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

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»**

**спеціальності 121 Інженерія програмного забезпечення**

**на тему: «Програмне забезпечення для автоматизації реєстрації паперових документів підприємства»**

Виконав:
студент IV курсу, групи КП-11
Защик Іван Олександрович _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Новак Дмитро Сергійович _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри інформаційних технологій, штучного інтелекту та кібербезпеки Національного університету харчових технологій, к.т.н., доцент
Мошенський Андрій Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет прикладної математики**
**Кафедра програмного забезпечення комп'ютерних систем**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

# ЗАВДАННЯ

**на дипломний проєкт студенту**

Защику Івану Олександровичу

1. Тема проєкту «Програмне забезпечення для автоматизації реєстрації паперових документів підприємства», керівник проєкту Новак Дмитро Сергійович, к.т.н., доцент, затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз предметної області та існуючих рішень;
   - аналіз мов програмування та технологій розроблення клієнтської та серверної частини розроблюваної системи;
   - аналіз архітектури та дизайну розробленої системи;
   - розроблення вказаного програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   - проєктування дизайну користувацького інтерфейсу (креслення);
   - схема бази даних (креслення);
   - вкладка із записами документів системи DocRegister (плакат);
   - вкладка із контрагентами системи DocRegister (плакат);

6. Консультанти розділів проєкту

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Онай М.В., доцент | | |

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою проєкту | 13.11.2024 | |
| 2. | Розроблення та узгодження технічного завдання | 22.11.2024 | |
| 3. | Розроблення структури ПЗ | 15.12.2024 | |
| 4. | Підготовка матеріалів першого розділу дипломного проєкту | 28.12.2024 | |
| 5. | Розроблення дизайну сторінок та графічних елементів | 01.02.2025 | |
| 6. | Підготовка матеріалів другого розділу дипломного проєкту | 19.02.2025 | |
| 7. | Програмна реалізація системи | 11.03.2025 | |
| 8. | Тестування ПЗ | 18.03.2025 | |
| 9. | Підготовка матеріалів третього розділу дипломного проєкту | 27.03.2025 | |
| 10. | Підготовка матеріалів четвертого розділу дипломного проєкту | 15.04.2025 | |
| 11. | Підготовка графічної частини дипломного проєкту | 22.04.2025 | |
| 12. | Оформлення документації дипломного проєкту | 27.05.2025 | |

Студент Іван ЗАЩИК

Керівник проєкту Дмитро НОВАК

# АНОТАЦІЯ

У сучасному діловому середовищі, де обсяг документації постійно зростає, ефективна обробка документів є критично важливою. Саме з цією метою був розроблений програмний комплекс для автоматизованого розпізнавання, аналізу та керування документацією. Рішення складається з двох частин – серверної, яка реалізована за допомогою Python та бібліотеки Tesseract OCR, та клієнтської, створеної у вигляді Windows-додатка.

Серверна частина приймає PDF-файли або зображення, виконує оптичне розпізнавання тексту (OCR) та повертає розпізнаний вміст. На стороні клієнта здійснюється подальший аналіз цього тексту із застосуванням регулярних виразів для автоматичного виявлення ключових реквізитів, таких як номер документа, дата підписання, контрагент тощо.

Користувачі мають змогу зручно працювати з базою записів: переглядати, фільтрувати, редагувати, завантажувати або видаляти документи. Передбачено також потужний функціонал для формування звітів: Excel-файл генерується на основі поточного набору записів (з урахуванням усіх фільтрів), а також формується ZIP-архів зі сканами, впорядкованими за датою та контрагентом.

Інтерфейс програми є інтуїтивно зрозумілим і адаптованим під щоденне використання, що робить систему корисною як для офісних працівників, так і для спеціалістів, відповідальних за документообіг. Завдяки поєднанню технологій OCR, автоматичного аналізу тексту та зручного експорту звітності, розроблений інструмент істотно підвищує ефективність обробки документів.

## ABSTRACT

In today's business environment, where the volume of documentation is constantly increasing, efficient document processing becomes critically important. For this purpose, a software system was developed to automate the recognition, analysis, and management of documents. The solution consists of a server component, implemented using Python and the Tesseract OCR library, and a client component in the form of a Windows application.

The server processes PDF files or images, performs optical character recognition (OCR), and returns the recognized text. On the client side, this text is analyzed using regular expressions to automatically extract key details such as document number, signing date, and counterparty name.

Users can conveniently interact with the records database: viewing, filtering, editing, downloading, or deleting documents. The system also provides advanced reporting functionality: an Excel file is generated based on the currently filtered dataset, and a ZIP archive is created with scans organized by date and counterparty.

The application's interface is intuitive and suitable for daily use, making the system valuable for office staff and documentation specialists alike. By combining OCR technology, automated text analysis, and convenient report export, the developed tool significantly improves the efficiency of document processing.

ДП.045490-01-90 Програмне забезпечення для автоматизації реєстрації паперових документів підприємства. Відомість проєкту

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | Документація проєкту | | |
| | | | |
| ДП.045490-02-91 | Програмне забезпечення | 5 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Технічне завдання | | |
| | | | |
| ДП.045490-03-81 | Програмне забезпечення | 70 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Пояснювальна записка | | |
| | | | |
| ДП.045490-04-51 | Програмне забезпечення | 4 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Програма та методика | | |
| | тестування | | |
| | | | |
| ДП.045490-05-34 | Програмне забезпечення | 10 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Керівництво користувача | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | | | |
| ДП.045490-06-99 | Програмне забезпечення | 1 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Структура клієнтського | | |
| | додатка. Структурна | | |
| | схема. | | |
| | | | |
| ДП.045490-07-99 | Програмне забезпечення | 1 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Схема бази даних. | | |
| | ERD-діаграма | | |
| | | | |
| ДП.045490-08-98 | Програмне забезпечення | 1 | |
| | для автоматизації | | |
| | реєстрації паперових | | |
| | документів підприємства. | | |
| | Компакт-диск | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

# ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕЄСТРАЦІЇ ПАПЕРОВИХ ДОКУМЕНТІВ ПІДПРИЄМСТВА

**Технічне завдання**

ДП.045490-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ЗАЩИК

2024

**ЗМІСТ**

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

**Назва розробки**: програмне забезпечення для автоматизації реєстрації паперових документів підприємства.

**Галузь застосування**: інформаційні технології.

## 2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

## 3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання як інформаційне забезпечення документообігу підприємства з метою автоматизації процесів реєстрації, обробки, зберігання та моніторингу паперових документів. Система DocRegister надає зручний інтерфейс для співробітників, які працюють з документацією, та забезпечує функції фільтрації, пошуку, формування звітів, розпізнавання вмісту сканованих документів, а також організацію взаємодії між працівниками під час опрацювання документів.

## 4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Розроблювана система повинна забезпечувати такі основні функції:

1) можливість реєстрації вхідних і вихідних документів, їхнього стану і типу;
2) можливість фільтрувати записи з документами за датою отримання, підписання та контрагентом;

3) збереження сканів у базі даних та забезпечення доступу до них зі сторони клієнтського додатку;
4) наявність сервера, який забезпечує розпізнавання тексту (OCR);
5) комунікація між клієнтською та серверною частиною побудована за RESTful-архітектурним стилем;
6) можливість розширення структури серверної частини.

Розробку клієнтської частини виконати на платформі Microsoft .NET з використанням технології WPF. Серверна частина реалізується на платформі Django з використанням Django REST Framework для побудови API та Tesseract OCR для розпізнавання тексту у відсканованих документах.

Додаткові вимоги:

1) наявність зручного та динамічного меню навігації у клієнтському застосунку;
2) вкладок для взаємодії з контрагентами, типами документів та їхніми статусами;
3) можливість виділити дублікати записів з документами;
4) наявність власного логотипа;
5) мінімалістичний інтерфейс.

## 5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Структура клієнтського додатка»;
   - «Проєктування дизайну користувацького інтерфейсу»;
   - «Схема бази даних»;

- «Діаграма прецендентів».

## 6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту ..... 13.11.2024
Розроблення та узгодження технічного завдання..... 22.11.2024
Розроблення структури ПЗ..... 15.12.2024
Розроблення дизайну сторінок та графічних елементів..... 01.02.2025
Програмна реалізація системи ..... 11.03.2025
Тестування ПЗ..... 18.03.2025
Підготовка графічної частини дипломного проєкту ..... 22.04.2025
Оформлення документації дипломного проєкту ..... 27.05.2025

## 7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2025 р.

# ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕЄСТРАЦІЇ ПАПЕРОВИХ ДОКУМЕНТІВ ПІДПРИЄМСТВА

**Пояснювальна записка**

ДП.045490-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ЗАЩИК

2025

**ЗМІСТ**

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface (інтерфейс прикладногопрограмування).

HTTP – Hyper Text Transfer Protocol (протокол передачі гіпертекстових документів).

ID – Identifier (ідентифікатор).

JSON – JavaScript Object Notation (формат обміну структурованими даними).

ML – Machine Learning (машинне навчання).

MVVM – Model-View-ViewModel (архітектурний шаблон розділення логіки в WPF).

OCR – Optical Character Recognition (оптичне розпізнавання символів).

ORM – Object-Relational Mapping (об'єктно-реляційне відображення).

PDF – Portable Document Format (універсальний формат електронних документів).

Regex – Regular Expression (регулярний вираз).

REST – Representational State Transfer (архітектурний стиль побудови API).

SQL – Structured Query Language (мова структурованих запитів).

UI – User Interface (інтерфейс користувача).

URL – Uniform Resource Locator (уніфікований локатор ресурсу).

UX – User Experience (користувацький досвід).

WPF – Windows Presentation Foundation (платформа для побудови графічного інтерфейсу в Windows).

XML – Extensible Markup Language (розширювана мова розмітки).

# ВСТУП

Сучасні підприємства щодня стикаються з великими обсягами паперових документів, які потребують ретельної організації, обліку та зберігання. Відсутність ефективної системи реєстрації та управління такими документами може призвести до втрати важливої інформації, плутанини в процесах та зниження продуктивності працівників. Особливо це актуально для малих і середніх бізнесів, які з різних причин не мають змоги впровадити повноцінні системи електронного документообігу.

На ринку існує низка рішень для автоматизації документообігу, однак багато з них є або надто дорогими, або мають надлишкову складність і функціонал, який не відповідає потребам користувачів. Саме тому виникла потреба у створенні простої, зручної та доступної системи, яка б забезпечувала базові, але важливі функції для обліку та роботи з паперовими документами.

Дипломний проєкт DocRegister має на меті розробку бюджетної альтернативи електронному документообігу. Система складається з двох основних компонентів: Windows-додатку з інтуїтивним інтерфейсом для взаємодії користувачів, а також вебдодатку, який виконує роль серверної частини з обробкою та зберіганням даних. Однією з ключових функцій DocRegister є OCR-розпізнавання документів із підтримкою кирилиці, що дозволяє автоматично визначати тип документа, контрагента, дату підписання тощо, і формувати структуровані дані для подальшої роботи.

DocRegister покликаний спростити процес реєстрації, зберігання та пошуку документів, забезпечуючи швидкий доступ до сканів і супровідної інформації. Завдяки таким можливостям як фільтрація, створення звітів та редагування даних, система стане надійним інструментом у повсякденній роботі співробітників підприємства.

# 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Актуальність автоматизації реєстрації паперового документообігу

У сучасних умовах цифровізації та постійного зростання обсягів інформації автоматизація процесів діловодства, зокрема реєстрації паперового документообігу, набуває особливої актуальності. Традиційна система реєстрації документів, заснована на ручному введенні даних у журнали або таблиці, є не лише трудомісткою, а й вразливою до помилок, затримок та втрати інформації. Відсутність централізованого контролю над документообігом часто призводить до дезорганізації, дублювання записів, а також ускладнює пошук і відстеження документів.

Автоматизація цього процесу дозволяє значно підвищити ефективність роботи установи чи підприємства, мінімізувати людський фактор та забезпечити точність і оперативність ведення документації. Створення спеціалізованого програмного забезпечення для реєстрації вхідних паперових документів дозволяє не лише оптимізувати час, витрачений на їх опрацювання, а й покращити контроль за обігом інформації, зберіганням скан-копій, фіксацією дат надходження, станів опрацювання та супровідних приміток.

З розвитком інформаційних технологій зросли й очікування до швидкості й прозорості внутрішніх процесів. Затримка в реєстрації вхідного документа може призвести до збоїв у подальшому виконанні доручень, юридичних непорозумінь або порушень термінів реагування. Автоматизована система реєстрації дає змогу миттєво отримувати статистику, формувати звіти, фільтрувати документи за різними критеріями, а також інтегруватися з іншими цифровими сервісами.

Окрему увагу слід звернути на фактор збереження даних. Ручна система ведення обліку є вразливою до втрати через людські помилки, пошкодження носіїв або відсутність резервного копіювання. Водночас

цифрова система, що забезпечує зберігання сканованих копій документів і метаданих у базі даних, зменшує ризики втрати інформації, а також спрощує її архівування та доступ.

Також важливим аспектом є економія часу й ресурсів. Працівники, які раніше витрачали години на заповнення журналів або пошук необхідного документа, отримують можливість сконцентруватися на більш значущих завданнях. Це підвищує загальну продуктивність та знижує рівень стресу, спричиненого перевантаженням.

У зв’язку з розширенням гнучких графіків роботи, віддаленої праці та необхідності дистанційного доступу до документів, автоматизовані системи також забезпечують мобільність і безперебійність документообігу навіть поза межами офісу.

Таким чином, автоматизація реєстрації паперового документообігу є важливим кроком на шляху до ефективного, безпечного та прозорого управління документацією. Вона сприяє зниженню витрат часу, покращенню контролю, зменшенню ризиків та підвищенню загальної ефективності організаційної діяльності.

### 1.2. Огляд існуючих додатків автоматизації документообігу

M-Files [1] – це система управління документами, яка автоматизує документообіг завдяки використанню метаданих замість традиційної ієрархії папок. M-Files дозволяє користувачам швидко знаходити потрібні документи, незалежно від того, де вони зберігаються. Платформа підтримує інтеграцію з Microsoft Office, SharePoint та іншими бізнес-додатками. Завдяки потужним функціям контролю версій і автоматичним робочим процесам, M-Files активно використовується в компаніях, які прагнуть зменшити обсяг ручної роботи. Крім того, система забезпечує високий рівень безпеки доступу до інформації, дозволяючи налаштовувати права користувачів відповідно до їх ролей та обов’язків. M-Files підтримує зберігання як структурованих, так і неструктурованих даних, що робить її

універсальним інструментом для організацій різного масштабу. Вигляд інтерфейсу M-Files продемонстровано на рис. 1.1.

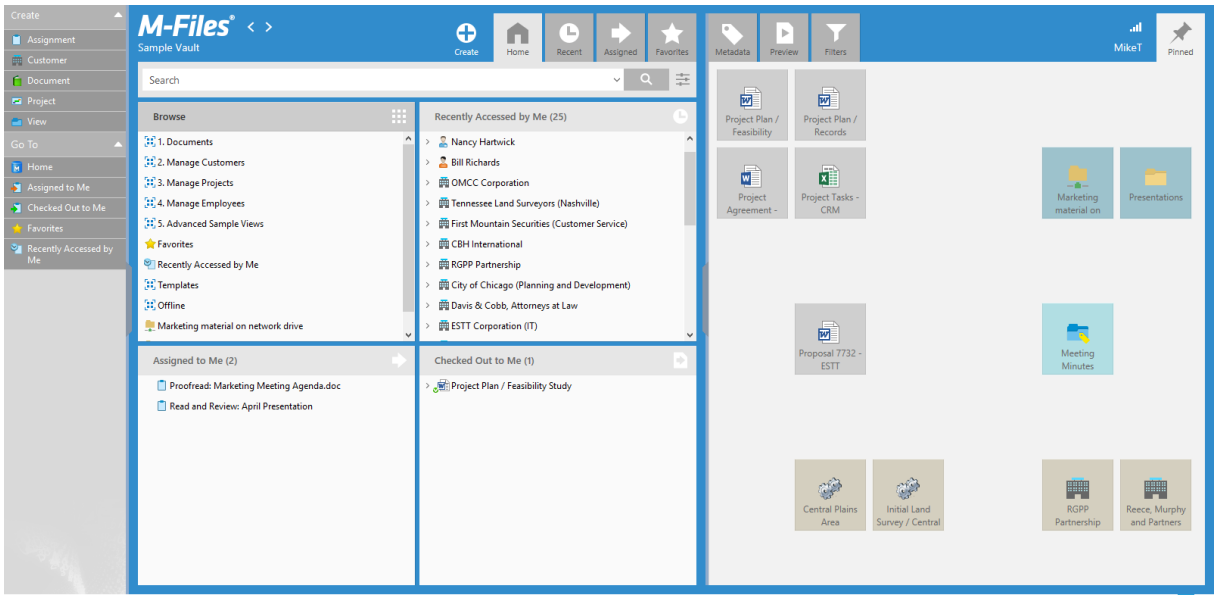



Рис. 1.1. M-Files

Переваги:

- автоматичний контроль версій документів;
- пошук за змістом та метаданими;
- інтеграція з популярними офісними продуктами (Office 365, Outlook, Teams);
- підтримка доступу через мобільні пристрої.

Недоліки:

- складна початкова конфігурація;
- висока вартість для малого бізнесу;
- потребує навчання персоналу для повного використання можливостей.

DocuWare [2] – це хмарне рішення для управління документами та автоматизації робочих процесів. DocuWare забезпечує безпечне зберігання цифрових копій документів, автоматичне сортування, маршрутизацію та архівацію даних. Підтримується розпізнавання тексту (OCR [4, 5]), що

дозволяє працювати навіть із відсканованими копіями паперових документів. Платформа популярна в урядових установах, фінансових організаціях та великих корпораціях. Вигляд інтерфейсу DocuWare продемонстровано на рис. 1.2.

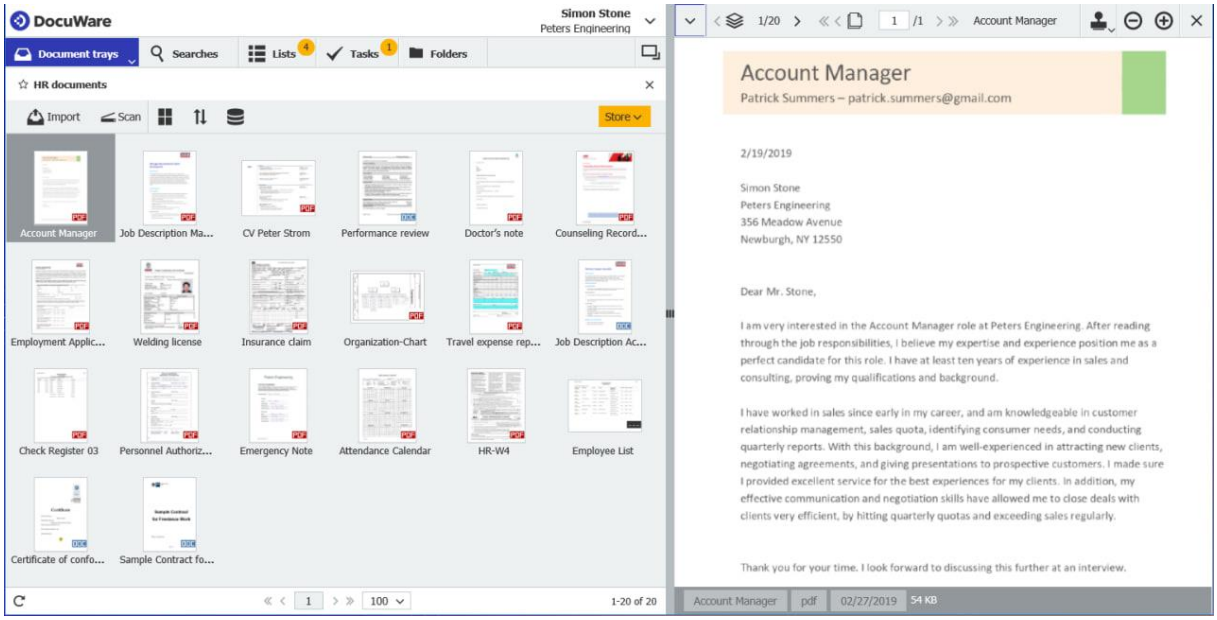



Рис. 1.2. DocuWare

Переваги:

- потужна система автоматизації робочих процесів;
- ocr для обробки сканованих документів;
- високий рівень безпеки даних;
- гнучка рольова модель доступу.

Недоліки:

- може бути надто складним рішенням для невеликих команд;
- вимагає стабільного підключення до Інтернету для роботи.

OnlyOffice Docs [3] – офісний пакет з можливістю спільного редагування документів у реальному часі. Має модуль керування документами, що дозволяє створювати структури папок, надавати доступ до файлів, залишати коментарі та вести журнал змін. Підтримує зберігання документів у локальній мережі або в хмарі. Часто використовується як

альтернатива Microsoft Office у внутрішніх системах документообігу. Крім основних текстових документів, OnlyOffice також дозволяє редагувати електронні таблиці та презентації з широким набором інструментів. Платформа сумісна з найпоширенішими форматами файлів, зокрема DOCX, XLSX та PPTX, що спрощує обмін документами між різними офісними рішеннями. Вигляд інтерфейсу OnlyOffice Docs продемонстровано на рис. 1.3.


Рис. 1.3. OnlyOffice Docs

Переваги:

- безкоштовна версія з відкритим кодом;
- підтримка всіх популярних форматів документів;
- можливість розгортання на власному сервері;
- інтеграція з Nextcloud, ownCloud, Alfresco (рішення для управління документами та спільної роботи).

Недоліки:

- обмежені функції документообігу у порівнянні зі спеціалізованими системами;
- відсутність повноцінної мобільної версії;

- слабша підтримка великих бізнес-процесів.

### 1.3. Результати проведеного аналізу

У результаті аналізу трьох сучасних систем автоматизації документообігу – M-Files, DocuWare та OnlyOffice Docs – було виявлено, що кожна з них має свої особливості, переваги та недоліки залежно від потреб і масштабу організації (табл. 1.1).

M-Files відзначається унікальним підходом до зберігання даних на основі метаданих, що дозволяє легко знаходити документи незалежно від місця їх розташування. Система ідеально підходить для компаній, які потребують гнучкого управління інформацією та контролю версій, проте потребує складного налаштування і навчання персоналу.

DocuWare – потужне хмарне рішення з широкими можливостями автоматизації, високим рівнем безпеки та підтримкою OCR для сканованих документів. Його функціонал орієнтований на великі компанії, урядові структури та фінансові установи. Однак через свою складність система може бути перевантаженою для невеликих команд.

OnlyOffice Docs – це офісний пакет із базовим функціоналом керування документами, який добре підходить для внутрішнього використання та спільного редагування. Він має відкритий код, підтримує всі популярні формати документів та дозволяє розгортання на власному сервері. Але у порівнянні зі спеціалізованими DMS-системами, функціональність документообігу у нього обмежена.

Таблиця 1.1

Порівняльна таблиця існуючих рішень

| Критерій | M-Files | DocuWare | OnlyOffice Docs |
|---|---|---|---|
| Тип рішення | Локальне / хмарне | Хмарне | Локальне / хмарне |

Продовження табл. 1.1

| | | | |
|---|---|---|---|
| Управління документами | Через метадані | Через структуру та автоматизацію процесів | Через структуру папок |
| Контроль версій | Автоматичний | Автоматичний | Історія змін |
| Підтримка OCR | Немає | Є | Немає |
| Інтеграція з іншими системами | Microsoft Office, SharePoint, Teams | Office, Outlook, ERP-системи | Nextcloud, ownCloud, Alfresco |
| Спільна робота з документами | Ні | Ні | У реальному часі |
| Мобільна версія | Підтримується | Підтримується | Обмежена |
| Можливість розгортання на власному сервері | Є | Немає | Є |
| Простота впровадження | Складна кофігурація | Високі вимоги до розгортання | Просте впровадження |
| Вартість | Висока | Висока | Безкоштовна версія з відкритим кодом |
| Цільова аудиторія | Середній та великий бізнес | Великі компанії, державні структури | Малі та середні компанії |

Отже, попри існування потужних систем автоматизації документообігу, більшість із них орієнтовані на великі організації або є

надто складними для малого і середнього бізнесу, який працює переважно з паперовими носіями. У цьому контексті DocRegister виступає як спеціалізоване, просте та бюджетне рішення, яке дозволяє підприємствам без великих інвестицій оптимізувати роботу з паперовими документами.

Вбудований OCR-механізм із підтримкою кирилиці, автоматичне розпізнавання ключових полів документа, інтуїтивно зрозумілий інтерфейс і локальна інсталяція роблять DocRegister ідеальним вибором для компаній, що прагнуть покращити документообіг без залучення складних і дорогих ECM/EDM систем.

Таким чином, DocRegister не просто доповнює ринок систем документообігу, а займає власну нішу, враховуючи реальні потреби та можливості малого і середнього бізнесу.

### 1.4. Обґрунтування необхідності створення "DocRegister"

Попри наявність великої кількості програм для управління завданнями та документообігом, більшість із них орієнтовані або на повноцінні системи електронного документообігу, або на прості органайзери без глибокої інтеграції з паперовими носіями. Водночас багато підприємств малого та середнього бізнесу все ще працюють з паперовими документами, не маючи фінансової чи технічної змоги впровадити складні та дорогі ECM/EDM-системи. Саме для такої аудиторії актуальним є створення спеціалізованої системи DocRegister.

DocRegister – це бюджетна альтернатива електронному документообігу, яка дозволяє структуровано реєструвати, зберігати та опрацьовувати скановані паперові документи. Завдяки вбудованому OCR-механізму з підтримкою кирилиці, система автоматично розпізнає тип документа, контрагента, дату підписання та формує з них інформативну назву, що значно пришвидшує процес обробки.

Важливою перевагою є простота використання, що не потребує складного навчання чи інтеграції з зовнішніми платформами. Усі базові

потреби – зберігання сканів, фільтрація, пошук, складання Excel-звітів – реалізовано у вигляді інтуїтивного Windows-додатку з клієнтським інтерфейсом, що взаємодіє з вебсервером для обробки та збереження даних.

Серед переваг власної розробки DocRegister:

- адаптація саме під реєстрацію паперових документів, а не універсальний документообіг;
- автоматизація рутинних операцій завдяки OCR та фільтрації;
- відсутність залежності від великих постачальників ПЗ та низька вартість впровадження;
- можливість розширення функціоналу в майбутньому під специфічні потреби компанії.

Крім того, DocRegister створений з урахуванням реальних робочих процесів і специфіки документообігу у малих та середніх підприємствах, де документи часто мають нестандартні формати й зберігаються локально. Використання вебсервера у поєднанні з локальним додатком дає можливість гнучко масштабувати систему, забезпечуючи доступ до даних з різних робочих місць без потреби складної інфраструктури.

З огляду на постійне зростання обсягів паперових документів і посилення вимог до їх обліку, DocRegister є своєчасним та ефективним рішенням, що дозволяє підвищити продуктивність праці, знизити ризики втрати важливої інформації та покращити загальний контроль за документообігом без суттєвих витрат часу і коштів.

Таким чином, створення DocRegister – це не просто розробка чергової системи для управління документами, а цільове рішення для бізнесів, які прагнуть навести лад у своєму документообігу без значних інвестицій у складні й неповоротні ECM-системи.

### 1.5. Висновок до розділу

У сучасних умовах ведення ділової діяльності автоматизація процесів документообігу набуває особливої актуальності. Зростання обсягів паперових документів, що циркулюють в організаціях, ускладнює їх оперативну обробку, пошук та зберігання, що призводить до втрат часу та підвищення ризиків помилок. Тому автоматизація реєстрації паперового документообігу є важливим кроком для підвищення ефективності роботи та забезпечення контролю за процесом обробки документів.

Проаналізовані існуючі рішення в сфері автоматизації документообігу показали, що на ринку доступні різноманітні додатки, які пропонують широкий функціонал: від реєстрації та сортування документів до автоматичного формування звітів і контролю доступу. Проте більшість із них мають складний інтерфейс, обмежені можливості адаптації під специфіку конкретного підприємства або потребують значних витрат на впровадження і підтримку.

Крім того, під час аналізу існуючих продуктів було виявлено, що їх функціональні можливості часто перевантажені зайвими опціями, які не використовуються в повсякденній роботі, що ускладнює навчання персоналу та знижує продуктивність. Це створює потребу у створенні більш гнучкого, простого у використанні і одночасно функціонально насиченого інструменту, який би відповідав реальним вимогам користувачів.

Проведений аналіз підтвердив необхідність розробки нового програмного продукту – "DocRegister", який має поєднати в собі інтуїтивно зрозумілий інтерфейс, широкий спектр необхідних функцій та можливість гнучкої налаштування під потреби конкретної організації. Такий підхід дозволить значно оптимізувати процеси документообігу, скоротити час на обробку документів та зменшити кількість помилок, що виникають при ручній реєстрації. Також система повинна забезпечувати

надійний захист даних і відповідати сучасним вимогам інформаційної безпеки.

Отже, на основі огляду предметної області та аналізу існуючих систем автоматизації можна зробити висновок про актуальність розробки "DocRegister" як сучасного та ефективного інструменту для автоматизації реєстрації паперового документообігу. Ця система має потенціал покращити організацію роботи з документами та забезпечити високий рівень контролю за їх рухом у межах підприємства.

# 2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

## 2.1. Вибір технологій для клієнтської частини

Для реалізації клієнтської частини було обрано WPF (Windows Presentation Foundation) [9, 10] – фреймворк для створення настільних застосунків під операційною системою Windows, що входить до складу платформи .NET [8]. Основною причиною такого вибору є потреба в потужному інструменті, здатному забезпечити повноцінний і зручний графічний інтерфейс користувача, який легко адаптується під задачі проєкту.

WPF забезпечує розробнику доступ до розширеної системи розмітки інтерфейсу за допомогою мови XAML, що дозволяє ефективно структурувати елементи UI, підтримувати шаблони та використовувати повторно створені компоненти. Крім того, WPF добре інтегрується з архітектурним підходом MVVM (Model-View-ViewModel), що дозволяє чітко розділити бізнес-логіку, інтерфейс користувача та обробку даних. Така структура спрощує тестування, масштабування та підтримку застосунку.

Серед альтернатив WPF є наступні варіанти:

- WinForms (Windows Forms) – це більш старий фреймворк для створення GUI-додатків на платформі .NET. Він простий у використанні та швидкий для створення простих форм, але має обмежені можливості щодо сучасної графіки, анімації та складного оформлення інтерфейсу. WinForms не підтримує розмітку на базі XAML, а архітектура не заохочує розділення логіки та представлення настільки ж чітко, як Model-View-ViewModel у WPF.
- UWP (Universal Windows Platform) – платформа, що орієнтована на створення універсальних додатків, які можуть працювати на різних пристроях Windows (ПК, планшети, Xbox). UWP підтримує

сучасний дизайн і має доступ до нових функцій Windows, але має обмежену сумісність зі старішими версіями ОС та вимагає специфічного підходу до розгортання (через Microsoft Store).

- WinUI – це сучасний UI-фреймворк від Microsoft, що є наступником UWP і підтримує нові стандарти дизайну Fluent Design. WinUI дозволяє створювати сучасні інтерфейси і має гарну інтеграцію з .NET 5/6+, але ще не так широко використовується, і підтримка більшості бізнес-додатків все ще базується на WPF.

Порівняння вищезгаданих технологій наведено в табл. 2.1.

Таблиця 2.1

Порівняльна таблиця існуючих рішень

| Характеристика | WinForms | WPF | UWP/WinUI |
|---|---|---|---|
| Підтримка графіки | Обмежена (GDI+) | Потужна, з підтримкою векторної графіки, анімації | Сучасна графіка, Fluent Design |
| Архітектура | Тісно пов’язана, складно масштабувати | MVVM, чітке розділення логіки | MVVM, сучасний підхід |
| Підтримка XAML | Ні | Так | Так |
| Кросплатформеність | Ні | Частково (через .NET Core) | Так (Windows 10+ пристрої) |
| Продуктивність | Вища для простих додатків | Вища для складних UI | Оптимізовано для сучасних пристроїв |

Продовження табл. 2.1

| | | | |
|---|---|---|---|
| Майбутнє | Обмежене (підтримка, але не розвиток) | Активний розвиток, підтримка в .NET | Активний розвиток, але нова платформа |

Отже, для поставлених задач WPF забезпечує найкраще співвідношення гнучкості, потужності та стабільності. Він дає змогу створити багатофункціональний, сучасний та візуально привабливий інтерфейс, який легко підтримувати та масштабувати завдяки архітектурі Model-View-ViewModel. При цьому WPF має глибоку інтеграцію з .NET екосистемою і підтримує локальні десктопні рішення на Windows без необхідності переходити на нові платформи або обмежуватися Microsoft Store.

Крім того, порівняно з WinForms, WPF надає набагато ширші можливості для роботи з графікою, анімаціями та адаптивністю інтерфейсу, що є важливим для користувацького досвіду. У той же час, UWP і WinUI, хоч і пропонують сучасні інструменти, поки не набули такої ж масової підтримки в бізнес-додатках і обмежені цільовою платформою Windows 10+.

Таким чином, WPF став оптимальним вибором, що відповідає вимогам проєкту щодо гнучкості, дизайну та підтримки складної бізнес-логіки у настільному застосунку.

### 2.2. Вибір системи керування базами даних

У якості СУБД для даного додатку було обрано PostgreSQL [12].

У якості основної системи керування базами даних для даного проєкту було обрано PostgreSQL. Це потужна реляційна СУБД, що підтримує ACID-транзакції [13], типізацію даних, складні SQL-запити, зв’язки між таблицями, а також індексацію та зберігання структурованих даних. У проєкті PostgreSQL забезпечує збереження інформації про

документи, результатів розпізнавання, метаданих, контрагентів, а також історії запитів користувача.

Однією з основних переваг PostgreSQL є строга типізація та підтримка зовнішніх ключів, що забезпечує цілісність та логічну послідовність даних. Це особливо важливо при роботі з пов’язаними сутностями, такими як скани документів, користувачі, організації, типи документів тощо. Додатково, підтримка JSON та JSONB дозволяє зберігати напівструктуровану інформацію у зручному форматі, що актуально при роботі з розпізнаними блоками тексту або динамічними даними.

PostgreSQL також вирізняється потужним SQL-двигуном, який дозволяє ефективно виконувати запити з умовами фільтрації, сортуванням, агрегацією та використанням віконних функцій. Це суттєво спрощує реалізацію складних запитів у бекенді для побудови аналітики, підрахунку статистики та формування звітів.

Ще однією вагомою причиною вибору цієї СУБД є можливість локального розгортання. У такому випадку сервер бази даних розгортається на тому ж пристрої, що й WPF-додаток, що дозволяє працювати навіть за відсутності інтернет-з’єднання. Це надає автономність та зменшує залежність від зовнішніх сервісів. Додатково, PostgreSQL є відкритим програмним забезпеченням, що не потребує ліцензійних платежів.

Серед альтернатив PostgreSQL [12] є MySQL [15], SQLite [14], MS SQL Server [17], Oracle [16] та інші. Нижче розглянемо їхні сильні та слабкі сторони.

MySQL – одна з найпопулярніших реляційних СУБД з відкритим кодом, яку часто використовують у вебзастосунках. MySQL відома своєю простотою, високою швидкістю роботи з читанням даних і широкою підтримкою хостинг-платформ. Проте вона має деякі обмеження у порівнянні з PostgreSQL – менш потужний SQL-двигун, обмежена

підтримка складних запитів, віконних функцій та транзакційної обробки. У випадках, де потрібна гнучка робота з даними, складна аналітика та підтримка нетипових типів (наприклад, JSONB), PostgreSQL забезпечує більш повний функціонал.

SQLite – надлегка вбудована СУБД, що зберігає дані у звичайному файлі. Вона ідеально підходить для невеликих проєктів, мобільних застосунків або випадків, коли потрібне збереження даних без розгортання окремого серверу. Проте SQLite не призначена для багатокористувацького доступу, не підтримує розширені можливості транзакцій, масштабування та складні SQL-запити на рівні, притаманному PostgreSQL. Через це вона не підходить для повноцінної клієнт-серверної архітектури, на якій базується наш проєкт.

Oracle Database – це потужна комерційна СУБД, яка широко використовується у великих корпоративних системах, особливо там, де потрібна масштабованість, висока доступність та безперервність бізнесу. Вона надає розширену підтримку процедурної логіки, розподілених транзакцій, реплікації та безпеки. Однак Oracle є платною системою з доволі високою вартістю ліцензій, що робить її менш доцільною для невеликих або середніх проєктів. У нашому випадку PostgreSQL обрано як безкоштовну альтернативу, що забезпечує подібний функціонал, достатній для вимог проєкту, але без фінансових витрат.

Microsoft SQL Server – також комерційна СУБД з широким набором можливостей, включно з інтеграцією з екосистемою Microsoft, зручною адміністративною панеллю (SSMS), аналітичними сервісами та підтримкою складних запитів. Її сильна сторона – тісна інтеграція з Windows, що може бути перевагою у деяких проєктах. Проте, на відміну від PostgreSQL, SQL Server у повнофункціональних редакціях вимагає ліцензування, а безкоштовна версія (SQL Server Express) має обмеження на розмір бази та використання ресурсів. PostgreSQL забезпечує більшу

гнучкість і можливості без таких обмежень, що робить її більш привабливою для локальних WPF-застосунків.

Отже, завдяки поєднанню стабільності, надійності, широкої функціональності та відсутності ліцензійних обмежень, PostgreSQL виявився найбільш конкурентоспроможним варіантом серед доступних СУБД. На відміну від комерційних рішень, він не вимагає фінансових витрат, а в порівнянні з легшими системами, як-от SQLite чи MySQL, забезпечує вищу гнучкість, кращу підтримку складних запитів, типізацію, роботу з JSON-структурами та цілісність даних. Це робить PostgreSQL оптимальним вибором для реалізації системи, що поєднує локальне збереження інформації, роботу з REST API та обробку розпізнаних документів.

### 2.3. Вибір технологій для OCR API

OCR планується бути розробленим як вебсервер, який імплементовує RESTful API для розпізнавання кириличного тексту спеціально під клієнтський додаток. Для цих цілей було обрано Python [6], Django та Django Rest Framework.

Python [6, 7] – це високорівнева мова програмування з активною спільнотою та великою кількістю бібліотек. Вона широко використовується в галузях машинного навчання, обробки зображень, наукових розрахунків та веброзробки. Завдяки наявності потужних бібліотек, таких як Pillow, OpenCV, Tesseract, PyMuPDF та ін., Python є ідеальним вибором для реалізації сервісів OCR. Легка інтеграція з API-сервісами, розвинена екосистема, зручне логування, обробка винятків і підтримка асинхронності дозволяють швидко створювати надійні рішення.

Django – це високорівневий фреймворк для веброзробки на Python, що дозволяє будувати масштабовані, безпечні та підтримувані застосунки. Він надає зручну архітектуру Model-View-Template, вбудовану панель адміністратора, ORM для роботи з базами даних, систему маршрутизації,

підтримку форм, а також механізми автентифікації. Для цього проєкту Django виконує роль основного серверного фреймворку, який обробляє HTTP-запити від клієнтського WPF-додатку, ініціює процес розпізнавання тексту на сканах та повертає результати у структурованому форматі.

Django Rest Framework (DRF) – це потужне розширення для Django, яке значно спрощує розробку RESTful API. Воно забезпечує зручну серіалізацію моделей, підтримку пагінації, фільтрації, автентифікації, прав доступу, обробки запитів і відповідей у форматі JSON. DRF ідеально підходить для реалізації сервісів, де клієнтський додаток (у цьому випадку – WPF) потребує чітко структурованого API для обміну даними з сервером. За допомогою DRF можна швидко створювати контролери (ViewSets), які обробляють запити OCR, повертати розпізнані значення, а також реалізовувати додаткові ендпоінти для управління історією, статистикою чи аналітикою.

Із альтернатив для реалізації цього модуля є наступні технології:

- Node.js [18] + Express [19]. Це популярний вибір для створення швидких і масштабованих REST API завдяки подієвій моделі та неблокуючому вводу/виводу. Проте для завдань комп’ютерного зору та OCR, реалізованих на Python, інтеграція може бути складнішою, оскільки основні бібліотеки для OCR краще підтримуються у Python-екосистемі.
- Java [20] + Spring Boot [21]. Потужний фреймворк для корпоративних застосунків із високою продуктивністю та надійністю. Проте розробка OCR-сервісу на Java може бути складнішою та повільнішою, а підтримка спеціалізованих бібліотек для обробки зображень і розпізнавання тексту менш розвинена порівняно з Python.
- Go (Golang) [22]. Відзначається високою швидкістю та ефективним використанням ресурсів, ідеальний для легких REST-сервісів. Однак Go має обмежену екосистему для складної обробки

зображень та машинного навчання, що робить його менш придатним для задач OCR у порівнянні з Python.

- .NET Core (C#) [23]. Добре підходить для створення вебсервісів у середовищі Windows, має потужні інструменти для розробки API. Але реалізація OCR безпосередньо у C# менш поширена, а інтеграція з найкращими OCR-бібліотеками часто вимагає додаткових обхідних рішень.

Таким чином, вибір Python з Django і DRF є обґрунтованим компромісом між простотою розробки, доступністю потужних бібліотек для OCR, а також можливістю швидко створювати масштабовані, надійні та підтримувані RESTful сервіси, оптимально адаптовані під потреби клієнтського додатку.

### 2.4. Вибір сервісів для хостингу OCR API

OCR API було розгорнуто у хмарній інфраструктурі Amazon Web Services (AWS) [24] на віртуальному сервері типу EC2 (Elastic Compute Cloud). Такий підхід забезпечує високу доступність, гнучкість масштабування та повний контроль над серверним середовищем. Використання EC2 дозволяє налаштовувати інстанси з різними характеристиками процесора, оперативної пам’яті, сховища та мережевих ресурсів, що дає змогу ефективно адаптуватися до змін навантаження на сервіс. У випадку збільшення кількості запитів до OCR API, можна швидко збільшити потужність існуючих інстансів або додати нові, організувавши балансування навантаження між ними. Це гарантує стабільність роботи та високу продуктивність системи навіть при пікових навантаженнях.

Операційна система, обрана для серверного середовища, – Ubuntu Server. Цей вибір обумовлений її стабільністю, широкою підтримкою спільноти та хорошою сумісністю з інструментами для веброзробки. На сервері було розгорнуто серверну частину на Django, налаштовано вебсервер (Nginx або Apache), а також організовано процеси фонової

обробки для асинхронного виконання задач, пов’язаних із розпізнаванням тексту. Така архітектура дозволяє розділяти навантаження між обробкою HTTP-запитів і важкими фоновими задачами, що підвищує загальну ефективність та надійність системи.

Використання AWS EC2 дає ряд переваг у порівнянні з іншими варіантами розгортання:

- Хмарні платформи Microsoft Azure та Google Cloud Platform (GCP).
  Ці платформи також пропонують подібні сервіси віртуальних машин із широким вибором конфігурацій і можливостями масштабування. Проте AWS EC2 вирізняється найбільшим вибором типів інстансів і найширшою екосистемою інтеграцій, що забезпечує більшу гнучкість для налаштувань під специфіку OCR-сервісу. Крім того, AWS має більш розвинену мережеву інфраструктуру та багаторівневі механізми безпеки, що є критичним для захисту даних.
- Контейнери (Docker, Kubernetes).
  Альтернативою запуску сервісу на віртуальних машинах є використання контейнерів, які забезпечують легковагу, швидке розгортання і автоматичне масштабування. Однак впровадження контейнеризації додає складності у підтримку та моніторинг, особливо на початкових етапах розробки. Для невеликих і середніх проєктів EC2 забезпечує простіше керування і більший контроль без необхідності впроваджувати складну оркестрацію.
- Віртуальні приватні сервери (VPS) від хостинг-провайдерів.
  VPS є дешевшим варіантом для розгортання вебсервісів, але вони часто мають обмежену масштабованість, слабкіші механізми автоматичного відновлення після збоїв та менше можливостей щодо балансування навантаження. Для сервісу OCR, де важлива висока доступність і стабільність, VPS може стати вузьким місцем при рості користувачів.

- Serverless-платформи (AWS Lambda, Azure Functions). Serverless дозволяє запускати функції на вимогу без управління інфраструктурою, що спрощує масштабування та знижує операційні витрати. Проте для OCR-завдань із тривалими та ресурсомісткими процесами обробки зображень обмеження часу виконання функцій і необхідність комплексної інтеграції роблять цей підхід менш зручним.

Отже, вибір AWS EC2 як платформи для розгортання OCR API став оптимальним рішенням, що поєднує необхідний рівень контролю, продуктивності і масштабованості. Це дозволяє підтримувати стабільну, безпечну та ефективну роботу сервісу, що є критично важливою частиною загальної системи розпізнавання документів.

### 2.5. Висновок до розділу

У результаті аналізу існуючих технологій для реалізації клієнтської частини, системи управління базами даних та OCR API було обрано найбільш оптимальні рішення, які відповідають поставленим вимогам та особливостям проєкту. Для створення клієнтського застосунку використано технологію WPF, яка дозволяє створювати сучасний, зручний та функціональний інтерфейс користувача з підтримкою архітектури Model-View-ViewModel. Це сприяє розділенню логіки від представлення, що полегшує подальшу підтримку і масштабування системи.

Для збереження та обробки даних обрана система керування базами даних PostgreSQL, що відзначається високою надійністю, масштабованістю та широкими можливостями для виконання складних запитів. Важливим фактором стала також можливість локального розгортання без додаткових витрат на ліцензії, що забезпечує ефективне використання ресурсів. PostgreSQL добре інтегрується з іншими компонентами системи і гарантує збереження цілісності даних. Крім того, активна спільнота та наявність великої кількості документації дозволяють

швидко вирішувати проблеми, що виникають під час розробки або супроводу бази даних.

Для розробки OCR API було вирішено використати мову Python разом з фреймворком Django та Django Rest Framework. Цей вибір обумовлений гнучкістю, широкою підтримкою бібліотек для обробки зображень та тексту, а також швидкістю розробки RESTful сервісів. Завдяки цьому API зможе ефективно взаємодіяти з іншими модулями системи та обробляти запити на розпізнавання тексту з високою точністю. Особливу увагу було приділено забезпеченню стабільності та безпеки API, включаючи захист від несанкціонованого доступу та обробку помилок на стороні сервера.

Таким чином, обрані технології утворюють надійну та масштабовану платформу, що забезпечує реалізацію всіх функціональних вимог проєкту. Вони поєднують у собі сучасні підходи до розробки, зручність підтримки і подальшого розвитку, що створює міцний фундамент для успішного впровадження системи.

Крім того, така технологічна база відкриває можливості для майбутньої інтеграції з зовнішніми сервісами та розширення функціональності, зокрема впровадження аналітичних модулів або машинного навчання для автоматизованої обробки вхідних даних.

# 3. АРХІТЕКТУРА ТА ДИЗАЙН РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Загальна архітектура DocRegister

Розроблюване програмне забезпечення DocRegister має модульну архітектуру, що складається з трьох основних компонентів: клієнтської частини, вебзастосунку, який виконує роль OCR API, та бази даних.

Клієнтська частина, реалізована за допомогою WPF, забезпечує користувачу інтерфейс для роботи з документами, контрагентами, типами документів та іншими елементами системи, взаємодіючи з серверною частиною через HTTP-запити.

Вебзастосунок, побудований на Python і Django, обробляє запити на розпізнавання тексту, формує структуровані дані та повертає їх клієнту.

PostgreSQL служить основною системою керування базами даних, де зберігаються всі необхідні дані, що дозволяє ефективно управляти інформацією про документи, контрагентів і типи документів.

Взаємодія між компонентами забезпечує стабільну роботу системи, дозволяючи швидко обробляти та зберігати великі обсяги даних, а також ефективно взаємодіяти з користувачем для виконання всіх необхідних операцій.

## 3.2. Загальна структура клієнтської частини

У клієнтській частині DocRegister використовується архітектура MVVM (Model-View-ViewModel) разом із сервісами, де відбувається обробка бізнес логіки (рис. 3.1). Також є окремий модуль DocAnalisis зі своїм контроллером, сервісом та утилітами, які використовуються при розпізнаванні тексту. Нижче наведено детальніший опис компонентів застосунку.

- Views: обробка користувацької взаємодії із застосунком та сам інтерфейс у вигляді XAML коду.

- Models: задання об’єктів, які відповідають структурі таблиць у базі даних.
- ViewModels: пов’язування Views та Models таким чином, щоб Views тригерили події, але логіка їхньої обробки була винесена у ViewModels, де також використовуються Models у поєднанні з Services, DBServices, DocAnalisis.
- Services: модуль, де зосереджена основна частина бізнес-логіки, яка стосується фільрації, формування Excel звітів, конфігурації бази даних та середовища, конвертацій між внутрішньопрограмними типами даних.
- DBServices: модуль, що відповідає за доступ до даних. По суті відіграє роль рівня репозиторіїв. Через цей модуль відбувається взаємодія з базою даних.
- DocAnalisis: інкапсулює логіку розпізнавання та аналізу тексту документів. Використовує API для розпізнавання тексту на сканах.

OCR API: API для розпізнавання тексту на файлах чи зображеннях. У якості цього модуля може використовуватись як сервіс Azure, так і OCR сервіс, розроблений разом із настільною частиною ПЗ.

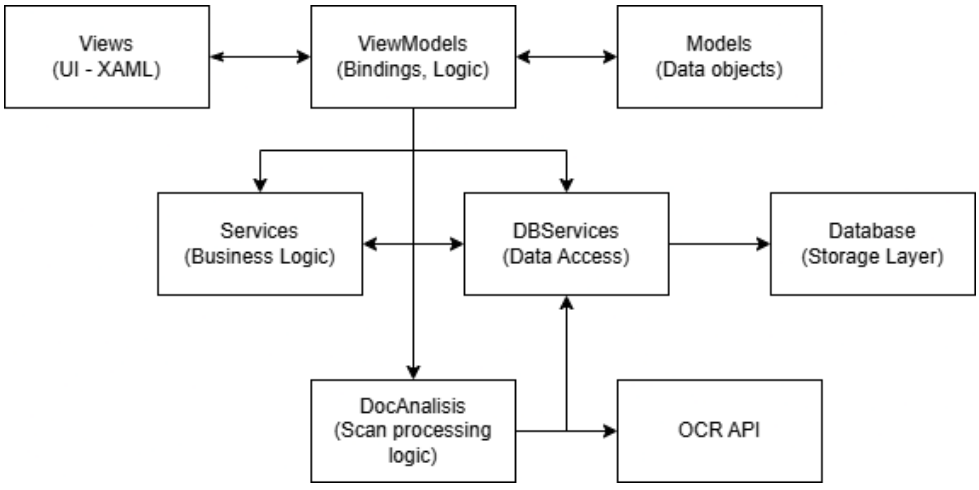



Рис. 3.1. Структура клієнтського додатка

## 3.3. Структура бази даних

Структура бази даних є важливою складовою під час розробки додатків, оскільки саме вона визначає спосіб зберігання, організації та обробки даних. У цьому розділі буде представлено детальний опис структури бази даних для розробленого додатку, який використовує PostgreSQL – потужну реляційну систему керування базами даних. Оскільки додаток працює локально і не потребує синхронізації через мережу, вибір локальної бази даних є оптимальним рішенням. PostgreSQL забезпечує надійне зберігання даних, підтримує складні запити та гарантує високу продуктивність, що робить її ідеальним варіантом для даного проєкту.

### *3.3.1. Структура бази даних*

Для застосунку DocRegister базу даних було спроєктовано таким чином, щоб забезпечити збереження інформації про документи, контрагентів, стани обробки документів та інші супутні дані. Нижче подано опис кожної таблиці, яка входить до структури бази даних.

Таблиця Contragents (табл. 3.1) містить інформацію про контрагентів – організації або фізичних осіб, з якими ведеться документообіг. Унікальний ідентифікатор id дозволяє пов’язати контрагента з відповідними документами в інших таблицях.

Таблиця 3.1

Contragents (Контрагенти)

| Field | Type |
|---|---|
| id | integer |
| contragent_name | character varying |

У таблиці Document_Types (табл. 3.2) зберігаються типи документів, такі як договори, акти, накладні тощо. Кожен тип має унікальний id та текстову назву.

Таблиця 3.2

Document_Types (Типи документів)

| Field | Type |
|---|---|
| id | integer |
| document_type | string |

Таблиця Documents (табл. 3.3) зберігає основну інформацію про документи. Вона включає номер документа, дату підписання, тип документа, зв’язок з контрагентом, а також скановану копію документа у вигляді байтового масиву.

Таблиця 3.3

Documents (Документи)

| Field | Type |
|---|---|
| id | integer |
| document_type_id | integer |
| document_number | character varying |
| sign_date | date |
| contragent_id | integer |
| scan | bytea |

Таблиця Records (табл. 3.4) відповідає за збереження записів реєстрації документів. Вона пов’язує документ з датою його отримання, поточним станом обробки та додатковими примітками.

Таблиця 3.4

Records (Записи)

| Field | Type |
|---|---|
| id | integer |
| receive_date | date |
| document_id | integer |
| state_id | integer |
| notes | text |

У таблиці States (табл. 3.5) перераховані можливі стани обробки документа, наприклад: "Новий", "В обробці", "Опрацьовано". Ці стани використовуються для моніторингу та контролю за виконанням документообігу.

Таблиця 3.5

States (Стани)

| Field | Type |
|---|---|
| id | integer |
| state_name | character varying |

### *3.3.2. Опис основних сутностей*

Сутність Contragents зберігає інформацію про кожного контрагента і містить наступні стовпці та обмеження:

Стовпці:

- id: унікальний ідентифікатор контрагента (PK);
- contragent_name: назва контрагента, унікальна.

Обмеження:

- contragent_name_unique: обмеження на унікальність contragent_name;
- contragents_pkey: основний ключ по полю id.

Сутність Document_Types зберігає інформацію про типи документів та містить наступні стовпці та обмеження:

Стовпці:

- id: унікальний ідентифікатор типу документа (PK);
- document_type: тип документа, унікальний.

Обмеження:

- document_type_unique: обмеження на унікальність document_type;
- document_Types_pkey: основний ключ по полю id.

Сутність Documents зберігає інформацію про документи, має зв’язки на таблиці Contragents та Document_Types і містить наступні стовпці:

- id: унікальний ідентифікатор документа (PK);
- document_type_id: ідентифікатор типу документа (FK на Document_Types);
- document_number: номер документа;
- sign_date: дата підписання документа;
- contragent_id: ідентифікатор контрагента (FK на Contragents);
- scan: сканована копія документа у вигляді бінарних даних.

Сутність Records зберігає інформацію про записи з документами, має зв’язки на таблиці Documents та States і містить наступні стовпці:

- id: унікальний ідентифікатор запису (PK);
- receive_date: дата отримання запису;
- document_id: ідентифікатор документа (FK на Documents);
- state_id: ідентифікатор стану (FK на States);
- notes: примітки до запису.

Сутність States зберігає інформацію про стани документів та містить наступні стовпці та обмеження:

Стовпці:

- id: унікальний ідентифікатор стану (PK);
- state_name: назва стану.

Обмеження:

- state_name_unique: обмеження на унікальність state_name;
- states_pkey: основний ключ по полю id.

Структура бази даних для DocRegister (рис. 3.2), описана вище, забезпечує надійне зберігання, високу продуктивність і можливість масштабування системи. Завдяки використанню локального сервера баз даних PostgreSQL, додаток отримує стабільну платформу для роботи з великими обсягами даних, гарантуючи цілісність і безпеку інформації.

Організація бази даних передбачає використання кількох основних таблиць: Contragents, Documents, Document_Types, Records та States. Кожна з цих таблиць відповідає за певний тип даних і пов'язана між собою через зовнішні ключі. Така побудова забезпечує логічну цілісність даних і спрощує роботу з ними на рівні запитів і бізнес-логіки додатку.

PostgreSQL дозволяє ефективно виконувати складні запити, завдяки підтримці транзакцій, індексів та оптимізації зберігання даних. Наявність унікальних обмежень (UNIQUE) і первинних ключів (PRIMARY KEY) гарантує унікальність важливих полів, таких як назва контрагента чи тип документа, що підвищує надійність системи.

Зовнішні ключі (FOREIGN KEY) між таблицями підтримують зв'язки між документами, записами і контрагентами, автоматично забезпечуючи контроль за узгодженістю даних навіть у разі оновлення або видалення пов'язаних записів. Завдяки цьому, DocRegister може безпечно управляти залежностями між об'єктами в системі.

Таким чином, обрана структура бази даних у DocRegister забезпечує гнучкість, стабільність і високу продуктивність. Локальне розгортання на PostgreSQL дозволяє легко контролювати стан системи, ефективно обробляти запити й адаптуватися до зростання обсягів даних або змін у вимогах бізнесу.

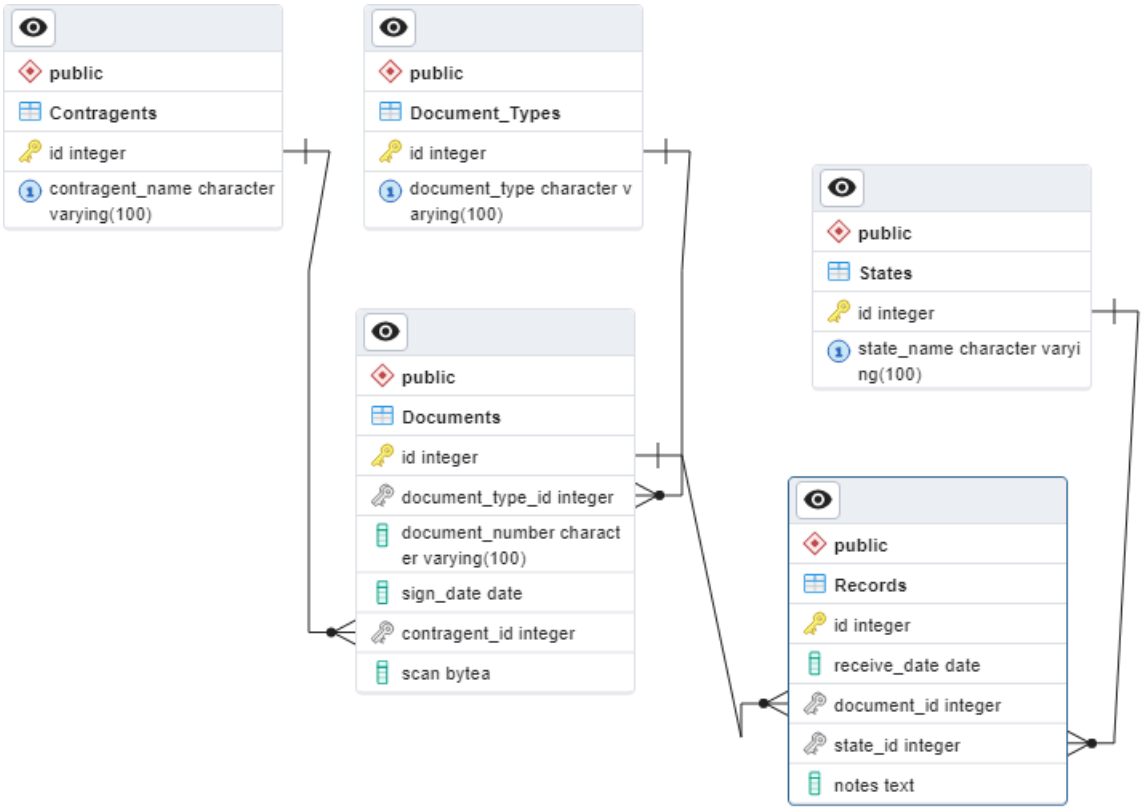



Рис. 3.2. ERD-діаграма бази даних

### 3.4. Дизайн користувацького інтерфейсу

Користувацький інтерфейс застосунку DocRegister розроблений з акцентом на простоту, ефективність та зручність у роботі з великою кількістю документів. Його структура складається з декількох

функціональних вкладок, що дозволяють користувачеві швидко орієнтуватися у програмі та виконувати всі необхідні дії з мінімальними зусиллями. Алгоритм проектування дизайну користувацького інтерфейсу наведено на рис. 3.3.

Головна вкладка – «Документи». Після запуску програми користувач автоматично потрапляє до вкладки «Документи», де зосереджено основні функціональні можливості. Тут можна переглядати документи у табличному форматі, очищати список відображення, застосовувати фільтри за датою, контрагентом або типом документу, додавати нові записи, а також редагувати або видаляти існуючі.

До кожного запису можна завантажити скан у форматі PDF. Окремо реалізовано функцію розпізнавання тексту на завантаженому файлі (OCR). У результаті аналізу автоматично формується запис із розпізнаними даними: номером документа, його типом, датою підписання, контрагентом, і відповідним фрагментом скану (виділена частина файлу, що відповідає конкретному документу).

Також користувач може завантажити звіт, що включає Excel-файл із таблицею документів і ZIP-архів із усіма сканами, впорядкованими за ієрархією: дата → контрагент → скан. Такий підхід забезпечує зручне зберігання та доступ до архівних матеріалів.

Вкладка – «Контрагенти». Ця вкладка реалізує функціонал довідника контрагентів. Користувач має змогу переглядати, додавати, редагувати та видаляти записи про контрагентів, які використовуються як при розпізнаванні сканів, так і при фільтрації чи редагуванні документів у головній вкладці. Інтерфейс виконано у вигляді таблиці з мінімалістичною панеллю керування.

Вкладка – «Типи документів». Аналогічним чином реалізовано управління типами документів. На відповідній вкладці користувач може створювати нові типи, змінювати або видаляти наявні. Вибрані типи

документів використовуються при автоматичному заповненні записів під час розпізнавання сканів, а також під час редагування та фільтрації.

Вкладка – «Стани документів». Ще однією важливою частиною інтерфейсу є вкладка «Стани документів». Вона дає змогу керувати довідником статусів документів – наприклад, «Чернетка», «У роботі», «Підписано», «Архівовано» тощо. Користувач має можливість створювати нові стани, редагувати їх або видаляти. Стани документів можуть бути використані для візуальної класифікації записів у таблиці документів, застосування фільтрів, а також для побудови звітності. Це дозволяє ефективніше контролювати життєвий цикл кожного документа.

Загалом дизайн інтерфейсу базується на модульному підході, використанні зрозумілих форм, іконок та підказок. Завдяки уніфікованому вигляду вкладок користувач швидко звикає до логіки роботи програми. Забезпечено повну підтримку клавіатурної навігації, що особливо корисно при роботі з великими обсягами даних. Такий підхід дозволяє не лише покращити взаємодію з системою, але й легко масштабувати її функціональність у майбутньому.

Окрему увагу приділено доступності та адаптивності інтерфейсу. Застосунок підтримує динамічне масштабування елементів інтерфейсу під різні роздільні здатності екранів, що забезпечує комфортну роботу як на стандартних моніторах, так і на широкоформатних або з високою щільністю пікселів. Контрастні кольори, читабельні шрифти та чітка візуальна ієрархія сприяють зручному сприйняттю інформації навіть при тривалому використанні програми.

Крім того, передбачено механізм повідомлень і підказок, що з’являються в процесі взаємодії користувача із системою. Повідомлення про помилки, підтвердження дій або системні сповіщення реалізовані у ненав’язливому стилі, не відволікаючи від основної роботи. Така система зворотного зв’язку підвищує прозорість роботи програми та допомагає уникнути помилок при виконанні рутинних операцій.

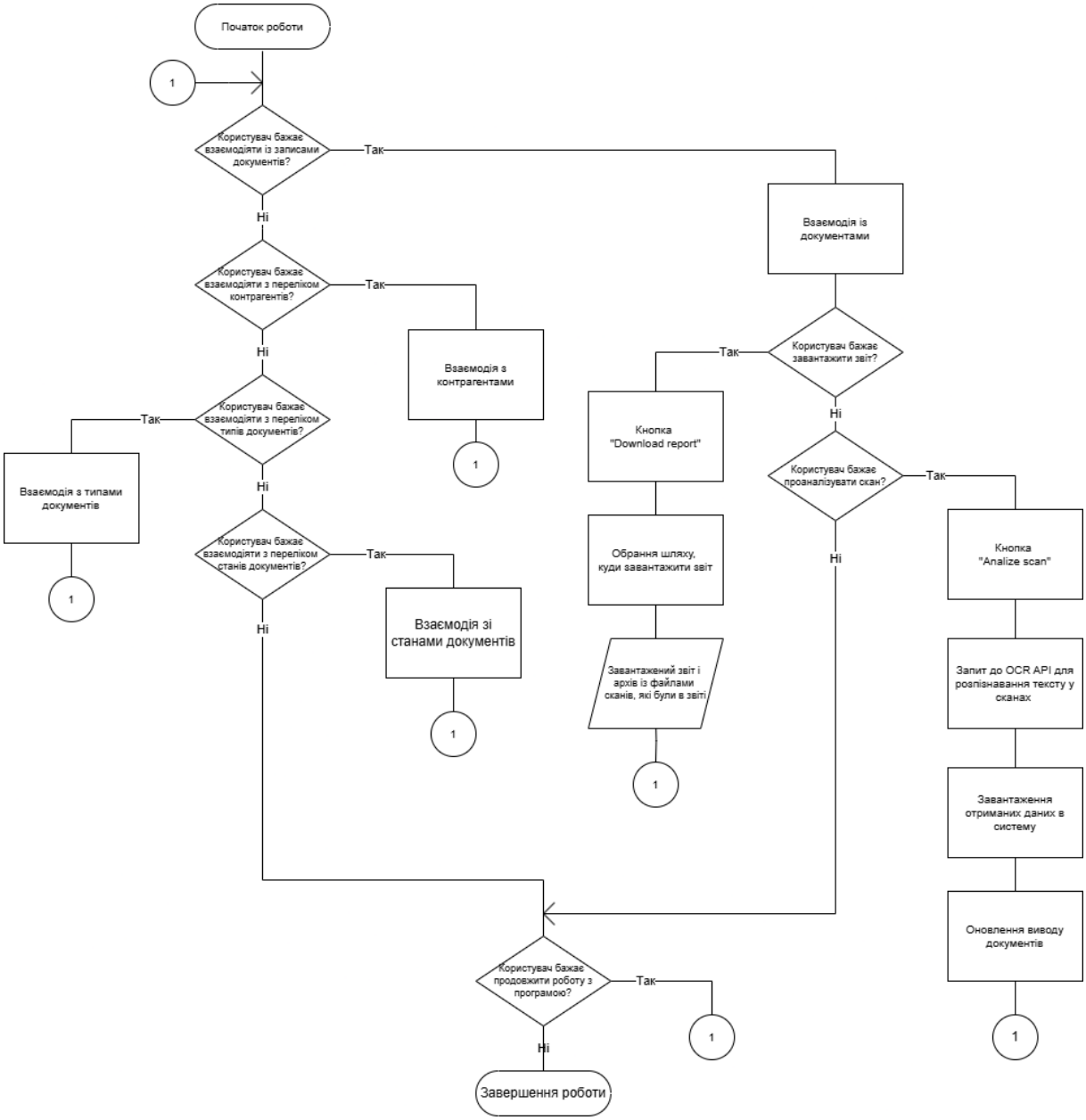



Рис. 3.3. Проєктування дизайну користувацького інтерфейсу

### 3.5. Висновок до розділу

У даному розділі було оглянути архітектуру, структуру бази даних та дизайн користувацького інтерфейсу програмного забезпечення DocRegister. Розроблена модульна архітектура, що складається з клієнтської частини на базі WPF, вебзастосунку для OCR API на Python/Django та реляційної бази даних PostgreSQL, забезпечує ефективну взаємодію між компонентами системи.

Використання патерну Model-View-ViewModel у клієнтській частині дозволяє розділити логіку відображення та бізнес-логіку, що підвищує підтримуваність і розширюваність коду. Сервісний підхід сприяє чіткому розподілу відповідальностей між модулями, зокрема у обробці бізнес-логіки, доступі до даних і розпізнаванні тексту.

Структура бази даних, побудована на основі PostgreSQL, підтримує цілісність, унікальність і зв’язність даних завдяки застосуванню первинних та зовнішніх ключів, унікальних обмежень і оптимізованих типів даних. Така організація бази даних гарантує надійність, високу продуктивність та масштабованість системи.

Дизайн користувацького інтерфейсу створений з урахуванням простоти та зручності використання, що дозволяє користувачу ефективно взаємодіяти з великими обсягами документів і виконувати всі необхідні операції швидко та інтуїтивно.

Окрім зазначених переваг, важливою складовою системи DocRegister є її розширюваність. Завдяки модульному підходу можливо легко інтегрувати нові функціональні блоки без потреби суттєвого перепроектування існуючих компонентів. Наприклад, у майбутньому можна реалізувати електронний цифровий підпис для документів або інтеграцію з зовнішніми системами документообігу. Такий потенціал для розширення робить систему придатною для використання в організаціях з різними вимогами та масштабами діяльності.

Отже, розроблена архітектура і дизайн системи DocRegister забезпечують стабільну, масштабовану та зручну у використанні платформу для управління документами, що відповідає сучасним вимогам і забезпечує високу якість обробки та збереження інформації.

# 4. РЕАЛІЗАЦІЯ КЛЮЧОВИХ ФУНКЦІОНАЛЬНОСТЕЙ

## 4.1. Реєстрація та облік документів

Функція реєстрації та обліку документів є основною в програмному забезпеченні DocRegister. Вона дозволяє створювати нові записи документів, зберігати до них скани, а також автоматично заповнювати дані за допомогою вбудованого модуля розпізнавання тексту (OCR). Це значно спрощує рутинну роботу з первинною документацією та забезпечує централізоване збереження всіх записів.

### *4.1.1. Додавання документа вручну*

Для додавання документа вручну (рис. 4.1), користувач заповнює рядок з наступними даними:

- дата отримання документа;
- контрагент;
- тип документа;
- номер документа;
- дата підписання документа;
- стан («Бухгалтерія», «Юр. відділ», «Директор» тощо);
- нотатки.

Також можна завантажити скан документа.

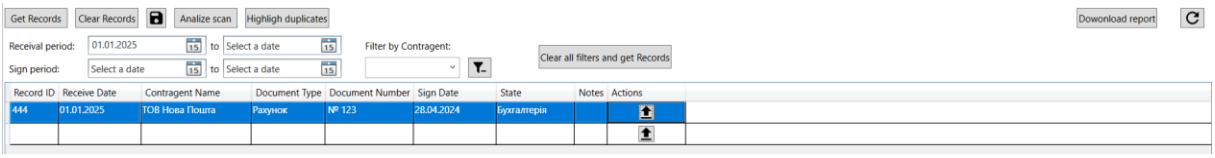



Рис. 4.1. Інтерфейс додавання нового документа

### *4.1.2. Розпізнавання документа з PDF (OCR)*

Користувач може завантажити скан документа, після чого активується функція розпізнавання.

Система автоматично:

- отримує текст з PDF-файлу;
- аналізує вміст та визначає тип документа, номер, дату, контрагента;
- створює нові записи з попередньо заповненими полями;
- зберігає фрагмент скану, де були знайдені відповідні дані.

#### *4.1.3. Перегляд, редагування та видалення документів.*

Усі документи відображаються у вигляді таблиці. Користувач може:

- фільтрувати документи за датою отримання, підписання, або контрагентом;
- очищувати таблицю перегляду;
- вносити зміни в існуючі записи;
- видаляти непотрібні або помилково додані документи;
- підсвічувати дублікати.

### 4.2. Автоматизоване розпізнавання вмісту документів

Однією з ключових функцій системи є автоматизоване розпізнавання тексту на документах за допомогою OCR (Optical Character Recognition). Це дає змогу значно пришвидшити заповнення полів документа шляхом витягування тексту безпосередньо зі скану.

#### *4.2.1. Розпізнавання тексту за допомогою API (Python + pytesseract).*

Було реалізовано RESTful API, який приймає PDF-файл від клієнта, обробляє його, витягує з нього текстову інформацію за допомогою бібліотеки pytesseract (рис. 4.2), яка виконує оптичне розпізнавання символів (OCR), а після завершення обробки формує і повертає клієнту відповідь у структурованому вигляді JSON, що містить розпізнаний текст.

Результат можна легко інтегрувати у зовнішні сервіси або бази даних, що значно спрощує процес цифровізації документообігу в організаціях.

```
12
13  class OCR:
14      def __init__(self, file: InMemoryUploadedFile):
15          self.file = file
16
17      def _get_image_from_file(self):
18          try:
19              image = Image.open(self.file)
20              return image
21          except Exception as e:
22              logger.error(f"Failed to get image from file. Details: {e}")
23
24      def _get_from_img_file(self) -> str:
25          try:
26              image = self._get_image_from_file()
27              return pytesseract.image_to_string(image, lang="ukr")
28          except Exception as e:
29              raise ValueError(f"Error in processing image: {e}")
30
31      def _get_from_pdf_file(self) -> list[str]:
32          with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as temp_file:
33              try:
34                  temp_file.write(self.file.read())
35                  temp_file_path = temp_file.name
36              except Exception as e:
37                  logger.error(f"Error in processing file. Details: {e} ")
38                  return []
39
40          try:
41              images = convert_from_path(temp_file_path)
42              extracted_text = [
43                  pytesseract.image_to_string(image, lang='ukr')
44                  for page_num, image in enumerate(images)
45              ]
46          finally:
47              os.remove(temp_file_path)
48
49          return extracted_text
50
51      def get_text(self) -> list[str]:
52          extension = os.path.splitext(self.file.name.lower())[1]
53
54          if extension == '.pdf':
55              return self._get_from_pdf_file()
56          elif extension in ('.png', '.jpg', '.jpeg'):
57              return [self._get_from_img_file()]
58          else:
59              raise ValueError("Unsupported file format")
```

Рис. 4.2. Фрагмент коду, який відповідає за розпізнавання тексту

### *4.2.2. Аналіз розпізнаного тексту на стороні клієнта.*

Отриманий JSON з текстом надсилається у Windows-додаток (клієнт), де відбувається його подальший аналіз. Для визначення ключових параметрів (тип документа, номер, дата, контрагент) використовуються регулярні вирази [11] (рис. 4.3).

```
2 references
public static string GetContragent(List<string> contragents, string page)
{
    string firstOccurringContragent = "";
    int earliestIndex = int.MaxValue;
    page = LatinToCyrillicNormalization(Regex.Replace(page, @"[""\s\r\n]+", ""));
    List<string> strippedContragents = RemoveContragentsPrefixes(contragents);

    foreach (var contragent in strippedContragents)
    {
        int index = page.IndexOf(LatinToCyrillicNormalization(contragent.Replace(" ", "")), StringComparison.OrdinalIgnoreCase);
        if (index != -1 && index < earliestIndex)
        {
            earliestIndex = index;
            firstOccurringContragent = contragent;
        }
    }

    return GetFullContragentName(firstOccurringContragent, contragents);
}
```

Рис. 4.3. Приклад аналізу розпізнаного тексту для визначення контрагента

Після завершення аналізу додаток автоматично створює записи з новими документами, які користувач може відредагувати та зберегти у потрібному вигляді. Це дозволяє значно скоротити час обробки документів і зменшити ймовірність помилок.

## 4.3. Формування звітів і експорт у форматі Excel

Функціонал формування звітів дозволяє користувачеві швидко отримати зведену інформацію у форматі Excel щодо записів, які наразі відображаються в додатку. Це особливо корисно для аналізу, архівування або подальшого обміну інформацією.

### *4.3.1. Експорт поточних записів у Excel.*

У додатку передбачена кнопка для завантаження звіту, яка формує Excel-файл на основі відфільтрованих даних, які в даний момент

відображаються у DataGrid. Це означає, що всі застосовані фільтри автоматично враховуються.

### *4.3.2. Архівування сканів документів*

Окрім Excel-звіту, користувач також отримує ZIP-архів, що містить усі скани документів, прив’язані до відібраних записів. Ці файли організовуються за визначеною ієрархічною структурою, яка дозволяє легко знайти необхідний документ за датою та контрагентом (рис. 4.4).

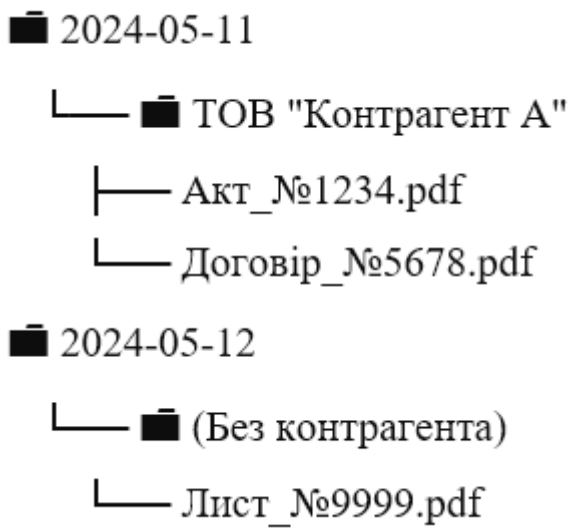



Рис. 4.4. Приклад ієрархічної структури архівованих сканів

## 4.4. Висновок до розділу

У цьому розділі було описано реалізацію основних функціональних можливостей програмного забезпечення DocRegister, що забезпечують ефективну роботу з документами. Реалізація реєстрації та обліку документів дозволяє користувачам створювати, редагувати, фільтрувати та видаляти записи, а також додавати скановані копії документів, що забезпечує централізоване зберігання і легкий доступ до інформації.

Автоматизоване розпізнавання тексту (OCR), реалізоване через REST API на Python з використанням бібліотеки pytesseract, значно оптимізує процес введення даних, дозволяючи системі автоматично

ідентифікувати тип документа, номер, дату та контрагента на основі завантажених сканів. Подальший аналіз тексту на стороні клієнта із застосуванням регулярних виразів підвищує точність автоматичного заповнення полів, зменшуючи кількість ручної роботи та помилок.

Функціонал формування звітів і експорту у форматі Excel дає змогу користувачам швидко отримувати зведені дані про документи з урахуванням застосованих фільтрів, а архівування сканів у ZIP-форматі із збереженням ієрархічної структури забезпечує зручний доступ і організацію збережених файлів.

Крім основних функцій, у DocRegister реалізовано низку додаткових можливостей, які сприяють покращенню користувацького досвіду. Наприклад, система підтримує візуальне виділення дублікатів документів, що дозволяє оперативно виявляти і усувати помилки або зайві записи. Це особливо важливо при роботі з великими обсягами даних, де ймовірність дублювання зростає. Також реалізована можливість сортування та групування записів за різними критеріями, такими як дата отримання, підписання, чи статус документа, що забезпечує більш гнучке управління інформацією.

Таким чином, реалізовані функції забезпечують комплексний, зручний та автоматизований підхід до управління документами, що значно підвищує продуктивність роботи користувачів і надійність збереження інформації.

# ВИСНОВКИ

У рамках розробки програмного забезпечення DocRegister було обрано поєднання сучасних технологій, що забезпечують надійність, продуктивність і зручність у використанні. Клієнтська частина створена на базі WPF, що дозволяє розробити інтуїтивно зрозумілий інтерфейс з підтримкою гнучкої роботи з даними, а серверна логіка реалізована на Python з використанням бібліотеки pytesseract для оптичного розпізнавання тексту (OCR). Такий підхід дав змогу створити ефективний інструмент для автоматизованого обліку та обробки документів, що відповідає вимогам сучасного бізнес-середовища.

Основна функціональність системи охоплює кілька важливих напрямів. По-перше, передбачено можливість ручного додавання та редагування документів із заповненням ключових полів: дати отримання, контрагента, типу документа, номера, дати підписання, стану та нотаток. Ця функція забезпечує користувачам гнучкість і контроль над інформацією. По-друге, інтеграція OCR-модуля дозволяє автоматично отримувати текст із PDF-файлів, що значно прискорює роботу, знижує ризик помилок і мінімізує необхідність ручного вводу. Розпізнаний текст надходить у вигляді структурованого JSON, який далі аналізується у клієнтському додатку за допомогою регулярних виразів для виділення ключових параметрів – типу документа, номеру, дати та контрагента.

Інтерфейс програми містить зручну таблицю для відображення усіх зареєстрованих документів із можливістю сортування, фільтрації за датами, контрагентами та іншими параметрами. Користувач може швидко знаходити потрібні записи, редагувати інформацію, а також видаляти зайві або некоректні документи. Додатково реалізовано функцію підсвічування дублікатів, що допомагає підтримувати чистоту бази даних і уникати плутанини.

Ще одним важливим компонентом системи є формування звітів у форматі Excel. Завдяки можливості експорту відфільтрованих даних користувач отримує зручний інструмент для аналізу, обміну інформацією та архівування. Паралельно формується ZIP-архів зі сканами документів, організованими за зрозумілою ієрархією, що базується на датах та контрагентах. Це робить архівування зручним і дозволяє швидко знаходити необхідні файли.

Інтеграція модуля OCR з клієнтською частиною суттєво підвищує продуктивність обробки документів, автоматизуючи заповнення основних полів і зменшуючи навантаження на користувача. Також це допомагає мінімізувати людські помилки, пов’язані з ручним введенням даних, що особливо важливо при роботі з великою кількістю первинної документації. Аналіз розпізнаного тексту з використанням регулярних виразів дозволяє адаптувати систему під різні формати документів, підвищуючи гнучкість і точність обробки.

Таким чином, DocRegister являє собою комплексне рішення для автоматизації документообігу, що поєднує в собі інтуїтивний інтерфейс, ефективну обробку даних та розширені можливості звітності. Це сприяє оптимізації робочих процесів, покращенню контролю за документами та підвищенню загальної продуктивності роботи організації. В перспективі передбачено розширення функціоналу шляхом інтеграції з іншими системами управління бізнес-процесами, що дозволить зробити DocRegister ще більш універсальним і корисним інструментом для користувачів.

## СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. M-Files | Document Management System [Електронний ресурс]. – Режим доступу: https://www.m-files.com. – Дата доступу: 11.02.2025 р.
2. DocuWare | Document Management System [Електронний ресурс]. – Режим доступу: https://start.docuware.com. – Дата доступу: 11.02.2025 р.
3. OnlyOffice Docs | Document Management System [Електронний ресурс]. – Режим доступу: https://www.onlyoffice.com. – Дата доступу: 11.02.2025 р.
4. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools [Електронний ресурс]. – Режим доступу: https://opencv.org/about. – Дата доступу: 16.02.2025 р.
5. Smith, R. (2007). An overview of the Tesseract OCR engine. Proceedings of the Ninth International Conference on Document Analysis and Recognition, с. 629–633. [Електронний ресурс] – Режим досутупу: https://ieeexplore.ieee.org/document/4376991. – Дата доступу: 22.02.2025 р.
6. van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace. – Дата доступу: 02.03.2025 р.
7. Beazley, D. M. (2009). Python Essential Reference (4th ed.). Addison-Wesley. – Дата доступу: 05.03.2025 р.
8. Troelsen, A., & Japikse, P. (2021). Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming. Apress. – Дата доступу: 07.03.2025 р.
9. Microsoft. (2023). WPF Overview. Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/introduction-to-wpf. – Дата доступу: 11.03.2025 р.

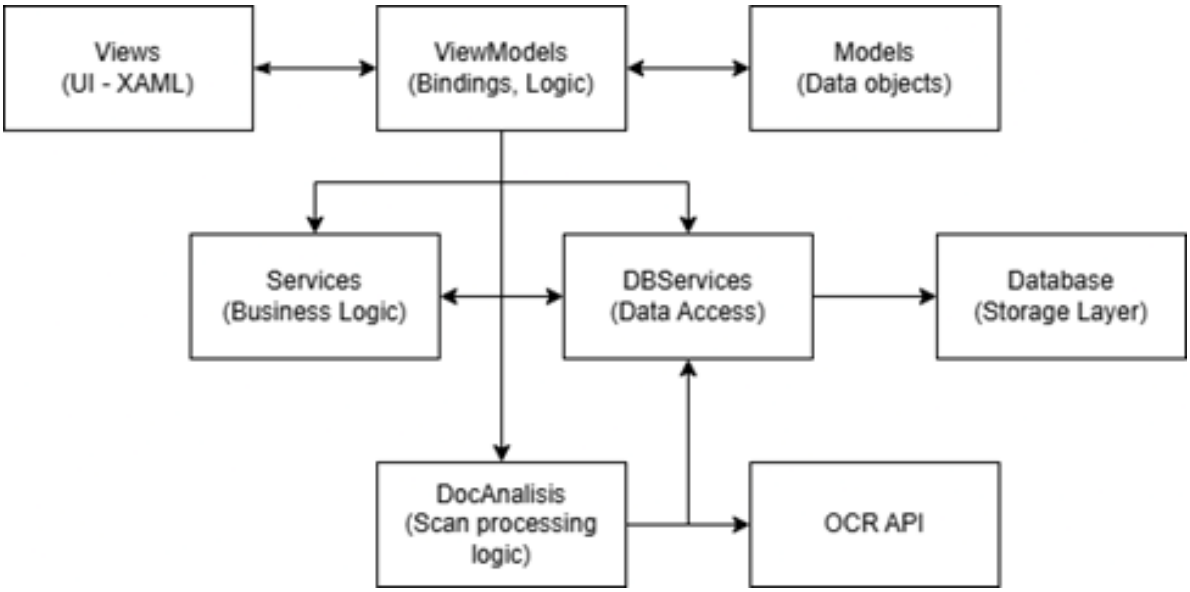
10. Freeman, A., & MacDonald, A. (2018). Pro WPF in C# 2010: Windows Presentation Foundation in .NET 4. Apress. – Дата доступу: 14.03.2025 р.
11. McFedries, P. (2013). Regular Expressions Pocket Reference (2nd ed.). O'Reilly Media. – Дата доступу: 17.03.2025 р.
12. PostgreSQL | СУБД [Електронний ресурс]. – Режим доступу: https://www.postgresql.org. – Дата доступу: 19.03.2025 р.
13. Date, C. J. (2003). An Introduction to Database Systems (8th ed.). Addison-Wesley. – Дата доступу: 20.03.2025 р.
14. SQLite | СУБД [Електронний ресурс]. – Режим доступу: https://sqlite.org. – Дата доступу: 21.03.2025 р.
15. MySQL | СУБД [Електронний ресурс]. – Режим доступу: https://www.mysql.com. – Дата доступу: 21.03.2025 р.
16. Oracle Database | СУБД [Електронний ресурс]. – Режим доступу: https://www.oracle.com/database. – Дата доступу: 21.03.2025 р.
17. Microsoft SQL Server | СУБД [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/en-us/sql-server. – Дата доступу: 22.03.2025 р.
18. Node.js | Середовище виконання JavaScript [Електронний ресурс]. – Режим доступу: https://nodejs.org. – Дата доступу: 22.03.2025 р.
19. Express | Вебфреймворк для Node.js [Електронний ресурс]. – Режим доступу: https://expressjs.com. – Дата доступу: 22.03.2025 р.
20. Java | Мова програмування [Електронний ресурс]. – Режим доступу: https://www.java.com. – Дата доступу: 22.03.2025 р.
21. Spring Boot | Вебфреймворк на Java [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-boot. – Дата доступу: 23.03.2025 р.
22. Golang | Мова програмування [Електронний ресурс]. – Режим доступу: https://go.dev. – Дата доступу: 24.03.2025 р.

23. .NET Core | Вебфреймворк на C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet. – Дата доступу: 24.03.2025 р.
24. Amazon Web Services | Провайдер хмарних технологій [Електронний ресурс]. – Режим доступу: https://aws.amazon.com. – Дата доступу: 25.03.2025 р.

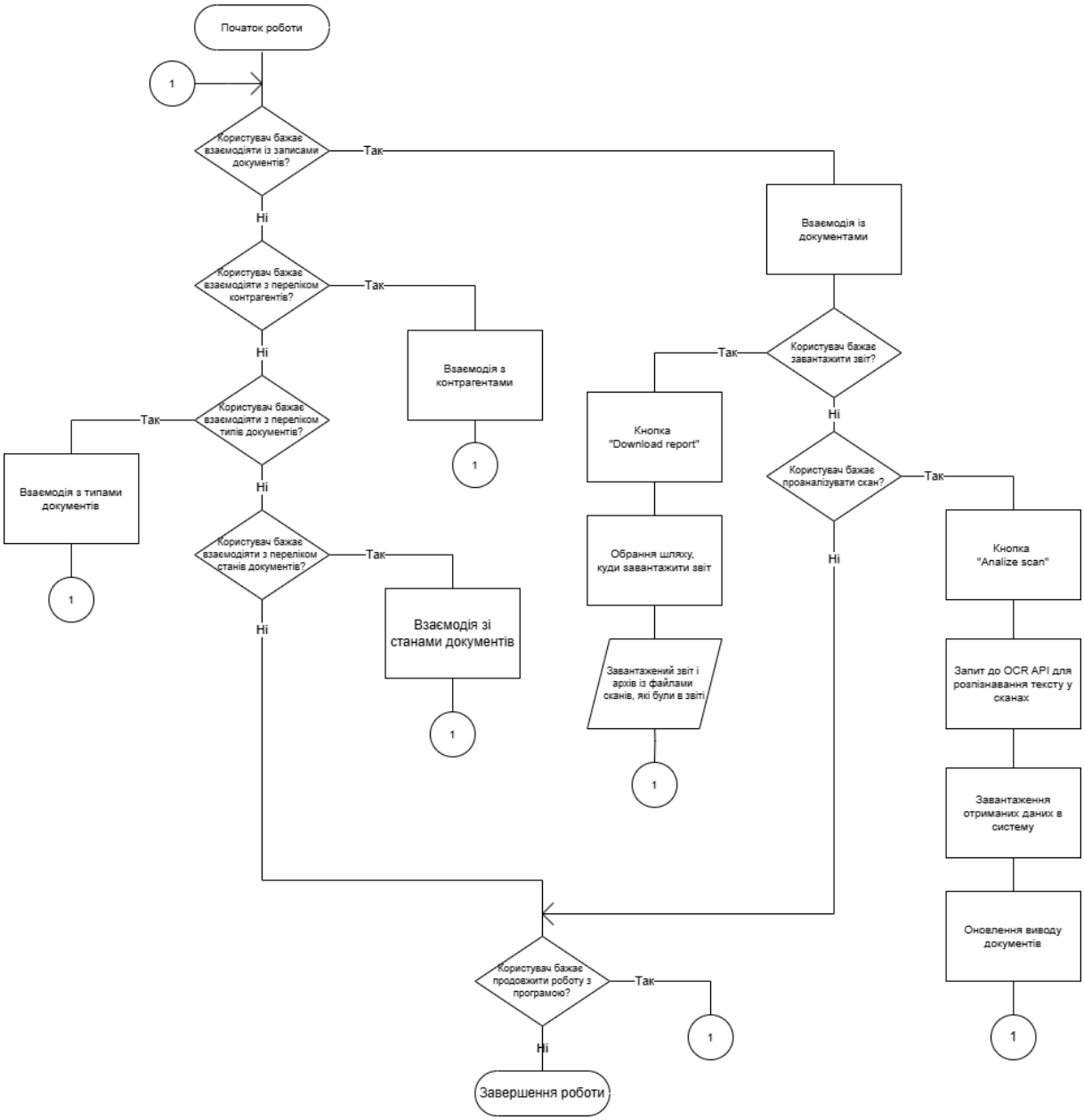
**ДОДАТКИ**

**Додаток 1**

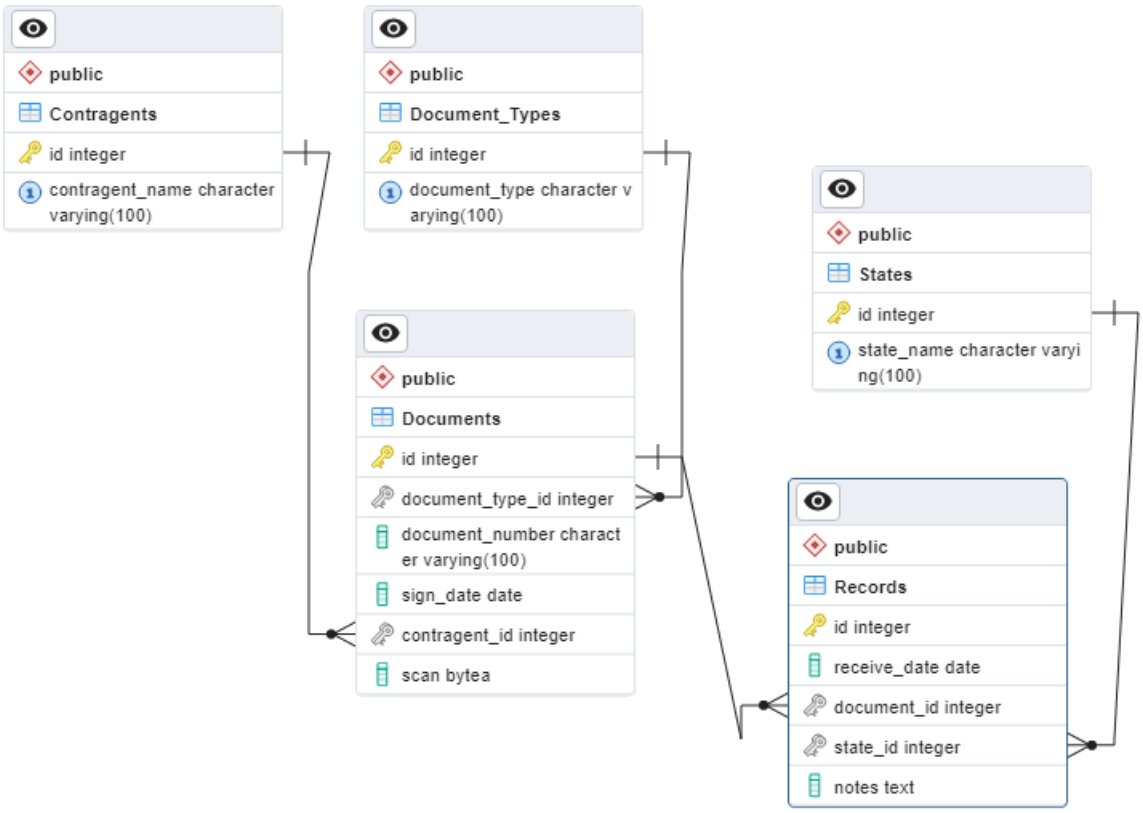
**Копії графічних матеріалів**




ДП.045490-06-99. Програмне забезпечення для автоматизації реєстрації паперових документів підприємства. Структура клієнтського додатка. Структурна схема

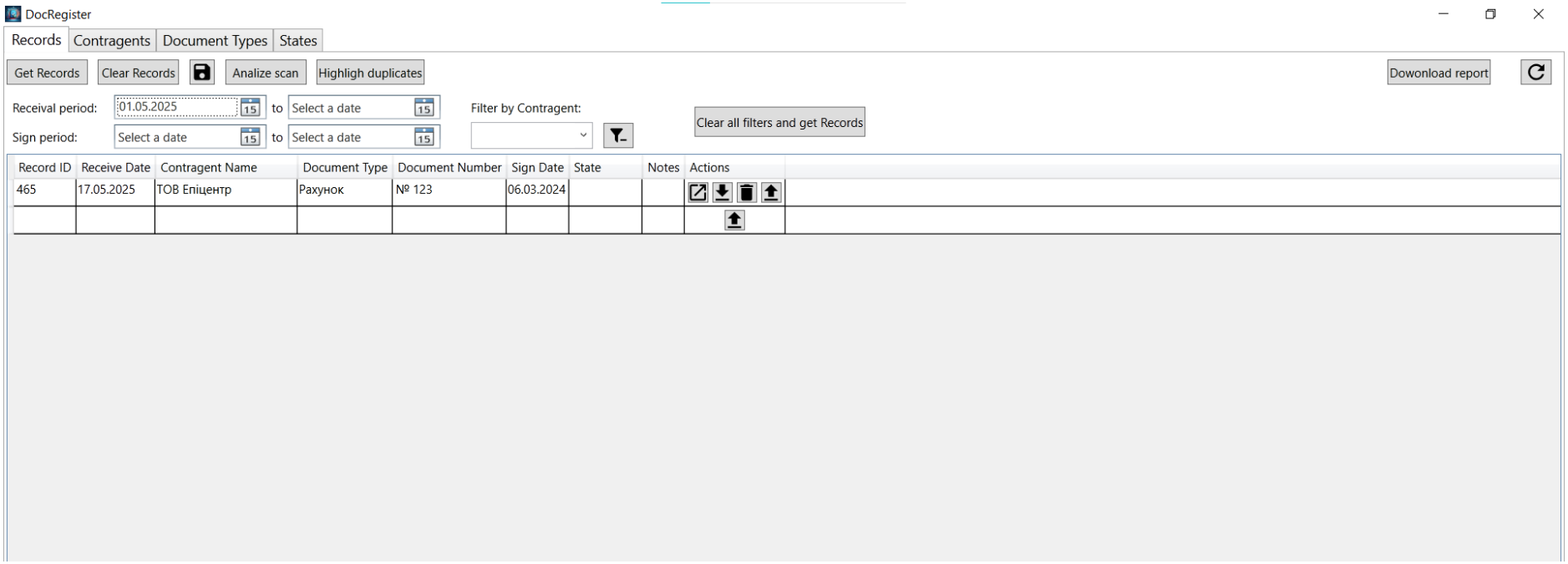



ДП.045490-07-99. Програмне забезпечення для автоматизації реєстрації паперових документів підприємства. Проєктування дизайну користувацького інтерфейсу. Схема алгоритму

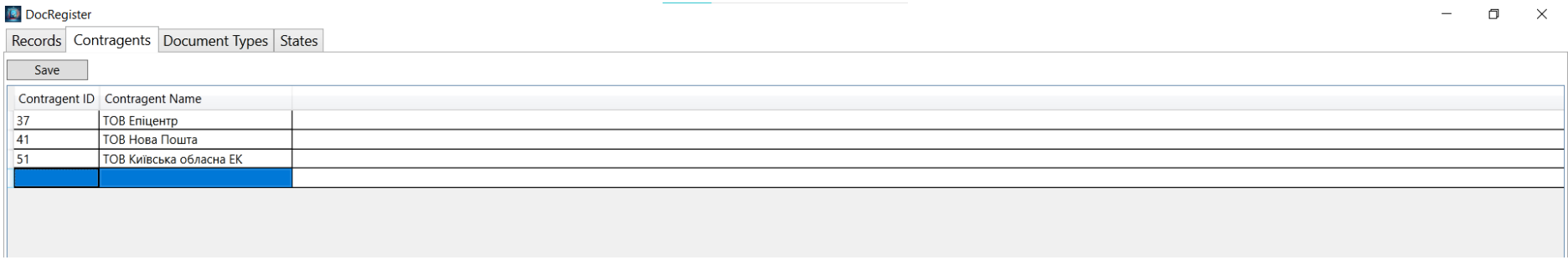



ДП.045490-08-99. Програмне забезпечення для автоматизації реєстрації паперових документів підприємства. Схема бази даних. ERD-діаграма

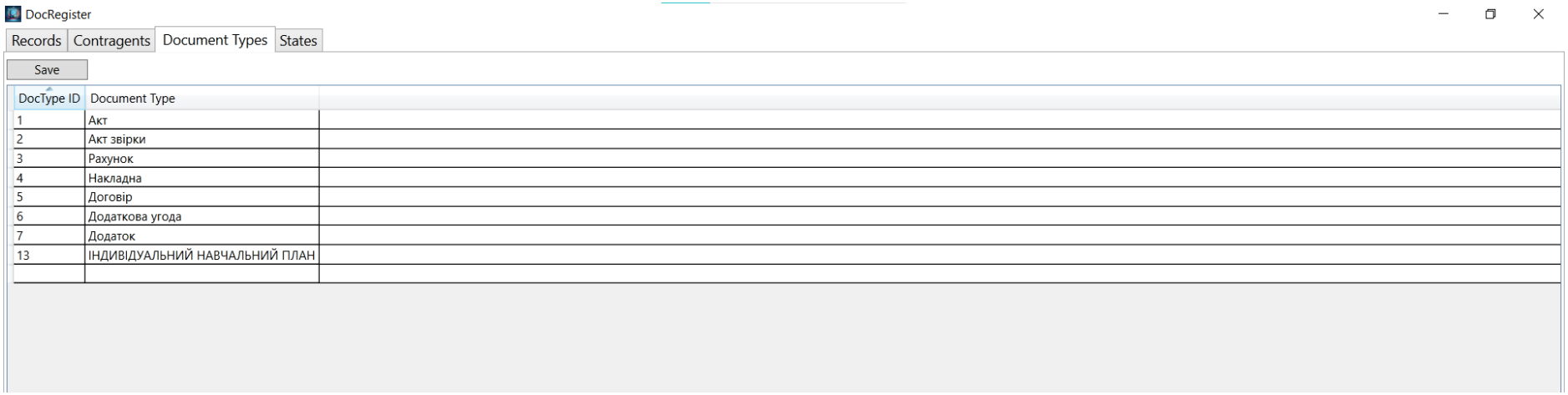

ДП.045490-09-99. Програмне забезпечення для автоматизації реєстрації паперових документів підприємства. Діаграма прецендентів. Схема алгоритму



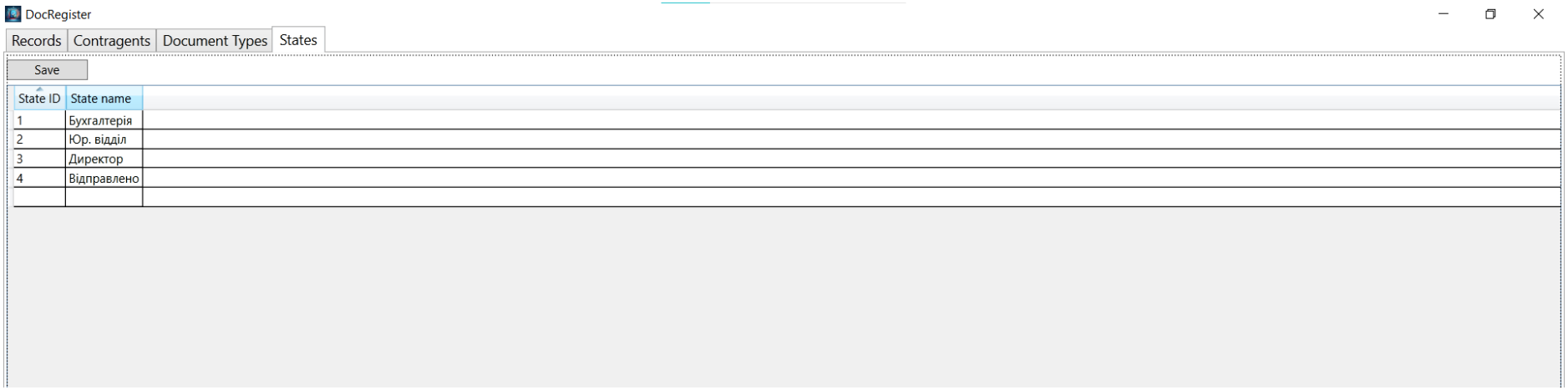

Вкладка із записами документів системи DocRegister
Защик І.О., група КП-11




Вкладка із контрагентами системи DocRegister
Защик І.О., група КП-11




Вкладка із типами документів системи DocRegister
Защик І.О., група КП-11




Вкладка із станами системи DocRegister
Защик І.О., група КП-11

## Додаток 2

## Лістинг програми

## Код з оптичним розпізнаванням символів (OCR):

```
import logging
import os
import tempfile
import pytesseract
from pdf2image import convert_from_path
from django.core.files.uploadedfile import InMemoryUploadedFile
from PIL import Image


logger = logging.getLogger(__name__)


class OCR:
    def __init__(self, file: InMemoryUploadedFile):
        self.file = file

    def _get_image_from_file(self):
        try:
            image = Image.open(self.file)
            return image
        except Exception as e:
            logger.error(f"Failed to get image from file. Details: {e}")

    def _get_from_img_file(self) -> str:
        try:
            image = self._get_image_from_file()
            return pytesseract.image_to_string(image, lang="ukr")
        except Exception as e:
            raise ValueError(f"Error in processing image: {e}")

    def _get_from_pdf_file(self) -> list[str]:
        with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as
temp_file:
            try:
                temp_file.write(self.file.read())
                temp_file_path = temp_file.name
            except Exception as e:
                logger.error(f"Error in processing file. Details: {e} ")
                return []

        try:
            images = convert_from_path(temp_file_path)
            extracted_text = [
                pytesseract.image_to_string(image, lang='ukr')
                for page_num, image in enumerate(images)
            ]
        finally:
            os.remove(temp_file_path)

        return extracted_text

    def get_text(self) -> list[str]:
        extension = os.path.splitext(self.file.name.lower())[1]

        if extension == '.pdf':
            return self._get_from_pdf_file()
        elif extension in ('.png', '.jpg', '.jpeg'):
            return [self._get_from_img_file()]
        else:
            raise ValueError("Unsupported file format"
```

## Код з аналізом розпізнаного тексту:

```
using System.IO;
using System.Text;
using System.Text.RegularExpressions;

namespace DocumentFlow.DocAnalisis
{
    public class TextAnalizer
    {
        private static List<string> _contragentsPrefixes = new()
        {
            "ТОВ", "Товариство з обмеженою відповідальністю",
            "ПрАТ", "Приватне акціонерне товариство",
            "ППІ"
        };

        private static List<string> _months = new()
        {
            "Січня", "Лютого", "Березня", "Квітня", "Травня", "Червня",
            "Липня", "Серпня", "Вересня", "Жовтня", "Листопада", "Груденя"
        };

        private static string RemovePrefix(string text, List<string>
prefixes)
        {
            if (string.IsNullOrEmpty(text))
            {
                return text;
            }

            foreach (var prefix in prefixes)
            {
                if (text.StartsWith(prefix,
StringComparison.OrdinalIgnoreCase))
                {
                    return text.Substring(prefix.Length).Trim();
                }
            }

            return text;
        }

        private static List<string> RemoveContragentsPrefixes(List<string>
contragents)
        {
            List<string> res = new();

            foreach (var c in contragents)
            {
                res.Add(TextAnalizer.RemovePrefix(c,
_contragentsPrefixes));
            }

            return res;
        }

        private static string GetFullContragentName(string contragent,
List<string> contragents)
        {
            if (string.IsNullOrEmpty(contragent))
            {
                return "";
            }
```

```
   foreach (var c in contragents)
            {
                if (c.Contains(contragent))
                {
                    return c;
                }
            }

            return "";
        }

        private static string LatinToCyrillicNormalization(string input)
        {
            Dictionary<char, char> crossScriptMapping = new
Dictionary<char, char>
            {
                // Latin to Cyrillic mappings
                { 'A', 'А' }, { 'B', 'В' }, { 'C', 'С' }, { 'E', 'Е' },
                { 'H', 'Н' }, { 'I', 'І' }, { 'K', 'К' }, { 'M', 'М' },
                { 'O', 'О' }, { 'P', 'Р' }, { 'T', 'Т' }, { 'X', 'Ч' },
                { 'Y', 'У' }, { 'L', 'Л' },
                { 'a', 'а' }, { 'c', 'с' }, { 'e', 'е' },
                { 'i', 'і' }, { 'o', 'о' }, { 'p', 'р' },
                { 'x', 'х' }, { 'y', 'у' }, { 'l', 'л' }
            };

            var normalized = new StringBuilder(input.Length);
            foreach (var c in input)
            {
                normalized.Append(crossScriptMapping.ContainsKey(c) ?
crossScriptMapping[c] : c);
            }

            return normalized.ToString();
        }


        public static string GetContragent(List<string> contragents, string
page)
        {
            string firstOccurringContragent = "";
            int earliestIndex = int.MaxValue;
            page = LatinToCyrillicNormalization(Regex.Replace(page,
@"[""\s\r\n]+", ""));
            List<string> strippedContragents =
RemoveContragentsPrefixes(contragents);

            foreach (var contragent in strippedContragents)
            {
                int index =
page.IndexOf(LatinToCyrillicNormalization(contragent.Replace(" ", "")),
StringComparison.OrdinalIgnoreCase);
                if (index != -1 && index < earliestIndex)
                {
                    earliestIndex = index;
                    firstOccurringContragent = contragent;
                }
            }

            return GetFullContragentName(firstOccurringContragent,
contragents);
        }
```

```
        public static string GetDocumentType(List<string> docTypes, string
page)
        {
            string firstOccurringDocType = "";
            int earliestIndex = int.MaxValue;
            List<string> foundDocTypes = new();

            foreach (var dt in docTypes)
            {
                int index = page.IndexOf(dt,
StringComparison.OrdinalIgnoreCase);
                if (index != -1 && index < earliestIndex)
                {
                    earliestIndex = index;
                    firstOccurringDocType = dt;
                    foundDocTypes.Clear();
                    foundDocTypes.Add(dt);
                }
                else if (index == earliestIndex)
                {
                    foundDocTypes.Add(dt);
                }
            }

            if (foundDocTypes.Count > 1)
            {
                firstOccurringDocType = foundDocTypes.OrderByDescending(dt
=> dt.Length).First();
            }

            return firstOccurringDocType;
        }

        public static string GetDocumentNumber(string page)
        {
            string pattern = @"(?:№|Mo)\s*\d+";

            Match match = Regex.Match(page, pattern,
RegexOptions.IgnoreCase);
            if (match.Success)
            {
                return match.Value;
            }
            else
            {
                return "";
            }
        }

        public static string GetDocumentSignDate(string page)
        {
            string monthPattern = string.Join("|", _months);
            string pattern =
$@"(?:\d{{1,2}}\s+({monthPattern})\s+\d{{4}}|\d{{1,2}}[./-]\d{{1,2}}[./-
]\d{{4}})";

            Match match = Regex.Match(page, pattern,
RegexOptions.IgnoreCase);
            if (match.Success)
            {
                return match.Value;
            }
            else
            {
```

```
                return "";
            }
        }
    }
}
```

### Код з генерацією Excel-звітів:

```
using ClosedXML.Excel;
using DocumentFlow.ViewModels.ModelWrappers;
using System.IO;
using System.IO.Compression;
using System.Windows.Forms;

namespace DocumentFlow.Services
{
    public class ExcelService
    {
        public static void DownloadReport(List<RecordWrapper>? records)
        {
            if (records == null || records.Count == 0)
            {
                System.Windows.MessageBox.Show("There must be some data to
extract.");
                return;
            }

            try
            {
                using (var folderDialog = new FolderBrowserDialog())
                {
                    folderDialog.Description = "Select Folder to Save
Files";

                    if (folderDialog.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
                    {
                        string selectedFolderPath =
folderDialog.SelectedPath;
                        string excelFilePath =
Path.Combine(selectedFolderPath, "Report.xlsx");
                        string zipFilePath =
Path.Combine(selectedFolderPath, "Report.zip");

                        using (var workbook = new XLWorkbook())
                        {
                            var worksheet =
workbook.Worksheets.Add("Report");

                            worksheet.Cell(1, 1).Value = "Receive Date";
                            worksheet.Cell(1, 2).Value = "Contragent Name";
                            worksheet.Cell(1, 3).Value = "Document Type";
                            worksheet.Cell(1, 4).Value = "Document Number";
                            worksheet.Cell(1, 5).Value = "Sign Date";
                            worksheet.Cell(1, 6).Value = "State";
                            worksheet.Cell(1, 7).Value = "Notes";

                            int row = 2;

                            foreach (var record in records)
                            {
                                worksheet.Cell(row, 1).Value =
record.ReceiveDate?.ToString("dd.MM.yyyy");
```

```
                                worksheet.Cell(row, 2).Value =
record.ContragentName;
                                worksheet.Cell(row, 3).Value =
record.DocumentType;
                                worksheet.Cell(row, 4).Value =
record.DocumentNumber;
                                worksheet.Cell(row, 5).Value =
record.SignDate?.ToString("dd.MM.yyyy");
                                worksheet.Cell(row, 6).Value =
record.State;
                                worksheet.Cell(row, 7).Value =
record.Notes;

                                row++;
                            }

                            worksheet.Columns().AdjustToContents();

                            var range = worksheet.Range($"A1:G{row - 1}");
                            var table = range.CreateTable();
                            table.Name = "ReportTable";
                            table.Theme = XLTableTheme.TableStyleMedium9;

                            workbook.SaveAs(excelFilePath);
                        }

                        using (var zipArchive = ZipFile.Open(zipFilePath,
ZipArchiveMode.Create))
                        {
                            foreach (var record in records)
                            {
                                if (record.Scan != null &&
record.ReceiveDate != null)
                                {
                                    string contragentName =
record.ContragentName ?? "NoContragent";
                                    string documentType =
record.DocumentType ?? "";
                                    string documentNumber =
record.DocumentNumber ?? "";

                                    string fileName =
$"{contragentName}_{documentType}_{documentNumber.Replace(" ", "")}.pdf";

                                    string dateFolder =
record.ReceiveDate.Value.ToString("dd-MM-yyyy");
                                    string contragentFolder =
Path.Combine(dateFolder, contragentName);

                                    var zipEntry =
zipArchive.CreateEntry(Path.Combine(contragentFolder, fileName));
                                    using (var zipStream = zipEntry.Open())
                                    {
                                        zipStream.Write(record.Scan, 0,
record.Scan.Length);
                                    }
                                }
                            }
                        }

                        MessageBox.Show($"Report generated:\nExcel file:
{excelFilePath}\nZIP file: {zipFilePath}");
                    }
                }
```

```
            }
            catch (Exception ex)
            {
                System.Windows.MessageBox.Show($"Error generating report:
{ex.Message}");
            }
        }
    }
```

## Код з фільтрацією записів:

```
using DocumentFlow.ViewModels.ModelWrappers;

namespace DocumentFlow.Services
{
    public class FiltrationService
    {
        public static List<RecordWrapper>?
FilterByStartReceivalDate(List<RecordWrapper> records, DateTime date)
        {
            return records
                ?.Where(record => record.ReceiveDate.HasValue &&
                        record.ReceiveDate >= date)
                .ToList();
        }

        public static List<RecordWrapper>?
FilterByEndReceivalDate(List<RecordWrapper> records, DateTime date)
        {
            return records
                .Where(record => record.ReceiveDate.HasValue &&
                        record.ReceiveDate <= date)
                .ToList();
        }

        public static List<RecordWrapper>?
FilterByStartSignDate(List<RecordWrapper> records, DateTime date)
        {
            return records
                .Where(record => record.SignDate.HasValue &&
                        record.SignDate >= date)
                .ToList();
        }

        public static List<RecordWrapper>?
FilterByEndSignDate(List<RecordWrapper> records, DateTime date)
        {
            return records
                .Where(record => record.SignDate.HasValue &&
                        record.SignDate <= date)
                .ToList();
        }

        public static List<RecordWrapper>?
FilterByContragent(List<RecordWrapper> records, string contragent)
        {
            return records
                .Where(record => record.ContragentName != null &&
                        record.ContragentName.Contains(contragent))
                .ToList();
        }
    }
}
```

## Код RecordsViewModel.cs:

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
using DocumentFlow.DocAnalisis.AnalisisControllers;
using DocumentFlow.Models;
using DocumentFlow.Services;
using DocumentFlow.Services.DBServices;
using DocumentFlow.ViewModels.ModelWrappers;
using Microsoft.Win32;
using System.Collections.ObjectModel;
using System.Diagnostics;
using System.IO;
using System.Windows;
using System.Windows.Media;

namespace DocumentFlow.ViewModels
{
    public partial class RecordsViewModel : BaseViewModel
    {
        #region Data handling
        public DBServiceRecords? RecordsService;
        public DBServiceContragents? ContragentsService;
        public DBServiceDocumentTypes? DocTypesService;
        public DBServiceStates? StatesService;
        public BaseAnalisisController? AnalisisController;
        #endregion

        #region Undo
        private List<RecordWrapper> _deletedRecords { get; set; } = new
List<RecordWrapper>();
        private int _deletedRecordsLimit;
        #endregion

        #region UI data
        public ObservableCollection<RecordWrapper>?
CurrentRecordsCollection { get; set; }
        private List<RecordWrapper>? AllRecords { get; set; }
        public ObservableCollection<ContragentWrapper>? Contragents { get;
set; }
        public ObservableCollection<DocumentTypeWrapper>? DocumentTypes {
get; set; }
        public ObservableCollection<StateWrapper>? States { get; set; }
        public List<string?> ContragentsInCombobox = new();
        public List<string?> DocTypesInCombobox = new();
        public List<string?> StatesInCombobox = new();
        #endregion

        #region Filters
        private bool _isAnyFilterApplied
        {
            get => (StartReceivalDateFilter != null ||
EndReceivalDateFilter != null ||
                    StartSignDateFilter != null || EndSignDateFilter != null
||
                    !string.IsNullOrEmpty(ContragentNameFilter));
        }

        [ObservableProperty]
        private DateTime? _startReceivalDateFilter;
        partial void OnStartReceivalDateFilterChanged(DateTime? value)
        {
            if (value != null && CurrentRecordsCollection != null)
            {
```

```
                var updatedRecords = ConversionService
                    .ToRecordWrappersCollection(FiltrationService

.FilterByStartReceivalDate(CurrentRecordsCollection.ToList(),
(DateTime)value));
                if (updatedRecords != null)
RefreshDataGrid(updatedRecords);
            }
            else
            {
                GetRecords();
            }
        }

        [ObservableProperty]
        private DateTime? _endReceivalDateFilter;
        partial void OnEndReceivalDateFilterChanged(DateTime? value)
        {
            if (value != null && CurrentRecordsCollection != null)
            {
                var updatedRecords = ConversionService
                    .ToRecordWrappersCollection(FiltrationService

.FilterByEndReceivalDate(CurrentRecordsCollection.ToList(),
(DateTime)value));
                if (updatedRecords != null)
RefreshDataGrid(updatedRecords);
            }
            else
            {
                GetRecords();
            }
        }

        [ObservableProperty]
        private DateTime? _startSignDateFilter;
        partial void OnStartSignDateFilterChanged(DateTime? value)
        {
            if (value != null && CurrentRecordsCollection != null)
            {
                var updatedRecords = ConversionService
                    .ToRecordWrappersCollection(FiltrationService

.FilterByStartSignDate(CurrentRecordsCollection.ToList(),
(DateTime)value));
                if (updatedRecords != null)
RefreshDataGrid(updatedRecords);
            }
            else
            {
                GetRecords();
            }
        }

        [ObservableProperty]
        private DateTime? _endSignDateFilter;
        partial void OnEndSignDateFilterChanged(DateTime? value)
        {
            if (value != null && CurrentRecordsCollection != null)
            {
                var updatedRecords = ConversionService
                    .ToRecordWrappersCollection(FiltrationService

.FilterByEndSignDate(CurrentRecordsCollection.ToList(), (DateTime)value));
```

```
                if (updatedRecords != null)
RefreshDataGrid(updatedRecords);
            }
            else
            {
                GetRecords();
            }
        }

        [ObservableProperty]
        private string? _contragentNameFilter;
        partial void OnContragentNameFilterChanged(string? value)
        {
            if (!string.IsNullOrEmpty(value) && CurrentRecordsCollection !=
null)
            {
                var updatedRecords = ConversionService
                    .ToRecordWrappersCollection(FiltrationService

.FilterByContragent(CurrentRecordsCollection.ToList(), value));
                if (updatedRecords != null)
RefreshDataGrid(updatedRecords);
            }
            else
            {
                GetRecords();
            }
        }
        #endregion

        public RecordsViewModel()
        {
            SetupDataHandlers();
            SetupCollections();
        }

        protected override void SetupDataHandlers()
        {
            RecordsService = new DBServiceRecords();
            ContragentsService = new DBServiceContragents();
            DocTypesService = new DBServiceDocumentTypes();
            StatesService = new DBServiceStates();
            AnalisisController = new CustomAnalisisController();
        }

        protected void SetupCollections()
        {
            // Init Records
            var records = RecordsService?.ExecuteSelectionQuery();

            if (records != null)
            {
                AllRecords =
ConversionService.ToRecordWrappersList(records);
                CurrentRecordsCollection =
ConversionService.ToRecordWrappersCollection(AllRecords);

                var firstDayOfCurrentMonth = new
DateTime(DateTime.Now.Year, DateTime.Now.Month, 1);
                StartReceivalDateFilter = firstDayOfCurrentMonth;
                OnStartReceivalDateFilterChanged(firstDayOfCurrentMonth);

                if (CurrentRecordsCollection != null)
                {
```

```
                    CurrentRecordsCollection.CollectionChanged +=
Records_CollectionChanged;
                }
            }

            // Init Contragents
            var contragents = ContragentsService?.ExecuteSelectionQuery();
            if (contragents != null)
            {
                Contragents = new
ObservableCollection<ContragentWrapper>(contragents.Select(contragent =>
new ContragentWrapper(contragent)));
                ContragentsInCombobox = Contragents.Select(c =>
c.ContragentName).ToList();
            }

            // Init DocumentTypes
            var documentTypes = DocTypesService?.ExecuteSelectionQuery();
            if (documentTypes != null)
            {
                DocumentTypes = new
ObservableCollection<DocumentTypeWrapper>(documentTypes.Select(dt => new
DocumentTypeWrapper(dt)));
                DocTypesInCombobox = DocumentTypes.Select(dt =>
dt.DocumentType).ToList();
            }

            // Init States
            var states = StatesService?.ExecuteSelectionQuery();
            if (states != null)
            {
                States = new
ObservableCollection<StateWrapper>(states.Select(s => new
StateWrapper(s)));
                StatesInCombobox = States.Select(s => s.State).ToList();
            }

            string? stackLimit =
Environment.GetEnvironmentVariable("RECORDS_STACK_LIMIT");
            _deletedRecordsLimit = (stackLimit != null) ?
Int32.Parse(stackLimit) : 30;
        }

        [RelayCommand]
        private void OnSave()
        {
            if (CurrentRecordsCollection == null) { return; }

            var modifiedRecords = CurrentRecordsCollection.Where(record =>
record.IsModified).ToList();
            List<RecordWrapper> recordsForUpdate = new();
            List<RecordWrapper> recordsForInsert = new();
            List<RecordWrapper> recordsForDelete = new();

            foreach (var record in modifiedRecords)
            {
                if (record.RecordId != null)
                {
                    recordsForUpdate.Add(record);
                }
                else
                {
                    recordsForInsert.Add(record);
                }
```

```
            }

            // Handle update
            if (recordsForUpdate.Count > 0)
            {
                RecordsService?.ExecuteUpdateQuery(recordsForUpdate,
                                                   Contragents?.ToList(),
                                                   DocumentTypes?.ToList(),
                                                   States?.ToList());
            }

            // Handle insert
            if (recordsForInsert.Count > 0)
            {
                RecordsService?.ExecuteInsertQuery(recordsForInsert,
                                                   Contragents?.ToList(),
                                                   DocumentTypes?.ToList(),
                                                   States?.ToList());
            }

            // Handle delete
            if (CurrentRecordsCollection != null)
            {
                foreach (var deletedRecord in _deletedRecords)
                {
                    recordsForDelete.Add(deletedRecord);
                }

                if (recordsForDelete.Count > 0)
                {
                    RecordsService?.ExecuteDeleteQuery(recordsForDelete);
                }

                _deletedRecords.Clear();
            }

            GetRecords();
        }

        private void Records_CollectionChanged(object? sender,
System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
        {
            if (e.Action ==
System.Collections.Specialized.NotifyCollectionChangedAction.Remove)
            {
                foreach (var item in e.OldItems)
                {
                    if (item is RecordWrapper removedRecord)
                    {
                        _deletedRecords.Add(removedRecord);

                        if (_deletedRecords.Count > _deletedRecordsLimit)
                        {
                            var excessCount = _deletedRecords.Count -
_deletedRecordsLimit;
                            _deletedRecords.RemoveRange(0, excessCount);
                        }
                    }
                }
            }
        }

        [RelayCommand]
        private void Undo()
```

```
        {
            if (_deletedRecords.Any() && CurrentRecordsCollection != null)
            {
                var recordToRestore = _deletedRecords.Last();
                _deletedRecords.Remove(recordToRestore);

                CurrentRecordsCollection.Add(recordToRestore);
            }
        }

        [RelayCommand]
        private void HighlightDuplicateRecords()
        {
            if (CurrentRecordsCollection == null) return;

            var copiedRecords = new ObservableCollection<RecordWrapper>(
               CurrentRecordsCollection
                   .Where(record => record.RecordReference != null)
                   .Select(record => new
RecordWrapper(record.RecordReference)
                   {
                       IsDuplicate = false,
                       HighlightColor = Brushes.Transparent
                   }));

            var groups = copiedRecords
                .GroupBy(r => new { r.ContragentName, r.DocumentNumber,
r.SignDate })
                .Where(g => g.Count() > 1)
                .ToList();

            var colorGenerator = new Random();
            var colorMap = new Dictionary<IGrouping<object, RecordWrapper>,
SolidColorBrush>();

            foreach (var group in groups)
            {
                var color = new SolidColorBrush(Color.FromRgb(
                    (byte)colorGenerator.Next(100, 200),
                    (byte)colorGenerator.Next(100, 200),
                    (byte)colorGenerator.Next(100, 200)
                ));
                colorMap[group] = color;
            }

            foreach (var record in copiedRecords)
            {
                var group = groups.FirstOrDefault(g => g.Contains(record));
                if (group != null)
                {
                    record.IsDuplicate = true;
                    record.HighlightColor = colorMap[group];
                }
                else
                {
                    record.HighlightColor = Brushes.Transparent;
                }
            }

            RefreshDataGrid(copiedRecords);
        }

        public void RefreshDataGrid(ObservableCollection<RecordWrapper>
updatedRecordsCollection)
```

```
        {
            if (updatedRecordsCollection != null &&
CurrentRecordsCollection != null)
            {
                CurrentRecordsCollection.Clear();
                foreach (var record in updatedRecordsCollection)
                {
                    CurrentRecordsCollection.Add(record);
                }
            }
        }

        private void RefreshContragents()
        {
            if (Contragents == null) return;

            Contragents.Clear();
            ContragentsInCombobox.Clear();

            var updatedContragents =
ContragentsService?.ExecuteSelectionQuery();
            if (updatedContragents != null)
            {
                foreach (var contragent in updatedContragents)
                {
                    Contragents.Add(new ContragentWrapper(contragent));
                    ContragentsInCombobox.Add(contragent.ContragentName);
                }
            }
        }

        private void RefreshDocumentTypes()
        {
            if (DocumentTypes == null) return;

            DocumentTypes.Clear();
            DocTypesInCombobox.Clear();

            var updatedDocumentTypes =
DocTypesService?.ExecuteSelectionQuery();
            if (updatedDocumentTypes != null)
            {
                foreach (var dt in updatedDocumentTypes)
                {
                    DocumentTypes.Add(new DocumentTypeWrapper(dt));
                    DocTypesInCombobox.Add(dt.DocType);
                }
            }
        }

        private void RefreshStates()
        {
            if (States == null) return;

            States.Clear();
            StatesInCombobox.Clear();

            var updatedStates = StatesService?.ExecuteSelectionQuery();
            if (updatedStates != null)
            {
                foreach (var s in updatedStates)
                {
                    States.Add(new StateWrapper(s));
                    StatesInCombobox.Add(s.StateName);
```

```
                }
            }
        }

        [RelayCommand]
        private void OpenPdf(byte[] scanData)
        {
            if (scanData != null)
            {
                string tempFilePath = Path.Combine(Path.GetTempPath(),
$"{Guid.NewGuid()}.pdf");
                File.WriteAllBytes(tempFilePath, scanData);

                Process.Start(new ProcessStartInfo(tempFilePath) {
UseShellExecute = true });
            }
            else
            {
                MessageBox.Show("No PDF data available.");
            }
        }

        [RelayCommand]
        private void DownloadPdf(Record record)
        {
            if (record == null) return;

            if (record.DocumentFK.Item2.Scan != null)
            {
                string contragentName =
record?.DocumentFK?.Item2?.ContragentFK?.Item2.ContragentName ?? "";
                string documentType =
record?.DocumentFK?.Item2?.DocumentTypeFK?.Item2.DocType ?? "";
                string documentNumber =
record?.DocumentFK?.Item2.DocumentNumber ?? "";

                string fileName =
$"{contragentName}_{documentType}_{documentNumber.Replace(" ", "")}.pdf";
                SaveFileDialog saveFileDialog = new SaveFileDialog
                {
                    Filter = "PDF files (*.pdf)|*.pdf",
                    DefaultExt = "pdf",
                    FileName = fileName
                };

                bool? result = saveFileDialog.ShowDialog();

                if (result == true)
                {
                    string filePath = saveFileDialog.FileName;
                    File.WriteAllBytes(filePath,
record.DocumentFK.Item2.Scan);
                }
            }
            else
            {
                MessageBox.Show("No PDF data available to download.",
"Download Failed", MessageBoxButton.OK, MessageBoxImage.Warning);
            }
        }

        [RelayCommand]
        private void UploadScan(Record record)
        {
```

```
            if (record == null) return;

            OpenFileDialog openFileDialog = new OpenFileDialog
            {
                Filter = "PDF files (*.pdf)|*.pdf",
                DefaultExt = "pdf"
            };

            bool? result = openFileDialog.ShowDialog();

            if (result == true)
            {
                var documentService = new DBServiceDocuments();

                string filePath = openFileDialog.FileName;
                byte[] scanData = File.ReadAllBytes(filePath);

                record.DocumentFK.Item2.Scan = scanData;
                documentService.UpdateDocumentScan(record);

                GetRecords();
            }
        }

        [RelayCommand]
        private void DeleteScan(Record record)
        {
            if (record == null)
            {
                MessageBox.Show("No scan data available to delete.",
"Delete Failed", MessageBoxButton.OK, MessageBoxImage.Warning);
                return;
            }

            var documentService = new DBServiceDocuments();
            documentService.DeleteDocumentScan(record);

            GetRecords();
        }

        [RelayCommand]
        private async Task Analize()
        {
            if (AnalisisController != null)
            {
                await AnalisisController.Analize(ContragentsInCombobox,
                                        DocTypesInCombobox,
                                        Contragents?.ToList(),
                                        DocumentTypes?.ToList());
                GetRecords();
            }
        }

        [RelayCommand]
        private void GetRecords()
        {
            if (_isAnyFilterApplied)
            {
                var records = RecordsService?.ExecuteSelectionQuery();

                if (records != null)
                {
                    AllRecords =
ConversionService.ToRecordWrappersList(records);
```

```
                    CurrentRecordsCollection?.Clear();
                    foreach (var record in AllRecords)
                    {
                        CurrentRecordsCollection?.Add(record);
                    }

                    if (StartReceivalDateFilter != null)
OnStartReceivalDateFilterChanged(StartReceivalDateFilter);
                    if (EndReceivalDateFilter != null)
OnEndReceivalDateFilterChanged(EndReceivalDateFilter);
                    if (StartSignDateFilter != null)
OnStartSignDateFilterChanged(StartSignDateFilter);
                    if (EndSignDateFilter != null)
OnEndSignDateFilterChanged(EndSignDateFilter);
                    if (!string.IsNullOrEmpty(ContragentNameFilter))
OnContragentNameFilterChanged(ContragentNameFilter);
                }
            }
            else
            {
                var records = RecordsService?.ExecuteSelectionQuery();

                if (records != null)
                {
                    AllRecords =
ConversionService.ToRecordWrappersList(records);

RefreshDataGrid(ConversionService.ToRecordWrappersCollectionNotNullable((Li
st<RecordWrapper>)AllRecords));
                }
            }
        }

        [RelayCommand]
        private void Refresh()
        {
            GetRecords();
            RefreshContragents();
            RefreshDocumentTypes();
            RefreshStates();
        }

        [RelayCommand]
        private void ClearRecords()
        {
            CurrentRecordsCollection?.Clear();
        }

        [RelayCommand]
        private void ClearContragentFilter()
        {
            ContragentNameFilter = null;
            GetRecords();
        }

        [RelayCommand]
        private void ClearFilters()
        {
            StartReceivalDateFilter = null;
            EndReceivalDateFilter = null;
            StartSignDateFilter = null;
            EndSignDateFilter = null;
            ContragentNameFilter = null;
```

```
            GetRecords();
        }

        [RelayCommand]
        private void DownloadReport()
        {

ExcelService.DownloadReport(CurrentRecordsCollection?.ToList());
        }
    }
}
```

## Код StatesPage.xaml:

```
using DocumentFlow.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace DocumentFlow.Views
{
    /// <summary>
    /// Interaction logic for StatesPage.xaml
    /// </summary>
    public partial class StatesPage : Page
    {
        public StatesPage(StatesViewModel viewModel)
        {
            DataContext = viewModel;
            InitializeComponent();
        }

        private ScrollViewer GetScrollViewer(UIElement element)
        {
            if (element == null)
                return null;

            ScrollViewer retour = null;
            for (int i = 0; i < VisualTreeHelper.GetChildrenCount(element)
&& retour == null; i++)
            {
                if (VisualTreeHelper.GetChild(element, i) is ScrollViewer)
                {
                    retour =
(ScrollViewer)(VisualTreeHelper.GetChild(element, i));
                }
                else
                {
                    retour =
GetScrollViewer(VisualTreeHelper.GetChild(element, i) as UIElement);
                }
            }
```

```
            return retour;
        }

        private void DocumentStatesDataGrid_PreviewMouseWheel(object
sender, MouseWheelEventArgs e)
        {
            if (Keyboard.IsKeyDown(Key.LeftShift) ||
Keyboard.IsKeyDown(Key.RightShift))
            {
                var dataGrid = sender as DataGrid;
                if (dataGrid != null)
                {
                    ScrollViewer scrollViewer = GetScrollViewer(dataGrid);
                    if (scrollViewer != null)
                    {
                        if (e.Delta > 0)
                        {

scrollViewer.ScrollToHorizontalOffset(scrollViewer.HorizontalOffset - 20);
                        }
                        else
                        {

scrollViewer.ScrollToHorizontalOffset(scrollViewer.HorizontalOffset + 20);
                        }
                        e.Handled = true;
                    }
                }
            }
        }

        private void DocumentStatesDataGrid_PreviewKeyDown(object sender,
KeyEventArgs e)
        {
            if (e.Key == Key.Escape)
            {
                var dataGrid = sender as DataGrid;
                if (dataGrid != null)
                {
                    dataGrid.SelectedItems.Clear();
                    Keyboard.ClearFocus();
                }
                e.Handled = true;
            }
        }
    }
}
```

**Додаток 3**
**Копія презентації**

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

# ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕЄСТРАЦІЇ ПАПЕРОВИХ ДОКУМЕНТІВ ПІДПРИЄМСТВА

Виконав: Зашик Іван Олександрович

Керівник: доцент кафедри П З К С , к . т . н . доцент ,
Новак Дмитро Сергійович

Київ – 2025

# ПОСТАНОВКА ЗАДАЧІ

**Мета проекту:** створення зручного та доступного програмного рішення для автоматизації реєстрації, обробки та управління паперовим документообігом підприємства в єдиному інтерфейсі.

**Завдання:**

1. Проаналізувати ринок на предмет застосунків для електронного документообігу
2. Порівняти аналоги з розроблюваною системою
3. Розробити перелік вимог для системи
4. Обрати технологічні рішення для розробки програмного забезпечення
5. Розробити систему відповідно до переліку вимог та обраних технологій
6. Провести тестування готового продукту

# АКТУАЛЬНІСТЬ

- За даними дослідження AIIM (Association for Intelligent Information Management), понад 70% компаній у світі вказують зниження витрат і підвищення ефективності як основну мотивацію для переходу на електронний документообіг.
- Середній час на пошук одного документа в паперовому архіві складає 18 хвилин, згідно з даними IDC, тоді як в електронному середовищі цей час скорочується до менше 1 хвилини.
- 94% підприємств, які впровадили електронний документообіг, відзначили суттєве скорочення часу на погодження документів, за оцінками McKinsey & Company.
- До 30% документів у паперовому вигляді можуть бути втрачені або пошкоджені протягом життєвого циклу (джерело: Iron Mountain)

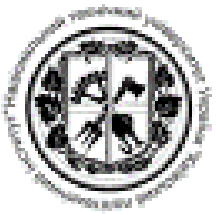
# АНАЛОГІЧНІ РІШЕННЯ


M-Files


OnlyOffice Docs


DocuWare

# ФУНКЦІОНАЛЬНІСТЬ РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



1. Додавання та редагування записів вручну.
2. Аналіз PDF документа за допомогою OCR та автоматична генерація записів і назв сканів.
3. Фільтрація документів за датою отримання, датою підписання та контрагентом.
4. Додавання та редагування контрагентів, типів документів, та їх станів
5. Генерація Excel-звіту та архівація сканів у зручному для пошуку фійлів форматі

# ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Клієнтська частина: C#, WPF
СУБД: PostgreSQL
OCR API: Python, Django, Tesseract



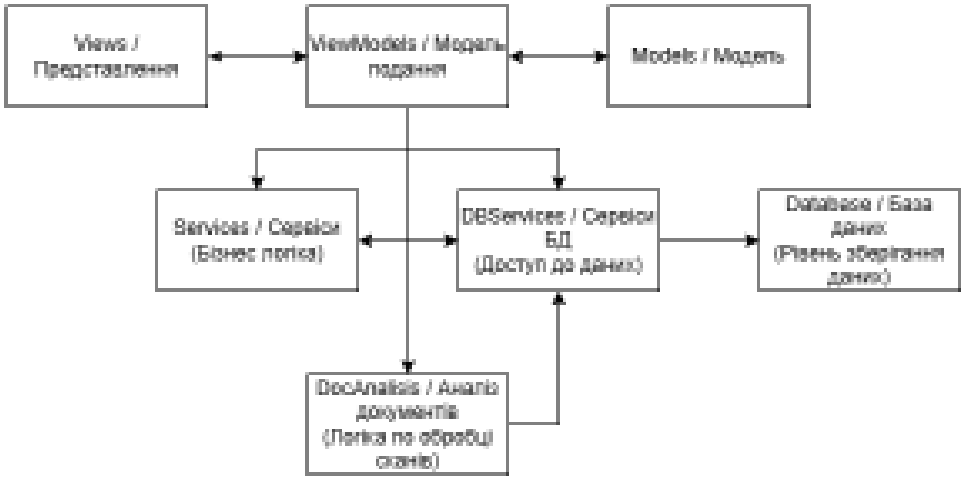

# АРХІТЕКТУРА ЗАСТОСУНКУ

## Загальна архітектура DocRegister

1. Клієнтська частина (настільний застосунок)
2. Вебсервер (OCR API)
3. База даних

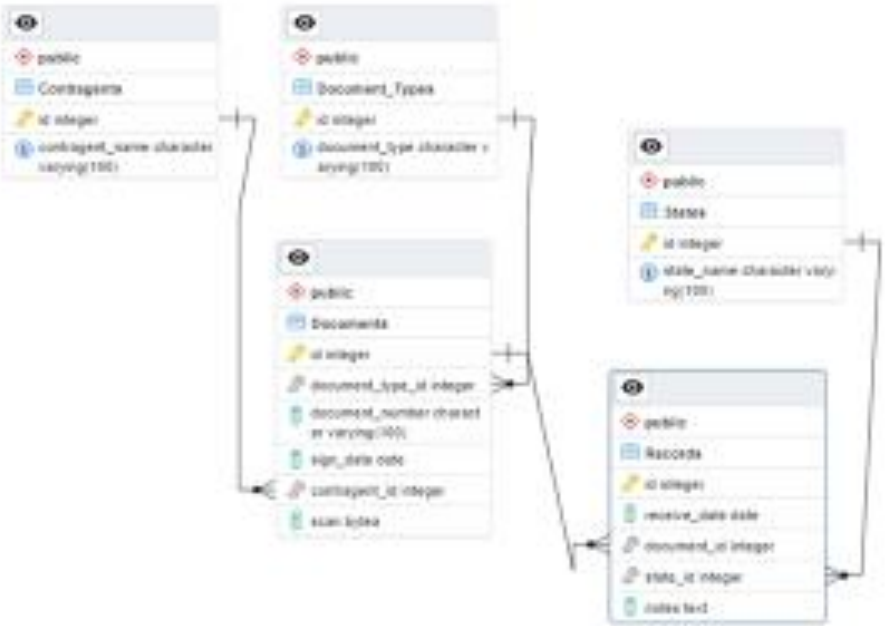
# АРХІТЕКТУРА ЗАСТОСУНКУ

## Архітектура клієнтського додатка



# АРХІТЕКТУРА ЗАСТОСУНКУ

## Структура бази даних



# ПОРІВНЯННЯ З АНАЛОГАМИ

Під час порівняння системи DocRegister з іншими конкурентами виявлено, що додаток має наступні переваги:

- адаптація саме під реєстрацію паперових документів, а не універсальний документообіг;
- автоматизація рутинних операцій завдяки OCR та фільтрації;
- відсутність залежності від великих постачальників ПЗ та низька вартість впровадження;
- можливість розширення функціоналу в майбутньому під специфічні потреби компанії

# ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ

Доступ до спільних документів від різних користувачів

Підтримка більшої кількості мов для OCR


Оптимізація користувацького досвіду за допомогою AI

# ВИСНОВКИ

1. Проведено аналіз ринку щодо аналогічних застосунків.
2. Проведено порівняння засобів розробки та обрані найдоцільніші технології.
3. Описано загальну структуру та архітектуру програмного застосунку.
4. Розроблено систему згідно з переліком вимог та обраних технологій
5. Сформовано рекомендації для подальшого розвитку та підтримки DocRegister.

# Дякую за увагу!

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

# ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕЄСТРАЦІЇ ПАПЕРОВИХ ДОКУМЕНТІВ ПІДПРИЄМСТВА

## Програма та методика тестування

ДП.045490-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ЗАЩИК

2024

**ЗМІСТ**

# 1. ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробувань є програмний продукт «DocRegister» – система для автоматизації реєстрації документообігу аграрного підприємства. Система складається з двох компонентів:

- windows-додаток – клієнтський інтерфейс для взаємодії користувачів із системою;
- веб-додаток (API) – серверна частина, реалізована на основі Django REST Framework (DRF), що забезпечує обробку, збереження та надання доступу до даних;
- модуль з OCR API, реалізований із використанням Tesseract OCR, який виконує оптичне розпізнавання символів, зокрема кирилиці, для автоматичного зчитування даних зі сканованих документів.

Проєкт реалізовано з застосуванням технологій Microsoft .NET (клієнтська частина) та Python/Django (серверна частина). Тестування охоплює функціональні можливості, інтерфейс користувача, продуктивність, безпеку та коректність роботи OCR-модуля.

# 2. МЕТА ТЕСТУВАННЯ

Метою тестування є підтвердження відповідності програмного продукту «DocRegister» встановленим вимогам щодо:

- функціональної працездатності всіх компонентів системи;
- коректної взаємодії Windows-клієнта із серверним API на базі DRF;
- правильної роботи OCR-модуля з Tesseract для коректного розпізнавання документів;
- відповідності форматів і протоколів передачі даних між клієнтом і сервером;
- забезпечення належного рівня безпеки збережених і переданих даних;

- зручності та інтуїтивності інтерфейсу користувача;
- відповідності дизайну затвердженим технічним вимогам.

## 3. МЕТОДИ ТЕСТУВАННЯ

Тестування проводиться методом Gray Box Testing, що передбачає аналіз як внутрішньої логіки та коду системи (включно з API та OCR-модулем), так і зовнішньої функціональності. Тестування виконуються на рівні системного тестування із застосуванням наступних підходів:

- ручне функціональне тестування інтерфейсу користувача – перевірка основних сценаріїв роботи Windows-додатку, зокрема введення даних, навігації та взаємодії з документами;
- тестування OCR-модуля – вручну перевірялася точність розпізнавання тексту на сканованих документах з різною якістю та форматами;
- тестування API – здійснювалося за допомогою інструменту Postman для перевірки коректності запитів і відповідей сервера, а також роботи основних функцій API.

## 4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування проводилося вручну та із застосуванням інструменту Postman для перевірки API.

Порядок тестування:

1) динамічне ручне тестування інтерфейсу – введення граничних та недопустимих значень у редаговані поля, перевірка поведінки системи;
2) ручне тестування функціональності Windows-додатку на відповідність вимогам;

3) тестування API через Postman – перевірка коректності запитів і відповідей, функціональної взаємодії з сервером;
4) тестування OCR-модуля – оцінка точності розпізнавання тексту на різних сканованих документах;
5) оцінка зручності використання інтерфейсу користувача.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“___” _____ 2025 р.

# ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ РЕЄСТРАЦІЇ ПАПЕРОВИХ ДОКУМЕНТІВ ПІДПРИЄМСТВА

**Керівництво користувача**

ДП.045490-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Дмитро НОВАК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Іван ЗАЩИК

2025

**ЗМІСТ**

## 1. Опис структури застосунку DocRegister

Застосунок DocRegister – це настільна програма для організації, перегляду та обробки великої кількості документів. Структура інтерфейсу побудована на основі вкладок, кожна з яких виконує окрему функціональну роль.

Основні вкладки застосунку:

- Документи – головна вкладка, де зосереджено всі ключові функції роботи з документами.
- Контрагенти – вкладка для керування довідником контрагентів.
- Типи документів – дозволяє додавати та редагувати типи документів.
- Стани документів – дозволяє управляти статусами документообігу.

Застосунок не є веб-ресурсом, тому не використовує HTML-сторінки.

Всі елементи інтерфейсу реалізовані засобами WPF (Windows Presentation Foundation) з підтримкою клавіатурної навігації, сучасного мінімалістичного дизайну та модульної архітектури.

Кожен модуль має доступ до спільної бази даних, що дозволяє інтегровано керувати інформацією. Для адміністративних задач використовується окрема панель налаштувань, доступна авторизованим користувачам з відповідними правами.

## 2. Опис вкладок інтерфейсу

Користувацький інтерфейс застосунку DocRegister реалізований із пріоритетом на простоту, швидкість доступу до функцій та зручність обробки великих обсягів документації. Інтерфейс поділений на кілька вкладок, кожна з яких відповідає за окремий напрям функціональності, що дозволяє користувачу швидко орієнтуватися в системі.

Головна вкладка – «Records» / «Записи» (рис. 1). Це основна робоча область застосунку, до якої користувач потрапляє одразу після запуску. Тут

у табличному вигляді відображаються всі наявні документи. Реалізовано можливість:

- фільтрації документів за датою, контрагентом, типом або станом;
- додавання, редагування та видалення записів;
- очищення таблиці від фільтрів;
- завантаження PDF-файлів (сканів) до кожного запису.

Також у цій вкладці доступна функція розпізнавання тексту (OCR) для PDF-сканів. Система автоматично аналізує файл і заповнює відповідні поля: тип документа, його номер, дату підписання, контрагента, а також зберігає фрагмент скану з відповідною частиною тексту.

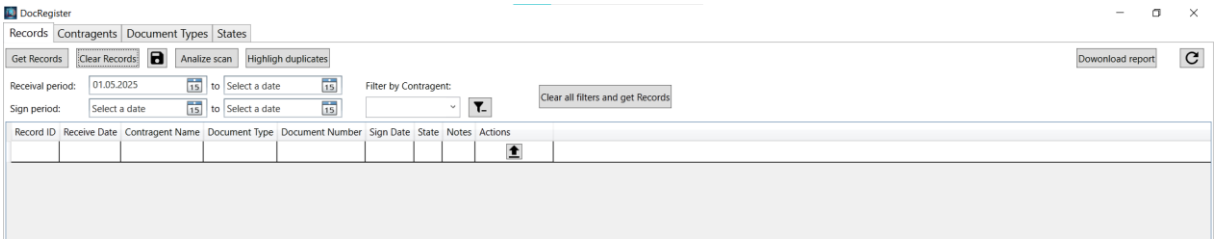



Рис. 1. Вкладка «Records» / «Записи»

Вкладка «Contragents» / «Контрагенти» (рис. 2). Ця вкладка реалізує роботу з довідником контрагентів. Користувач може:

- переглядати існуючі контрагенти у вигляді таблиці;
- додавати нових контрагентів;
- редагувати або видаляти наявні записи.

Контрагенти використовуються в документах при фільтрації, редагуванні та під час автоматичного заповнення інформації при OCR-аналізі.

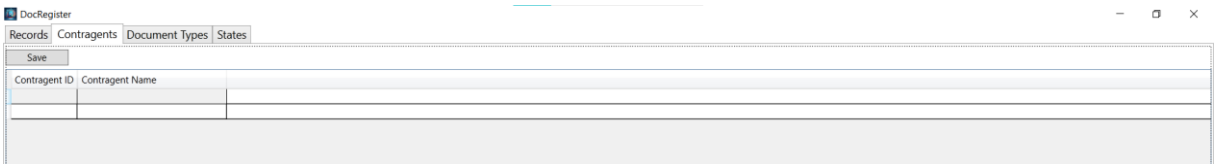



Рис. 2. Вкладка «Contragents» / «Контрагенти»

Вкладка «Document Types» / «Типи документів» (рис. 3). На цій вкладці керується перелік типів документів. Користувачу доступні функції:

- створення нових типів;
- редагування наявних;
- видалення непотрібних типів.

Типи документів застосовуються як у процесі ручного введення даних, так і при автоматичному розпізнаванні тексту.

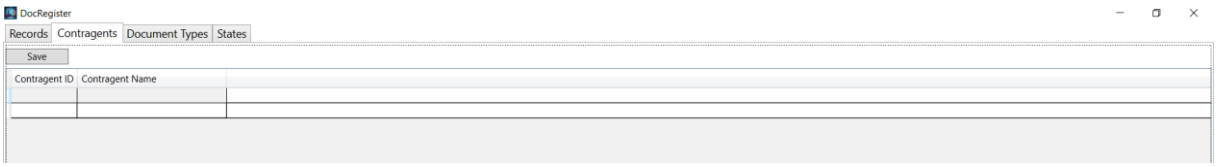



Рис. 3. Вкладка «Document Types» / «Типи документів»

Вкладка «States» / «Стани» (рис. 4). Ця вкладка дозволяє визначати і змінювати стани документів. Вони використовуються для візуального маркування в таблиці документів.


Рис. 4. Вкладка «States» / «Стани»

Загальний дизайн інтерфейсу дотримується модульного підходу: усі вкладки мають уніфіковане компонування, зрозумілі іконки та текстові підказки. Також передбачена повноцінна клавіатурна навігація для прискореної роботи з великим обсягом даних. Така структура інтерфейсу дозволяє не лише покращити досвід користувача, а й легко розширювати функціональність у майбутньому.

## 3. Додавання та редагування записів

У DocRegister усі дії з документами, контрагентами та типами документів виконуються безпосередньо в головній таблиці відповідної вкладки, реалізованій через компонент DataGrid (WPF). Інтерфейс дозволяє редагувати записи з мінімальними діями – без окремих кнопок чи форм.

Нові записи додаються шляхом заповнення останнього рядка таблиці, де всі поля є редагованими. Після введення даних у ключові поля (наприклад, дата отримання, тип документа, номер документа) потрібно натиснути кнопку «Save» / «Зберегти», тоді запис додається до бази даних.

Для модифікації даних потрібно внести потрібні змінити та натиснути кнопку «Save» / «Зберегти».

Для видалення запису, потрібно натиснути на відповідний ID, тоді, після виділення всього рядка, натиснути Backspace на клавіатурі. Після цього необхідно натиснути кнопку «Save» / «Зберегти».

## 4. Завантаження та розпізнавання сканів документів

Застосунок DocRegister підтримує дві незалежні функції роботи зі сканованими документами у форматі PDF: додавання сканів вручну та аналіз PDF файла.

При додаванні сканів до існуючих записів, кожному запису в таблиці можна прикріпити скан документа:

- У стовпці «Actions» / «Дії» відображається кнопка для завантаження скану на випадок, якщо файл ще не доданий.
- Після вибору PDF-файлу, він зберігається в базі та закріплюється за відповідним записом.
- У разі потреби, користувач може відкрити цей файл для перегляду або замінити його іншим.

Ця функціональність використовується для зберігання вже відомих документів, коли всі інші поля (тип, номер, дата, контрагент тощо) заповнено вручну, або оновлення скана для конкретного запису.

Також застосунок дозволяє імпортувати PDF-файли з метою автоматичного створення нових записів:

- Після завантаження файл обробляється за допомогою OCR (оптичного розпізнавання символів).
- На основі вмісту визначаються ключові атрибути: тип документа, номер, дата підписання, контрагент тощо.
- Якщо документ є багатосторінковим, кожна сторінка розглядається окремо – це дає змогу виділити кілька документів з одного файлу.

Ця функція значно пришвидшує обробку документів, отриманих у великій кількості, особливо коли немає попередньо створених записів.

## 5. Генерація звітності та архівація документів

Застосунок DocRegister надає можливості для формування звітів та створення архівів документів у зручному для зберігання або передачі вигляді. Це дозволяє користувачам швидко підготувати вибірку документів за певними критеріями та зберегти її локально у структурованій формі.

Додаток дозволяє згенерувати звіт, який включає такі етапи:

- Формування Excel-файлу. Користувач може сформувати Excel-файл, який містить усі ключові дані документів та з урахуванням поточного стану таблиці, включаючи всі активні фільтри (дату, контрагента тощо). Дані експортуються у зручному для аналізу табличному форматі.
- Архівація сканів. Окрім таблиці, застосунок дозволяє згенерувати ZIP-архів, що містить усі PDF-файли, прикріплені до відповідних записів. Назви файлів формуються автоматично за шаблоном: {Контрагент}_{ТипДокумента}_{НомерДокумента}.pdf

- Завантаження Excel-файлу та архіву до обраної теки. Після генерації звіту користувачеві пропонується вибрати місце збереження обох файлів – Excel-таблиці та ZIP-архіву. Застосунок автоматично обробляє обидва завдання паралельно, що забезпечує зручність і економію часу.