і видає відповідне повідомлення — без переривання потоку або виведення помилок.

Алгоритм параметризованого пошуку реалізований у вигляді послідовного фільтрування, яке враховує усі задані параметри користувача: дату створення, формат, кількість сторінок, ключові слова. Основні акценти зроблено на стійкість, швидкодію та зручність інтеграції з графічним інтерфейсом. Архітектура дозволяє легко розширити або замінити окремі етапи в майбутньому — наприклад, додати індексацію, розподілений пошук або підтримку нових форматів документів.

2.6 Забезпечення управління процесом пошуку

Під час реалізації програми постала практична проблема: пошук файлів у великій директорії може займати значний час. Якщо цей процес виконувати у головному потоці, який одночасно відповідає за графічний інтерфейс, програма «зависає» — не реагує на натискання кнопок, не оновлює прогресбар, і користувач сприймає це як збій. Тому виникла потреба в розмежуванні основного потоку інтерфейсу та фонової логіки пошуку, тобто у використанні багатопоточності.

У Python існує кілька підходів до паралельного виконання коду. Найпростішим є модуль threading, який дозволяє запускати окремі потоки всередині одного процесу. Цей варіант ідеально підходить для задач, що не

потребують складних обчислень, але займають час на введення/виведення, як у випадку з обробкою файлів. Альтернативним є модуль multiprocessing, який запускає окремі процеси і краще підходить для обчислювально інтенсивних задач, однак ускладнює обмін даними. Третім варіантом є асинхронне програмування з використанням asyncio, яке ефективне для роботи з мережею, але погано інтегрується з tkinter, оскільки ця бібліотека не підтримує асинхронний цикл подій. З огляду на специфіку розробки було обрано threading, оскільки пошук не потребує складних обчислень, більшість часу витрачається на зчитування файлів, а сама бібліотека проста у реалізації й найкраще поєднується з tkinter.

У графічному інтерфейсі реалізовано кнопку «Почати пошук». При її натисканні активується функція, яка не викликає пошук безпосередньо, а створює новий потік і передає йому цільову функцію пошуку. Алгоритм дій передбачає, що після натискання кнопки в основному потоці створюється об’єкт Thread, що запускає функцію пошуку у фоновому режимі. Пошук відбувається паралельно з роботою головного інтерфейсу, який продовжує оновлювати прогресбар, приймати кліки та дозволяє зупинити пошук.

Такий підхід забезпечує плавну, не блокуючу роботу інтерфейсу під час активного пошуку, що є критично важливим для користувацького досвіду. Крім запуску пошуку, користувач повинен мати змогу зупинити процес у будь-який момент. Для цього реалізовано механізм на основі логічного прапорця. При натисканні кнопки «Зупинити» змінна self.stop_search_flag встановлюється у значення True, а під час кожної ітерації циклу, що перебирає файли, виконується перевірка цього прапорця. Якщо він активний, пошук негайно припиняється шляхом виходу з функції. Після завершення або примусової зупинки кнопка «Зупинити» блокується. Такий підхід дозволяє безпечно контролювати пошук ізсередини, без складної взаємодії між потоками або спроб насильно завершити потік, що вважається небажаною практикою в Python.

У більшості середовищ розробки пряма взаємодія з графічним інтерфейсом із фонових потоків заборонена. Проте tkinter дозволяє частково оновлювати елементи GUI з іншого потоку, зокрема значення прогресбару, текст у мітках або вміст текстових полів. Для цього використовується пряме оновлення відповідних атрибутів, наприклад, self.progress['value'], self.progress_label.config(...), або вставка рядків у ScrolledText для результатів пошуку. Щоб зміни стали видимими негайно, періодично викликається self.root.update_idletasks(), яка примусово оновлює інтерфейс.

Оскільки пошук охоплює велику кількість файлів, можливі помилки, пов’язані з нестандартними умовами: відсутність прав доступу, пошкоджені документи, нестандартні кодування. Усі ці винятки можуть призвести до аварійного завершення потоку, якщо їх не обробити. Тому кожна операція читання файлу обгорнута у конструкцію try-except, яка перехоплює помилки і виводить повідомлення в консоль. Завдяки цьому програма продовжує працювати навіть у разі виявлення проблемних файлів.

Під час активного пошуку кнопка запуску деактивується, щоб користувач не міг випадково запустити кілька паралельних пошуків. Натомість активується кнопка зупинки. Після завершення процесу, незалежно від того, чи був пошук завершений автоматично, чи зупинений вручну, інтерфейс повертається у початковий стан: кнопка зупинки блокується, а кнопка запуску стає доступною.

Реалізований підхід має низку переваг. Програма не зависає навіть при обробці тисяч файлів, користувач бачить прогрес у реальному часі, пошук можна зупинити у будь-який момент, а вся логіка проста, стабільна і легко масштабована. Водночас не використовується складна синхронізація потоків, що знижує ризики помилок. Така реалізація багатопотокової обробки дозволила зробити програму зручною навіть для тривалих задач. Підхід з логічним прапорцем і контролем GUI з головного потоку є класичним рішенням у подібних застосунках і може бути адаптований у майбутньому —

зокрема, для паралельної обробки кількох директорій або створення черги завдань.

2.7 Відкриття знайдених файлів

Однією з важливих функцій програми після успішного пошуку є можливість відкриття знайдених документів безпосередньо з інтерфейсу. Це дозволяє користувачу не лише знаходити потрібні файли, але й одразу отримувати до них доступ — без потреби вручну шукати їх у файловій системі. Фактично, програма повинна виконувати роль повноцінного засобу навігації по релевантних документах.

У багатьох подібних системах відкриття результатів реалізовано через подвійне натискання або окрему кнопку «Відкрити». Після аналізу варіантів реалізації було прийнято рішення використати найпростіший і водночас інтуїтивно зрозумілий підхід — відкриття файлу після одинарного кліку лівою кнопкою миші на відповідному рядку в списку результатів. Це рішення дозволяє зекономити простір у вікні, не потребує додавання окремих кнопок, мінімізує кількість дій користувача та зберігає чистоту і простоту інтерфейсу.

Для реалізації цього підходу використано текстову область із прокруткою (ScrolledText), яка містить список усіх знайдених файлів. Кожен шлях виводиться у новому рядку, і ця область є інтерактивною — вона реагує на натискання миші, тому до неї можна прив’язати подію відкриття файлу. Після вставлення шляху до текстового поля програма асоціює з ним обробник події натискання. Під час кліку визначається координата клацання, яка перетворюється на індекс рядка у полі. Далі зчитується текст цього рядка, що фактично є повним шляхом до файлу. Якщо такий файл існує, програма намагається його відкрити.

Для відкриття використано виклик системного засобу за замовчуванням. У середовищі Windows це реалізовано через os.startfile(). Хоча у Linux або macOS могли б використовуватись інші команди на кшталт xdg-open або open, у межах даного проєкту, орієнтованого на Windows,

реалізовано саме варіант із os.startfile(). Такий підхід дозволяє відкривати будь-який тип файлу засобами тієї програми, яка асоційована з відповідним розширенням на комп’ютері користувача. Наприклад, PDF відкриється в Adobe Acrobat або Chrome, DOCX — у Microsoft Word або LibreOffice, TXT — у Блокноті чи Notepad++.

Обрана реалізація є універсальною, не потребує створення власного переглядача документів, не залежить від конкретного формату і забезпечує максимальну гнучкість. Водночас під час розробки було враховано, що файл міг бути переміщений, видалений або пошкоджений після завершення пошуку. Тому спроба відкриття обгорнута у конструкцію обробки помилок. Якщо файл не існує або виникає помилка при спробі відкриття, повідомлення про це не виводиться користувачу у вигляді окремого діалогового вікна, щоб не створювати додатковий стрес або плутанину, але фіксується в консольному виводі для розробника. У майбутніх версіях можливо реалізувати більш зручне повідомлення в самій програмі з поясненням причини помилки.

Щоб користувач зрозумів, що файли в списку можна відкривати, над полем результатів розміщено пояснювальний напис: «Натиснувши на файл, ви зможете його відкрити». Цей напис оновлюється при зміні мови інтерфейсу і відповідає загальній логіці багатомовності програми.

Загалом механізм відкриття файлів безпосередньо з вікна результатів був реалізований максимально просто та інтуїтивно. Він не вимагає від користувача додаткових дій або налаштувань, не залежить від формату файлу і забезпечує миттєвий доступ до результатів пошуку. Такий підхід суттєво підвищує практичну цінність застосунку і робить його зручним інструментом для щоденного використання.

2.8 Підтримка багатомовності інтерфейсу

Сучасні прикладні програмні продукти повинні орієнтуватися на широку аудиторію користувачів, що зумовлює необхідність забезпечення багатомовної підтримки інтерфейсу. У контексті реалізації даного програмного забезпечення для параметризованого пошуку документів, було передбачено механізм динамічного перемикання мови інтерфейсу між українською та англійською без необхідності перезапуску застосунку.

Зміна мови реалізується через окрему кнопку в правому верхньому куті вікна програми. Натискання цієї кнопки викликає відповідну функцію, яка змінює мовну змінну системи та оновлює всі текстові елементи інтерфейсу згідно з обраною мовою. В якості базового підходу використано словникове представлення інтерфейсних міток, згруповане за мовними ключами. Такий підхід забезпечує не лише гнучкість при масштабуванні системи локалізації, але й простоту підтримки й редагування міток у разі зміни вимог.

У структурі коду функція отримання міток get_labels() повертає словник ключових текстів, у якому кожен елемент відповідає певному елементу інтерфейсу (наприклад, назва кнопки, напис у полі тощо). При перемиканні мови викликається метод update_labels(), який переглядає всі відповідні GUI-компоненти та оновлює їхні текстові значення відповідно до поточної мовної змінної. Завдяки цьому всі елементи — від назв кнопок до підписів над текстовими полями — миттєво адаптуються під обрану локалізацію.

Слід зазначити, що реалізація підтримки багатомовності вбудована безпосередньо в інтерфейсну частину програми, без потреби у зовнішніх мовних файлах або модульних бібліотеках локалізації. Це значно спрощує структуру проекту та уможливлює централізоване оновлення термінології. При цьому всі текстові значення зосереджені у межах одного методу, що підвищує зрозумілість та спрощує супровід коду.

Багатомовна підтримка охоплює всі ключові повідомлення, з якими взаємодіє користувач під час роботи з програмою: від назв полів введення до

діалогових вікон повідомлень про помилки чи результати пошуку. Застосування такого підходу дозволяє зберігати консистентність термінології незалежно від мови, а також забезпечує можливість розширення мовної підтримки шляхом додавання нових словників з відповідною локалізацією.

У цілому, реалізований підхід до багатомовності не лише підвищує доступність застосунку, але й демонструє придатність програми до використання в середовищах з різномовною аудиторією. Логіка реалізації є узагальнюваною та відкритою до подальшого вдосконалення у разі потреби в підтримці інших мов.

2.9 Обробка винятків при роботі з файлами та взаємодія з користувачем

У процесі розробки прикладного програмного забезпечення, орієнтованого на пошук інформації у великій кількості електронних документів, ключовим фактором стабільності функціонування є здатність системи адекватно реагувати на помилки, що виникають під час виконання основних операцій [30]. Особливу увагу при реалізації було приділено обробці виняткових ситуацій, пов’язаних із роботою з файловою системою, відкриттям і читанням документів різних форматів, а також валідацією параметрів, введених користувачем у графічному інтерфейсі.

Значна частина потенційно критичних ситуацій пов’язана з обробкою PDF-документів. З огляду на складну структуру цього формату, можливе виникнення низки помилок: від захисту паролем до відсутності текстового шару, що унеможливлює вилучення вмісту. У разі пошкодження структури документа або невідповідності формату, бібліотека PyMuPDF, яка використовується в системі, може генерувати винятки, що призводять до аварійного завершення програми, якщо їх не перехопити. Для запобігання таким ситуаціям усі операції читання PDF-файлів обгорнуті в захищені конструкції з обробкою винятків. У разі помилки файл просто пропускається,

а інформація про збій виводиться в службову консоль. Подібний підхід дозволяє гарантувати продовження обробки інших файлів незалежно від інцидентів, пов’язаних з окремими об’єктами.

Формат DOCX, попри загальну поширеність, також створює низку потенційних ускладнень. Файли можуть бути пошкодженими, містити нетипову структуру або взагалі не містити жодного абзацу тексту. Бібліотека python-docx має низку внутрішніх перевірок, однак не завжди здатна коректно обробити неконсистентні документи. Програмою реалізовано перехоплення таких винятків, завдяки чому будь-який DOCX-документ, що не піддається аналізу, ігнорується без порушення загального алгоритму пошуку. Повідомлення про помилку у взаємодії з DOCX також надсилається у службовий потік, а не відображається користувачеві, що дозволяє зберегти цілісність інтерфейсу та уникнути надмірної інформації для кінцевого користувача.

Текстові файли у форматі .txt часто містять дані, збережені в різних кодуваннях. Це може призвести до ситуацій, у яких файл технічно відкривається, однак його вміст не може бути прочитаним без втрати частини символів. Особливо часто подібне спостерігається у документах, створених у середовищах, що не підтримують UTF-8. Для обробки таких випадків було реалізовано відкриття файлів із параметром ігнорування помилок кодування. Це дозволяє уникнути аварійного припинення пошуку через наявність некоректних символів і забезпечити обробку навіть частково зіпсованих текстових файлів.

Окрему групу винятків становлять ситуації, пов’язані не з файлами, а з діями самого користувача. Йдеться про введення некоректних параметрів пошуку — таких як неправильно задані дати, числові значення, що не є цілими числами, або порожні поля у всіх критичних параметрах одночасно. Для обробки таких сценаріїв в інтерфейсі передбачено валідацію введених даних та відображення відповідних повідомлень через вікна типу messagebox. Кожне таке повідомлення містить конкретну вказівку на те, що саме

потребує виправлення, що забезпечує користувача зворотним зв’язком у зручній формі. Наприклад, у разі введення дати у форматі, відмінному від YYYY-MM-DD, користувач бачить попередження про некоректний формат дати, а саме поле не передається в основний алгоритм пошуку.

Усі повідомлення для користувача реалізовані з урахуванням мовної локалізації, відповідно до вибраної мови інтерфейсу. Це означає, що як інформаційні, так і помилкові повідомлення, що з’являються під час взаємодії, відображаються тією мовою, яка була обрана у головному вікні програми. Таким чином, забезпечено не лише технічну стійкість системи, але й комфорт користувача під час роботи з нею.

Сумарно реалізовані механізми забезпечують високу стійкість програмного забезпечення до непередбачуваних ситуацій. Застосування конструкцій з обробкою винятків, перевірка введених даних, мінімізація критичних точок відмови та продумана взаємодія з користувачем є тими факторами, що дозволяють програмі функціонувати ефективно навіть за наявності складних або нестандартних вхідних умов. Це особливо важливо в умовах обробки великої кількості документів, де навіть один пошкоджений або неправильний файл не повинен впливати на результат усього пошуку.

РОЗДІЛ 3
ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Завершення розробки та інтеграція функцій

На фінальному етапі розробки програмного забезпечення важливу роль відіграє процес інтеграції всіх компонентів у цілісну, стабільну систему, яка відповідає вимогам до функціональності, зручності у використанні та надійності. У контексті створення програмного застосунку для параметризованого пошуку документів завершальна фаза роботи передбачала не лише технічне поєднання окремих модулів, а й повноцінне забезпечення взаємодії між графічним інтерфейсом користувача, пошуковим алгоритмом, обробкою файлів та логікою контролю процесу.

Однією з ключових задач на цьому етапі стало забезпечення зручної архітектури програми, яка дозволяє масштабувати застосунок у майбутньому, забезпечує простоту обслуговування та розширення функціоналу. З цією метою було прийнято рішення реалізувати всю логіку в межах одного основного класу DocumentSearcher, що інкапсулює в собі всі важливі методи й компоненти: створення інтерфейсу, обробку подій, запуск пошуку, перевірку файлів та виведення результатів.

Такий підхід виправданий у випадках розробки настільних застосунків середнього масштабу, де немає потреби у складній багаторівневій структурі, властивій вебсервісам або корпоративним системам. Завдяки інкапсуляції стало можливим централізовано керувати всіма процесами, що значно спростило тестування, відлагодження й подальше документування коду.

Інтерфейс користувача реалізований за допомогою бібліотеки tkinter, яка є стандартною частиною мови Python. Основною причиною вибору tkinter стала його інтегрованість у Python, простота використання, можливість створення інтерфейсів без потреби у додаткових зовнішніх

залежностях. Попри свою простоту, tkinter дозволяє створювати достатньо функціональні й естетично прийнятні інтерфейси для настільних програм.

У межах інтерфейсу було реалізовано низку ключових елементів, що забезпечують повноцінну взаємодію користувача з програмою. Зокрема, передбачено поле введення ключових слів для пошуку, яке дозволяє задавати текстові критерії фільтрації. Також реалізовано параметри для вибору типів файлів, які підлягають обробці — підтримуються формати PDF, TXT та DOCX. Додатково користувач має змогу обмежити пошук за кількістю сторінок документа, що особливо актуально для PDF, а також за діапазоном дати створення файлів. Важливим елементом є прогрес-бар, який наочно демонструє хід виконання пошуку та забезпечує зворотний зв’язок у режимі реального часу. Результати пошуку виводяться у спеціальне вікно, де кожен знайдений файл представлено повним шляхом. Це вікно є інтерактивним — користувач може клікнути на рядок із результатом для відкриття відповідного документа.

Особливу увагу приділено локалізації інтерфейсу — підтримуються дві мови: українська та англійська. Це забезпечується завдяки внутрішньому словнику з перекладами та можливістю перемикання мови за допомогою відповідної кнопки. Такий підхід демонструє гнучкість інтерфейсу та сприяє зручності користувача у багатомовному середовищі.

На даному етапі також було важливо забезпечити коректну взаємодію між елементами інтерфейсу та логікою пошуку. Для цього було реалізовано механізм запуску пошуку в окремому потоці з використанням модуля threading. Такий підхід дає змогу уникнути «заморожування» графічного інтерфейсу під час тривалих операцій обробки великої кількості файлів.

Обробка подій, як-от натискання кнопки «Почати пошук», виконується через відповідні колбек-функції, які ініціюють окремий потік для виклику методу start_search. Крім цього, реалізовано кнопку «Зупинити», яка дозволяє користувачу вручну припинити процес пошуку, що особливо актуально у

випадках із великою кількістю документів або виявлення помилки в параметрах пошуку.

Для відображення динамічного прогресу пошуку був доданий прогрес-бар, значення якого оновлюється пропорційно кількості оброблених файлів. Це надає користувачеві візуальне уявлення про тривалість операції та покращує UX (user experience).

Інтеграція функцій не може вважатися завершеною без реалізації обробки потенційних помилок. Програма повинна залишатися стабільною навіть у разі недоступності файлу, неправильного формату дати, пошкодженого PDF або DOCX, або відсутності прав доступу до тек.

Для цього у всіх ключових місцях програми використовуються конструкції try-except, які дозволяють уникнути аварійного завершення програми та виводити користувачу повідомлення про суть помилки у зрозумілій формі через messagebox.

Завершення розробки та інтеграція компонентів стали важливим етапом, який об’єднав всі раніше реалізовані модулі в єдиний, цілісний і функціональний продукт. Вибрані підходи до структури, реалізації GUI, асинхронного виконання та обробки виключень дозволили створити стабільний застосунок, який повністю виконує поставлені задачі. Після завершення інтеграції система була готова до проведення повного тестування, яке розглядається у наступному підрозділі.

3.2 Методика перевірки працездатності

Після завершення етапу розробки програмного забезпечення особливу увагу було приділено процесу перевірки працездатності. Методично правильно організоване тестування дозволяє виявити недоліки як у логіці роботи програми, так і у реалізації окремих функцій. У межах цієї роботи основною метою тестування було підтвердити відповідність функціональності програмного продукту попередньо визначеним вимогам.

Було прийнято рішення реалізувати ручне (мануальне) тестування, оскільки застосунок має графічний інтерфейс користувача, який передбачає активну взаємодію через віконні елементи — поля вводу, чекбокси, кнопки тощо. Для перевірки логіки програми такого типу найбільш зручним і ефективним є саме ручне тестування за заздалегідь підготовленими сценаріями. Це також дозволяє виявити не лише технічні помилки, але й потенційні незручності в інтерфейсі або логіці взаємодії, які можуть залишитися поза увагою при автоматизованому підході.

Для забезпечення репрезентативності тестування була створена тестова директорія (рис. 3.1), яка містила файли у форматах PDF, DOCX і TXT з різною кількістю сторінок (від 1 до 150), з різними датами створення (від 2023 до 2025 року), а також із текстами, що включали як наявні, так і відсутні ключові слова. До складу тестових даних також увійшли спеціально створені проблемні файли: порожні, зашифровані.

Имя	Дата изменения	Тип	Размер
150_Storinok_pdf_test	17.05.2025 20:12	Документ Adobe Acr...	254 КБ
2023-06-03_txt_test	17.05.2025 20:17	Текстовый документ	1 КБ
2025-05-05_docs_test	05.05.2025 20:04	Документ Office Op...	13 КБ
2025-05-05_pdf_test	05.05.2025 19:59	Документ Adobe Acr...	83 КБ
2025-05-05_txt_test	05.05.2025 19:55	Текстовый документ	1 КБ
Certificate	01.06.2023 13:05	Документ Adobe Acr...	181 КБ
Coursera VMYFUGM2GZNZ	04.06.2024 12:53	Документ Adobe Acr...	373 КБ
Б21_д_122 _4 курс (2)	22.01.2025 11:41	Документ Adobe Acr...	370 КБ
МЕТОДИЧНІ_РЕКОМЕНДАЦІЇ_ДО_ВИКОНАННЯ...	17.05.2025 20:48	Документ Office Op...	274 КБ
Сертифікат	25.11.2024 11:23	Документ Adobe Acr...	664 КБ
Сертифікат_Кісельов	26.10.2024 17:33	Документ Adobe Acr...	553 КБ
Zashifrovaniy	17.05.2025 20:58	Документ Adobe Acr...	79 КБ

Рисунок 3.1 – Тестова директорія

Тестування здійснювалося на ноутбуці середнього класу з операційною системою Windows 10, 8 ГБ оперативної пам’яті, процесором Intel i5, встановленим Python 3.10 та необхідними бібліотеками (PyMuPDF, python-docx, tkinter).

Сценарії тестування охоплювали як звичайні, так і граничні випадки. Перевірявся пошук без жодних фільтрів, лише за ключовими словами, що

дозволяло переконатися у правильному розпізнаванні тексту у PDF, DOCX і TXT файлах. Окремо перевірялась робота фільтрації за типом файлу — користувач вибирав, наприклад, лише PDF, і очікувалося, що решта форматів буде проігнорована. Іншим сценарієм було використання діапазону дат: при вказанні межі, наприклад, від 2023-01-01 до 2024-12-31, програма мала ігнорувати всі файли, створені поза межами зазначеного періоду. Фільтрація за кількістю сторінок застосовувалася лише до PDF-файлів, де ця характеристика визначається точно. У випадку DOCX і TXT очікувалося попередження про неможливість використання фільтру за сторінками. Тестувалися також помилкові та неповні введення, зокрема неправильний формат дати або порожнє поле ключових слів — у таких випадках мала з’являтися відповідна підказка або повідомлення. Перевірявся механізм переривання пошуку: при запуску обробки великої директорії (~3000 файлів), після натискання кнопки «Зупинити», програма мала коректно зупинити процес. Окремо тестувалась функція відкриття файлів: з отриманого списку обирався будь-який документ, і перевірялося, чи відкривається він стандартною програмою операційної системи. Крім того, було протестовано перемикання мови інтерфейсу — очікувалося, що всі підписи, кнопки та попередження будуть правильно локалізовані.

Кожен тестовий сценарій мав заздалегідь визначені очікувані результати. Наприклад, якщо ключове слово мало зустрічатися у трьох документах, саме вони повинні були з’явитися у результатах пошуку. Якщо було обрано фільтрацію лише по PDF, документи у форматах DOCX і TXT не мали відображатися. При спробі використати фільтрацію за кількістю сторінок у TXT-файлі система повинна була вивести відповідне попередження. У випадку переривання пошуку прогрес-бар мав зупинитися, а кнопка «Зупинити» змінити стан на неактивний.

У всіх описаних випадках, поведінка програми відповідала очікуванням. Методика перевірки працездатності базувалася на практичному тестуванні із симуляцією типових сценаріїв використання. Такий підхід

дозволив глибоко проаналізувати реакцію системи на різні вхідні умови та переконатися у її надійності, стабільності та зручності в роботі.

3.3 Результати перевірки

Після завершення тестування програмного забезпечення було зібрано та систематизовано результати, які відображають фактичну працездатність і якість реалізованого функціоналу. Тестування проводилось згідно з методикою, описаною у попередньому підрозділі, і дозволило оцінити не лише логічну коректність роботи застосунку, а й його стабільність, швидкодію та поведінку в умовах підвищеного навантаження.

У більшості протестованих сценаріїв програма демонструвала поведінку, що відповідала очікуванням.

Пошук за ключовими словами у PDF, DOCX та TXT-файлах відбувався стабільно: система коректно знаходила всі документи, що містили задані слова, без хибних пропусків або зайвих результатів (рис. 3.2).

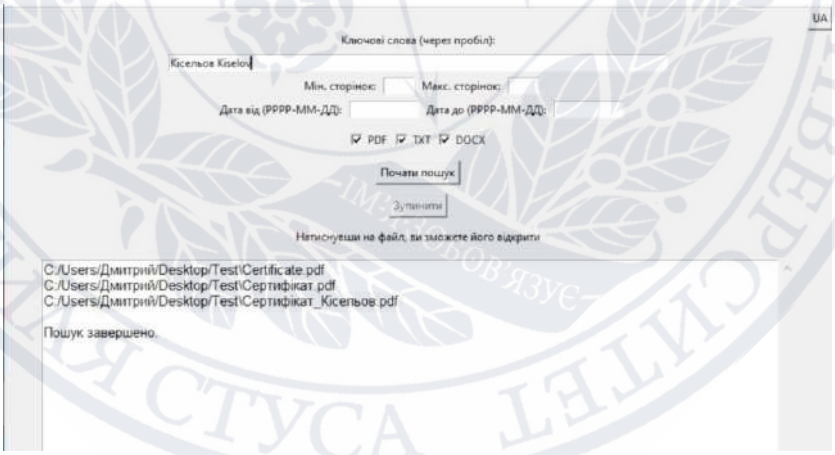

Рисунок 3.2 – Здійснення пошуку документів за двома ключовими словами (Кісельов та Kiselov)

Фільтрація за форматом працювала точно — у разі вибору лише PDF, у списку результатів не з’являлися файли інших типів (рис. 3.3).

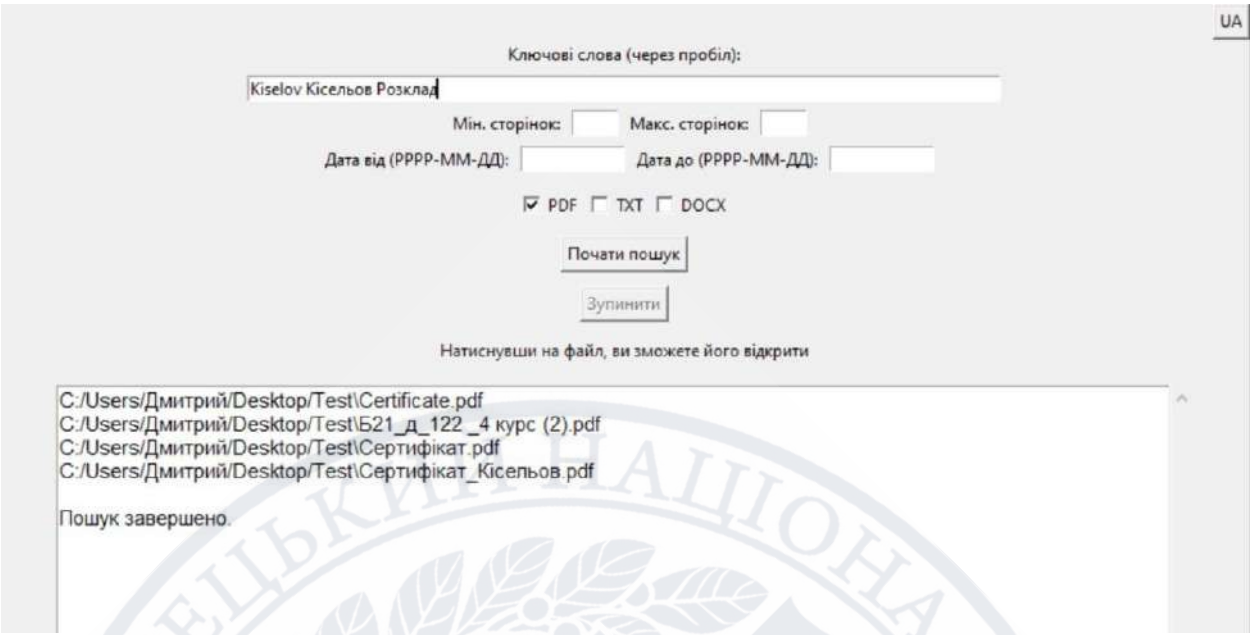

Рисунок 3.3 – Здійснення пошуку документів за трьома ключовими словами (Kiselov, Кісельов, Розклад), з фільтрацією за форматом (лише PDF файли)

Аналогічно, при встановленні діапазону дат у результатах не було жодного файлу, створеного за межами зазначеного періоду, що свідчить про правильну реалізацію механізму порівняння об'єктів типу datetime (рис. 3.4).

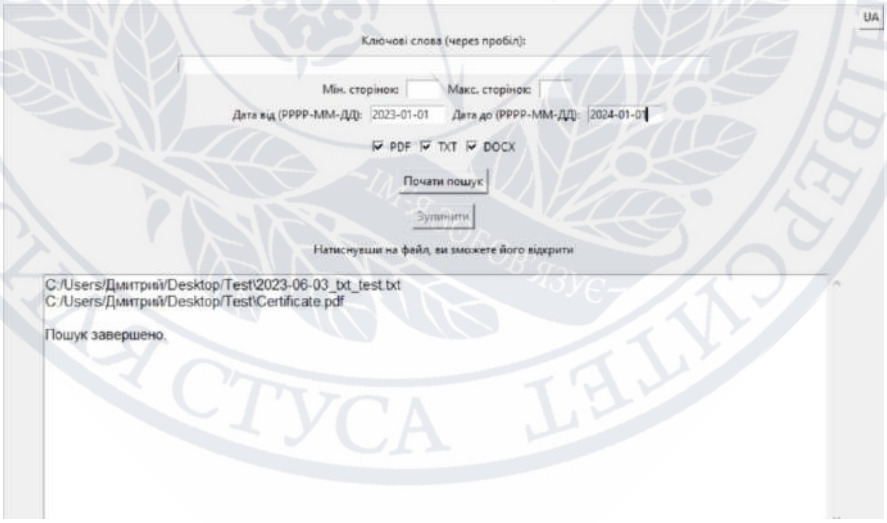

Рисунок 3.4 – Здійснення пошуку документів з датою створення в діапазоні від 01.01.2023 до 01.01.2024 року

Визначення кількості сторінок у PDF-файлах також відповідало заданим умовам: документи, які не відповідали вказаним обмеженням, не включалися до результатів (рис. 3.5).

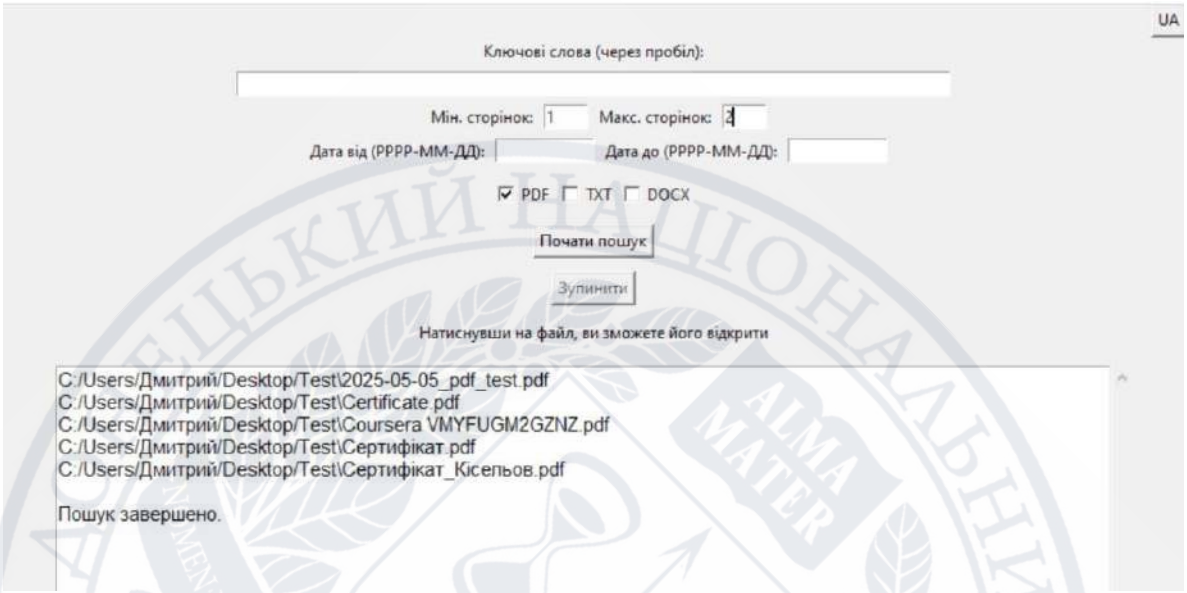

Рисунок 3.5 – Здійснення пошуку документів за кількістю сторінок (від 1 до 2)

У випадках, коли користувач помилково вводив некоректні параметри (наприклад, неправильний формат дати (рис. 3.6) або залишав поля порожніми (рис. 3.7)), програма видавала відповідне попередження через вікна типу messagebox, тим самим запобігаючи несподіваному завершенню або некоректній поведінці.

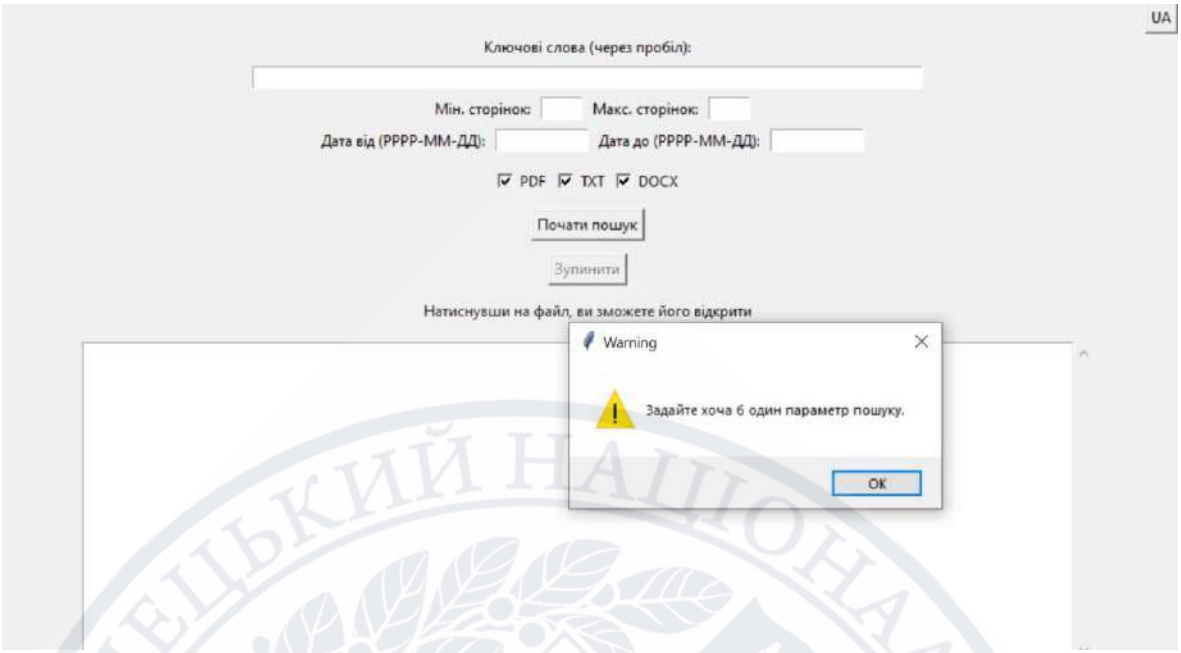

Рисунок 3.6 – Помилка при не введенні жодного параметра для пошуку

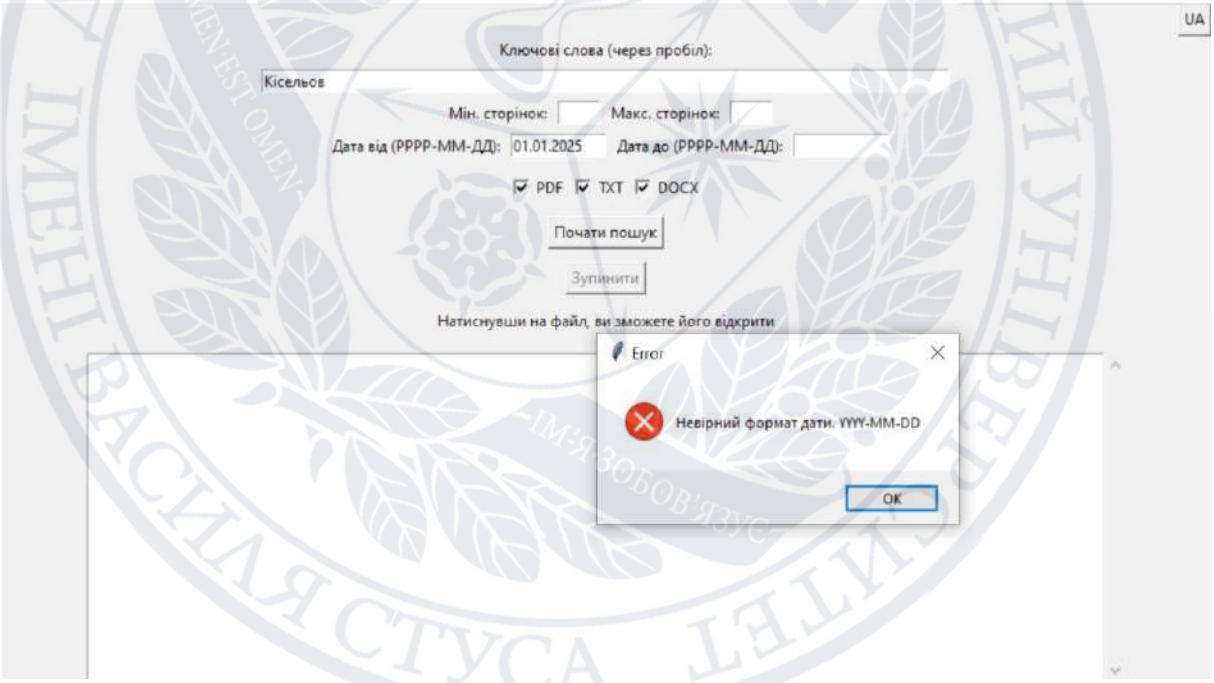

Рисунок 3.7 – Помилка при введенні дати в невірному форматі

Механізм зупинки пошуку виконувався коректно: при натисканні кнопки «Зупинити» обхід файлів миттєво припинявся, інтерфейс повертався у початковий стан, а відповідні елементи змінювали свою активність згідно з логікою.

Функція відкриття файлу також працювала надійно — при натисканні на шлях у результатах відкривався відповідний документ у стандартному додатку системи, наприклад PDF відкривався в браузері або Adobe Reader (рис. 3.8).

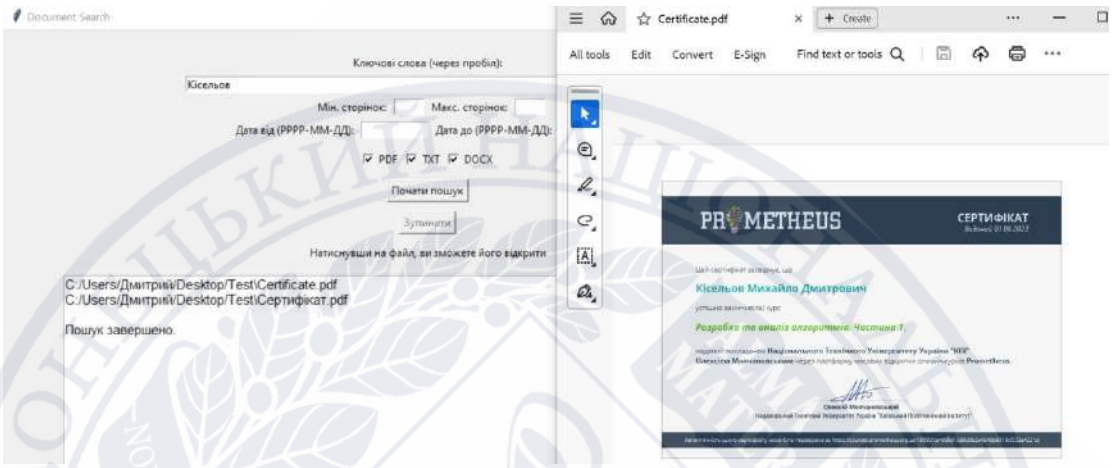

Рисунок 3.8 – Відкриття файлу одним натисканням на нього

Перемикання мови інтерфейсу відбувалося динамічно: усі підписи, мітки, кнопки та текстові підказки адаптувалися до вибраної мови без потреби перезапуску програми (рис. 3.9).

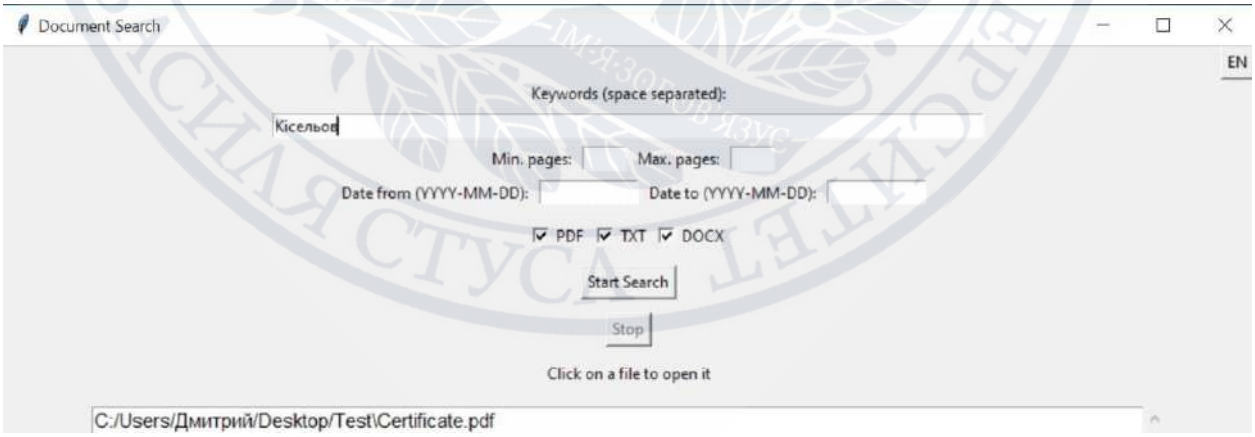

Рисунок 3.9 – Перемикання мови інтерфейсу за допомогою кнопки UA/EN

Разом з тим, у ході перевірки були зафіксовані деякі особливості, які не є критичними, але можуть бути покращені у майбутніх версіях. Наприклад,

при скануванні великої кількості файлів (понад 5000) спостерігалося помітне зниження швидкодії, зокрема в оновленні графічного інтерфейсу, що пов’язано з частими викликами оновлення ScrolledText та Progressbar, які навантажують GUI-потік. Також були виявлені проблеми з відображенням тексту в деяких TXT-файлах із нестандартним кодуванням (cp1251, utf-16) — попри використання errors='ignore', бібліотека іноді повертала некоректні символи. Ще однією особливістю стало те, що за наявності захищених паролем PDF-файлів програма їх пропускала без повідомлення, що можна покращити, реалізувавши інформування користувача про такі випадки. Крім того, після перемикання мови деякі елементи системних діалогів, залишалися англомовними. Це пояснюється обмеженнями самої бібліотеки tkinter, яка не підтримує повної локалізації системних компонентів.

Загалом, результати перевірки підтверджують високу якість реалізації програмного забезпечення. Усі ключові функції працюють відповідно до очікувань, а виявлені незначні обмеження не мають критичного впливу на загальну працездатність або зручність використання застосунку. Таким чином, програмний продукт може вважатися готовим до використання в умовах реального середовища для задач параметризованого пошуку документів різних форматів.

3.4 Аналіз ефективності

Ефективність програмного забезпечення є критично важливою характеристикою, яка визначає не лише його технічну якість, а й практичну придатність для кінцевого користувача. Особливо це актуально у випадках, коли мова йде про роботу з великими обсягами даних або взаємодію з файловою системою — як це має місце у розробленому програмному забезпеченні для параметризованого пошуку документів.

Оцінка ефективності даної системи проводилась за сукупністю кількох ключових аспектів: швидкодія, стабільність при навантаженні, чутливість до

масштабів даних, реакція графічного інтерфейсу на інтенсивні процеси, а також відсутність системних збоїв або втрат даних.

Одним із перших аспектів, який було проаналізовано, є загальна швидкість виконання основної операції — пошуку документів відповідно до вказаних користувачем параметрів. У типовому сценарії, коли кількість об'єктів для аналізу не перевищує кількох сотень, система демонструє надзвичайно оперативну реакцію. Це досягається завдяки використанню ефективного обходу файлової системи через os.walk, а також за рахунок легких засобів обробки тексту.

Однак із ростом кількості файлів до кількох тисяч зростає не лише обсяг обчислювальних операцій, а й кількість перевірок, які необхідно виконати над кожним файлом: фільтрація за типом, дата створення, вміст, кількість сторінок (для PDF), орієнтація на ключові слова. Кожна з цих перевірок додає обчислювального навантаження, і це прямо впливає на швидкість обробки.

Попри це, розроблена програма демонструє задовільну продуктивність і зберігає здатність обробляти великі обсяги даних без критичних затримок. Водночас, певне сповільнення спостерігається не на рівні самого алгоритму пошуку, а у частині, яка стосується оновлення графічного інтерфейсу, зокрема виводу результатів у вікно ScrolledText. Часте оновлення вмісту цього елемента, особливо після обробки кожного окремого файла, може спричинити затримки в роботі GUI, хоча сам пошук при цьому продовжується у фоновому потоці без порушень логіки.

Окрему увагу слід приділити поведінці програми при зростанні обсягів даних. В рамках тестування перевірялися різні сценарії, починаючи з невеликої кількості файлів (100–500) і до великих масивів (3000–5000 документів). Зі збільшенням кількості об'єктів обробки очікувано зростає час виконання пошуку, однак програма зберігає свою працездатність. Вона не зависає, не виводить помилок, не завершуються аварійно. Це підтверджує наявність належної реалізації багатопотоковості: пошук виконується в

окремому потоці, завдяки чому головний потік, відповідальний за інтерфейс, зберігає контроль над візуальним відображенням прогресу.

Це рішення — запуск пошукової логіки в окремому потоці за допомогою модуля threading — було прийнято свідомо, з метою уникнення блокування основного вікна під час інтенсивних операцій. Простіше кажучи, користувач бачить, що система «живе» і працює, навіть якщо в реальному часі обробляється велика кількість файлів. Це забезпечує позитивне користувацьке враження і створює відчуття контролю над процесом.

Інтерфейс користувача — ще один аспект, який має значення для оцінки ефективності. Розроблена програма використовує tkinter — досить простий, проте надійний фреймворк для створення GUI у Python. В процесі тестування виявлено, що при великих обсягах даних (понад 1000 файлів) деякі елементи інтерфейсу, зокрема прогрес-бар та вікно виводу результатів, починають реагувати з деякою затримкою. Це не є критичним, але вказує на потенційні можливості для оптимізації — наприклад, можна змінити логіку оновлення текстового поля так, щоб нові результати виводились не після кожного файлу, а після певної кількості (наприклад, кожні 10 або 50).

Окрім цього, було протестовано кнопку «Зупинити». Вона дозволяє у будь-який момент перервати процес пошуку, що особливо актуально у випадках, коли параметри були задані неправильно або користувач випадково обрав велику директорію. Реалізація механізму зупинки через логічний прапорець (stop_search_flag) виявилася ефективною і дозволяє не лише перервати пошук без втрати стабільності, а й коректно оновити стан інтерфейсу після припинення обробки.

Надійність програми — ще один параметр, який входить до поняття ефективності. Під час тестування жодного разу не спостерігалося аварійного завершення програми, навіть при навмисному створенні помилкових ситуацій — наприклад, спроба обробити пошкоджений PDF або відкрити файл, до якого відсутній доступ. Всі такі випадки були обгорнуті в

конструкції try-except, що дозволяє вивести помилку в консоль (або ж у майбутньому — у лог-файл), не припиняючи роботу всього застосунку.

Додатково варто відзначити стабільність взаємодії з файловою системою — під час виконання багатьох десятків пошуків поспіль не було зафіксовано жодного конфлікту, зависання або порушення прав доступу. Це свідчить про правильну реалізацію як логіки обробки директорій, так і перевірки типів файлів.

Таким чином, розроблений застосунок демонструє загалом високий рівень ефективності. Він здатний працювати з тисячами документів різних форматів, підтримує багатопоточну логіку для забезпечення стабільної роботи інтерфейсу, реагує на дії користувача у реальному часі, дозволяє переривати обробку на будь-якому етапі, і стійкий до неочікуваних ситуацій. Попри окремі незначні затримки при дуже великих навантаженнях, загальна оцінка продуктивності є позитивною. Програма цілком відповідає очікуванням щодо швидкості, стабільності та гнучкості в роботі з різноманітними наборами даних.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було здійснено комплексне дослідження підходів до організації ефективного доступу до інформації у файлових системах. Поставленою метою було створення зручного, інтуїтивно зрозумілого та функціонального настільного застосунку, здатного виконувати фільтрацію та пошук документів за вмістом, типом файлу, датою створення та кількістю сторінок.

Результатом роботи стало програмне рішення з графічним інтерфейсом, яке забезпечує автоматизований аналіз великої кількості документів і дозволяє користувачам швидко знаходити потрібні файли відповідно до заданих параметрів. Програма підтримує декілька поширених форматів (PDF, DOCX, TXT), має функцію динамічного оновлення прогресу пошуку, можливість зупинення процесу на вимогу користувача, а також підтримує багатомовність.

У ході розробки були застосовані сучасні програмні засоби та технології, що забезпечило високу стабільність, продуктивність і надійність системи. Проведене тестування підтвердило правильність реалованого функціоналу, стійкість до помилок користувача і придатність програми до реального використання.

Таким чином, результати бакалаврської роботи демонструють успішне досягнення поставленої мети. Розроблений застосунок є ефективним інструментом для швидкого пошуку документів у локальному середовищі та може бути використаний як основа для подальшого розвитку систем автоматизованого керування цифровими архівами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhang, A., Lipton, Z. C., Li, M., Smola, A. J. Cambridge Dive into Deep Learning, 2023. 548 с.

2. Motwani, B. Data Analytics using Python. New Delhi, 2020. 760 с.

3. Sarkar, D. Text Analytics with Python. Berkeley, 2019. 674 с.

4. Rocchi W. Cybersecurity and Privacy Law Handbook. Birmingham, 2022. 230 с.

5. Blokdyk G. Document Security. Emereo Publishing, 2020. 303 с.

6. Saurav S. Python Apps on Visual Studio Code. BPB Publications, 2024. 200 с.

7. Speight A. Visual Studio Code for Python Programmers. Wiley, 2021. 256 с.

8. Del Sole, A. Visual Studio Code Distilled. Berkeley, 2021. 276 с.

9. Forbes E. Learning Concurrency in Python. Birmingham, 2017. 350 с.

10. Nguyen Q. Mastering Concurrency in Python. Birmingham, 2018. 446 с.

11. Nelli F. Parallel and High Performance Programming with Python. New Delhi, 2023. 392 с.

12. Fowler M. Python Concurrency with asyncio. New York, 2022. 376 с.

13. Dey S. Python Image Processing Cookbook. Birmingham, 2020. 438 с.

14. Willman J. Beginning PyQt. Berkeley, 2020. 310 с.

15. Fitzpatrick, M. Create GUI Applications with Python and Qt5. London, 2022. 997 с.

16. Moore A. D. Python GUI Programming with Tkinter. Birmingham, 2021. 452 с.

17. Moore, A. D., Harwani, B. M. Python GUI Programming. Birmingham, 2019. 746 с.

18. Roseman M. Modern Tkinter for Busy Python Developers. Independently published, 2020. 272 с.

19. Bahadure N. B. Building Modern GUIs with tkinter and Python. Birmingham, 2023. 484 с.

20. Fadheli A. Practical Python PDF Processing. Kindle Edition, 2024. 165 с.

21. Driscoll M. PDF Processing with Python. Independently published, 2018. 428 с.

22. Whitington J. PDF Explained. O'Reilly Media, 2011. 144 с.

23. Canning J. Data Structures and Algorithms in Python. Hoboken, 2022. 928 с.

24. Scratch M. Python Algorithms. Independently published, 2020. 126 с.

25. Agarwal B. Hands-On Data Structures and Algorithms with Python. Birmingham, 2022. 496 с.

26. Abella H. 300+ Python Algorithms. Independently published, 2024. 344 с.

27. Hetland, M. L. Beginning Python. Berkeley, 2024. 607 с.

28. Somnath P. Advanced Python Programming. Birmingham, 2022. 606 с.

29. Crickard P. Data Engineering with Python. Birmingham, 2020. 356 с.

30. Clark W. E. Python Exception Handling Made Easy. London, 2025. 257 с.

Додаток А

Код додатку

```
import os
import tkinter as tk
from tkinter import filedialog, messagebox, scrolledtext
import re
import fitz
import docx
from datetime import datetime
import threading
from tkinter import ttk
import time


class DocumentSearcher:
    def __init__(self, root):
        self.root = root
        self.root.title("Document Search")
        self.root.geometry("750x750")


        self.lang = "uk"
        self.labels = self.get_labels()


        self.stop_search_flag = False


        self.lang_btn = tk.Button(root, text="UA",
command=self.switch_language)
        self.lang_btn.pack(anchor='ne')


        self.query_entry_label = tk.Label(root,
text=self.labels["keywords"])
```

```
        self.query_entry_label.pack()
        self.query_entry = tk.Entry(root, width=90)
        self.query_entry.pack(pady=5)

        frame1 = tk.Frame(root)
        frame1.pack()

        self.min_pages_lbl = tk.Label(frame1,
text=self.labels["min_pages"])
        self.min_pages_lbl.grid(row=0, column=0, padx=5)
        self.min_pages = tk.Entry(frame1, width=5)
        self.min_pages.grid(row=0, column=1)

        self.max_pages_lbl = tk.Label(frame1,
text=self.labels["max_pages"])
        self.max_pages_lbl.grid(row=0, column=2, padx=5)
        self.max_pages = tk.Entry(frame1, width=5)
        self.max_pages.grid(row=0, column=3)

        frame2 = tk.Frame(root)
        frame2.pack(pady=5)

        self.date_from_lbl = tk.Label(frame2,
text=self.labels["date_from"])
        self.date_from_lbl.grid(row=0, column=0, padx=5)
        self.date_from = tk.Entry(frame2, width=12)
        self.date_from.grid(row=0, column=1)

        self.date_to_lbl = tk.Label(frame2,
text=self.labels["date_to"])
        self.date_to_lbl.grid(row=0, column=2, padx=5)
        self.date_to = tk.Entry(frame2, width=12)
```

```
        self.date_to.grid(row=0, column=3)


        frame3 = tk.Frame(root)

        frame3.pack(pady=5)


        self.pdf_var = tk.BooleanVar(value=True)

        self.pdf_check = tk.Checkbutton(frame3, text="PDF",
variable=self.pdf_var)

        self.pdf_check.grid(row=0, column=0)


        self.txt_var = tk.BooleanVar(value=True)

        self.txt_check = tk.Checkbutton(frame3, text="TXT",
variable=self.txt_var)

        self.txt_check.grid(row=0, column=1)


        self.docx_var = tk.BooleanVar(value=True)

        self.docx_check = tk.Checkbutton(frame3, text="DOCX",
variable=self.docx_var)

        self.docx_check.grid(row=0, column=2)


        self.search_button = tk.Button(root,
text=self.labels["start"], command=self.run_search_thread)

        self.search_button.pack(pady=5)


        self.stop_button = tk.Button(root,
text=self.labels["stop"], command=self.stop_search,
state="disabled")

        self.stop_button.pack(pady=5)


        self.progress_label = tk.Label(root, text="0%")

        self.progress_label.pack()

        self.progress = ttk.Progressbar(root, length=600,
mode='determinate')

        self.progress.pack(pady=5)
```

```
        self.progress_label.pack_forget()
        self.progress.pack_forget()


        self.open_file_label = tk.Label(root,
text=self.labels["click_to_open"], fg="black")
        self.open_file_label.pack(pady=5)


        self.result_box = scrolledtext.ScrolledText(root,
width=100, height=15, fg="black", bg="white", font=("Arial",
11))
        self.result_box.pack(pady=10)
        self.result_box.bind("<Button-1>",
self.open_selected_file)


    def switch_language(self):
        self.lang = "en" if self.lang == "uk" else "uk"
        self.labels = self.get_labels()
        self.update_labels()


    def update_labels(self):
        self.root.title(self.labels["title"])

self.query_entry_label.config(text=self.labels["keywords"])
        self.search_button.config(text=self.labels["start"])
        self.stop_button.config(text=self.labels["stop"])
        self.min_pages_lbl.config(text=self.labels["min_pages"])
        self.max_pages_lbl.config(text=self.labels["max_pages"])
        self.date_from_lbl.config(text=self.labels["date_from"])
        self.date_to_lbl.config(text=self.labels["date_to"])

self.open_file_label.config(text=self.labels["click_to_open"])
        self.lang_btn.config(text="UA" if self.lang == "uk" else
"EN")
```

```
    def get_labels(self):
        if self.lang == "uk":
            return {
                "title": "Пошук документів",
                "keywords": "Ключові слова (через пробіл):",
                "min_pages": "Мін. сторінок:",
                "max_pages": "Макс. сторінок:",
                "date_from": "Дата від (РРРР-ММ-ДД):",
                "date_to": "Дата до (РРРР-ММ-ДД):",
                "start": "Почати пошук",
                "stop": "Зупинити",
                "click_to_open": "Натиснувши на файл, ви зможете
його відкрити"
            }
        else:
            return {
                "title": "Document Search",
                "keywords": "Keywords (space separated):",
                "min_pages": "Min. pages:",
                "max_pages": "Max. pages:",
                "date_from": "Date from (YYYY-MM-DD):",
                "date_to": "Date to (YYYY-MM-DD):",
                "start": "Start Search",
                "stop": "Stop",
                "click_to_open": "Click on a file to open it"
            }

    def run_search_thread(self):
        thread = threading.Thread(target=self.start_search)
        thread.start()
```

```
    def start_search(self):
        self.progress_label.pack()
        self.progress.pack()

        keywords = self.query_entry.get().strip().split()
        min_pages = self.min_pages.get().strip()
        max_pages = self.max_pages.get().strip()
        date_from = self.date_from.get().strip()
        date_to = self.date_to.get().strip()

        min_pages = int(min_pages) if min_pages.isdigit() else
None
        max_pages = int(max_pages) if max_pages.isdigit() else
None

        try:
            date_from = datetime.strptime(date_from, "%Y-%m-%d")
if date_from else None
            date_to = datetime.strptime(date_to, "%Y-%m-%d") if
date_to else None
        except ValueError:
            messagebox.showerror("Error", "Невірний формат дати.
YYYY-MM-DD")
            return

        if not any([keywords, min_pages, max_pages, date_from,
date_to]):
            messagebox.showwarning("Warning", "Задайте хоча б
один параметр пошуку.")
            return

        file_types = []
        if self.pdf_var.get():
            file_types.append('.pdf')
```

```
        if self.txt_var.get():
            file_types.append('.txt')
        if self.docx_var.get():
            file_types.append('.docx')

        if not file_types:
            file_types = ['.pdf', '.txt', '.docx']

        # Перевірка неможливості визначення точної кількості сторінок у txt/docx
        if (self.txt_var.get() or self.docx_var.get()) and (min_pages is not None or max_pages is not None):
            messagebox.showerror("Помилка", "Неможливо точно визначити кількість сторінок в Word або txt-файлі.")
            self.progress.pack_forget()
            self.progress_label.pack_forget()
            return

        self.result_box.delete(1.0, tk.END)
        self.stop_search_flag = False
        self.progress_label.config(text="0%")
        self.progress['value'] = 0

        root_dir = filedialog.askdirectory(title="Оберіть теку для пошуку")
        if not root_dir:
            self.progress.pack_forget()
            self.progress_label.pack_forget()
            return

        all_files = [os.path.join(dp, f) for dp, dn, filenames in os.walk(root_dir) for f in filenames]
```

```
        all_files = [f for f in all_files if
f.lower().endswith(tuple(file_types))]
        total_files = len(all_files)

        if total_files == 0:
            self.progress_label.config(text="0%")
            self.progress.pack_forget()
            self.progress_label.pack_forget()
            return

        self.stop_button.config(state="normal")
        matched_count = 0
        start_time = time.time()

        for idx, filepath in enumerate(all_files):
            if self.stop_search_flag:
                break
            filename = os.path.basename(filepath)
            try:
                created_time =
datetime.fromtimestamp(os.path.getctime(filepath))
                if date_from and created_time < date_from:
                    continue
                if date_to and created_time > date_to:
                    continue

                content = ""
                page_count = None

                if filename.endswith(".txt"):
                    with open(filepath, 'r', encoding='utf-8',
errors='ignore') as f:
```

```
                    content = f.read().lower()

            elif filename.endswith(".pdf"):
                with fitz.open(filepath) as doc:
                    page_count = len(doc)
                    if (min_pages is not None and page_count
< min_pages) or \
                            (max_pages is not None and page_count
> max_pages):
                        continue
                    content = "".join(page.get_text() for
page in doc).lower()

            elif filename.endswith(".docx"):
                content =
self.extract_docx_text(filepath).lower()
                word_count = len(content.split())
                approx_pages = word_count // 500 + 1
                if (min_pages is not None and approx_pages <
min_pages) or \
                    (max_pages is not None and approx_pages >
max_pages):
                    continue
            else:
                continue

            if keywords and not any(re.search(r'\b' +
re.escape(word.lower()) + r'\b', content) for word in keywords):
                continue

            self.result_box.insert(tk.END, filepath + "\n")
            self.result_box.yview(tk.END)
            matched_count += 1
```

```
            except Exception as e:
                print(f"Помилка зчитування файлу {filepath}:
{e}")

            progress_percent = round((idx + 1) / total_files *
100, 2)
            self.progress['value'] = progress_percent
self.progress_label.config(text=f"{progress_percent:.2f}%")

            self.root.update_idletasks()

        if not self.stop_search_flag:
            self.progress['value'] = 100
            self.progress_label.config(text="100.00%")

        if matched_count == 0:
            self.result_box.insert(tk.END, "Файли не знайдено."
if self.lang == "uk" else "No files found.")

        self.result_box.insert(tk.END, "\nПошук завершено.\n" if
self.lang == "uk" else "\nSearch completed.\n")

        self.stop_button.config(state="disabled")
        self.progress.pack_forget()
        self.progress_label.pack_forget()

    def stop_search(self):
        self.stop_search_flag = True

    def extract_docx_text(self, path):
        text = ""
        try:
```

```
            doc = docx.Document(path)
            for para in doc.paragraphs:
                text += para.text + "\n"
        except Exception as e:
            print(f"Помилка читання DOCX {path}: {e}")
        return text


    def open_selected_file(self, event):
        try:
            index =
self.result_box.index(f"@{event.x},{event.y}")
            selected_file = self.result_box.get(f"{index}
linestart", f"{index} lineend").strip()
            if os.path.isfile(selected_file):
                os.startfile(selected_file)
        except Exception as e:
            print(f"Помилка відкриття файлу: {e}")


if __name__ == "__main__":
    root = tk.Tk()
    app = DocumentSearcher(root)
    root.mainloop()
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____
_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)