МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем
_____Юрій КОНДРАТЕНКО
«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

СИСТЕМА РОЗПІЗНАВАННЯ І КЛАСИФІКАЦІЇ ФІНАНСОВИХ ДОКУМЕНТІВ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Олександр ОБУШКО
«___»_____ 2025 р.

Керівник канд. фіз.-мат. наук, доцент _____ Інесса КУЛАКОВСЬКА
«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем
_____ Юрій КОНДРАТЕНКО
«____» _____ 20___ р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

_____ _ _Обушко Олександр Олександрович_____
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Система розпізнавання і класифікації фінансових документів_____.

Керівник роботи: Кулаковська Інесса Василівна, доцент кафедри інтелектуальних інформаційних систем, кандидат фізико-математичних наук, доцент_____.
(прізвище, ім'я, по батькові, посада, науковий ступінь, вчене звання)

Затверджена наказом ЧНУ ім. Петра Могили від « 18 » листопада 2024 р. №321

2. Строк представлення кваліфікаційної роботи «_11_» _червня__ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: програмна реалізація інформаційної системи для автоматичного розпізнавання та класифікації паперових ТТН, що дозволить вилучати ключові реквізити та інтегрувати дані у внутрішні системи підприємств. Для початку роботи необхідний набір зразків ТТН

від різних постачальників у вигляді сканованих копій або фотографій для навчання та тестування системи.

4. Перелік питань, що підлягають розробці: створення алгоритмів для попередньої обробки зображень ТТН, розробка модуля оптичного розпізнавання тексту та семантичного аналізу для вилучення реквізитів. Також необхідно реалізувати механізм навчання на основі шаблонів, спроєктувати базу даних для зберігання інформації та розробити графічний інтерфейс користувача для взаємодії з системою.

5. Перелік графічних матеріалів: презентація_____
_____.

Керівник роботи _____ Інесса КУЛАКОВСЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олександр ОБУШКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Система розпізнавання і класифікації фінансових документів

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання	18.12.2024	20.12.2024	виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	01.02.2025	виконано
3	Огляд аналогічних систем щодо розпізнавання документів	01.02.2025	10.02.2025	виконано
4	Огляд наукових публікацій на тему OCR	10.02.2025	25.02.2025	виконано
5	Огляд та вибір системи розпізнавання, інших бібліотек та СУБД для реалізації системи розпізнавання ТТН	25.02.2025	01.03.2025	виконано
6	Проєктування системи	01.03.2025	14.03.2025	виконано
7	Реалізація системи	14.03.2025	15.05.2025	виконано
8	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	виконано
9	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	виконано
10	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	виконано
11	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	виконано
12	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	виконано

Керівник роботи _____ Інесса КУЛАКОВСЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олександр ОБУШКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Обушко Олександра Олександровича

на тему: **“СИСТЕМА РОЗПІЗНАВАННЯ І КЛАСИФІКАЦІЇ ФІНАНСОВИХ ДОКУМЕНТІВ”**

Актуальність даного дослідження полягає у необхідності підвищення ефективності документообігу ТТН, розробці програмного забезпечення з використанням технологій комп’ютерного зору та оптичного розпізнавання символів. Це дозволить автоматизувати обробку паперових документів, що зменшить навантаження на операторів та пришвидшить їх роботу.

Об’єктом роботи є процеси розпізнавання та класифікації текстової інформації з паперових ТТН у системах електронного документообігу.

Предметом роботи є методи та технології автоматизації обробки ТТН за допомогою комп’ютерного бачення та оптичного розпізнавання символів (OCR).

Метою роботи є дослідження інформаційної системи для розпізнання та класифікації паперових ТТН, яка дозволить автоматизувати процес витягу основних реквізитів документа та інтегрувати данні у внутрішні інформаційні системи підприємств.

В результаті виконання роботи було досліджено інформаційну систему для розпізнавання та класифікації паперових ТТН, яка автоматизує процес вилучення основних реквізитів документа, а також розроблено програмне забезпечення, що реалізує функціонал розпізнавання та класифікації документів.

Робота складається з чотирьох розділів. У першому проведено аналіз предметної області, огляд останніх публікацій та аналогів, сформовано постановку задачі. Другий присвячений методам розробки системи та необхідним для цього технологіям. У третьому наведено опис проєктування системи розпізнавання та класифікації фінансових документів. В четвертому – опис програмної реалізації та

результати тестування. Загальний обсяг роботи – 101 сторінка. Кваліфікаційна робота містить 1 додаток, 22 рисунків, 5 таблицю і 34 джерел посилання.

Ключові слова: автоматизація обробки документів, оптичне розпізнавання символів (OCR), семантичний аналіз тексту, цифрова трансформація підприємств, модульна архітектура системи.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Obushko Oleksandr

“FINANCIAL DOCUMENT RECOGNITION AND CLASSIFICATION SYSTEM”

The relevance of this research lies in the need to increase the efficiency of document flow in TTN, the development of software using computer vision and optical character recognition technologies. This will allow automating the processing of paper documents, which will reduce the burden on operators and speed up their work.

The object of the work is the processes of recognition and classification of text information from paper TTN in electronic document flow systems.

The subject of the work is methods and technologies for automating TTN processing using computer vision and optical character recognition (OCR).

The purpose of the work is to study an information system for recognizing and classifying paper TTN, which will allow automating the process of extracting the main details of the document and integrating data into the internal information systems of enterprises.

As a result of the work, an information system for recognizing and classifying paper TTN was studied, which automates the process of extracting the main details of the document, and software was developed that implements the functionality of document recognition and classification.

This work consists of four sections. The first section analyzes the subject area, reviews recent publications and analogues, and formulates the problem statement. The second section is devoted to the methods of system development and the technologies necessary for this. The third section describes the design of the system for recognizing and classifying financial documents. The fourth section describes the software implementation and testing results.

The overall scope of the work is 101 pages. Thesis contains 1 application, 22 figures, 5 tables and 34 references in it.

Key words: document processing automation, optical character recognition (OCR), semantic text analysis, digital transformation of enterprises, modular system architecture.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – база даних

Валідація – процес перевірки коректності та правдоподібності отриманих або введених даних на відповідність визначеним правилам.

ДСТУ – державний стандарт України

ІПН – індивідуальний податковий номер

Метадані – структуровані дані, що описують, пояснюють або надають інформацію про інші дані.

Парсинг – аналіз вхідних даних для виділення їх складових частин та інтерпретації.

ПДВ – податок на додану вартість

СУБД – система управління базами даних

ТТН – товарно-транспортна накладна

ЄДРПОУ – Єдиний державний реєстр підприємств та організацій України

API – Application Programming Interface

CRM – Customer Relationship Management

ERP – Enterprise Resource Planning

GDPR – General Data Protection Regulation

GUI – Graphical User Interface

JNI - Java Native Interface

JVM – Java Virtual Machine

LTS – Long-Term Support

OCR – Optical Character Recognition

RPA – Robotic Process Automation

SAP – Systems, Applications and Products

SDK - Software Development Kit

SQL – Structured Query Language

UI – User Interface

ВСТУП

Цифрова трансформація підприємств суттєво змінює підходи до ведення документації, особливо в галузях торгівлі та логістики. Попри активне впровадження електронного документообігу, значна частина процесів все ще залежить від паперових накладних, що потребують ручного внесення даних у комп'ютерні системи. Це спричиняє значні витрати часу, підвищує ризик помилок і збільшує адміністративні витрати.

Автоматизація обробки товарно-транспортних накладних (ТТН) вимагає застосування передових методів комп'ютерного бачення та оптичного розпізнавання символів (OCR). З огляду на різноманітність форматів ТТН, ключовою задачею є розробка гнучкої системи, здатної коригувати спотворення та враховуватиме рівномірне освітлення під час сканування або фотографування документів.

Використання методів попередньої обробки, таких як покращення контрасту, видалення артефактів та підвищення роздільної здатності, сприяє більшій точності подальшого текстового аналізу. Традиційні OCR-системи добре працюють із стандартними шрифтами та чітко надрукованими документами, проте у випадку змінних форматів ТТН можуть виникати труднощі з інтерпретацією нестандартних символів, розпізнаванням числових полів або коректним відокремленням розділових знаків. Для вирішення таких проблем доцільно використовувати додаткові алгоритми семантичного аналізу тексту та валідації числових реквізитів.

Метою даної роботи є дослідження інформаційної системи для розпізнання та класифікації паперових ТТН, яка дозволить автоматизувати процес витягу основних реквізитів документа та інтегрувати данні у внутрішні інформаційні системи підприємств. Для досягнення цілі виконано певний перелік завдань, до яких входить:

- проаналізувати сферу цифровізації фінансових документів;
- дослідити інформаційні технології для цифровізації паперових документів;
- спроєктувати систему розпізнавання ТТН;
- створити програмну реалізацію системи розпізнавання ТТН та проаналізувати її.

Модульна архітектура дозволяє розширювати функціонал системи, інтегрувати її з зовнішніми сервісами та застосовувати для обробки інших типів документів. Ці технологічні рішення сприяють підвищенню ефективності управління документацією та впровадженню сучасних підходів до цифрової трансформації підприємств.

1 АНАЛІЗ СФЕРИ ЦИФРОВІЗАЦІЇ ФІНАНСОВИХ ДОКУМЕНТІВ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери

На сьогоднішній день будь-яке підприємство, що здійснює діяльність у сферах торгівлі або перевезень зобов'язане вести паперову документацію. Водночас в умовах цифрової трансформації бізнес прагне максимально перевести цю документацію в електронний формат, оскільки електронний документообіг забезпечує миттєвий доступ до інформації, зручне зберігання та можливість інтеграції з різними системами або хмарними сервісами [1]. Здебільшого на практиці застосовується гібридний підхід: тобто первинний облік ведеться на паперових бланках, після чого працівники вручну переносять ключові реквізити до комп'ютерної системи [2]. Цей процес відрізняється високою трудомісткістю: працівники витрачають значну частину робочого дня на перенесення даних, що сильно впливає на швидкість обробки інформації та підвищує ризик помилок [3].

Особливу групу документів у логістичній та торговій галузі становлять ТТН. Вони є основним підтвердженням факту передачі вантажу між підприємствами та містять сукупність данних: від даних про відправника й одержувача до детального опису найменування товарів, їх одиниць виміру, кількості, ціни за одиницю та вартості [4]. Хоча загальні правила структурування ТТН закріплені в нормативних актах, в реальному житті ці шаблони підлаштовуються під потреби підприємств. Компанії можуть використовувати власні шрифти, розміщувати логотипи, штампи чи штрих-коди, додавати колонтитули та графічні елементи, вносити табличні надбудови або дублювати текст кількома мовами. В результаті кожен бланк, хоча і належить до одного виду документа, має унікальні параметри оформлення [5].

Підприємства, які працюють із великим обсягом накладних, часто обробляють тисячі ТТН щодня [6]. Серед них компанії, що доставляють продукти харчування та лікарські засоби, замовляють будівельні матеріали та промислові вантажі, займаються перевезенням зерна, промислової сировини або хімічних

речовин, постачають меблі, одяг, взуття й вироби індивідуальної гігієни. У кожному сегменті та підприємстві застосовуються свої бланки, які відповідають технологічним та нормативним вимогам, наприклад, наявність окремих колонок для маркування температурного режиму продуктів чи позначки про перевірку цілісності упаковки [7]. Враховуючи, що система розроблятиметься здебільшого для звичайних торгових точок, таких як продуктові магазини, вона повинна ефективно обробляти формати бланків, які найчастіше зустрічаються в цьому сегменті, без потреби дорогого кодування кожного нового шаблону [8].

Також існує проблема різнорідності якості вхідних зображень. Нерівномірне освітлення під час фотографування або сканування призводить до тіней та засвічених ділянок, що ускладнює чітке відокремлення меж полів документа. Викривлення, спричинені кутом нахилу камери або неточним вирівнюванням аркуша, можуть змінювати пропорції рядків і стовпців таблиць, що порушує структуру даних [9]. Низька роздільна здатність або сильне стиснення у форматі JPEG часто призводять до розмитості символів та появи блокового шуму – артефактів, які перешкоджають коректному розпізнаванню літер і цифр. Складки, зморшки чи пошкодження паперу створюють додаткові лінії та контурні візерунки, які алгоритм може помилково трактувати як текстові елементи [10]. Для покращення результатів перед основним розпізнаванням застосовують прості методи обробки: вирівнювання аркуша, базове коригування яскравості та контрасту, легке згладжування шуму й обрізання небажаних країв. Це дозволяє отримати чистіше зображення та підвищити точність витягнення тексту й цифр без використання складних алгоритмів [11].

Традиційні OCR-двигуни демонструють високу точність на документах зі стабільною структурою та добре відсканованими зображеннями. Але при роботі з реальними бланками ТТН, де шаблони можуть відрізнятися між собою, а символи бути тонкими або незвичними, вони можуть виявляють свої обмеження: цифри іноді «збиваються» з місця, розділові знаки можуть неправильно сприйматися як інші символи, а значення у різних стовпцях – змішуватися між собою [12]. Навіть

незначна помилка в розпізнаванні кількості товару чи загальної суми здатна призвести до неточностей у бухгалтерській звітності, викликати затримки у взаєморозрахунках із постачальниками та збільшити трудовитрати на перевірку й виправлення даних [13]. У бізнесі, що обробляє сотні й тисячі документів щодня, важливо досягти високого рівня точності, аби запобігти накопиченню системних похибок. Саме тому сучасні рішення поєднують класичний OCR із додатковими перевірками контексту, валідацією числових полів та автоматичним виправленням типових помилок – що суттєво підвищує якість розпізнавання в реальних умовах [14].

Автоматизація обробки товарно-транспортних накладних передбачає комплексну послідовність дій: пошук полів із даними, розпізнавання тексту та переведення його в зручний для аналізу формат [15]. У предметній сфері малих торгових точок, зокрема продуктових магазинів, ключовими факторами є стійкість системи до змін у якості сканів або фотографій. Автоматичне вилучення реквізитів із ТТН створює міцний фундамент для подальшої аналітики: моніторингу обсягів постачання в реальному часі, управління запасами та формування прозорих звітів без затримок [16]. Швидша обробка документів дає змогу скоротити ручні операції та перенаправити ресурси працівників на виконання інших завдань.

Таким чином, аналіз предметної області підкреслює необхідність гнучких алгоритмів попередньої обробки зображень і розпізнавання, а також чіткої структури даних для забезпечення ефективної інтеграції з обліковими або складськими системами.

1.2 Огляд та аналіз наявних аналогів та публікацій

Для проєктування власного рішення доцільно детально вивчити індустріальні та відкрито-джерельні OCR-інструменти, що нині доступні на ринку. Нижче наведено огляд чотирьох ключових систем із розширеним описом їх архітектури, підходів та основних переваг і недоліків.

1.2.1 Індустріальні та open-source рішення

UiPath Document Understanding – це рішення від компанії UiPath, яке спрощує переведення сканованих та цифрових документів у структуровані дані. Ідея виникла у 2019-2020 роках, коли зростання обсягів паперових рахунків та накладних стало проблемою для бізнесу. Перший реліз 2020.10 вже міг автоматично обробляти PDF, зображення та поштові вкладення: система сама визначала, чи це інвойс, накладна чи контракт, і далі зчитувала основні поля.

У 2021 році вийшла довготривала версія 2021.10 LTS з поліпшеним пошуком полів і новим інструментом навчання: користувачі могли завантажувати кілька зразків документів, позначати, де саме знаходяться сума, дата чи номер і через хвилини система навчалася на цій інформації. У 2022 році додано підтримку розмітки таблиць: тепер вона розбивала рядки та стовпці автоматично, навіть якщо макети різнилися від постачальника до постачальника.

У 2023 році в налаштуваннях з’явилися готові шаблони під звичні форми: інвойси SAP, рахунки 1С, форми державних органів – усе це можна було підключити одним кліком. А в оновленні 2024.4 впровадили можливість працювати без доступу до хмари, встановлюючи всі необхідні компоненти локально на сервері компанії.

Технічно Document Understanding складається з п’яти основних частин. Спершу система «дивиться» на документ – це модуль класифікації, який швидко визначає тип файлу. Далі йде модуль OCR: він перетворює зображення у текст, використовуючи перевірені движки у фоновому режимі. Потім модуль виділення даних шукає ключові поля, використовуючи поєднання правил (наприклад, завжди береться слово поряд з «Сума») та машинного навчання за зібраними прикладами. Четвертий крок – перевірка людиною в зручному інтерфейсі: оператор бачить усі знайдені поля у вигляді таблиці і може швидко виправити неточності. На

завершення система експортує дані у вигляді Excel, JSON або XML, які легко завантажити в будь-які облікові чи ERP-системи.

Документ-студія для налаштувань працює без коду: достатньо перетягнути блоки «Класифікація», «OCR», «Екстракція», «Перевірка» та «Експорт» у робоче поле та задати базові параметри. Якщо потрібно обробити новий тип рахунків, досить завантажити кілька зразків і натиснути «Навчити», а система автоматично побудує модель вилучення полів. Усі зміни зберігаються як частина процесу RPA, тож їх можна запускати за розкладом чи вручну в UiPath Orchestrator.

Розгортати Document Understanding можна у хмарі UiPath Automation Cloud або інсталяцією на власних серверах. Для великих підприємств доступна гібридна модель: критичні дані обробляються локально, а менш чутливі відправляються у хмару для швидшого розпізнавання. Система підтримує багатокористувацьку роботу – кілька операторів можуть одночасно валідувати документи у веб-інтерфейсі.

Типові сценарії включають автоматизацію закупівель: зчитування рахунків постачальників, валідація по сотнях полів, накопичення статистики за оплатами; обробку банківських виписок та договорами, де важливо швидко знаходити ключові дати й суми; а також створення архівів searchable PDF для легкого пошуку по тексту. Щоб система працювала стабільно, радять:

- використовувати сканери зі стабільним освітленням та не менше 200 DPI;
- попередньо приведення документів до однорідного формату (PDF або TIFF);
- надання прикладів різних макетів під час навчання моделі;
- регулярну перевірку статистики помилок у Orchestrator Insights для вчасного донавчання.

Переваги:

- абсолютно безкоштовний, ліцензія Apache 2.0;
- підтримка широкого спектра мов (>100) та кастомних моделей;
- кросплатформеність (Linux, Windows, macOS);

- розвинена спільнота, детальна документація та приклади коду.

Недоліки:

- чутливий до низької якості сканів, шуму та тіней;
- вимагає ретельної передобробки зображень (deskew, threshold);
- обмежений у сегментації складних багатоколонних макетів;
- відсутність зручного графічного інтерфейсу для моніторингу процесів.

OCRopus – це безкоштовна програма для розпізнавання тексту на відсканованих документів. Її почали розробляти у 2007 році в дослідницькому підрозділі Google. Програма призначена для покрокової обробки зображень: від чищення скану до перетворення тексту в цифрову форму. Кожен етап – це окрема частина програми, яку можна змінити або налаштувати. Завдяки цьому користувач може адаптувати систему під свої потреби.

Спочатку програма прибирає зайве з картинки: фоновий шум, затемнення, плями або перекоси. Це робиться для того, щоб залишився тільки чорний текст на білому фоні. Потім документ ділиться на частини: абзаци, заголовки, колонки або таблиці. Далі програма шукає рядки тексту і в кожному рядку – окремі слова. Після цього розпізнає, що саме написано: наприклад, якщо символи погано видно або перекручені, система намагається здогадатися, що це за слово. Усе це виконується поетапно, що дозволяє змінювати чи вдосконалювати кожен крок окремо.

Після обробки результат можна зберегти як звичайний текстовий файл або документ із точним розташуванням тексту на сторінці – це зручно, якщо потрібно створити файл, у якому можна буде шукати слова, як у PDF. Програма також дозволяє вручну вказати, де на зображенні знаходяться конкретні слова чи фрази – це допомагає навчитись краще працювати з подібними документами в майбутньому.

OCRopus часто використовують дослідники, архівісти, бібліотеки чи невеликі компанії, які хочуть мати повний контроль над тим, як саме розпізнається текст. Програма не має готового віконного інтерфейсу, як інші подібні рішення,

зате її можна налаштувати «під себе», змінюючи послідовність дій. Це зручно, коли потрібно працювати з нестандартними або старими документами.

OCRopus добре працює з багатьма форматами зображень і дає змогу автоматично обробляти одразу велику кількість файлів. Щоб її встановити, потрібно мати операційну систему Linux або Mac та базові знання роботи з командним рядком. Для стабільної роботи бажано мати достатньо пам’яті та сучасний комп’ютер.

Переваги:

- можна налаштувати майже все, що стосується процесу розпізнавання;
- дає змогу бачити, що відбувається на кожному кроці обробки;
- можна доповнювати своїми правилами або модулями;
- підходить для експериментів або нестандартних задач.

Недоліки:

- немає зручного графічного інтерфейсу, все через текстові команди;
- потрібно самому налаштовувати роботу, що займає час;
- не завжди справляється з таблицями або багатоколонними документами;
- потрібні базові технічні знання для початку роботи.

Tesseract – це одна з найвідоміших безкоштовних програм для розпізнавання тексту з зображень або сканів документів. Її почали розробляти ще в 1980-х роках у дослідницьких лабораторіях Hewlett-Packard. Через кілька десятиліть програму підтримала компанія Google, яка відкрила її код і почала активно розвивати. З того часу Tesseract постійно оновлюється та підтримується спільнотою по всьому світу.

Основна функція програми – «читати» текст на зображеннях. Наприклад, якщо ви маєте сфотографовану товарно-транспортну накладну або рахунок, програма може розпізнати текст із зображення й перетворити його у звичайний цифровий текст, який можна редагувати, копіювати або зберігати у форматі документа. Це особливо зручно, коли потрібно швидко внести багато даних у комп’ютерну систему без ручного введення.

Tesseract підтримує понад 100 мов, у тому числі українську, англійську, німецьку, французьку та навіть мови, які пишуться справа наліво. Це дає змогу використовувати його в різних країнах і для різних типів документів – від стандартних бланків до газет чи книжок.

Щоб результат розпізнавання був точним, важливо, щоб зображення було якісним: без тіней, чітким, з хорошим освітленням. Якщо текст на зображенні розмитий або нахилений, програма може помилятися. Проте навіть у таких випадках є способи покращити результат: наприклад, можна попередньо обробити зображення, вирівняти його або прибрати зайві елементи.

Ще однією сильною стороною Tesseract є його здатність навчатися. Якщо підприємство використовує нестандартні шрифти або специфічні шаблони документів, можна додатково навчити програму, щоб вона краще їх розуміла. Наприклад, якщо компанія отримує накладні від різних постачальників, і в кожного з них трохи відрізняється структура бланку, систему можна налаштувати під кожен із них.

Tesseract працює на основних операційних системах – Windows, Linux і macOS. Його можна використовувати окремо або інтегрувати в інші програми. Наприклад, розробник може створити систему, яка автоматично обробляє скановані документи та зберігає дані у вигляді таблиць. Також доступні доповнення до програми наприклад: графічні оболонки, які роблять її зручнішою для тих, хто не працює з командами.

Програма підходить як для домашнього використання, так і для компаній. У невеликому магазині її можна використовувати для сканування накладних, у бібліотеці – для оцифрування книжок, у компанії – для автоматизації обліку [14].

Станом на листопад 2024 року останньою версією є Tesseract 5.5.0. У ній було покращено точність, додано підтримку нових мов і шаблонів документів. Розпізнавання стало швидшим, і тепер програма краще справляється з нестандартним розташуванням тексту.

Переваги:

- безкоштовна й доступна кожному;
- працює з понад 100 мовами, включаючи українську;
- дає змогу навчити систему під конкретні документи;
- працює на різних операційних системах;
- підходить як для малих, так і для великих проєктів.

Недоліки:

- потребує якісного зображення для кращого результату;
- не має простого віконного інтерфейсу – більшість дій виконується через команди;
- може давати помилки, якщо текст перекошений чи розмитий;
- для повного налаштування потрібні базові знання комп'ютера.

1.2.2 Наукові статті та публікації

An OCR Engine for Printed Receipt Images using Deep Learning Techniques (Sayallar C., 2023) [17].

У своїй науковій праці Sayallar та його колеги з Університету Коджаелі (Туреччина) представили власну систему для оптичного розпізнавання тексту з друкованих чеків і товарних накладних – Nacsoft OCR. На їхню думку більшість існуючих систем OCR, таких як Tesseract або Google Vision API, мають низку обмежень у роботі з нестандартними шаблонами, низькою якістю зображення або специфічними мовами, зокрема турецькою. Мета авторів – створити компактний і простий у впровадженні інструмент, який забезпечить високу точність розпізнавання без потреби в потужному обладнанні.

Особливу увагу приділяють тому, що більшість відкритих датасетів із зображеннями чеків не доступні для широкого використання через наявність персональних даних (суми покупок, імена, адреси, ІПН). Через це команда розробників самостійно зібрала власний корпус датасет на турецькій та англійській

мовах, а також провела ручну розмітку даних, що дозволило побудувати повноцінну навчальну вибірку для тренування моделі.

Архітектура Nacsoft OCR складається з трьох основних етапів обробки:

- першим етапом є попередня обробка зображення (Image Preprocessing). На цьому етапі система виконує видалення всіх зайвих елементів навколо чека, які не несуть корисної інформації (руки, тіні, фрагменти інших предметів). Потім система визначає контури самого документа та виправляє перспективні спотворення. Тобто, якщо фотографія зроблена під кутом, зображення «розгортається», вирівнюється й обрізається до прямокутної форми. Для цього використовуються прості фільтри, алгоритми фіксації геометричних меж і переведення кольорового зображення в чорно-білий режим. Таким чином формується чітке, однорідне поле для подальшого аналізу;
- другий етап – сегментація тексту (Text Segmentation). Система аналізує очищене зображення та знаходить на ньому області, які виглядають як блоки тексту – зокрема, слова або групи символів. Для цього реалізовано аналіз щільності пікселів (ті частини, де пікселі скупчені в компактні зони, сприймаються як потенційні слова). Потім кожне слово виокремлюється як окремий фрагмент зображення. Якщо сегмент виявився надто маленьким (наприклад, це лише крапка або смітинка), або надто великим (наприклад, фрагмент двох злиплих слів) – система їх фільтрує. Таким чином відсіюються артефакти, а слова виділяються чітко й рівно;
- останній етап – розпізнавання тексту (Text Recognition). Кожне отримане слово система подає на вхід нейронній мережі, яка складається з кількох простих шарів, натренованих на зображеннях турецьких чеків. Мережа спочатку формує внутрішнє представлення форми літер (векторну ознаку), а потім збирає зі знайдених ознак конкретне слово. Щоб уникнути помилок з турецькими діакритичними символами, автори ввели стандартне правило заміни рідкісних букв на близькі латинські (наприклад: "Ğ" → "G", "Ç" → "C"). Це дало змогу підвищити точність і знизити кількість помилкових відповідностей. Після завершення

розпізнавання система формує таблицю з усіма знайденими словами та їх позицією на зображенні.

Завдяки простій структурі модель працює досить швидко навіть на комп'ютерах із середньою продуктивністю. Усі модулі реалізовано як окремі Python-скрипти, тому систему можна адаптувати під різні формати або інтегрувати в більші робочі процеси. Вихідна інформація експортується у форматах TXT або JSON, що дозволяє швидко імпортувати її у бухгалтерські, складські або CRM-системи.

Переваги:

- відкритий код і модульна архітектура, яку можна адаптувати до різних мов і шрифтів;
- власноруч зібраний датасет забезпечує відповідність реальним умовам і кращу точніст;
- система протестована на тих самих зразках, що й Tesseract, EasyOCR та Google Vision API, показала вищу точність для турецькомовних чеків;
- система не потребує дорогого обладнання й може бути розгорнута навіть у невеликому магазині або на сервері з базовою конфігурацією;
- проста процедура налаштування.

Недоліки:

- модель навчена виключно на турецьких чеках тому без донавчання погано працює з іншими мовами та шрифтами;
- необхідність вручну додавати кожен новий шаблон, шрифт або формат документа до навчального набору;
- у разі сильного викривлення, зм'ятості або засвічення скану етап попередньої обробки може не спрацювати належним чином;
- відсутність повноцінного користувацького інтерфейсу. Система доступна лише у вигляді Python-скриптів, тому вимагає технічних знань для інтеграції.

Попри ці обмеження, розробка Sayallar і його команди демонструє, що невелика нейронна система, спеціалізована під конкретний тип документів, може

суттєво перевершити загальноприйняті OCR-рішення і є прикладом того, як можна адаптувати глибоке навчання до потреб малого бізнесу, торгових мереж або спеціалізованих архівів. Основними передумовами успіху в подібних проєктах залишаються якісний датасет, адаптивність до нових вхідних форматів і простота масштабування системи.

Automated Receipt Parsing (Alex Yue, 2018) [18].

У цій статті Alex Yue з Прінстонського університету запропонував практичну та ефективну систему автоматичного витягу інформації з фотографій касових чеків. Він звернув увагу на те, що більшість існуючих систем мають два основні недоліки: високу вартість або недостатню точність. Особливо проблемною виявляється ситуація, коли знімок чека зроблений у складних умовах. Наприклад на телефон при поганому освітленні, на папері з візерунком або з нерівним розміщенням паперу. Мета автора – створити алгоритм, який працюватиме стабільно навіть у таких ситуаціях, залишаючись водночас простим і доступним.

Запропонований підхід включає кілька ключових етапів:

а) виявлення меж документа, для чого автор поєднав одразу три методи:

1) перший метод орієнтується на чітко виражені контури. Він шукає довгі прямі лінії, які добре виділяються на фоні;

2) другий групує короткі фрагменти, які разом можуть утворити межу, навіть якщо вона слабко видно;

3) третій оцінює зміну освітлення між ділянками зображення, дозволяючи виявити межі навіть тоді, коли вони нечіткі через тіні чи бліки.

Комбінування цих трьох способів дозволяє визначити, де саме на фото розташовано чек і де закінчується фон.

б) після того як межі визначено, програма вирізає чек із зображення. Якщо фотографія зроблена під кутом то система вирівнює її, тобто робить так, щоб текст був розміщений горизонтально. Завдяки цьому на наступному етапі текст краще читається, а символи не викривляються;

в) на третьому етапі програма розбиває зображення на дрібні частини, де кожна частина містить окреме слово або фрагмент рядка. Таким чином, кожен шматочок проходить через розпізнавання окремо, що дозволяє досягти вищої точності. Цей підхід також зменшує вплив проблем зі зображенням, наприклад коли тінь покриває лише частину чека;

г) наступний етап оптичне розпізнавання символів (OCR). На кожну ділянку зі словом застосовується класичний OCR-алгоритм, який перетворює зображення в текст. Завдяки тому, що фон уже очищено, а текст випрямлено, модуль працює з підвищеною точністю. Автор не створював власний OCR, а інтегрував один із доступних механізмів;

д) після застосовування OCR-алгоритму серед усього розпізнаного тексту система шукає ключові слова, наприклад: Total або Date. Коли вона знаходить такі слова, то дивиться поруч, шукаючи числа або дати. Наприклад, якщо після слова "Total" написано "123.45", то ця цифра визначається як фінальна сума покупки;

е) на заключному етапі отримані дані виводяться у зручному форматі – як простий текстовий файл або таблиця Excel. Це дозволяє одразу імпортувати результати в облікову систему або базу даних магазину.

Автор протестував систему на реальних знімках чеків, включно з тими, що мали неякісний фон, перекоси або затемнення. Результати показали, що точність залишається задовільною навіть за поганих умов, що особливо важливо для малих підприємств або користувачів без професійного сканувального обладнання.

Ця робота демонструє, що навіть простий підхід із поетапною обробкою може дати дуже добрі результати при умові правильної організації процесу.

Переваги:

- працює швидко навіть на звичайному комп’ютері, бо кожен етап оптимізований;
- підходить для чеків із будь-яким оформленням і тлом;
- висока надійність завдяки поєднанню кількох методів пошуку меж документа;

- простота інтеграції в інші програми або робочі процеси;
- дає зрозумілий результат, який легко обробляється в системах обліку.

Недоліки:

- гірше справляється з зображеннями, де документ сильно зім'ятий або має серйозні пошкодження;
- якщо фон має той самий колір, що й чек (наприклад, світлий стіл), межі можуть визначатися неточно;
- розпізнавання ключових слів працює тільки з конкретною мовою (англійською) – для інших мов потрібна адаптація;
- без донавчання система не працює стабільно з рідкісними або нестандартними чеками (наприклад, з нестандартним розміщенням дати чи суми).

Calamari + Post-correction (Drobac S., Lindén K., 2020) [19].

У цій роботі Drobac та Lindén дослідили ефективність сучасних нейронних OCR-систем у поєднанні з алгоритмами автоматичної посткорекції результатів. Основним об'єктом дослідження стали цифрові копії історичних газет, надрукованих фінською та шведською мовами в період із 1771 по 1929 роки. Ці архівні матеріали мали численні особливості: суміш двох мов на одній сторінці, використання кількох типів шрифтів (Antiqua та Fraktur), зношений папір, неякісна оцифровка, нерівна верстка тощо. Через це більшість існуючих систем розпізнавання, включно з комерційними рішеннями, демонстрували високий рівень помилок.

Метою дослідників було побудувати універсальну модель, яка однаково добре працює з обома мовами і стилями шрифтів, без попередньої класифікації тексту за мовою або типографікою. Вони також прагнули мінімізувати помилки після розпізнавання за допомогою постобробки, оскільки помилки у рідкісних словах або незвичних поєднаннях символів призводили до серйозного викривлення змісту. Запропоноване рішення складається з двох основних компонентів:

- модуль розпізнавання на базі Calamari. Автори використали відкриту OCR-платформу Calamari, яка застосовує поєднання згорткових і рекурентних

нейронних мереж. Особливістю їхнього підходу стало створення п’яти окремо натренованих систем, які паралельно розпізнають один і той самий текст. Після цього система застосовує голосування: якщо більшість моделей дають однаковий результат, він вважається найімовірнішим і зараховується як фінальний. Такий підхід знижує випадкові помилки й підвищує стабільність результатів навіть у складних умовах (коли символи погано видно через старіння паперу). Моделі були навчені на реальних газетах із точно верифікованим текстом (ground truth). Ці приклади включали всі можливі варіації: великі й малі літери, старі орфографічні форми, нестандартні розділові знаки та міжрядкові інтервали. Таким чином, система не потребувала попереднього визначення мови чи шрифту – вона вже знала, як виглядає текст у різних комбінаціях;

- модуль автоматичної посткорекції. Навіть після використання потужної нейронної моделі залишаються незначні помилки: злиття букв, неправильне розпізнавання схожих символів (наприклад, "c" і "e"). Для автоматичного виправлення таких ситуацій було впроваджено посткорекцію. Вона працює за принципом зіставлення отриманого слова зі словником і перевірки його на правильність. Якщо слово не відповідає жодному зі знайомих – система пропонує найближчий варіант з бази. Цей процес було реалізовано за допомогою скінченних автоматів (Finite-State Transducers), які дозволяють створити гнучкі, але точні правила заміни та перевірки тексту. Наприклад, якщо модель прочитала слово як "finnisk", а правильний варіант – "finsk" (шведською – фінський), посткоректор автоматично підставить правильне. Це значно знижує загальний показник помилок, який після корекції впав до менш ніж 2% у фінських текстах та близько 2.7% у шведських.

Крім того, дослідники підкреслюють важливість такого підходу саме для історичних корпусів, де не завжди можна знайти достатньо навчального матеріалу для кожної газети або номера. Універсальна модель із корекцією дозволяє ефективно масштабувати процес на великі масиви даних без втрати точності.

Переваги:

- підтримка кількох мов та шрифтів в рамках однієї моделі без необхідності попереднього поділу;
- автоматичне виправлення типових помилок після OCR значно підвищує якість даних;
- ефективне голосування між моделями покращує стабільність розпізнавання;
- відкрите програмне забезпечення з можливістю масштабування для обробки великих обсягів документів;
- актуальність для архівів, бібліотек та наукових установ, які оцифровують історичні документи.

Недоліки:

- для тренування моделі потрібен значний обсяг розмічених даних (ground truth);
- голосування між кількома моделями потребує більше ресурсів та часу;
- посткорекція не справляється з великими структурними помилками або змістовними неточностями;
- незвичні елементи оформлення (заголовки, оголошення, складні таблиці) можуть розпізнаватись гірше, оскільки рідко трапляються у тренувальних датасетах;
- розробка власних правил корекції потребує глибшого аналізу мови та частотності помилок.

1.3 Постановка задачі

Актуальність даного дослідження полягає у необхідності підвищення ефективності документообігу ТТН, розробці програмного забезпечення з використанням технологій комп’ютерного зору та оптичного розпізнавання символів. Це дозволить автоматизувати обробку паперових документів, що зменшить навантаження на операторів та пришвидшить їх роботу.

Метою даної роботи є дослідження інформаційної системи для розпізнання та класифікації паперових ТТН, яка дозволить автоматизувати процес витягу основних реквізитів документа та інтегрувати данні у внутрішні інформаційні системи підприємств.

Предмет роботи: методи та технології автоматизації обробки товарно-транспортних накладних (ТТН) за допомогою комп’ютерного бачення та оптичного розпізнавання символів (OCR).

Об'єкт роботи: процеси розпізнавання, аналізу та валідації текстової інформації з паперових ТТН у системах електронного документообігу.

Для досягнення мети необхідно вирішити наступні задачі:

- проаналізувати сферу цифровізації фінансових документів;
- дослідити інформаційні технології для цифровізації паперових документів;
- спроєктувати систему розпізнавання ТТН;
- створити програмну реалізацію системи розпізнавання ТТН та проаналізувати її.

Серед основних вимог до програмної реалізації системи можна виділити:

- точне розпізнавання тексту;
- робота з фіксованими шаблонами, які задає користувач (наприклад, для постачальника А та постачальника Б);
- зрозумілий та зручний для користувача дизайн інтерфейсу;
- масштабування – систему має бути легко підстроїти під інші типи документів у майбутньому;
- зберігання даних у зручному для обробки вигляді.

Реалізація цих задач дасть нам створити компактну, адаптивну та ефективну систему, орієнтовану як на малі так і на великі торгові підприємства. Вона має спростити обробку ТТН, підвищити точність облікових операцій і зменшити залежність від ручного введення даних.

Висновки до розділу 1

У першому розділі було проведено детальний аналіз предметної області, пов’язаної з автоматичним розпізнаванням і обробкою фінансових документів, зокрема товарно-транспортних накладних та чеків. З огляду на широку поширеність паперової документації в малому та середньому бізнесі, особливо в торгових точках, актуальність задачі переходу до електронного документообігу є беззаперечною. Ручне перенесення даних із накладних до інформаційних систем залишається тривалим, затратним і схильним до помилок [20].

Було розглянуто типові особливості ТТН, такі як варіативність шаблонів, різнорідність шрифтів, неоднорідна якість зображень, що ускладнює застосування класичних систем оптичного розпізнавання. Окрему увагу приділено сучасним підходам до обробки таких документів, зокрема системам з використанням глибокого навчання, механізмів інтерактивного донавчання та автоматичної посткорекції.

Проведено порівняльний аналіз наявних інструментів. Також розглянуто низку наукових праць, які описують практичні реалізації алгоритмів для роботи з фінансовими документами. Аналіз показав, що для забезпечення високої точності та адаптивності необхідно поєднувати різні техніки: попередню обробку зображень, динамічну сегментацію, навчання на власних шаблонах і подальшу перевірку результатів.

Тому було сформульовано такі вимоги до майбутньої системи: вона має бути гнучкою, адаптованою до різних форматів ТТН, підтримувати механізм швидкого навчання на нових шаблонах і забезпечувати інтеграцію з обліковими системами підприємств.

2 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ОЦИФРУВАННЯ ПАПЕРОВИХ ДОКУМЕНТІВ

2.1 Методи для вирішення задачі

У сучасному цифровому середовищі однією із ключових задач для підприємств різного масштабу стало ефективне та надійне опрацювання документів, особливо паперових чи сканованих. Процес обробки таких документів вручну займає значні ресурси, включаючи час, людські зусилля та ймовірність помилок [3]. У зв’язку з цим методи, що дозволяють автоматизувати витяг даних з документів, стали предметом інтенсивного дослідження та впровадження в бізнес-середовище.

У межах запропонованого проєкту було розглянуто декілька ефективних підходів до розв’язання проблеми автоматичного розпізнавання реквізитів документів. Основна мета полягає у створенні системи, яка дозволяє автоматично обробляти паперові рахунки, витягувати з них ключову інформацію (наприклад, назви товарів, кількість, ціну, постачальника тощо) та зберігати її у структурованому вигляді в базі даних. Такий підхід дозволяє суттєво зменшити ручну працю і прискорити облік у підприємствах, а також зменшує кількість людських помилок, покращує якість ведення бухгалтерського і товарного обліку [14].

Першочергово була проаналізована структура бізнес-процесів, пов’язаних з отриманням, обробкою та збереженням рахунків. У типовій ситуації, замовник формує заявку на товар, яка надсилається постачальнику. У відповідь постачальник надсилає рахунок, який містить опис товару, вартість і кількість. У реальному середовищі такі рахунки часто надаються у вигляді сканованих копій або фотографій, тобто зображень. Саме це створює виклик, оскільки машина не здатна «прочитати» зображення так, як це робить людина. Для забезпечення такої здатності потрібне багаторівневе програмне рішення, що поєднує обробку зображень, розпізнавання тексту та аналіз змісту документа.

Для того щоб машина могла обробляти такі документи, було запропоновано багаторівневу методику, яка поєднує класичні методи обробки зображень із сучасними підходами до інтерпретації тексту. У рамках даного проекту запропоновано розробити систему, що виконує декілька послідовних етапів: спочатку документ оцифровується, потім за допомогою відповідних алгоритмів розпізнаються символи, слова та структура документа [12]. Далі відбувається аналіз змісту тексту, витяг необхідних реквізитів і їх збереження у формалізованій формі. Система будується таким чином, щоб не лише розпізнавати текст, але й правильно інтерпретувати контекст, наприклад, розрізняти ціну товару від його кількості чи назви.

Однією з основних технологій, які використовуються у даному проекті, є технологія оптичного розпізнавання символів (OCR). Ця технологія дозволяє перетворити графічне зображення на текстовий вміст. OCR не є новою технологією, однак її ефективність значно зросла із впровадженням методів штучного інтелекту [7]. Сучасні системи OCR вже не просто читають текст, а можуть адаптуватися до різних шрифтів, нахилу зображення, якості скану тощо. Завдяки цьому вдалося підвищити точність і надійність системи. Більше того, сучасні OCR-системи здатні навчатися на прикладах і вдосконалювати свої результати з часом. Це дозволяє значно скоротити час адаптації системи до нових типів документів і зменшити кількість помилок у розпізнаванні.

На етапі попередньої обробки зображення застосовуються методи, що дозволяють покращити якість вхідного зображення. Це зокрема, вирівнювання, контрастування, фільтрація шумів, видалення фону або сторонніх елементів, які можуть заважати аналізу [8]. Ці процедури дозволяють збільшити точність розпізнавання, особливо у випадках, коли зображення зроблене в неналежних умовах або на пристроях з низькою роздільною здатністю. Якість вхідного зображення має критичне значення для подальших етапів обробки, оскільки навіть найсучасніші алгоритми не здатні компенсувати повністю зашумлені або змазані документи.

Після того як текст розпізнано, важливим етапом є його семантичний аналіз [21]. Тут система повинна не лише бачити слова, а й розуміти, які з них є значущими для подальшого збереження. Наприклад, фраза "кількість: 10" повинна бути розпізнана як конкретне значення, яке стосується певного товару. Для цього використовуються шаблони і логіка, яка дозволяє визначити, де саме на документі розташовані ключові поля. Також система повинна вміти працювати з різноманіттям форматів рахунків, які можуть сильно відрізнятись залежно від постачальника. Семантичний аналіз, у свою чергу, доповнюється можливістю використання словників, базових правил логічного аналізу тексту та статистичних моделей, що дозволяє виявляти типові формати розміщення реквізитів [22].

Ще одним важливим етапом у побудові такої системи є нормалізація даних. Наприклад, ціна може бути вказана у різних форматах: з пробілами, роздільниками, символами валюти. Для забезпечення точності обліку ці значення повинні бути приведені до єдиного формату. Також у системі передбачено можливість розпізнавання дати, назви компанії постачальника, номеру телефона постачальника, ЄДРПОУ постачальника, назви товару, одиниці виміру, кількості, ціни товару, загальної вартості замовлення тощо. Це дозволяє побудувати єдину структуру бази даних, що у свою чергу спрощує пошук, аналіз та обробку отриманих відомостей у майбутньому. Також у нормалізації даних використовуються техніки перетворення регістрів, форматів чисел, стандартизації назв одиниць виміру та валютних символів.

У деяких випадках, коли документ не піддається повному автоматичному розпізнаванню, система дозволяє користувачу вручну верифікувати або виправити дані. Це забезпечує додаткову гнучкість і гарантує коректність збереженої інформації. Також можливе навчання системи на прикладах – якщо користувач декілька разів виправляє однакову помилку, система в подальшому буде враховувати ці корективи. Саме здатність до самонавчання є надзвичайно цінною характеристикою таких систем, адже в реальному середовищі користувачі стикаються з різноманітними, іноді непередбачуваними форматами документів.

У рамках даного проекту система реалізована за принципом шаблонного підходу. Це означає, що під час первинної обробки ТТН від певного постачальника система позначає ключові поля (наприклад, номер, дата, постачальник, найменування товарів, сума тощо), після чого зберігає шаблон цього документа. У майбутньому при повторному надходженні ТТН від того ж постачальника, система автоматично розпізнає відповідний шаблон і виконує витяг даних без додаткових налаштувань. Такий підхід забезпечить високу точність розпізнавання на знайомих шаблонах. Для роботи з іншими постачальниками система потребує навчання на їхніх документах, після чого також починає працювати в автоматичному режимі [23].

Загалом, запропоновані методи поєднують простоту впровадження, ефективність та можливість поступового вдосконалення системи відповідно до потреб користувача. Розробка таких рішень відповідає на актуальні виклики цифровізації документообігу і дає можливість підприємствам підвищити свою продуктивність, зменшити витрати часу та ресурсів, і одночасно покращити точність обробки документації. Саме поєднання класичних підходів і новітніх технологій машинного навчання дозволяє побудувати адаптивну, гнучку і розширювану систему автоматичного розпізнавання і обробки документів, яка здатна ефективно функціонувати в умовах реального бізнес-середовища.

2.2 Технології розробки системи

Розробка інформаційної системи, орієнтованої на автоматизоване розпізнавання реквізитів документів, вимагає ретельно продуманого вибору програмних інструментів. Від правильності цього вибору залежить не лише технічна реалізація окремих компонентів, але й загальна ефективність, надійність, масштабованість та стійкість системи до помилок. Враховуючи специфіку предметної області, було здійснено поглиблений аналіз можливих варіантів реалізації кожного з компонентів: починаючи від мови програмування і закінчуючи

базами даних та модулями обробки зображень. У цьому підрозділі викладено міркування щодо вибору технологій, а також порівняльний аналіз альтернатив.

2.2.1 Вибір мови програмування

В основі створення системи лежить об'єктно-орієнтована мова програмування Java [24]. Історично Java зарекомендувала себе як одна з найстабільніших та наймасштабованіших мов, що застосовується у проектах різного масштабу – від невеликих настільних додатків до розподілених корпоративних систем. Її гасло "write once, run anywhere", тобто "написав один раз – працює скрізь" стало основою кросплатформеного підходу, що дозволяє запускати програми на будь-якій операційній системі, де встановлено віртуальну машину Java.

Ключовою перевагою Java є її стабільна та перевірена часом віртуальна машина (JVM), яка виконує код незалежно від апаратної платформи. Це особливо актуально у контексті проєкту, де передбачається розгортання системи як на персональних комп’ютерах, так і потенційно на серверних рішеннях або навіть у хмарному середовищі. Високий рівень безпеки, контроль доступу, надійна система управління пам’яттю та велика кількість бібліотек роблять Java привабливою для критично важливих застосунків.

З практичної точки зору, мова Java також забезпечує високу продуктивність під час реалізації багатопотокових обчислень. Це критично важливо при роботі з обробкою зображень, особливо в тих випадках, коли потрібно одночасно розпізнавати дані з кількох документів. Інтеграція багатопотоковості дозволяє скористатися перевагами сучасних багатоядерних процесорів, що значно прискорює обробку. Крім того, у Java є розвинений механізм винятків, який дозволяє ефективно обробляти помилки та виключення, що виникають у процесі виконання, не порушуючи при цьому цілісності виконання інших модулів.

Важливо також зазначити, що для Java існує величезна кількість документації, як офіційної, так і створеної спільнотою, а також десятки тисяч

відкритих проектів з прикладами реалізації схожих рішень. Це значно полегшує як початкову реалізацію системи, так і її подальшу підтримку, оскільки нові розробники можуть швидко ознайомитися з проектом та долучитися до його вдосконалення. Таким чином, Java виявляється не лише ефективним, але й стратегічно вигідним вибором для довготривалих проєктів.

2.2.2 Вибір бібліотек та інструментів розпізнавання

Для реалізації основного функціоналу – розпізнавання тексту з документів було обрано бібліотеку Nicomsoft OCR. Ця бібліотека вирізняється високою точністю розпізнавання, стабільністю роботи з кирилицею, зокрема українською мовою, що є критично важливим для проєктів орієнтованих на державні установи чи фінансові компанії [25]. На відміну від багатьох безкоштовних систем, які потребують додаткового налаштування та тренування локальних моделей, Nicomsoft OCR демонструє якість «з коробки», скорочуючи час впровадження системи [26].

Ще однією вирішальною перевагою даної бібліотеки є можливість локального розгортання без залучення зовнішніх веб-сервісів. Це гарантує повну конфіденційність оброблених даних та мінімізує ризики сторонніх витоків інформації, що особливо важливо при роботі з персональними або фінансовими документами. Такий підхід відповідає світовим стандартам захисту даних, включаючи GDPR та національні вимоги.

Nicomsoft OCR дозволяє гнучко налаштовувати зони розпізнавання, створювати шаблони для різних типів бланків, що суттєво підвищує точність і швидкість обробки. Інтерфейс API бібліотеки написаний на Java, що забезпечило безшовну інтеграцію з основним кодом системи без необхідності проміжних протоколів.

У табл. 2.1 наведено стислий огляд ключових класів та основних методів, що використовуються під час інтеграції з бібліотекою Nicomsoft OCR.

Таблиця 2.1 – Опис класів та основних методів Nicomsoft OCR

Клас	**Опис**	**Основні методи**
Engine	Головний клас для управління процесом OCR. Завантажує бінарні модулі, ініціалізує систему, проводить розпізнавання.	initialize() – ініціалізація движка з заданими параметрами; setConfig(HCFG) – завантаження конфігурації; addImage(HIMG) – додавання зображення для обробки; recognize() – запуск процесу розпізнавання; getResult() – отримання текстового результату.
HCFG	Клас конфігурації OCR. Містить параметри обробки, шляхи до ресурсів, налаштування мови та зон.	setLanguage(String) – вибір мови розпізнавання; setPageSegMode(int) – режим сегментації сторінки; setRecognitionZones(List<Rectangle>) – визначення зон розпізнавання.
HIMG	Карта зображення, призначена для OCR. Підтримує завантаження з файлу, масиву байтів, пам’яті.	loadFromFile(String) – завантаження зображення з диску; loadFromMemory(byte[]) – завантаження з масиву байтів; getWidth()/getHeight() – отримання розмірів зображення.
HBLK	Представляє блок тексту, визначений під час зонування та сегментації.	getBoundingBox() – координати блоку; extractText() – отримання тексту з блоку.
HSCAN	Службовий клас для сканування документів.	scanToHIMG() – сканування і повернення HIMG-об’єкта; selectDevice(String) – вибір сканера.
HSVR	Клас для збереження результатів розпізнавання у різні формати (TXT, PDF, XML).	saveAsTXT(String) – збереження як текстовий файл; saveAsPDF(String) – збереження у PDF з підтримкою пошуку; saveAsXML(String) – збереження з розміткою.

Кінець таблиці 2.1

HOCR	Модуль виклику алгоритму розпізнавання символів.	run() – виконання OCR на заданому зображенні; getHOCR() – отримання HOCR-вихідних даних для подальшого аналізу.
Constant	Набір констант для налаштувань OCR.	–
Error	Опис кодів та повідомлень про помилки.	getErrorCode() – отримання коду помилки; getErrorMessage() – читабельне повідомлення.

Для реалізації графічного інтерфейсу користувача застосована бібліотека Swing – перевірений часом засіб створення інтерфейсів у Java. Swing підтримує побудову як простих діалогових вікон, так і складних багато-віконних додатків з реактивною обробкою подій. Завдяки цьому ми зрозробимо інтуїтивно зрозумілий інтерфейс, у якому оператор може обирати файл документа, навчати систему, переглядати проміжні результати та вносити корекції.

У якості допоміжних технологій використано Apache Commons IO, що спростило роботу з файловою системою, та Apache POI для створення звітів і експорту результатів у формати Microsoft Excel. Для організованого логування подій, помилок та інформаційних повідомлень застосовано Log4j, що дало можливість вести детальний аудит роботи системи та прискорити діагностику можливих несправностей.

2.2.3 Вибір системи управління базами даних

Реляційну СУБД для зберігання метаданих документів, інформацію про користувачів, шаблонів реквізитів та історії обробки обрано Firebird SQL [27]. Ця система управління базами даних відзначається можливістю працювати у вбудованому режимі без необхідності окремого серверного процесу, що спрощує розгортання і підтримку застосунку. Для невеликих і середніх обсягів інформації

Firebird демонструє чудову продуктивність, забезпечуючи швидке виконання як простих SELECT-запитів, так і складних JOIN-операцій.

Вбудована підтримка SQL-92, можливість створення тригерів та збережених процедур дозволяє реалізувати частину бізнес-логіки безпосередньо на рівні бази даних [28]. GBAK надає інструмент для створення резервних копій і відновлення сховища з мінімальним простоєм системи. Безпека доступу реалізується через визначення ролей та привілеїв на рівні таблиць і стовпців, а також можливість шифрування файлів бази даних на диску.

Інтеграція Firebird з Java здійснюється через офіційний спеціальний драйвер, що підтримує пул підключень, автоматичне відновлення з'єднання та налаштування тайм-аутів. Також у середовищі Firebird доступні інструменти моніторингу виконання запитів, які допомагають аналізувати план виконання та виявляти вузькі місця в індексах чи складних запитах. Завдяки своїй кросплатформенності та стійкості до збоїв Firebird SQL є надійним сховищем даних для систем, що обробляють конфіденційну інформацію. Цей вибір обґрунтовано поєднанням простоти в адмініструванні, високої продуктивності на відносно малих об'ємах даних та можливістю вбудованого розгортання поруч із основним додатком.

Firebird SQL не потребує окремого серверного оточення: база даних у вигляді одного або кількох файлів може зберігатися в каталозі програми, мінімізуючи складність розгортання та оновлення. При цьому СУБД підтримує повну транзакційність (здатність виконувати набір операцій як єдине ціле), що гарантує цілісність даних навіть у випадку аварійного завершення програми. Збережені процедури, тригери та механізми контролю доступу дозволяють реалізувати складну бізнес-логіку безпосередньо на рівні СУБД.

2.2.4 Технологічна інтеграція

Компоненти системи реалізовані у вигляді окремих модулів, кожен з яких виконує певну функцію – обробку зображення, розпізнавання тексту, взаємодію з

базою даних або виведення результату. Такий підхід забезпечує модульність архітектури та полегшує масштабування системи у разі необхідності розширення її функціоналу. Модулі взаємодіють через чітко визначені програмні інтерфейси, що спрощує підтримку та тестування системи.

У разі потреби система може бути інтегрована з іншими корпоративними рішеннями. Це реалізується або шляхом прямого доступу до бази даних, або через реалізацію прикладного інтерфейсу (API), що дозволяє обмінюватися даними між системами у реальному часі. Такий підхід забезпечує масштабованість та гнучкість інформаційної взаємодії між різними інформаційними платформами.

2.2.5 Порівняння з альтернативними технологіями

В процесі вибору технологій розглядались й альтернативні варіанти. Наприклад, використання мови програмування C# на платформі .NET могло б забезпечити подібну функціональність, однак така реалізація була б обмежена платформою Windows, що суперечить вимогам кросплатформності. Python, хоча і є гнучкою та популярною мовою з багатим набором бібліотек для обробки зображень, менш ефективно працює у середовищах, де потрібна висока продуктивність у багатопотокових задачах (див. табл. 2.2).

Таблиця 2.2 – Порівняння мов програмування

	Java	**C#**	**Python**
Кросплатформеність	Висока	Здебільшого Windows	Висока
Продуктивність	Висока	Висока	Середня
Швидкість виконання	Вища за Python	Вища за Python	Найнижча
Складність розробки	Середня	Середня	Висока
Багатопотоковість	Розвинена	Розвинена	Обмежена
Інтерфейс користувача	Swing, JavaFX	Windows Forms, WPF	Tkinter, PyQt, Kivy

Щодо СУБД, то тут розглядалися варіанти на зразок MySQL та PostgreSQL. Проте вони виявились надмірно потужними для завдань невеликої локальної

системи, яка не потребує масштабованості на рівні великих серверних рішень. Firebird натомість, забезпечує необхідну функціональність при збереженні простоти та ефективності у використанні (див. табл. 2.3).

Таблиця 2.3 – Порівняння систем баз даних

	Firebird SQL	**MySQL**	**PostgreSQL**
Складність адміністрування	Низька	Середня	Висока
Потреба у сервері	Може працювати вбудовано	Потребує сервера	Потребує сервера
Продуктивність на малих об'ємах даних	Висока	Середня	Середня

Таким чином, інтеграція Java, Nicomsoft OCR, Swing, Firebird SQL та допоміжних бібліотек створила збалансований технологічний стек, який задовольняє вимоги проекту щодо продуктивності, надійності, безпеки та кросплатформності.

Висновки до розділу 2

У другому розділі було детально розглянуто предметну область автоматизованого розпізнавання реквізитів документів та проведено комплексний аналіз технологічних рішень для розробки інформаційної системи і встановлено ключові завдання. Для кожного з компонентів системи було обґрунтовано вибір інструментів, що забезпечують максимальну ефективність та відповідність бізнес-вимогам.

Обрано мову програмування Java, яка завдяки стабільності, широкій екосистемі бібліотек та можливостям багатопотоковості відповідає високим вимогам до продуктивності та масштабованості. Переваги Java підтверджені як у сценаріях локального розгортання, так і в умовах інтеграції із хмарними або корпоративними середовищами.

Для ядра розпізнавання тексту ми обрали бібліотеку Nicomsoft OCR. Вона гарантує високий рівень точності обробки, адаптована до кирилиці і не потребує зовнішніх сервісів, що забезпечує конфіденційність даних і скорочує час інтеграції.

Для побудови клієнтського інтерфейсу обрано бібліотеку Swing – перевірене рішення для створення додатків на Java. Як сховище метаданих, шаблонів і результатів OCR обрана СУБД Firebird SQL. Її переваги – відсутність необхідності у зовнішній інсталяції серверного компоненту та гарантоване збереження цілісності даних у разі збоїв. Додаткові інструменти – Apache Commons IO, Apache POI і Log4j 2 реалізують завдання підтримки файлових операцій, генерації звітів у форматах Microsoft Office і гнучкого логування.

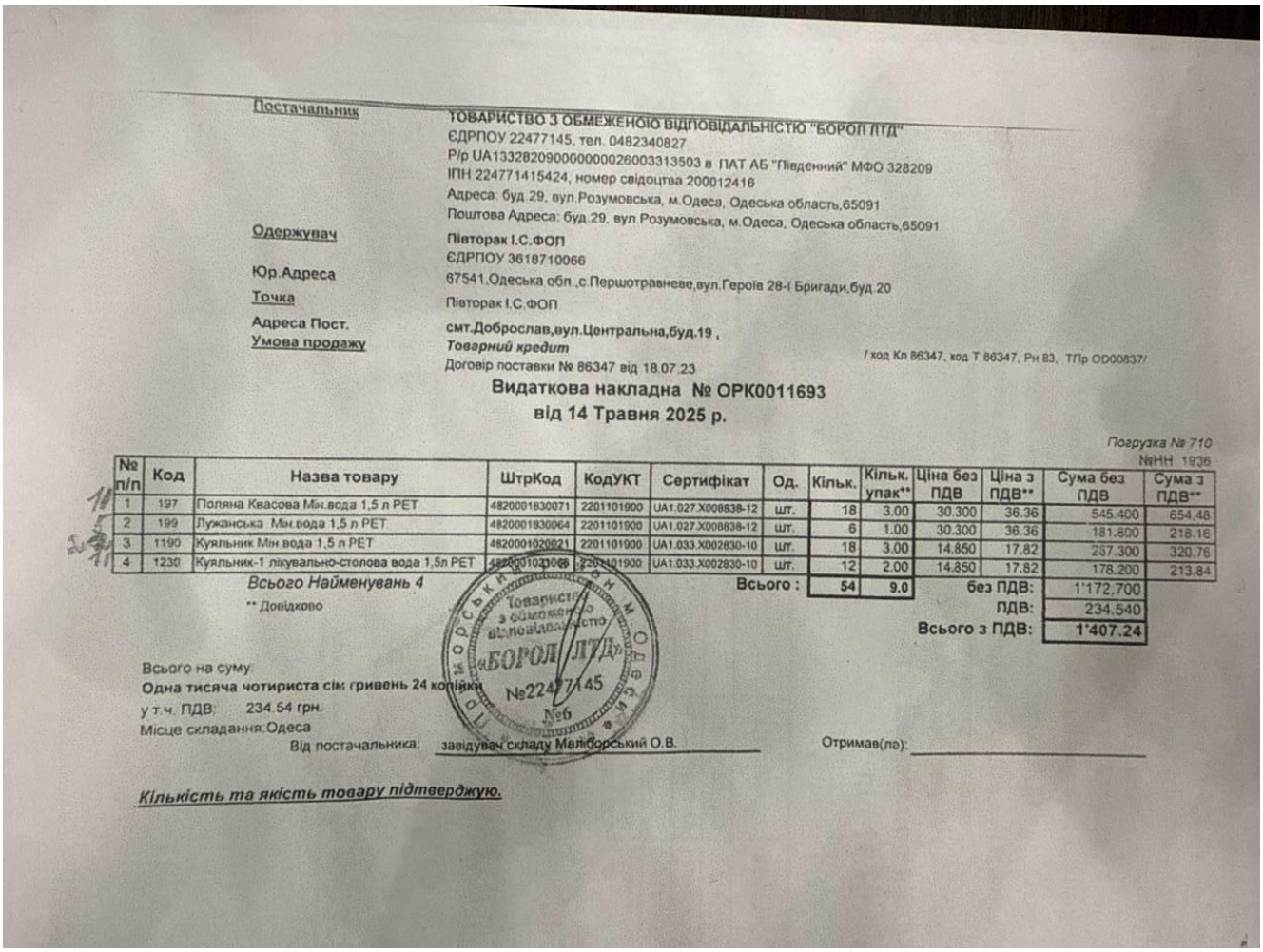
Таким чином, результати розділу доводять, що стратегія поєднання Java з Nicomsoft OCR, вбудованою БД Firebird та допоміжними технологіями є оптимальною для створення системи автоматизованого розпізнавання реквізитів ТТН. Запропоноване рішення забезпечує високу точність, надійність, захист даних та здатне масштабуватися, відповідаючи як поточним, так і перспективним бізнес-вимогам.

3 ПРОЄКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ФІНАНСОВИХ ДОКУМЕНТІВ

3.1 Опис вхідних даних та структури системи

В процесі функціонування системи автоматизованого розпізнавання реквізитів із зображень ТТН, головну роль грає підготовка, обробка, класифікація та аналіз вхідних даних. Вхідними даними розробленої інформаційної системи є фотографії паперових документів, отримані переважно за допомогою вбудованих камер мобільних пристроїв, планшетів або традиційних офісних сканерів.

Через відсутність загального стандарту цифрового подання ТТН, документи можуть значно відрізнятися структурою, стилем оформлення, шрифтами, вирівнюванням елементів, кількістю полів та їх розташуванням (див. рис. 3.1) [4]. На практиці в документообігу підприємств часто спостерігається сильна різноманітність шаблонів, кожен з яких має свої особливості. Це створює додатковий виклик для автоматизованого розпізнавання і вимагає гнучкої побудови системи [11].



Постачальник ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "БОРОЛ ЛТД"
ЄДРПОУ 22477145, тел. 0482340827
Р/р UA133282090000000026003313503 в ПАТ АБ "Південний" МФО 328209
ІПН 224771415424, номер свідоцтва 200012416
Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091
Поштова Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091
Одержувач Півторак І.С.ФОП
ЄДРПОУ 3618710066
Юр.Адреса 67541,Одеська обл.,с.Першотравневе,вул.Героїв 28-ї Бригади,буд.20
Точка Півторак І.С.ФОП
Адреса Пост. смт.Доброслав,вул.Центральна,буд.19 ,
Умова продажу *Товарний кредит*
Договір поставки № 86347 від 18.07.23
/ код Кл 86347, код Т 86347, Рн 83, ТПр ОД00837/

Видаткова накладна № ОРК0011693
від 14 Травня 2025 р.

Погрузка № 710
№НН 1936

№ п/п	Код	Назва товару	ШтрКод	КодУКТ	Сертифікат	Од.	Кільк.	Кільк. упак**	Ціна без ПДВ	Ціна з ПДВ**	Сума без ПДВ	Сума з ПДВ**
1	197	Поляна Квасова Мін.вода 1,5 л PET	4820001830071	2201101900	UA1.027.X008838-12	шт.	18	3.00	30.300	36.36	545.400	654.48
2	199	Лужанська Мін.вода 1,5 л PET	4820001830064	2201101900	UA1.027.X008838-12	шт.	6	1.00	30.300	36.36	181.800	218.16
3	1190	Куяльник Мін.вода 1,5 л PET	4820001020021	2201101900	UA1.033.X002830-10	шт.	18	3.00	14.850	17.82	267.300	320.76
4	1230	Куяльник-1 лікувально-столова вода 1,5л PET	4820001020069	2201101900	UA1.033.X002830-10	шт.	12	2.00	14.850	17.82	178.200	213.84
		Всього Найменувань 4					Всього :	54	9.0		без ПДВ:	1'172.700
		** Довідково									ПДВ:	234.540
											Всього з ПДВ:	1'407.24

Товариство з обмеженою відповідальністю «БОРОЛ ЛТД» №22477145

Всього на суму:
Одна тисяча чотириста сім гривень 24 копійки
у т.ч. ПДВ: 234.54 грн.
Місце складання:Одеса
Від постачальника: завідувач складу Маліборський О.В. Отримав(ла):

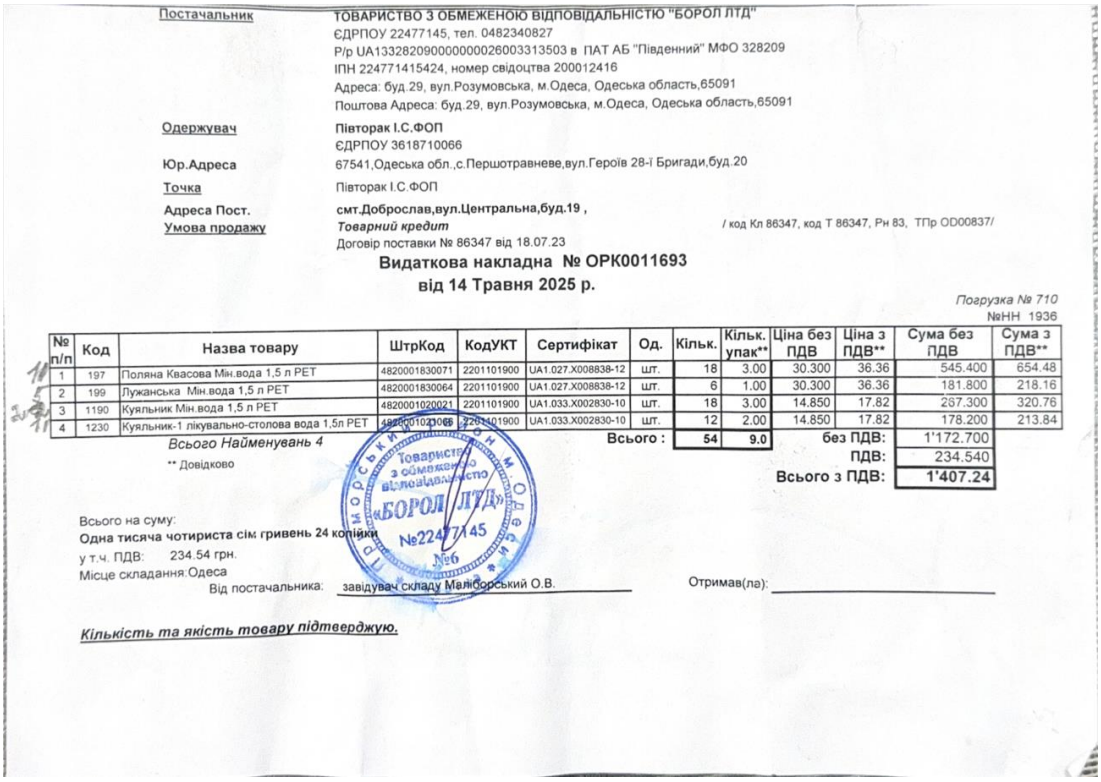
Кількість та якість товару підтверджую.

Рисунок 3.1 – Приклад зображення ТТН

Для забезпечення ефективності системи, особлива увага приділяється роботі з фото, знятими в польових умовах. Наприклад, на складах, у пунктах відвантаження або під час транспортування. Такі фото досить часто мають низьку якість, недостатню освітленість, спотворення геометрії або сторонні об’єкти в кадрі. У зв’язку з цим система повинна автоматично адаптувати зображення до прийнятного формату: вирівнювати, очищати від шумів, підвищувати контрастність та коригувати викривлення, спричинені неправильним кутом зйомки [29].

Попередня обробка зображення виконує роль фільтрації та підготовки. На цьому етапі здійснюється приведення зображення до стандартного розміру, очищення від зайвих деталей, виправлення перекосів, спричинених неправильним нахилом під час фотографування, а також визначення положення документа, щоб

він був правильно орієнтований для подальшої обробки (див. рис. 3.2). Всі ці дії спрямовані на покращення якості вхідного матеріалу, що є вирішальним фактором для подальшої точності OCR-аналізу. Висока якість попередньої обробки дозволяє компенсувати недоліки апаратного забезпечення користувача.



Постачальник ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "БОРОЛ ЛТД"
ЄДРПОУ 22477145, тел. 0482340827
Р/р UA133282090000000026003313503 в ПАТ АБ "Південний" МФО 328209
ІПН 224771415424, номер свідоцтва 200012416
Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091
Поштова Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091

Одержувач Півторак І.С.ФОП
ЄДРПОУ 3618710066
Юр.Адреса 67541,Одеська обл.,с.Першотравневе,вул.Героїв 28-ї Бригади,буд.20
Точка Півторак І.С.ФОП
Адреса Пост. смт.Доброслав,вул.Центральна,буд.19 ,
Умова продажу *Товарний кредит* / код Кл 86347, код Т 86347, Рн 83, ТПр OD00837/
Договір поставки № 86347 від 18.07.23

Видаткова накладна № ОРК0011693
від 14 Травня 2025 р.

Погрузка № 710
№НН 1936

№ п/п	Код	Назва товару	ШтрКод	КодУКТ	Сертифікат	Од.	Кільк.	Кільк. упак**	Ціна без ПДВ	Ціна з ПДВ**	Сума без ПДВ	Сума з ПДВ**
1	197	Поляна Квасова Мін.вода 1,5 л PET	4820001830071	2201101900	UA1.027.X008838-12	шт.	18	3.00	30.300	36.36	545.400	654.48
2	199	Лужанська Мін.вода 1,5 л PET	4820001830064	2201101900	UA1.027.X008838-12	шт.	6	1.00	30.300	36.36	181.800	218.16
3	1190	Куяльник Мін.вода 1,5 л PET	4820001020021	2201101900	UA1.033.X002830-10	шт.	18	3.00	14.850	17.82	267.300	320.76
4	1230	Куяльник-1 лікувально-столова вода 1,5л PET	4820001020069	2201101900	UA1.033.X002830-10	шт.	12	2.00	14.850	17.82	178.200	213.84
		Всього Найменувань 4				Всього :	54	9.0		без ПДВ:	1'172.700	
		** Довідково								ПДВ:	234.540	
										Всього з ПДВ:	1'407.24	

Товариство з обмеженою відповідальністю «БОРОЛ ЛТД» №22477145 №6

Всього на суму:
Одна тисяча чотириста сім гривень 24 копійки
у т.ч. ПДВ: 234.54 грн.
Місце складання:Одеса
Від постачальника: завідувач складу Маліборський О.В. Отримав(ла):

Кількість та якість товару підтверджую.

Рисунок 3.2 – Зображення ТТН після обробки

На наступному етапі відбувається семантична класифікація елементів зображення. Це означає визначення інформаційних зон, таких як «шапка документа», «таблична частина», «підсумковий фінансовий блок» [30]. Залежно від ідентифікованої зони до неї застосовуються відповідні правила розпізнавання, включно з підключенням окремих словників, шаблонів і граматичних моделей. Це дозволяє підвищити точність витягу реквізитів у складних випадках, коли один і той самий символ або фрагмент тексту може трактуватися по-різному в залежності від контексту.

Далі відбувається етап вилучення ключових реквізитів, які мають важливе значення для логістичних, фінансових та облікових процесів. Для кожного документа система ідентифікує набір полів, що є критично важливими для

подальшого збереження в базі даних. Перелік таких реквізитів був сформований на основі аналізу типових форм ТТН, законодавчих вимог та потреб користувачів.

До основних реквізитів, які мають бути вилучені з кожної товарно-транспортної накладної, належать:

- номер ТТН;
- повна назва фірми постачальника;
- код ЄДРПОУ;
- банківський рахунок у форматі IBAN;
- поштова адреса постачальника;
- номер договору;
- повна назва товару;
- штрих-код товарної позиції;
- одиниця виміру продукції;
- кількість одиниць;
- ціна з урахуванням ПДВ;
- сума з ПДВ по кожній позиції;
- загальна сума замовлення з урахуванням ПДВ;
- ПІБ особи, що відвантажила товар;
- ПІБ особи, що отримала товар.

У структурі ТТН ці реквізити зазвичай поділяються на логічні блоки. У верхній частині документа (так звана «шапка») розміщені загальні відомості про сторони угоди, реквізити, а також дата та номер накладної. Центральну частину займає таблична форма з переліком товарів, де вказані їх найменування, коди, кількість, ціна, одиниці виміру та інші атрибути. Внизу ТТН традиційно розташовуються підсумкові суми та підписи відповідальних осіб.

Архітектура системи побудована за модульним принципом і включає наступні складові: модуль обробки зображень, модуль оптичного розпізнавання тексту, модуль логічного аналізу, блок витягу реквізитів, інтерфейс користувача та модуль взаємодії з базою даних. Кожен компонент виконує окрему функцію,

взаємодіючи з іншими через стандартизовані програмні інтерфейси. Такий підхід забезпечує гнучкість, розширюваність і можливість масштабування системи при зміні вимог або додаванні нових функцій.

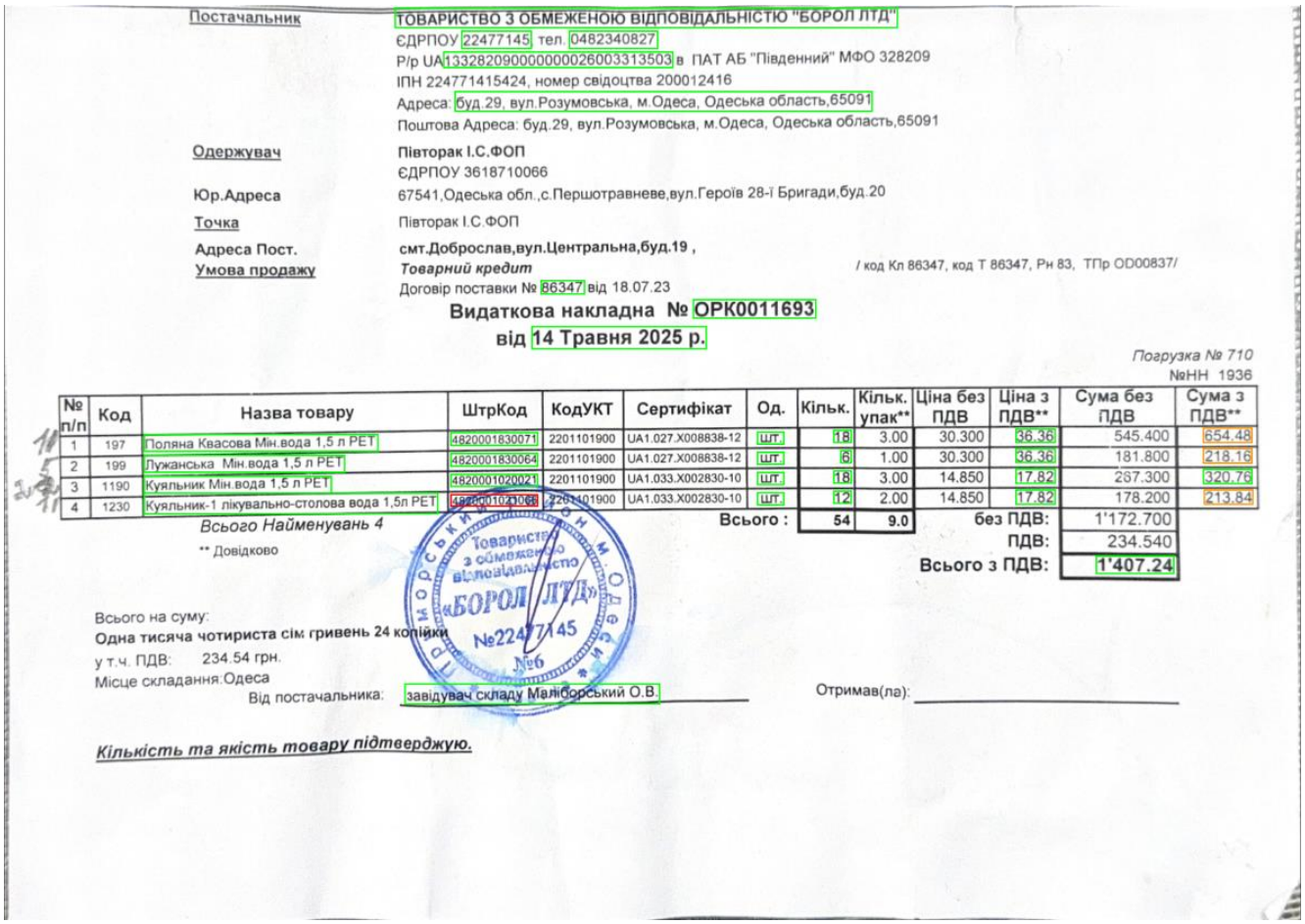
Важливою особливістю є підтримка шаблонного навчання: при першому опрацюванні ТТН певного формату користувач має можливість вказати ключові дані вручну, система шукає розташування полів, після чого зберігає шаблон їх розташування. Надалі для обробки подібних документів застосовується вже відомий шаблон без необхідності повторного налаштування. Це дає змогу суттєво підвищити продуктивність і точність розпізнавання в «реальних умовах».

Для візуальної ілюстрації особливостей вхідних зображень, на рис. 3.3 представлено приклад типового скану ТТН, в якому система позначила розміщення розпізнаних реквізитів.

Зелені зони – повністю розпізнані реквізити, яким не потрібна додаткова перевірка.

Помаранчеві зони – вірогідність правильного розпізнання реквізиту менша за 80% і потребує додаткової перевірки оператором.

Червоні зони – не розпізнаний реквізит, вимагає ручного введення оператором.



Постачальник ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "БОРОЛ ЛТД"
ЄДРПОУ 22477145 тел. 0482340827
Р/р UA133282090000000026003313503 в ПАТ АБ "Південний" МФО 328209
ІПН 224771415424, номер свідоцтва 200012416
Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091
Поштова Адреса: буд.29, вул.Розумовська, м.Одеса, Одеська область,65091

Одержувач Півторак І.С.ФОП
ЄДРПОУ 3618710066
Юр.Адреса 67541,Одеська обл.,с.Першотравневе,вул.Героїв 28-ї Бригади,буд.20
Точка Півторак І.С.ФОП
Адреса Пост. смт.Доброслав,вул.Центральна,буд.19 ,
Умова продажу *Товарний кредит* / код Кл 86347, код Т 86347, Рн 83, ТПр OD00837/
Договір поставки № 86347 від 18.07.23

Видаткова накладна № ОРК0011693
від 14 Травня 2025 р.

Погрузка № 710
№НН 1936

№ п/п	Код	Назва товару	ШтрКод	КодУКТ	Сертифікат	Од.	Кільк.	Кільк. упак**	Ціна без ПДВ	Ціна з ПДВ**	Сума без ПДВ	Сума з ПДВ**
1	197	Поляна Квасова Мін.вода 1,5 л PET	4820001830071	2201101900	UA1.027.X008838-12	шт	18	3.00	30.300	36.36	545.400	654.48
2	199	Лужанська Мін.вода 1,5 л PET	4820001830064	2201101900	UA1.027.X008838-12	шт	6	1.00	30.300	36.36	181.800	218.16
3	1190	Куяльник Мін.вода 1,5 л PET	4820001020021	2201101900	UA1.033.X002830-10	шт	18	3.00	14.850	17.82	267.300	320.76
4	1230	Куяльник-1 лікувально-столова вода 1,5л PET	4820001020038	2201101900	UA1.033.X002830-10	шт	12	2.00	14.850	17.82	178.200	213.84
						Всього :	54	9.0		без ПДВ:	1'172.700	
										ПДВ:	234.540	
										Всього з ПДВ:	1'407.24	

Всього Найменувань 4
** Довідково

Товариство з обмеженою відповідальністю «БОРОЛ ЛТД» №22477145 №6

Всього на суму:
Одна тисяча чотириста сім гривень 24 копійки
у т.ч. ПДВ: 234.54 грн.
Місце складання:Одеса
Від постачальника: завідувач складу Маліборський О.В. Отримав(ла):

Кількість та якість товару підтверджую.

Рисунок 3.3 – Зображення ТТН з виділеними реквізитами для розпізнавання

Таким чином, побудова системи базується на детальному аналізі вхідного зображення, зонуванні, розпізнаванні тексту, семантичному аналізі та збереженні структурованої інформації у базі даних. Структура системи повністю відповідає вимогам до автоматизованої обробки паперових документів та адаптована для роботи з великою кількістю нестандартизованих шаблонів ТТН, забезпечуючи гнучкість і точність в умовах змінного бізнес-середовища.

3.2 Проєктування інформаційної системи

Етап проєктування інформаційної системи є визначальним для її подальшої ефективності, гнучкості, ергономічності, а також потенціалу масштабування та модернізації. В умовах зростання обсягів документообігу, автоматизація обробки первинної документації, зокрема ТТН, стає необхідністю для операційної

ефективності. Ручна обробка ТТН пов'язана зі значними часовими затратами, ризиком помилок та обмеженими можливостями масштабування [5].

Розроблювана інформаційна система має на меті автоматизацію повного циклу обробки ТТН: від завантаження зображення до витягування реквізитів, їх верифікації та збереження у базі даних. Ключовими цілями є підвищення швидкості та ефективності обробки, забезпечення високої точності даних, адаптація до різних форматів ТТН, надання зручного інтерфейсу та підґрунтя для інтеграції з іншими корпоративними системами.

Архітектура інформаційної системи включає декілька взаємопов'язаних функціональних модулів підсистем, що послідовно обробляють документ. Основними модулями інформаційної системи є:

- модуль попередньої обробки зображень;
- модуль оптичного розпізнавання тексту (OCR);
- модуль семантичного аналізу та вилучення реквізитів;
- модуль керування шаблонами та навчання;
- модуль збереження та обробки даних;
- модуль взаємодії з користувачем (графічний інтерфейс);
- модуль логування, безпеки та обліку подій.

Загальну архітектуру системи, що ілюструє взаємодію цих підсистем, представлено на рис. 3.4.

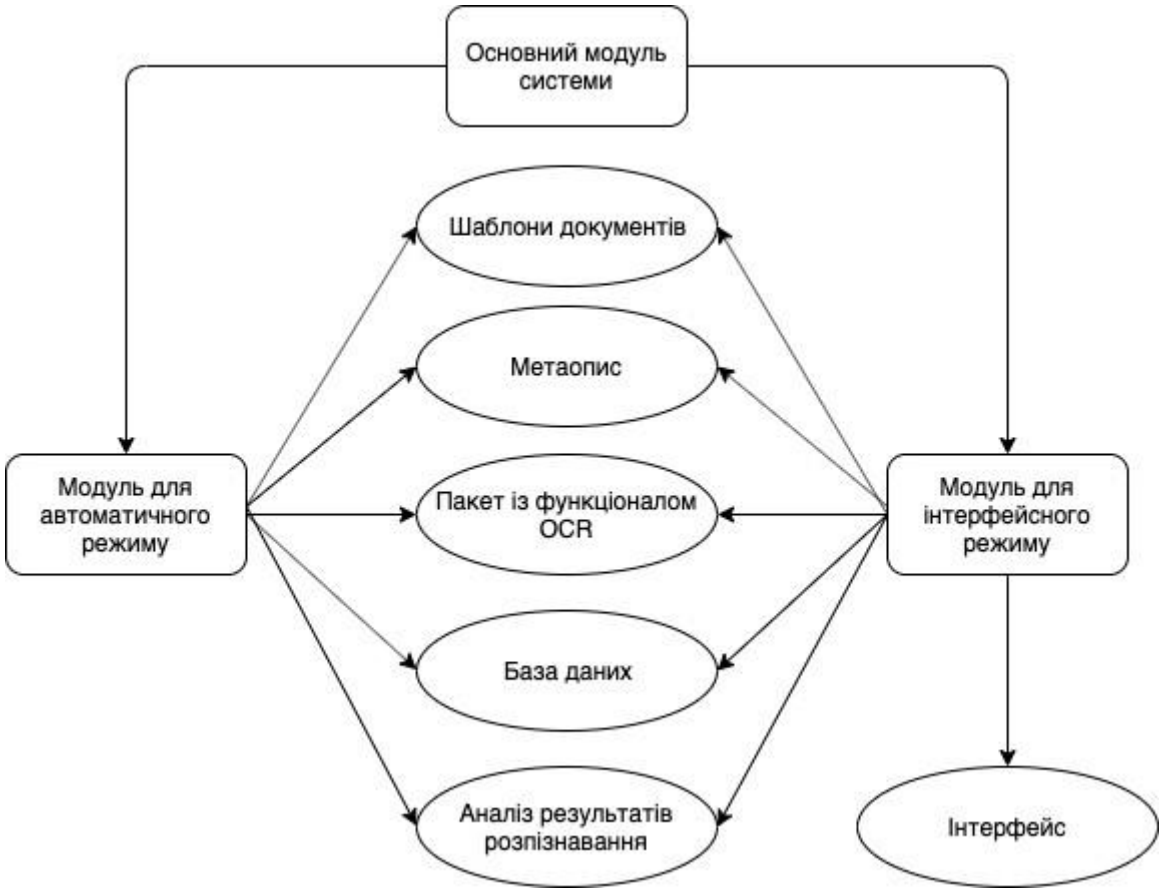


Рисунок 3.4 – Архітектура системи у загальному вигляді

Модуль попередньої обробки зображень відповідає за прийом зображень документів (JPEG, PNG, TIFF, PDF) та їх підготовку до етапу оптичного розпізнавання. Вона вирішує проблеми, пов'язані з різноманітністю форматів та низькою якістю зображень, такі як нерівномірне освітлення, наявність шумів та геометричні спотворення. Основні кроки обробки включають уніфікацію формату, бінаризацію, усунення шумів, вирівнювання геометрії та підвищення контрасту. Результатом роботи підсистеми є очищене зображення, оптимально підготовлене для подальшої обробки.

Модуль оптичного розпізнавання тексту (OCR), реалізована на базі бібліотеки Nicomsoft OCR, виконує перетворення текстової інформації з растрового зображення у цифровий формат [9]. Бібліотека Nicomsoft OCR підтримує українську мову та забезпечує високу точність розпізнавання.

Підсистема ідентифікує символи, слова та рядки, надаючи на виході розпізнаний текст, потенційно з інформацією про координати та рівень впевненості [10].

Модуль семантичного аналізу та вилучення реквізитів здійснює інтелектуальний аналіз текстового потоку, отриманого від OCR-підсистеми, з ключовою метою ідентифікації та подальшого вилучення семантично значущих реквізитів документа. До таких реквізитів належать, наприклад, унікальний номер ТТН, дата її складання, повні найменування та коди ЄДРПОУ, а також числові значення, що відображають суми, кількість, ціну тощо. Процедура аналізу передбачає попереднє логічне зонування документа, тобто його умовний поділ на типові інформаційні блоки, такі як «Шапка документа», «Блок постачальника», «Блок покупця», «Товарна таблиця», «Підсумкові суми» та інші. Для кожної такої зони та для кожного очікуваного реквізиту застосовується набір спеціалізованих правил та евристичних алгоритмів, що включають використання регулярних виразів для пошуку структурованих даних (дат, номерів, кодів), пошук за ключовими словами-маркерами, аналіз просторової близькості текстових фрагментів, а також валідацію типів даних (наприклад, перевірка, чи є витягнуте значення числом або коректною датою). Особлива увага приділяється алгоритмам обробки товарних таблиць, які можуть мати значну варіативність у своїй структурі, включаючи наявність багаторядкових комірок або об'єднаних стовпців [31].

Модуль керування шаблонами та навчання забезпечує адаптивність системи до різних форм ТТН. При первинній обробці документа невідомого раніше формату, користувач вводить вручну ключові дані, після чого система знаходить і запам'ятовує координати полів. Система зберігає ці координати та характеристики як шаблон, який надалі використовується для автоматичного витягування даних з аналогічних документів, що з часом підвищує точність обробки.

Модуль збереження та обробки даних відповідає за надійне зберігання всієї інформації, що генерується та використовується системою, у реляційній базі даних Firebird SQL. Це включає як вихідні зображення документів, так і розпізнані

реквізити, створені шаблони, системні логи та інші метадані. База даних потрібна для реалізації наступних процесів:

- зберігання зображень документів;
- зберігання шаблонів документів;
- збереження розпізнаних реквізитів.

Вибір Firebird SQL обґрунтований такими факторами, як відсутність ліцензійних відрахувань, належна продуктивність та надійність. Структура бази даних, представлена у табл. 3.1, розроблена для ефективного зберігання всієї необхідної інформації.

Таблиця 3.1 – Структура бази даних інформаційної системи

Таблиця	**Призначення**
Documents	Зберігання загальних відомостей про документи (ID, дата, постачальник, статус обробки)
Templates	Координати полів шаблонів, асоційовані з типом документа та постачальником
Requisites	Вилучені реквізити з документів (назва поля, значення, координати)
Users	Дані про користувачів системи (логіни, ролі, права доступу)
Logs	Журнал системних подій, помилок та дій користувачів

Модуль взаємодії з користувачем (UI), реалізована на технології Java Swing. Вона надає користувачам інструменти для завантаження документів, перегляду результатів розпізнавання, верифікації, корекції даних та управління шаблонами. Інтерфейс включає функції відображення зображення, візуального підсвічування реквізитів, панель для редагування даних, інструменти для створення шаблонів та систему сповіщень. Приклад графічного інтерфейсу користувача, що демонструє головне вікно програми з відображенням документа,

розпізнаними зонами, панеллю розпізнаного тексту та панеллю реквізитів, наведено на Рис. 3.5.

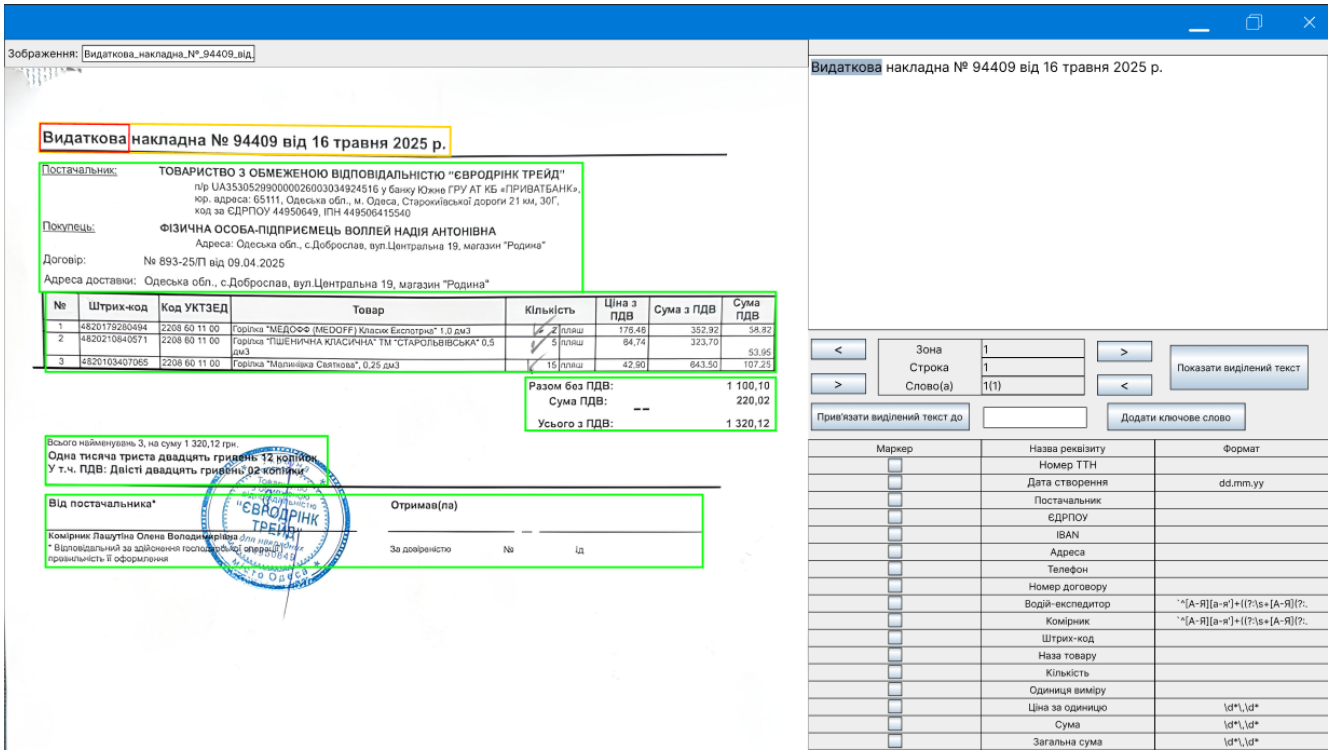


Рисунок 3.5 – Головне вікно програми

Модуль логування, безпеки та обліку подій забезпечує фіксацію всіх ключових операцій, помилок та дій користувачів, що є важливим для діагностики, оцінки продуктивності та аудиту безпеки. Аспекти надійності та безпеки даних також охоплюються цією підсистемою та детальніше розглядаються далі.

Забезпечення високого рівня надійності зберігання даних та контролю безпеки доступу до системи є одним із пріоритетних завдань проектування. З цією метою реалізовано комплексний аудит дій користувачів, в рамках якого всі значущі операції, що виконуються в системі (такі як вхід/вихід, завантаження та модифікація документів, створення або зміна шаблонів, редагування конфігураційних параметрів), детально протоколюються у спеціальному захищеному журналі. Це дозволяє не тільки відстежувати історію роботи з кожним

документом, але й аналізувати активність користувачів для виявлення потенційних загроз або неправомірних дій.

Впроваджено гнучку систему диференційованого доступу до функціональних можливостей системи, що базується на визначенні ролей користувачів (наприклад, Адміністратор, Оператор, Користувач для перегляду). Кожна роль наділяється специфічним набором прав, що чітко обмежує доступ до певних функцій та даних, запобігаючи несанкціонованому втручанню. Безпека бази даних забезпечується використанням стійких механізмів автентифікації, розмежуванням прав доступу на рівні об'єктів СУБД, а також можливістю застосування технологій шифрування для захисту конфіденційної інформації. Невід'ємною частиною забезпечення надійності є регулярне резервне копіювання та тестування процедур відновлення даних, що дозволяє мінімізувати ризики втрати інформації внаслідок апаратних збоїв або інших непередбачених ситуацій. Усі ці аспекти є невід'ємною частиною функціоналу підсистеми логування, безпеки та обліку подій.

При проектуванні системи значна увага також приділялася забезпеченню її інтеграційних можливостей з іншими існуючими або майбутніми корпоративними інформаційними системами. Це може бути реалізовано шляхом надання механізмів експорту оброблених та верифікованих даних у стандартизованих та широко використовуваних форматах, таких як CSV, XML або JSON, що дозволить легко імпортувати їх до систем бухгалтерського обліку, ERP-систем або інших аналітичних платформ. У перспективі також розглядається розробка повноцінного прикладного програмного інтерфейсу (API), який дозволить іншим системам здійснювати програмну взаємодію з системою обробки ТТН в режимі реального часу, наприклад, для автоматизованого завантаження нових документів або отримання актуального статусу їх обробки.

Масштабованість системи забезпечується на кількох рівнях. Передбачена можливість вертикального масштабування шляхом нарощування апаратних ресурсів сервера (збільшення потужності процесора, обсягу оперативної пам'яті,

ємності дискового простору). Завдяки модульній архітектурі та використанню клієнт-серверного режиму роботи СУБД Firebird SQL, створюються також передумови для часткової горизонтальної масштабованості, що може включати розгортання окремих ресурсоємних компонентів, таких як сервіс OCR, на декількох серверах для розподілу навантаження [32]. Структура бази даних та алгоритми обробки запитів оптимізуються для ефективної роботи зі зростаючими обсягами даних, забезпечуючи стабільну продуктивність системи в довгостроковій перспективі.

Спроектована інформаційна система обробки товарно-транспортних накладних являє собою комплексне програмне рішення, що ефективно поєднує сучасні методи цифрової обробки зображень, передові технології оптичного розпізнавання тексту, інтелектуальні алгоритми семантичного аналізу та надійні механізми збереження структурованих даних у захищеному інформаційному середовищі. Система була розроблена з неухильним дотриманням фундаментальних принципів модульності архітектури, гнучкості конфігурації та потенціалу для подальшого масштабування, що в сукупності забезпечує високий рівень зручності її використання, здатність до швидкої адаптації до нових форматів ТТН та широкі можливості для інтеграції з іншими корпоративними інформаційними системами підприємств.

Ретельне та всебічне проектування кожного окремого компонента системи, починаючи від етапу попередньої підготовки та очищення зображень і завершуючи складними механізмами верифікації розпізнаних даних та динамічної шаблонізації, дозволило закласти міцний технологічний фундамент для досягнення високих показників точності розпізнавання. Це у свою чергу, сприяє суттєвому зменшенню обсягу рутинного операційного навантаження на персонал, відповідальний за обробку первинної документації, та перетворенню цього трудомісткого процесу на ефективний, прозорий та повністю контрольований бізнес-процес. Впровадження такої системи в операційну діяльність підприємства здатне значно підвищити загальну ефективність його функціонування, мінімізувати фінансові та часові

ризики, пов'язані з неминучими помилками ручного введення даних, та надати керівництву цінну, оперативну та достовірну фактографічну інформацію для підтримки прийняття своєчасних та обґрунтованих управлінських рішень. Подальший розвиток системи може передбачати розширення її функціональних можливостей за рахунок інтеграції нових технологій та алгоритмів, спрямованих на досягнення ще вищого рівня автоматизації та інтелектуалізації процесів електронного документообігу.

3.3 Функціонал пошуку, фільтрації та класифікації документів

Для забезпечення ефективної роботи користувачів з великими обсягами оброблених товарно-транспортних накладних, система надає можливості пошуку, фільтрації та класифікації документів. Ці функції є невід’ємною частиною "Модуля взаємодії з користувачем (UI)" та тісно інтегровані з "Модулем збереження та обробки даних", дозволяючи оперативно знаходити необхідні ТТН на основі різних критеріїв. Основними інструментами класифікації є фільтрація за постачальником та за датою створення ТТН. Приклад інтерфейсу, що демонструє табличне представлення списку ТТН з елементами керування для фільтрації та пошуку, наведено на Рис. 3.6. На цьому рисунку видно розташовані елементи "Всі постачальники" (випадаючий список для вибору критерію фільтрації), поле "Фільтр" (для введення значення фільтра) та поле "Пошук" (для повнотекстового пошуку).

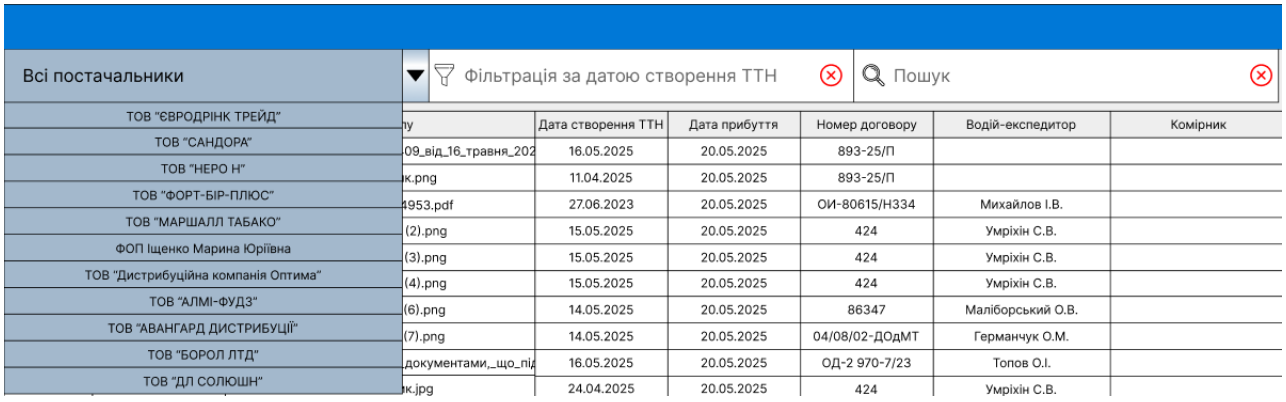


Рисунок 3.6 – Меню фільтрацій та повнотекстового пошуку

Класифікація та фільтрація ТТН за постачальником дозволяє користувачеві швидко відібрати та переглянути всі документи, що надійшли від конкретного постачальника. Ідентифікація постачальника відбувається на етапі розпізнавання документа (вилучення таких реквізитів, як назва компанії) та під час створення або застосування шаблону, де постачальник є одним із ключових ідентифікаторів [2]. Збережена в базі даних інформація про постачальника використовується для побудови відповідних запитів. В графічному інтерфейсі користувача, як показано на Рис. 3.6, передбачено випадаючий список, який дозволяє обрати конкретного постачальника. Після вибору постачальника (наприклад: САНДОРА), система динамічно оновлює список відображуваних документів, залишаючи лише ті, що відповідають запиту.

Фільтрація за постачальником є надзвичайно корисною для аналізу історії взаємовідносин з конкретними контрагентами, перевірки обсягів поставок, звірки розрахунків або швидкого пошуку документа від певного відправника у випадку виникнення спірних питань.

Класифікація та фільтрація ТТН за датою. Можливість фільтрувати документи за датою створення є ще одним важливим інструментом для ефективного управління документообігом. Система дозволяє користувачам відбирати ТТН за конкретну дату, за певний період (місяць, рік) або за довільно обраним діапазоном дат. Інформація про дату документа вилучається з відповідного реквізиту ТТН під час розпізнавання та зберігається у структурованому форматі. У користувацькому інтерфейсі для фільтрації за датою користувач може ввести бажану дату або діапазон у поле " Фільтрація за датою створення ТТН".

Фільтрація за датою є незамінною для формування періодичної звітності, аналізу динаміки поставок, відстеження документів у хронологічному порядку, а також для архівування або пошуку документів, що відносяться до певних облікових періодів.

Повнотекстовий пошук та комбінована фільтрація. Окрім фільтрації за окремими колонками, інтерфейс (див. рис. 3.6) також містить поле "Пошук". Цей елемент призначений для здійснення повнотекстового пошуку за різними реквізитами документів, такими як номер ТТН, ПІБ водія чи комірника тощо [13]. Користувач може ввести ключове слово, номер документа або будь-який інший текстовий фрагмент, і система відобразить усі ТТН, що містять введений запит.

Для більш точного та цілеспрямованого пошуку система підтримує можливість комбінування фільтрів. Користувач може одночасно застосувати фільтр за постачальником (обравши його у випадаючому списку) та за датою (ввівши її, або діапазон до відповідного поля). Також результати фільтрації можуть бути додатково уточнені за допомогою повнотекстового пошуку у полі "Пошук". Комбінована фільтрація значно розширює аналітичні можливості системи та дозволяє користувачам швидко отримувати саме ту вибірку документів, яка їм необхідна для вирішення конкретних завдань.

Впровадження функціоналу класифікації та фільтрації документів за постачальником і датою, а також можливості повнотекстового пошуку, суттєво підвищує зручність роботи з системою, скорочує час на пошук потрібної інформації та надає користувачам інструменти для аналізу оброблених документів. Це у свою чергу, сприяє оптимізації робочих процесів та підвищенню загальної ефективності управління документацією на підприємстві.

3.4 Основні режими роботи застосунку

Система спроєктована для функціонування у двох основних режимах, що забезпечує необхідний рівень гнучкості та дозволяє оптимізувати процес обробки документів залежно від конкретних умов та вимог:

- **автоматичний режим обробки** активується системою при надходженні нових зображень документів. У цьому режимі система автономно виконує повний цикл обробки: починаючи з попередньої підготовки зображення, система намагається ідентифікувати документ шляхом зіставлення його з наявними

у базі даних шаблонами. Якщо відповідний шаблон успішно знайдено, застосовується процедура OCR, після чого відбувається вилучення реквізитів згідно з правилами, визначеними у шаблоні. Наступним кроком є автоматична валідація отриманих даних (перевірка на відповідність форматам, діапазонам значень, контрольним сумам тощо) [15]. Успішно оброблені дані зберігаються у базі. У випадку, якщо для документа не вдалося знайти відповідний шаблон або виникли суттєві розбіжності під час валідації, такий документ автоматично направляється до черги для подальшої ручної обробки оператором. Ключовою перевагою даного режиму є висока швидкість обробки значних масивів типових, стандартизованих документів при мінімальному залученні людських ресурсів.

- **ручний режим обробки** призначений для обробки документів, які не змогли бути коректно оброблені в автоматичному режимі, а також у випадках, коли внутрішні регламенти підприємства вимагають обов'язкової процедури верифікації для всіх або окремих категорій документів. У цьому режимі оператор за допомогою графічного інтерфейсу отримує доступ до документа з черги ручної обробки. Система відображає зображення документа та за наявності, результати попереднього автоматичного розпізнавання. Оператор здійснює ретельну перевірку коректності вилучених даних, вносить необхідні виправлення, доповнює пропущені поля. Надзвичайно важливою функцією цього режиму є можливість створення нового шаблону, якщо для поточного документа відповідний шаблон відсутній. Цей режим гарантує високу точність даних навіть для складних, нестандартних або пошкоджених документів та надає системі здатність до поступової адаптації та навчання на нових форматах [16].

Висновки до розділу 3

Спроектована інформаційна система обробки ТТН являє собою комплексне програмне рішення, що ефективно поєднує сучасні методи цифрової обробки зображень, передові технології оптичного розпізнавання тексту, інтелектуальні алгоритми семантичного аналізу та надійні механізми збереження структурованих

даних у захищеному інформаційному середовищі. Система була розроблена з неухильним дотриманням модульності архітектури, гнучкості конфігурації та потенціалу для подальшого масштабування, що в сукупності забезпечує високий рівень зручності її використання, здатність до швидкої адаптації до нових форматів ТТН та широкі можливості для інтеграції з іншими корпоративними інформаційними системами підприємств.

Ретельне проектування кожного окремого компонента системи, починаючи від етапу попередньої підготовки та очищення зображень і завершуючи складними механізмами верифікації розпізнаних даних та динамічної шаблонізації, дозволило закласти міцний фундамент для досягнення високих показників точності розпізнавання. Це сприяє суттєвому зменшенню обсягу рутинного навантаження на персонал, який відповідальний за обробку первинної документації та перетворенню цього трудомісткого процесу на ефективний, прозорий та повністю контрольований бізнес-процес [33]. Впровадження такої системи в операційну діяльність підприємства здатне значно підвищити загальну ефективність його функціонування, мінімізувати фінансові та часові ризики, пов'язані з неминучими помилками ручного введення даних, та надати керівництву цінну, оперативну та достовірну інформацію для підтримки прийняття своєчасних та обґрунтованих управлінських рішень. Подальший розвиток системи може передбачати розширення її функціональних можливостей за рахунок інтеграції нових технологій та алгоритмів, спрямованих на досягнення ще вищого рівня автоматизації та інтелектуалізації процесів електронного документообігу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ АВТОМАТИЧНОЇ ОБРОБКИ ДОКУМЕНТІВ

4.1 Опис програмної реалізації

Програмна реалізація інформаційної системи для автоматизованого розпізнавання та класифікації фінансових документів, зокрема товарно-транспортних накладних (ТТН), була здійснена на об'єктно-орієнтованій мові програмування Java. Вибір Java як основної технологічної платформи був зумовлений низкою її фундаментальних переваг, серед яких особливо значущими є високий рівень кросплатформенності, що забезпечується функціонуванням віртуальної машини Java (JVM), наявність розгалуженої та зрілої екосистеми доступних бібліотек та фреймворків, розвинені та ефективні засоби для реалізації багато-потокових обчислень, а також надійна та автоматизована система управління пам'яттю. Кінцевим артефактом розробки є виконуваний JAR-файл (Java Archive), який інкапсулює всі необхідні компоненти системи та дозволяє забезпечити легкість її розгортання та запуску на широкому спектрі операційних систем (Windows, Linux, macOS) без необхідності внесення будь-яких змін до вихідного коду. Конфігурування роботи системи, а також визначення її специфічної поведінки в залежності від потреб користувача, здійснюються шляхом передачі відповідних параметрів командного рядка під час запуску програми. Ці параметри дозволяють гнучко керувати ключовими аспектами функціонування, такими як вибір режиму роботи (інтерактивний інтерфейсний або автоматичний пакетний), визначення шляхів до конфігураційних файлів та каталогів з даними, передача ідентифікаторів конкретних документів, що підлягають обробці, та інші специфічні налаштування, що впливають на логіку виконання.

В основу архітектури розробленої інформаційної системи покладено модульний принцип. Такий підхід до проєктування передбачає логічний поділ всієї функціональності системи на окремі, відносно незалежні та слабко зв'язані між собою компоненти – модулі. Кожен такий модуль несе відповідальність за

виконання чітко визначеного та обмеженого набору завдань, що сприяє кращій організації коду, полегшує його розуміння, тестування та подальший супровід. Центральним елементом системи виступає керуючий модуль (або ядро системи), який несе відповідальність за ініціалізацію всіх підсистем та компонентів, аналіз (парсинг) вхідних параметрів, переданих користувачем при запуску та подальшу координацію взаємодії між програмними модулями для забезпечення коректного та послідовного виконання всієї бізнес-логіки закладеної в систему. На рис. 4.1 представлено архітектуру системи у загальному вигляді, вона демонструє концептуальну схему взаємозв'язку основного модуля системи з модулями, які відповідають за автоматичний та інтерфейсний режими роботи, а також їхню спільну взаємодію з ключовими ресурсами (такими як база шаблонів документів, метаописи, пакет функціоналу OCR, база даних, модуль аналізу результатів розпізнавання та графічний інтерфейс).

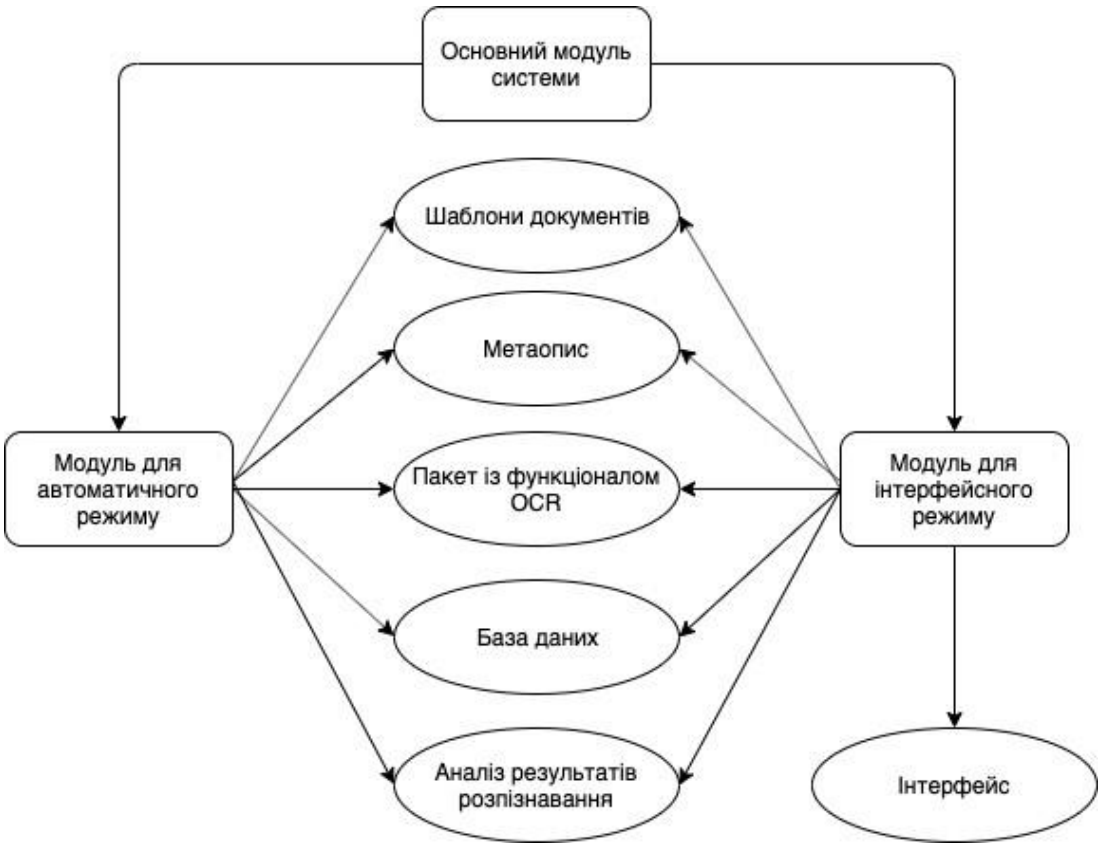


Рисунок 4.1 – Архітектура системи у загальному вигляді

Система спроєктована для підтримки двох основних режимів функціонування:

- **інтерфейсний режим (UI - User Interface).** Цей режим призначений для безпосередньої інтерактивної взаємодії з користувачем (оператором системи). У ньому користувачеві надаються всі необхідні інструменти для візуального контролю над процесом розпізнавання документів, можливості ручного введення та оперативної корекції даних у випадку виявлення помилок автоматичного розпізнавання. Надзвичайно важливою функцією інтерфейсного режиму є також забезпечення механізмів для навчання системи, що реалізується шляхом створення нових та редагування існуючих шаблонів документів;
- **автоматичний режим (batch).** Він орієнтований на пакетну обробку великих обсягів документів без необхідності прямого втручання оператора на кожному окремому кроці обробки. Автоматичний режим є оптимальним рішенням для виконання рутинних завдань з обробки значної кількості стандартизованих документів, для яких у системі вже існують попередньо навчені шаблони.

Вибір конкретного режиму роботи інформаційної системи здійснюється користувачем під час її запуску шляхом передачі відповідного значення для параметра командного рядка -mode (наприклад, -mode UI або -mode BATCH).

Усі модулі, що входять до складу системи, в процесі свого функціонування мають регламентований доступ до низки спільних ресурсів, які є критично важливими для забезпечення її коректної, стабільної та ефективної роботи. До таких ключових спільних ресурсів належать:

- *база шаблонів документів* – централізоване та структуроване сховище, де зберігаються метадані про логічну структуру та точне просторове розташування реквізитів для різних типів документів, а також для документів, що надходять від різних постачальників або контрагентів. Кожен шаблон у цій базі є формалізованим описом, який дозволяє системі однозначно ідентифікувати та вилучати необхідну інформацію з документа відповідного типу;

– *метаописи реквізитів* – детальні формалізовані описи кожного з очікуваних реквізитів, які підлягають вилученню з документів. Ці описи включають таку важливу інформацію, як тип даних реквізиту (наприклад, рядок, ціле число, дата, булеве значення), можливі формати його представлення у вихідному документі (наприклад, різні формати запису дат або числових значень), а також набір правил валідації, які застосовуються системою для перевірки коректності та правдоподібності розпізнаних значень (див. рис. 4.2);

```
<TMPL>
  <ELM NM="InvoiceNumberValue" NMBR="1" REGEX="\d*" RL="51"
 RU="171" RW="596" RH="42" PZ="1" PL="1" PW="3" NW="1" />
  <ELM NM="InvoiceDateValue" NMBR="2" REGEX="dd/mm/yy" RL="51"
 RU="171" RW="596" RH="42" PZ="1" PL="1" PW="5" NW="1" />
  <ELM NM="SupplierName" NMBR="3" REGEX="" RL="51" RU="229"
 RW="788" RH="187" PZ="2" PL="1" PW="2" NW="6" />
  <ELM NM="IBAN" NMBR="4" REGEX="UA\d*" RL="51" RU="229" RW="788"
 RH="187" PZ="2" PL="2" PW="2" NW="1" />
  <ELM NM="LegalAddress" NMBR="5" REGEX="" RL="51" RU="229"
 RW="788" RH="187" PZ="2" PL="3" PW="3" NW="9" />
  <ELM NM="EDRPOU" NMBR="6" REGEX="\d*" RL="51" RU="229" RW="788"
 RH="187" PZ="2" PL="4" PW="4" NW="1" />
  <ELM NM="ContractNumber" NMBR="7" REGEX="\d*" RL="51" RU="229"
 RW="788" RH="187" PZ="2" PL="7" PW="3" NW="1" />
  <ELM NM="ProductName" NMBR="8" REGEX="" RL="60" RU="416"
 RW="1059" RH="116" PZ="3" PL="2" PW="7" NW="7" />
  <ELM NM="Barcode" NMBR="9" REGEX="\d*" RL="60" RU="416"
 RW="1059" RH="116" PZ="3" PL="2" PW="2" NW="1" />
  <ELM NM="TotalAmount" NMBR="10" REGEX="\d*\.\d*" RL="756"
 RU="540" RW="363" RH="77" PZ="4" PL="4" PW="4" NW="1" />
</TMPL>
```

Рисунок 4.2 – Приклад метаданих шаблону

У табл. 4.1 можна побачити пояснення значення до кожного тегу метаданих.

Таблиця 4.1 – Значення тегів метаданих

TMPL	Початок метаданих шаблону
ELM	Початок метаданих елементу
NM	Назва елементу
NMBR	Номер елементу за порядком
REGEX	Регулярний вираз, якому має відповідати значення
RL	Відступ блоку тексту зліва
RU	Відступ блоку тексту зверху
RW	Ширина блоку
RH	Висота блоку
PZ	Номер блоку
PL	Номер строки
PW	Номер слова
NW	Кількість слів

- *поточний оброблюваний документ* – об'єктна інкапсуляція документа, який на даний момент знаходиться в процесі обробки системою. Цей об'єкт агрегує в собі, як саме графічне зображення документа, так і всі проміжні та кінцеві результати його обробки на різних етапах технологічного ланцюга (результати попередньої обробки зображення, розпізнаний текстовий масив, вилучені та валідовані реквізити);
- *результати розпізнавання* – текстові дані, отримані від OCR-двигуна після завершення процесу обробки графічного зображення документа, а також детальна службова інформація про точні координати кожного розпізнаного елемента (такого як окремий символ, слово або текстовий рядок) на вихідному зображенні. Ця координатна інформація є надзвичайно важливою для подальшого семантичного аналізу та прив'язки розпізнаних даних до логічної структури документа;

– *підключення до бази даних*. Програмний інтерфейс (зазвичай реалізований за допомогою JDBC для Java-систем), що забезпечує надійну та ефективну взаємодію з обраною системою управління базами даних (в даному випадку Firebird SQL) для персистентного збереження та оперативного отримання всіх необхідних даних (шаблонів, вилучених реквізитів, системних логів, налаштувань користувачів тощо).

Кожен програмний модуль системи спроєктований таким чином, щоб максимально інкапсулювати специфічну бізнес-логіку, пов'язану з його функціональним призначенням. Модулі використовують власні, чітко визначені класи та об'єкти для управління внутрішніми даними та процесами, що повністю відповідає фундаментальним принципам об'єктно-орієнтованого проєктування. Такий підхід сприяє підвищенню рівня внутрішньої функціональної зв'язності та цілісності кожного окремого модуля та водночас зниженню рівня зв'язності між різними компонентами системи. Це в свою чергу значно полегшує процеси розробки, індивідуального тестування модулів, подальшого супроводу системи та її майбутньої модернізації або розширення функціоналу.

Програмна реалізація інформаційної системи представлена набором взаємодіючих Java-класів. Основним класом програми, що слугує точкою входу (main метод) при запуску виконуваного JAR-файлу, є OCR_Application. Його головне функціональне призначення – аналіз (парсинг) аргументів командного рядка, які були передані користувачем під час запуску. На основі аналізу цих параметрів, зокрема ключового параметра -mode (який може приймати значення UI або BATCH), а також інших специфічних параметрів, таких як -cfg_path (шлях до каталогу з конфігураційними файлами, що містять наприклад, налаштування підключення до бази даних, параметри OCR-двигуна), -doc_pi_import_id (унікальний ідентифікатор документа в базі даних, який необхідно відкрити для обробки в інтерфейсному режимі), -batch_count (максимальна кількість документів для обробки в рамках одного запуску пакетного режиму) та -batch_goal_to (визначає цільовий етап, до якого повинна дійти обробка кожного документа в

пакетному режимі, наприклад, DATA_EXTRACT_CHECK_SAVE означає повний цикл від завантаження до збереження перевірених даних), клас OCR_Application приймає рішення щодо того, який саме режим роботи системи має бути активований. Після визначення необхідного режиму OCR_Application створює екземпляр відповідного керуючого класу. OCR_Batch для автоматичного (пакетного) режиму або FwDrMainForm для інтерфейсного режиму. Новоствореному об'єкту передаються всі необхідні конфігураційні дані та параметри, отримані з командного рядка. Окрім цього, клас OCR_Application відповідає за початкову ініціалізацію глобальних ресурсів системи, таких як конфігурація системи логування (в даній реалізації використовується стандартний java.util.logging.Logger, доступний через статичне поле OCR_APPLICATION.logger), завантаження основних файлів налаштувань, а також за потреби, встановлення первинного з'єднання з базою даних на самому ранньому етапі роботи програми. Важливою частиною функціоналу цього класу є також обробка виняткових ситуацій, пов'язаних з некоректними або відсутніми параметрами запуску. У таких випадках система коректно інформує користувача про помилку (виводячи повідомлення в консоль) та можливо, коректно завершувати свою роботу, щоб уникнути непередбачуваної поведінки.

Клас **OCR_Batch** інкапсулює всю складну логіку роботи інформаційної системи в автоматичному (пакетному) режимі. При своїй ініціалізації він отримує від керуючого класу OCR_Application набір параметрів, що детально визначають обсяг та глибину запланованої обробки документів. Зокрема, параметр -batch_count дозволяє обмежити кількість документів, що підлягають обробці протягом одного сеансу роботи системи, що може бути корисним для розподілу навантаження на системні ресурси або для виконання тестових запусків на обмеженій вибірці даних. Параметр -batch_goal_to визначає кінцевий етап обробки, до якого має дійти кожен документ у пакеті (наприклад: значення DATA_EXTRACT_CHECK_SAVE означає виконання повного технологічного циклу обробки, від завантаження зображення та попередньої обробки до вилучення реквізитів, їх валідації та фінального

збереження перевірених даних у базу). Процес обробки документів в автоматичному режимі є послідовним: система ітерує по набору документів, обраних з бази даних відповідно до певних критеріїв. Критерієм вибору документів для обробки зазвичай є їхній поточний статус (наприклад, "не оброблено", "помилка розпізнавання на попередньому етапі", "потребує повторної обробки після оновлення шаблонів"). Для кожного обраного документа виконується стандартизована послідовність операцій, що включає завантаження зображення та його попередню обробку, оптичне розпізнавання тексту, пошук відповідного шаблону в базі даних, виокремлення реквізитів на основі знайденого шаблону, валідацію отриманих даних відповідно до визначених бізнес-правил та, у випадку успішної валідації, збереження результатів у базу даних. У випадку виникнення будь-яких помилок на одному з перерахованих етапів (наприклад, система не змогла знайти відповідний шаблон для документа, або виникли критичні помилки валідації, які не вдалося виправити автоматично), детальна інформація про таку подію обов'язково фіксується у системному лозі, а сам документ може бути позначений спеціальним статусом (наприклад, "потребує ручного розпізнавання") для подальшого аналізу та обробки оператором в інтерфейсному режимі. Загальну логіку цього процесу, де відображено послідовність кроків обробки кожного документа, включаючи умовні переходи та розгалуження у випадку успішного чи неуспішного знаходження шаблону, представлено на рис. 4.2.

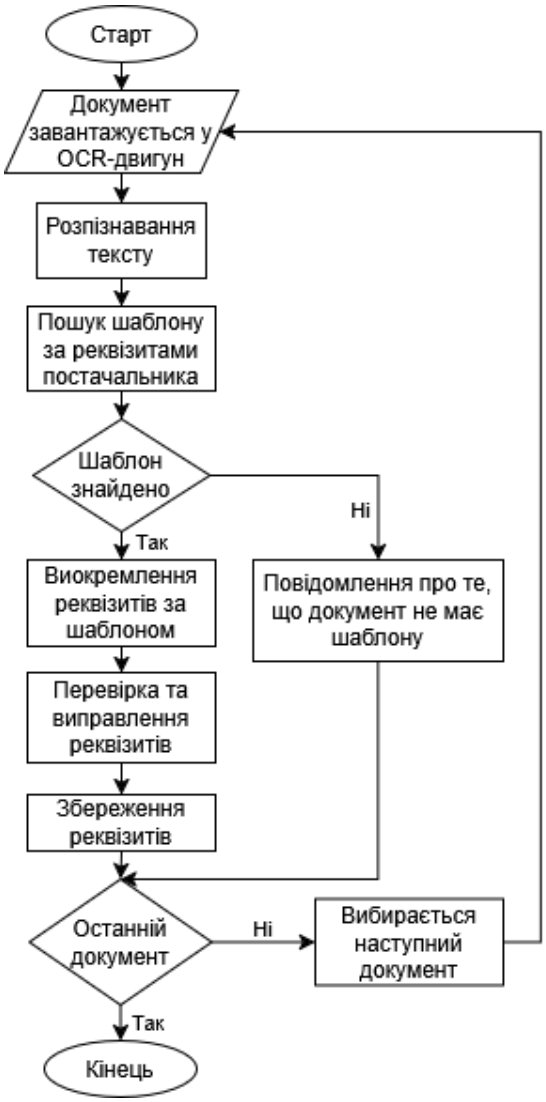


Рисунок 4.2 – Алгоритм роботи системи у автоматичному режимі

Центральним компонентом графічного інтерфейсу користувача (GUI), який реалізований за допомогою стандартної бібліотеки Java Swing, є клас **FwDrMainForm**. Він відповідає за створення головного вікна програми, розміщення на ньому всіх необхідних елементів керування (кнопок, текстових полів, таблиць, панелей тощо) та управління їхньою поведінкою та взаємодією з користувачем. Інтерфейс користувача спроєктований таким чином, щоб надати оператору максимально зручні та інтуїтивно зрозумілі інструменти для виконання завдань візуалізації документа, перегляду розпізнаного тексту, роботи з панеллю реквізитів та, що особливо важливо, для створення та редагування шаблонів

документів. Приклад головного вікна програми, що ілюструє типове розташування елементів, наведено на рис. 4.3.

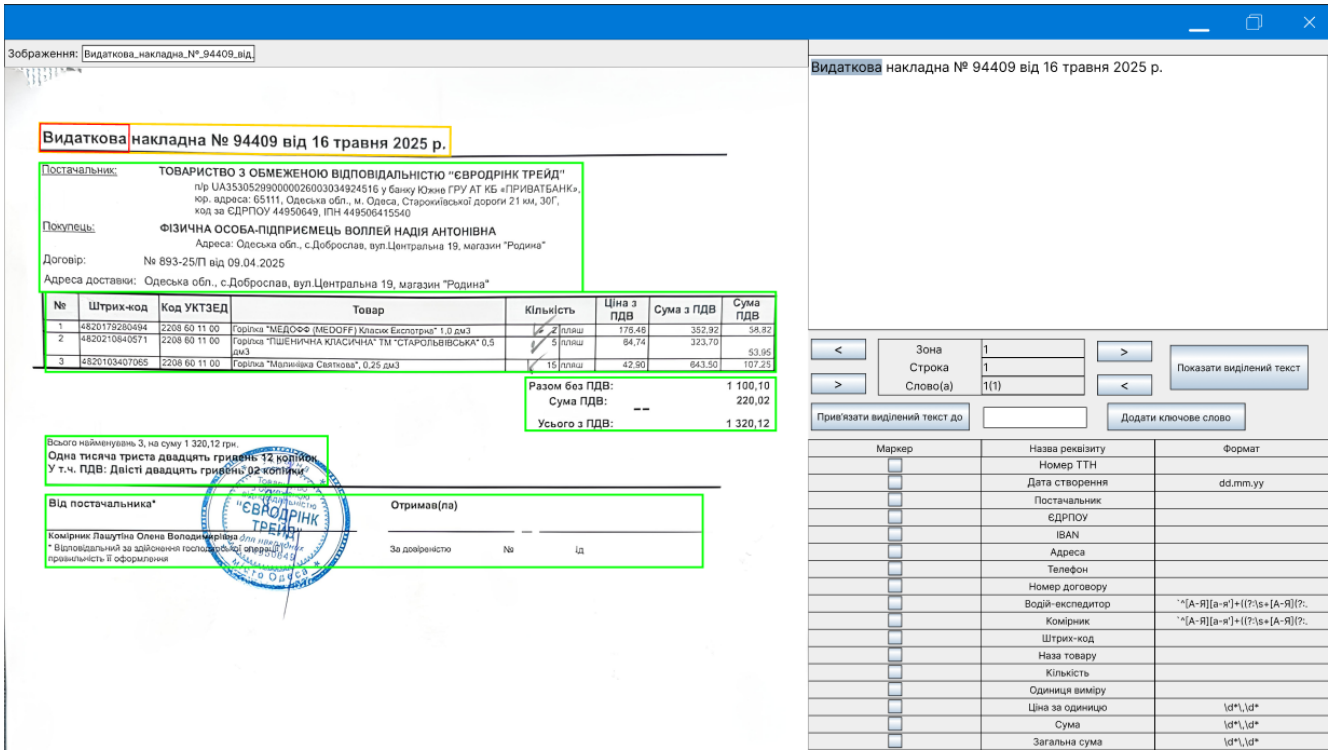
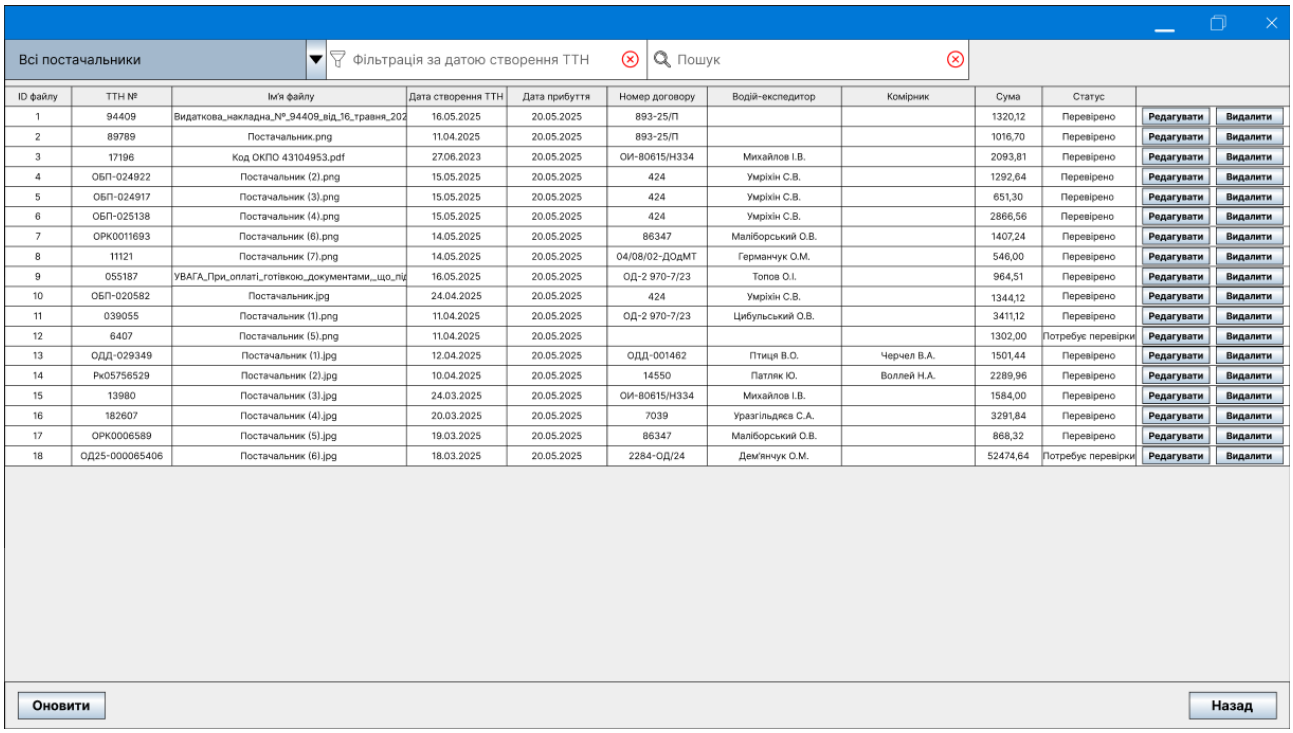


Рисунок 4.3 – Головне вікно програми

На цьому рисунку можна бачити, що зліва розташована область для перегляду зображення оброблюваного документа, праворуч зверху – текстове поле, де відображається результат роботи OCR-двигуна, а праворуч знизу – таблиця або набір полів для введення, відображення та редагування значень реквізитів. Для забезпечення повного циклу роботи користувача в інтерфейсному режимі, від завантаження документа до збереження результатів та навчання системи, передбачені додаткові елементи інтерфейсу. Наприклад, для вибору документа з бази даних використовується спеціальне діалогове вікно, приклад якого наведено на рис. 4.4. Це вікно дозволяє користувачеві переглядати список доступних для обробки документів та обирати необхідний для подальшої роботи, що є першим кроком у процесі навчання або верифікації.



ID файлу	ТТН №	Ім'я файлу	Дата створення ТТН	Дата прибуття	Номер договору	Водій-експедитор	Комірник	Сума	Статус		
1	94409	Видаткова_накладна_№_94409_від_16_травня_202	16.05.2025	20.05.2025	893-25/П			1320,12	Перевірено	Редагувати	Видалити
2	89789	Постачальник.png	11.04.2025	20.05.2025	893-25/П			1016,70	Перевірено	Редагувати	Видалити
3	17196	Код ОКПО 43104953.pdf	27.06.2023	20.05.2025	ОИ-80615/Н334	Михайлов І.В.		2093,81	Перевірено	Редагувати	Видалити
4	ОБП-024922	Постачальник (2).png	15.05.2025	20.05.2025	424	Умрікін С.В.		1292,64	Перевірено	Редагувати	Видалити
5	ОБП-024917	Постачальник (3).png	15.05.2025	20.05.2025	424	Умрікін С.В.		651,30	Перевірено	Редагувати	Видалити
6	ОБП-025138	Постачальник (4).png	15.05.2025	20.05.2025	424	Умрікін С.В.		2866,56	Перевірено	Редагувати	Видалити
7	ОРК0011693	Постачальник (6).png	14.05.2025	20.05.2025	86347	Маліборський О.В.		1407,24	Перевірено	Редагувати	Видалити
8	11121	Постачальник (7).png	14.05.2025	20.05.2025	04/08/02-ДОдМТ	Германчук О.М.		546,00	Перевірено	Редагувати	Видалити
9	055187	УВАГА_При_оплаті_готівкою_документами_що_під	16.05.2025	20.05.2025	ОД-2 970-7/23	Топов О.І.		964,51	Перевірено	Редагувати	Видалити
10	ОБП-020582	Постачальник.jpg	24.04.2025	20.05.2025	424	Умрікін С.В.		1344,12	Перевірено	Редагувати	Видалити
11	039055	Постачальник (1).png	11.04.2025	20.05.2025	ОД-2 970-7/23	Цибульський О.В.		3411,12	Перевірено	Редагувати	Видалити
12	6407	Постачальник (5).png	11.04.2025	20.05.2025				1302,00	Потребує перевірки	Редагувати	Видалити
13	ОДД-029349	Постачальник (1).jpg	12.04.2025	20.05.2025	ОДД-001462	Пташ В.О.	Черчел В.А.	1501,44	Перевірено	Редагувати	Видалити
14	Рк05756529	Постачальник (2).jpg	10.04.2025	20.05.2025	14550	Патляк Ю.	Воллей Н.А.	2289,96	Перевірено	Редагувати	Видалити
15	13980	Постачальник (3).jpg	24.03.2025	20.05.2025	ОИ-80615/Н334	Михайлов І.В.		1584,00	Перевірено	Редагувати	Видалити
16	182607	Постачальник (4).jpg	20.03.2025	20.05.2025	7039	Уразгільдяєв С.А.		3291,84	Перевірено	Редагувати	Видалити
17	ОРК0006589	Постачальник (5).jpg	19.03.2025	20.05.2025	86347	Маліборський О.В.		868,32	Перевірено	Редагувати	Видалити
18	ОД25-000065406	Постачальник (6).jpg	18.03.2025	20.05.2025	2284-ОД/24	Дем'янчук О.М.		52474,64	Потребує перевірки	Редагувати	Видалити

Рисунок 4.4 – Скріншот у меню вибору документа

Після розпізнавання документа та відображення реквізитів, користувач може взаємодіяти з таблицею реквізитів, де є можливість візуального позначення співпоставлених полів, як це показано на рис. 4.5. Така таблиця зазвичай містить назву реквізиту, його розпізнане значення та індикатори статусу ("перевірено", "потребує перевірки"), що допомагає оператору швидко оцінити результати автоматичного розпізнавання.

Маркер	Назва реквізиту	Формат
☐	Номер ТТН	
☐	Дата створення	dd.mm.yy
☐	Постачальник	
☐	ЄДРПОУ	
☐	IBAN	
☐	Адреса	
☐	Телефон	
☐	Номер договору	
☐	Водій-експедитор	`^[А-Я][а-я']+((?:\s+[А-Я](?:.
☐	Комірник	`^[А-Я][а-я']+((?:\s+[А-Я](?:.
☐	Штрих-код	
☐	Наза товару	
☐	Кількість	
☐	Одиниця виміру	
☐	Ціна за одиницю	\d*\,\d*
☐	Сума	\d*\,\d*
☐	Загальна сума	\d*\,\d*

Рисунок 4.5 – Таблиця реквізитів

Важливим етапом є прив’язка документа або створюваного шаблону до конкретного постачальника, що здійснюється через відповідне меню вибору, приклад якого можна побачити на рис. 4.6. Це дозволяє системі групувати шаблони та застосовувати їх більш точно, враховуючи специфіку документів від різних постачальників.

Рисунок 4.6 – Меню вибору постачальника

Перед фінальним збереженням шаблону користувачеві надається спеціалізоване вікно для перегляду та підтвердження всіх введених даних, як ілюструється на рис. 4.7. У цьому вікні оператор може ще раз переконатися у коректності визначених зон та значень реквізитів, що мінімізує ризик збереження неточного шаблону.

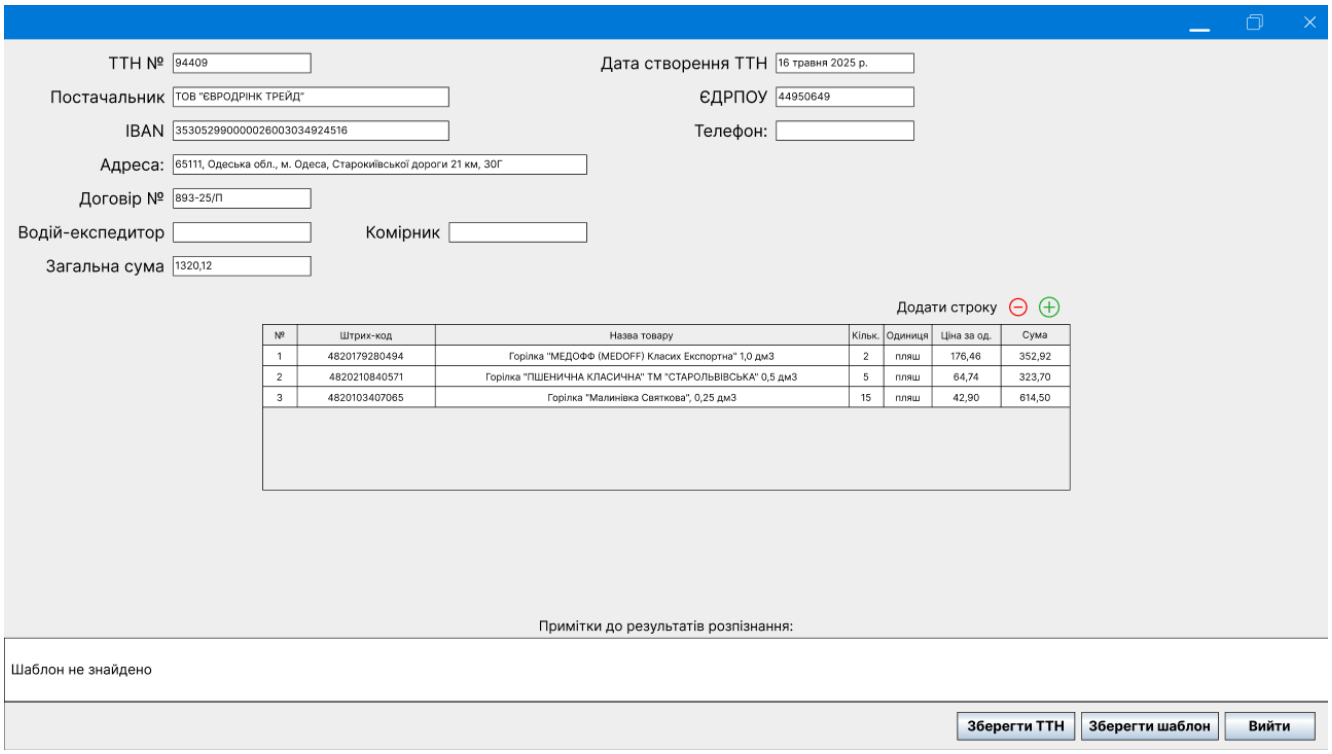


Рисунок 4.7 – Вікно збереження шаблонів

Успішне завершення операції збереження шаблону підтверджується відповідним інформаційним повідомленням, приклад якого наведено на рис. 3.8. Це сповіщення інформує користувача про те, що новий шаблон було успішно додано до бази даних системи і він готовий до використання.

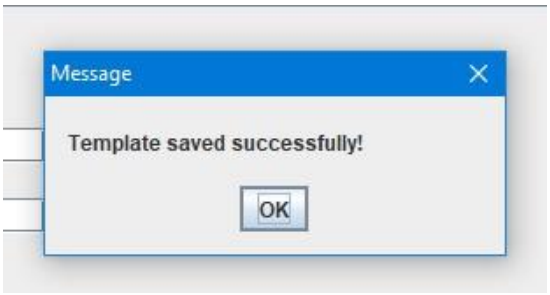


Рисунок 4.8 – Повідомлення про успішно збережений шаблон

Загальний алгоритм роботи користувача в інтерфейсному режимі, що включає послідовні кроки вибору документа, ручного розпізнавання та співставлення реквізитів, вибору постачальника, перевірки введених даних та

фінального збереження шаблону, представлений у вигляді блок-схеми на рис. 4.9. Ця схема наочно демонструє логіку взаємодії користувача з системою на етапі навчання.

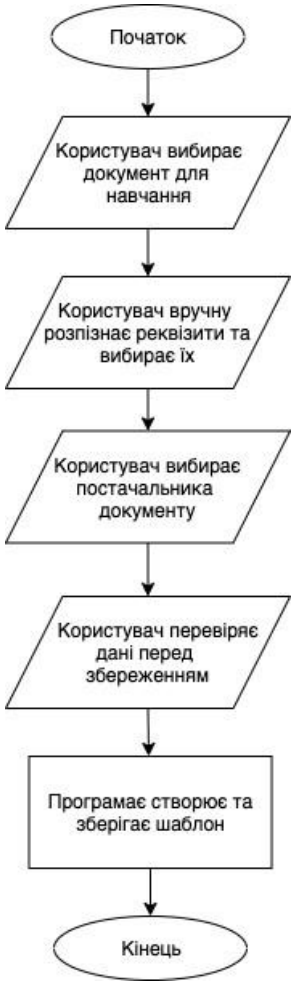


Рисунок 4.9 – Алгоритм роботи у інтерфейсному режимі

Сценарії використання (прецеденти) системи в інтерфейсному режимі, такі як "Навчання та створення шаблонів бланків" або "Вибрати документ для навчання", були детально описані та візуалізовані за допомогою діаграм прецедентів. Ці діаграми допомагають зрозуміти основні варіанти взаємодії користувача з системою та її функціональні можливості в контексті управління шаблонами (див. рис. 4.10 та рис. 4.11). Наведені діаграми прецедентів візуалізують ключові взаємодії користувача з системою в інтерфейсному режимі. Рис. 4.10

демонструє загальний огляд можливих дій, тоді як рис. 4.11 деталізує один з найважливіших сценаріїв – процес створення нового шаблону документа.

Рисунок 4.10 – Прецеденти високого рівня для інтерфейсного режиму

Рисунок 4.11 – Детельна діаграма прецеденту «Створення шаблону»

Клас **FwDrMainForm** реалізує всю необхідну логіку обробки подій від елементів керування та активно взаємодіє з іншими ключовими класами системи для виконання відповідних операцій.

Клас **OCR_ENGINE** – один із найважливіших компонентів системи, оскільки він інкапсулює всю складну логіку взаємодії з зовнішньою бібліотекою оптичного розпізнавання тексту Nicomsoft OCR. Для забезпечення можливості виклику функцій OCR-бібліотеки (які зазвичай написані на мовах програмування C або C++) з Java-коду, використовується спеціальна проксі-бібліотека JNSOCR. Ця бібліотека надається розробниками Nicomsoft OCR у складі їхнього SDK і реалізує механізм JNI, що дозволяє подолати розрив між керованим середовищем Java та нативним кодом. Клас OCR_ENGINE надає зручний та високорівневий програмний інтерфейс для інших модулів системи, тим самим абстрагуючись від специфічних деталей реалізації та особливостей використання конкретної OCR-бібліотеки. Такий підхід до проєктування дозволяє у майбутньому, за виникнення такої потреби, відносно легко замінити Nicomsoft OCR на інший OCR-двигун (наприклад, Tesseract OCR або комерційний аналог) без необхідності кардинальної перебудови всієї логіки системи, обмежившись лише модифікацією класу OCR_ENGINE. До основних методів, що надаються класом OCR_ENGINE, належать ініціалізація OCR-двигуна, завантаження зображення документа з файлу або з масиву байтів, запуск процесу оптичного розпізнавання, отримання повного текстового представлення документа, пошук заданого текстового фрагмента на зображенні та повернення його координат, створення нового шаблону документа та реалізація логіки пошуку відповідного існуючого шаблону. Приклад методу loadImg() для завантаження зображення на рис. 4.12.

```
public int loadImg() {
    byte[] fileContent;
    int err = -1;
    setStage(-1); //  зображення не завантажено
    fileContent = docProcessed.getInvoiceImage(); // отримали зображення документа
    if (fileContent != null) { // є зображення
        err = engineObj.Img_LoadFromMemory(fileContent, fileContent.length);
        if (err == 0) {
            setStage(Constant.OCRSTEP_FIRST);
            OCR_APPLICATION.logger.log(Level.INFO, "зображення успішно завантажено");
        } else {
            OCR_APPLICATION.logger.log(Level.SEVERE, "зображення не завантажено err={0}", err);
        }
    } else {
        OCR_APPLICATION.logger.log(Level.SEVERE, "зображення не отримано з БД err={0}", err);
    }
    return err;
}
```

Рисунок 4.12 – Код методу loadImg()

Метод run(int lastStep, boolean inOtherThread) запускає процес розпізнавання (див. рис. 4.13).

```
public int run(int lastStep, boolean inOtherThread) {
    int err = -1;
    if (getStage() > 0 & lastStep > getStage()) {
        err = engineObj.Img_OCR(getStage(), lastStep, inOtherThread ? Constant.OCRFLAG_THREAD : Constant.OCRFLAG_NONE);
        // ... (обробка помилок та встановлення стадії) ...
    } else {
       OCR_APPLICATION.logger.log(Level.SEVERE, "стадія не дійсна err={0}", err);
    }
    return err;
}
```

Рисунок 4.13 – Метод run(int lastStep, boolean inOtherThread)

Клас **RecognizeStage** використовується для управління поетапним виконанням процесу розпізнавання, що дозволяє оптимізувати процес обробки.

DOC_ENGINE відповідає за реалізацію специфічної бізнес-логіки обробки документів певного типу, переважно ТТН. Його основні завдання включають застосування шаблону (завантаження та парсинг XML-метаданих шаблону), виокремлення значень реквізитів на основі цих метаданих та розпізнаного тексту, валідацію та корекцію отриманих даних за допомогою методів класу DocMetaType, та підготовку даних до збереження у БД. Ключовим методом є analyze(), який координує процес вилучення та перевірки всіх реквізитів для документа на основі переданого шаблону. Фрагмент методу analyze() зображено на рис. 4.14.

```
public int analyze(DocTemplate docTemplate, int precision, ArrayList<DocRecv> result) {
    int errKod = 0;
    extractReport="";
    ArrayList<Dimension> shiftList;
    Dimension bestshift;
    shiftList = getShift(docTemplate,precision);
    for (DocMeta recvDef : docTemplate.getData()) {
       DocMetaType rType=recvDef.getType();
        if (rType.getTypeId() > 0) {
            DocRecv recv = new DocRecv();
            addResult(result, recv);
            recv.setRecvDef(recvDef);
            rType.setRegEx(recvDef.getRegex());
            // ... (цикли по методах вилучення та зміщеннях) ...
            if (textRecognizer.findData(recv,shiftList.get(i),m,precision)){
                // ... (логіка перевірки та покращення rType.check() та rType.refine()) ...
            }
            // ...
            if (recv.getStatus()!=0){
                errKod++ ;
            }
        }
    }
    return errKod;
}
```

Рисунок 4.14 – Фрагмент методу analyze()

Метод *getShift()* визначає можливе зміщення поточного документа відносно еталонного шаблону.

Клас **DocMetaType**, реалізований як перелічуваний тип (*enum*), відіграє роль сховища метаінформації про реквізити. Для кожного типу реквізиту він визначає опис, тип ID, регулярний вираз за замовчуванням, а також реалізує абстрактні методи *check(DocRecv dataItem,String prot)* для перевірки відповідності значення реквізиту визначеним правилам та *refine(DocRecv dataItem,Integer nStep,String prot)* для спроби автоматичної корекції типових помилок OCR. Приклад реалізації для FLOAT_VALUE дивитись на рис. 4.15.

```
FLOAT_VALUE {
    // ... (getDecription, getTypeId, etc.) ...
    @Override
    public boolean check(DocRecv dataItem, String prot) {
        String rValue=dataItem.getRecvValue();
        // ... (перевірка на null, порожнечу) ...
        try {
            float net = Float.parseFloat(rValue);
            dataItem.setStatus(0);   // відповідає формату
            return true;
        } catch (NumberFormatException ex) {
            dataItem.setStatus(6);   // не відповідає формату
            prot=prot+"елемент "+dataItem.getRecvDef().getName()+" = "+rValue+" направильний формат";
            return false;
        }
    }
    @Override
    public boolean refine(DocRecv dataItem, Integer nStep, String prot) {
        boolean isLast=false;
        String rValue=dataItem.getRecvValue();
        switch (nStep) {
            case 1:   // видалення пробілів
                dataItem.setRecvValue(rValue.replaceAll("\\s", ""));
                break;
            case 2:   // видалення неправильних символів
               dataItem.setRecvValue(rValue.replaceAll("[^0123456789.,]", ""));
                break;
            case 3:   // заміна комм на крапку
                rValue= rValue.replace(",",".");
                dataItem.setRecvValue(rValue);
                break;
            // ... (інші кроки покращення) ...
            default:
                isLast=true;
                break;
        }
        return isLast;
    }
}
```

Рисунок 4.15 – Скрипт для FLOAT_VALUE

Центральним елементом, що забезпечує адаптивність та гнучкість розробленої інформаційної системи, є механізм створення та використання шаблонів документів. Цей механізм дозволяє системі "навчатися" на нових типах документів або на документах від нових постачальників, які можуть мати унікальну структуру та розташування реквізитів. Процес створення шаблону зазвичай відбувається в інтерфейсному режимі під контролем оператора. Спочатку користувач завантажує зображення нового документа, після чого система виконує OCR. Далі, для кожного реквізиту, користувач вводить значення вручну або виділяє відповідний текстовий фрагмент. На основі цієї інформації OCR_ENGINE визначає

точні координати зони реквізиту, його відносне положення та можливі ключові слова-якорі. Вся зібрана інформація збирається у стандартизовану XML-структуру. У цій структурі теги ELM описують окремі елементи (реквізити) із зазначенням їх атрибутів, таких як назва (NM), відносні координати блоку (RL, RU, RW, RH), номер блоку (PZ), номер рядка (PL), номер слова (PW), кількість слів (NW) та регулярний вираз (REGEX). Ця XML-структура зберігається у відповідному полі таблиці OCR_DOC_TEMPLATE бази даних, асоціюючись з певним постачальником та типом документа.

При подальшій роботі системи в автоматичному режимі, після ідентифікації постачальника, система завантажує відповідний шаблон. Клас DOC_ENGINE парсить XML-метадані шаблону та використовує збережені атрибути для точного вилучення значень реквізитів з розпізнаного тексту. Такий підхід забезпечує високий рівень адаптивності та точності розпізнавання.

4.2 Тестування системи

Етап тестування є критично важливою фазою розробки інформаційної системи, спрямованої на автоматизоване розпізнавання документів. Головною метою тестування було оцінювання ефективності та надійності розробленої системи в умовах, наближених до реальної експлуатації. Ключовим показником ефективності визначено відсоток документів, які система успішно розпізнає в автоматичному режимі. Завдання тестування також включає виявлення потенційних проблем, типових помилок та перевірку стабільності роботи.

Обрана методологія базується на комплексному підході з використанням репрезентативної вибірки документів, що відображала різноманітність типів бланків, шрифтів та якості друку. Тестування імітувало реальний сценарій використання: на першому етапі для кожного нового типу бланка система "навчалася" в інтерфейсному режимі шляхом ручної обробки оператором одного або декількох зразків та створення відповідного шаблону. Після формування бази шаблонів система переводилася в автоматичний режим для пакетної обробки

решти документів. Такий підхід дозволяв оцінити якість автоматичного розпізнавання та ефективність механізму навчання.

Перший етап, проведений після початкової реалізації, мав на меті виявлення проблемних зон для подальшого доопрацювання. Було ідентифіковано характерні помилки: неточності OCR-бібліотеки при розпізнаванні окремих символів (особливо схожих такі як "/", "I", "i", "1") через нестандартні шрифти або низьку якість друку [34]; різноманітність форматів дат ("22.04.2025", "22 Травня 2025"), що вимагало розробки гнучкого модуля їх парсингу та нормалізації; наявність випадкових пробілів або помилкових знаків пунктуації у числових реквізитах (сумах), що призводило до помилок валідації. Ці спостереження були враховані при доопрацюванні відповідних модулів, зокрема блоку перевірки та валідації даних (методів *refine()* та *check()* у класі DocMetaType).

Другий етап був спрямований на кількісну оцінку ефективності оптимізованої системи. Було сформовано набір з 84 різних типів бланків ТТН від 11 постачальників. Після попереднього навчання системи на зразках кожного типу бланка, було запущено пакетну обробку. Результати показали, що з 84 документів система успішно (повністю та коректно, без ручного втручання) обробила 68 документів. Таким чином, узагальнений показник успішності автоматичного розпізнавання склав приблизно 80.95%. Цей результат свідчить про достатньо високу ефективність системи, враховуючи складність завдання.

Аналіз неуспішних випадків виявив декілька основних причин: неможливість знайти відповідний шаблон (через відсутність ключових слів-маркерів або низьку якість скану), наявність значних дефектів друку.

Незважаючи на певний відсоток документів, що потребували ручної обробки, досягнутий показник у 81% успішних автоматичних розпізнавань свідчить про значний потенціал системи. Подальше вдосконалення алгоритмів пошуку шаблонів, розширення можливостей модуля попередньої обробки зображень та збагачення бази знань новими шаблонами дозволить у майбутньому підвищити точність та надійність.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи було детально описано процес програмної реалізації інформаційної системи для автоматизованого розпізнавання та класифікації фінансових документів, а також представлено результати її комплексного тестування. Програмна реалізація була виконана на мові Java з використанням модульної архітектури, що забезпечило гнучкість та можливість подальшого розширення системи. Було описано ключові класи, такі як OCR_Application, OCR_Batch, FwDrMainForm, OCR_ENGINE, DOC_ENGINE та DocMetaType, що відповідають за ініціалізацію, управління режимами роботи, взаємодію з OCR-бібліотекою Nicomsoft, логіку обробки документів за шаблонами та визначення метаданих реквізитів. Особливу увагу приділено механізму створення та використання шаблонів, що є основою адаптивності системи до різних форматів ТТН. Графічний інтерфейс користувача, реалізований на Java Swing, надає інструменти для навчання системи та верифікації даних.

Тестування системи проводилося у два етапи та мало на меті всебічну оцінку її ефективності та надійності. Застосована методологія включала навчання системи на зразках документів та подальшу автоматичну обробку тестового набору. В ході тестування було виявлено та усунуто низку початкових недоліків, пов'язаних з точністю розпізнавання окремих символів та обробкою різноманітних форматів даних. Кількісна оцінка ефективності на вибірці документів від 11 постачальників продемонструвала показник успішності автоматичного розпізнавання на рівні 80.95%. Аналіз неуспішних випадків дозволив визначити напрямки для подальшого вдосконалення системи. Загалом, результати тестування підтвердили працездатність розробленого програмного продукту та його потенціал для підвищення ефективності обробки фінансових документів.

ВИСНОВКИ

У даній кваліфікаційній роботі було успішно досліджено інформаційні системи для розпізнання та класифікації паперових ТТН, яка дозволяє автоматизувати процес витягу реквізитів документа та інтегрувати дані у внутрішні інформаційні системи підприємств.

Для досягнення поставленої мети було виконано низку завдань:

- проаналізовано сферу цифровізації фінансових документів;
- досліджено інформаційні технології для цифровізації паперових документів;
- спроєктовано систему розпізнавання ТТН;
- створено програмну реалізацію системи розпізнавання ТТН та проаналізовано її.

Розроблена інформаційна система має значний практичний потенціал, оскільки дозволяє суттєво скоротити час на обробку ТТН, зменшити кількість помилок, пов'язаних з людським фактором, та підвищити загальну ефективність документообігу на підприємствах. Подальший розвиток системи може включати розширення підтримки типів документів, інтеграцію з хмарними сервісами та впровадження більш просунутих методів машинного навчання.

Таким чином, усі поставлені у кваліфікаційній роботі завдання було виконано, а мету – досягнуто. Розроблено функціональну інформаційну систему, що підтверджує її практичну цінність та можливість впровадження на підприємствах для автоматизації обробки фінансових документів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Digitalization in Business World. *TimeCamp* : вебсайт. URL: https://www.timecamp.com/blog/2018/02/5-ways-digitization-is-changing-business-world/ (дата звернення: 21.12.2024)

2. How OCR Can Help Employees Fight Through Most Mundane Tasks. *InData Labs* : вебсайт. URL: https://indatalabs.com/blog/optical-character-recognition (дата звернення: 21.12.2024)

3. The Benefits of OCR Systems for Your Business. *StartUp Mindset* : вебсайт. URL: https://startupmindset.com/the-benefits-of-optical-character-recognition-systems-for-your-business/ (дата звернення: 22.12.2024)

4. Market Analysis Report. *Grand View Research* : вебсайт. URL: https://www.grandviewresearch.com/industry-analysis/optical-character-recognition-market (дата звернення: 22.12.2024)

5. 10 Advantages of Using OCR in Business. *LeadChange* : вебсайт. URL: https://leadchangegroup.com/10-advantages-of-using-ocr-in-business/ (дата звернення: 23.12.2024)

6. Automating Invoice Processing with OCR and Deep Learning. *Nanonets* : вебсайт. URL: https://nanonets.com/blog/invoice-ocr/ (дата звернення: 24.12.2024)

7. How to Build Custom Deep Learning Based OCR models? *Nanonets* : вебсайт. URL: https://nanonets.com/blog/attention-ocr-for-text-recogntion/ (дата звернення: 25.12.2024)

8. Creating a Modern OCR Pipeline Using Computer Vision and Deep Learning. *Dropbox.Tech* : вебсайт. URL: https://blogs.dropbox.com/tech/2017/04/creating-a-modern-ocr-pipeline-using-computer-vision-and-deep-learning/ (дата звернення: 25.12.2024)

9. Tesseract Documentation Tesseract Open Source Engine. *GitHub* : вебсайт. URL: https://github.com/tesseract-ocr/tesseract (дата звернення: 26.12.2024)

10. A Beginner’s Guide to Tesseract OCR. *Medium* : вебсайт. URL: https://medium.com/better-programming/beginners-guide-to-tesseract-ocr-using-python-10ecbb426c3d (дата звернення: 26.12.2024)

11. Computer Vision Market Research. *MarketsAndMarkets* : вебсайт. URL: https://www.marketsandmarkets.com/Market-Reports/computer-vision-market-186494767.html (дата звернення: 04.01.2025)

12. Azad A. K. M., Misbahuddin M. Cloud Enabled Text Reader for Individuals with Vision Impairment. *Advances in Internet of Things*. 2017. Vol. 07, no. 04. P. 97–111. URL: https://doi.org/10.4236/ait.2017.74007 (дата звернення: 04.06.2025).

13. Busta M., Neumann L., Matas J. Deep TextSpotter: An End-to-End Trainable Scene Text Localization and Recognition Framework. *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, 22–29 October 2017. 2017. URL: https://doi.org/10.1109/iccv.2017.242 (дата звернення: 04.01.2025).

14. What Is OCR and How Will It Help You Grow Your Business? *Insanelab* : вебсайт. URL: https://insanelab.com/blog/mobile-development/ocr-how-will-it-help-grow-business/ (дата звернення: 07.01.2025)

15. El Bahi H., Zatni A. Text recognition in document images obtained by a smartphone based on deep convolutional and recurrent neural network. *Multimedia Tools and Applications*. 2019. Vol. 78, no. 18. P. 26453–26481. URL: https://doi.org/10.1007/s11042-019-07855-z (дата звернення: 07.01.2025)

16. Reading Text in the Wild with Convolutional Neural Networks / M. Jaderberg et al. *International Journal of Computer Vision*. 2015. Vol. 116, no. 1. P. 1–20. URL: https://doi.org/10.1007/s11263-015-0823-z (дата звернення: 14.03.2025).

17. Sayallar C., Sayar A., Babalik N. An OCR Engine for Printed Receipt Images using Deep Learning Techniques. *International Journal of Advanced Computer Science and Applications*. 2023. Vol. 14, no. 2. URL: https://doi.org/10.14569/ijacsa.2023.0140295 (дата звернення: 22.03.2025)

18. Yue A. Automated Receipt Image Identification, Cropping, and Parsing, 2018.

URL: https://www.cs.princeton.edu/courses/archive/spring18/cos598B/public/projects/System/COS598B_spr2018_ReceiptParsing.pdf (дата звернення: 18.03.2025)

19. Drobac S., Lindén K. Optical character recognition with neural networks and post-correction with finite state methods. *International Journal on Document Analysis and Recognition (IJDAR)*. 2020. Vol. 23, no. 4. P. 279–295. URL: https://doi.org/10.1007/s10032-020-00359-9 (дата звернення: 21.03.2025)

20. Digitization, Digitalization, And Digital Transformation: Confuse Them At Your Peril. *Forbes* : вебсайт. URL: https://www.forbes.com/sites/jasonbloomberg/2018/04/29/digitization-digitalization-and-digital-transformation-confuse-them-at-your-peril/#679fe0942f2c (дата звернення: 25.03.2025)

21. X. Tang, Y. Lai, Y. Liu, Y. Fu, and R. Fang, “Visual-Semantic Transformer for Scene Text Recognition,” Dec. 2021. URL: http://arxiv.org/abs/2112.00948 (дата звернення: 27.03.2025)

22. Eger S., Brück T. v. d., Mehler A. A Comparison of Four Character-Level String-to-String Translation Models for (OCR) Spelling Error Correction. *The Prague Bulletin of Mathematical Linguistics*. 2016. Vol. 105, no. 1. P. 77–99. URL: https://doi.org/10.1515/pralin-2016-0004 (дата звернення: 28.03.2025).

23. K. Wang, B. Babenko, and S. Belongie, “End-to-end scene text recognition,” in 2011 International Conference on Computer Vision, Nov. 2011, pp. 1457–1464, iSSN: 2380-7504.

24. Is Java still relevant? *SDTimes* : вебсайт. URL: https://sdtimes.com/java/is-java-still-relevant/ (дата звернення: 17.01.2025)

25. M. Gjoreski, G. Zajkovski, A. Bogatinov, G. Madjarov, D. Gjorgjevikj, and H. Gjoreski. Optical character recognition applied on receipts printed in macedonian language. 04 2014.

26. Document Understanding Process - User Guide. GitHub : вебсайт. URL: https://github.com/UiPath-Services/StudioTemplates/blob/develop/DocumentUnderstandingProcess/contentFiles/a

ny/any/pt0/VisualBasic/UserGuide/Document%20Understanding%20Process%20-%20User%20Guide.pdf (дата звернення: 23.02.2025)

27. Helen Borrie - The Firebird Book: A Reference for Database Developers - 2017 - 1128 c.

28. Code Quality Comparison of Firebird, MySQL, and PostgreSQL. *Medium* : вебсайт. URL: https://medium.com/@CPP_Coder/code-quality-comparison-of-firebird-mysql-and-postgresql-53e39fc3298d (дата звернення: 07.03.2025)

29. O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” May 2015, arXiv:1505.04597 URL: http://arxiv.org/abs/1505.04597 (дата звернення: 23.04.2025)

30. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Apr. 2015, arXiv:1409.1556. URL: http://arxiv.org/abs/1409.1556 (дата звернення: 28.04.2025)

31. Springmann, U., Lüdeling, A.: OCR of historical printings with an application to building diachronic corpora: a case study using the RIDGES herbal corpus 2016. URL: https://arxiv.org/abs/1608.02153 (дата звернення: 29.04.2025)

32. Wick, C., Reul, C., Puppe, F.: Calamari—a high-performance tensorflow-based deep learning package for optical character recognition 2018. URL: https://arxiv.org/abs/1807.02004 (дата звернення: 12.03.2025)

33. Shi, X. Bai, and S. J. Belongie. Detecting oriented text in natural images by linking segments. CoRR, abs/1703.06520, 2017.

34. Reul, C., Springmann, U., Wick, C., Puppe, F.: State of the art optical character recognition of 19th century Fraktur scripts using open source engines. 2018 URL: https://arxiv.org/abs/1810.03436 (дата звернення: 11.04.2025)

ДОДАТОК А

Код програмної реалізації

Клас OCR_Engine

```
import NSOCR.NSInt;
import NSOCR.Constant;
import NSOCR.Engine;
import NSOCR.HBLK;
import static com.sun.javafx.util.Utils.split;
import static fwdocumentrecognizer.OCR_APPLICATION.fwOcrAPP;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import static java.lang.Math.abs;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class OCR_ENGINE {

    private BufferedImage fullSizeBmp;
    private final EngineObj engineObj;
    private EngineObj engineObj2;
    //private final String licenseKey = "AB2A4DD5FF2A";
    private int stage;  //** стадія процесу
    private int nBlocks; // число блоків
    private Invoice docProcessed; // документ котрий обробляється
    public static int TRANSFORM_COEF = 10000;
    private boolean drawBinary = true;

    public OCR_ENGINE() {
        setStage(-2); // двигун не проініціалізован
        engineObj = new EngineObj();
        if (engineObj.isReady()) {
            setStage(-1); // двигун проініціалізован
        }
    }

    public void setDocProcessed(Invoice newDoc) {
        docProcessed = newDoc;
        fullSizeBmp = null;
    }
    public Invoice getDocProcessed() {
        return docProcessed;
    }

    public int getStage() {
```

```
        return stage;
    }

    public void setStage(int newStage) {
        stage = newStage;
    }

    public int loadCfg(String configFile) {
        // TODO: implement
        return 0;
    }

    public int saveCfg(String configFile) {
        // TODO: implement
        return 0;
    }

    public int loadImg() {
        byte[] fileContent;
        int err = -1;
        setStage(-1); //  зображення не завантажено
        fileContent = docProcessed.getInvoiceImage(); // отримали зображення
документа
        if (fileContent != null) { // є зображення
            err = engineObj.Img_LoadFromMemory(fileContent, fileContent.length);
            if (err == 0) {
                setStage(Constant.OCRSTEP_FIRST);
                OCR_APPLICATION.logger.log(Level.INFO, "image loaded
successfully");  //Logger.getLogger(OCR_ENGINE.class.getName())
            } // успішно завантажений
            else {
                OCR_APPLICATION.logger.log(Level.SEVERE, "image not loaded
err={0}", err);
            }
        } else {
            OCR_APPLICATION.logger.log(Level.SEVERE, "image not obtained fom DB
err={0}", err);
        }
        return err;
    }

    public int run(int lastStep, boolean inOtherThread) {
        int err = -1;
        if (getStage() > 0 & lastStep > getStage()) {
            err = engineObj.Img_OCR(getStage(), lastStep, inOtherThread ?
Constant.OCRFLAG_THREAD : Constant.OCRFLAG_NONE);
            if (err != 0) {
                OCR_APPLICATION.logger.log(Level.SEVERE, "image not recognised
err={0}", err);
            } else {
                setStage(lastStep);
            }
        } else {
```

```
            OCR_APPLICATION.logger.log(Level.SEVERE, "stage not valid err={0}",
err);
        }
        return err;
    }

    public int run(boolean inOtherThread) {
        int err = -1;
        if ((getStage() >= 0) & (getStage() < Constant.OCRSTEP_LAST)) {
            err = engineObj.Img_OCR(Constant.OCRSTEP_FIRST, Constant.OCRSTEP_LAST,
inOtherThread ? Constant.OCRFLAG_THREAD : Constant.OCRFLAG_NONE);
            if (err != 0) {
                OCR_APPLICATION.logger.log(Level.SEVERE, "image not recognised
err={0}", err);
            } else {
                setStage(Constant.OCRSTEP_LAST);
            }
        } else {
            OCR_APPLICATION.logger.log(Level.SEVERE, "stage not valid err={0}",
err);
        }
        return err;
    }

    public StringBuffer getText(Integer zoneNumber) {
        StringBuffer text = new StringBuffer();
        if (zoneNumber == null) {
            engineObj.Img_GetImgText(text, Constant.FMT_EXACTCOPY);
            return text;
        } else {
            HBLK BlkObj = new HBLK();
            engineObj.Img_GetBlock(zoneNumber - 1, BlkObj);
            getEngine().Blk_GetText(BlkObj, text, Constant.FMT_EXACTCOPY);
            return text;
        }
    }

    public String getText(int zone, int line, int word, int wordCount) {
        String res = "";
        StringBuffer text = new StringBuffer();
        HBLK BlkObj = new NSOCR.HBLK();// id зони
        int znCnt = engineObj.Img_GetBlockCnt();  //число зон
        if ((zone <= znCnt) & (zone > 0)) {  // номер зони в потрібному діапазоні
            engineObj.Img_GetBlock(zone - 1, BlkObj); //отримали зону
            int lnCnt = getEngine().Blk_GetLineCnt(BlkObj);   // число строк
            if ((line <= lnCnt) & (line > 0)) {  // номер лінії в потрібному
діапазоні
                int wdCnt = getEngine().Blk_GetWordCnt(BlkObj, line - 1); //число
слів
                if ((word <= wdCnt) & (word > 0)) {  // номер слова в потрібному
діапазоні
                    getEngine().Blk_GetWordText(BlkObj, line - 1, word - 1, text);
//get word text
                    res = text.toString().trim();
```

```
                    if (wordCount > 1) {
                        for (int i = word + 1; i < word + wordCount; i++) {
                            if ((i <= wdCnt) & (i > 0)) {  // номер слова в
потрібному діапазоні
                                StringBuffer textI = new StringBuffer();
                                getEngine().Blk_GetWordText(BlkObj, line - 1, i -
1, textI); //get word text
                                String sI = textI.toString().trim();
                                if (!sI.isEmpty()) {
                                    if (res.isEmpty()) {
                                        res = sI;
                                    } else {
                                        res = res + " " + sI;
                                    }
                                }
                            }
                        }
                    } else {
                        if (word == -1) {
                            getEngine().Blk_GetLineText(BlkObj, line - 1, text); //get
line text
                        } else {
                            //text.setLength(0);//номер слова вне діапазона
                        }
                    }
                } else {
                    if (line == -1) {
                        getEngine().Blk_GetText(BlkObj, text, Constant.FMT_EXACTCOPY);
                    } else {
                        //text.setLength(0);//номер слова вне діапазона
                    }
                }
            } else {
                //text.setLength(0);//номер зони вне діапазона
            }
            return res;
        }
        public int saveText(){
            Integer blockCount=0;
            Integer res=-1;
            Invoice invoice=docProcessed;
            blockCount =
DocFileOCRBlock.getDocFileOCRBlockCount(invoice.getDb().getMainDB(),
invoice.getIdImg());
            if (blockCount==null) return -1;
            if (blockCount>0){//delete old text
                if(!DocFileOCRBlock.delDocFileOCRBlocks(invoice.getDb().getMainDB(),
invoice.getIdImg()) ) return -1;   // unsuccessful deletion
            }
            try {

                invoice.getDb().getMainDB().setAutoCommit(false);
                try {
```

```
                for (int i = 1; i <= getZonesNumber(); i++) {
                     StringBuffer blockText = getText(i);
                    int[] zp = getZoneProp(i);
                    int blockType = zp[10];
                    int blockXPOS = zp[0];
                    int blockYPOS = zp[1];
                    int blockWIDTH = zp[2];
                    int blockHEIGHT = zp[3];
                    if (DocFileOCRBlock.addDocFileOCRBlock(invoice.getDb(),
invoice.getIdImg(), blockType, blockText.toString(),
blockText.toString().hashCode(), blockXPOS, blockYPOS, blockWIDTH, blockHEIGHT) >
0) {
                        blockCount++;
                    } else {
                        OCR_APPLICATION.logger.log(Level.INFO, "New block
{0}insertion failed    for DOC_FILE_ID = {1}", new Object[]{blockCount,
invoice.getIdImg()});
                        invoice.getDb().getMainDB().rollback();
                    }
                }
                if (Invoice.saveRecognizeStageToDb(invoice.getDb().getMainDB(),
invoice.getIdentifier(), RecognizeStage.RECOGNIZED_NOT_ANALYZED, 0, true) == 0) {
                     invoice.getDb().getMainDB().rollback();
                }
                invoice.getDb().getMainDB().commit();
                OCR_APPLICATION.logger.log(Level.INFO, "{0} blocks  inserted for
DOC_FILE_ID = {1}", new Object[]{blockCount, invoice.getIdImg()});
                res=blockCount;

            } finally {
               invoice.getDb().getMainDB().setAutoCommit(true);
            }
        } catch (SQLException ex) {
             OCR_APPLICATION.logger.log(Level.SEVERE, null, ex);
        }
       return res;
    }

    public int[] getZoneProp(int zoneNumber) {
        int j;
        int[] prop = new int[12];// список властивостей
        HBLK BlkObj = new NSOCR.HBLK();// id зони
        NSOCR.NSInt Xpos = new NSOCR.NSInt(0);
        NSOCR.NSInt Ypos = new NSOCR.NSInt(0);
        NSOCR.NSInt Width = new NSOCR.NSInt(0);
        NSOCR.NSInt Height = new NSOCR.NSInt(0);
        StringBuffer txt = new StringBuffer();

        engineObj.Img_GetBlock(zoneNumber - 1, BlkObj); //get Block object
        prop[10] = getEngine().Blk_GetType(BlkObj); //get zone type
        getEngine().Blk_GetRect(BlkObj, Xpos, Ypos, Width, Height); //get zone
position and size
        prop[0] = Xpos.GetValue();
        prop[1] = Ypos.GetValue();
```

```
        prop[2] = Width.GetValue();
        prop[3] = Height.GetValue();
        engineObj.Img_GetSize(Width, Height);
        prop[4] = prop[0] * OCR_ENGINE.TRANSFORM_COEF / Width.GetValue();
        prop[6] = prop[2] * OCR_ENGINE.TRANSFORM_COEF / Width.GetValue();
        prop[5] = prop[1] * OCR_ENGINE.TRANSFORM_COEF / Height.GetValue();
        prop[7] = prop[3] * OCR_ENGINE.TRANSFORM_COEF / Height.GetValue();
        prop[8] = getEngine().Blk_GetLineCnt(BlkObj); //get number of text lines
for zone

        // res = NSOCR.Engine.Blk_Inversion(BlkObj,
NSOCR.Constant.BLK_INVERSE_GET); //get zone inversion
        // res = NSOCR.Engine.Blk_Rotation(BlkObj, NSOCR.Constant.BLK_ROTATE_GET);
//get zone rotation
        // res = NSOCR.Engine.Blk_Mirror(BlkObj, NSOCR.Constant.BLK_MIRROR_GET);
//get zone mirror flag
        // res = NSOCR.Engine.Blk_GetBackgroundColor(BlkObj); //get zone
background color
        // NSOCR.Engine.Blk_GetText(BlkObj, txt, NSOCR.Constant.FMT_EDITCOPY);
//get zone text
        // System.out.println(txt);
        prop[9] = 0;
        for (j = 0; j < prop[8]; j++) {
            getEngine().Blk_GetLineText(BlkObj, j, txt); //get line text
            if (txt.length() > prop[9]) {
                prop[9] = txt.length();
            }
        }
        //   NSOCR.Engine.Blk_GetTextRect(BlkObj, j, -1, Xpos, Ypos, Width,
Height); //get text line position and size

        //   System.out.println(txt);
        //   wcnt = NSOCR.Engine.Blk_GetWordCnt(BlkObj, j); //get number of words
of "j"-th text line of zone
        //  for (k = 0; k < wcnt; k++)
        //  {
        //    NSOCR.Engine.Blk_GetTextRect(BlkObj, j, k, Xpos, Ypos, Width,
Height); //get word position and size
        //    res = NSOCR.Engine.Blk_GetWordFontColor(BlkObj, j, k); //get word
font color
        //    res = NSOCR.Engine.Blk_GetWordFontSize(BlkObj, j, k); //get word
font size
        //    res = NSOCR.Engine.Blk_GetWordFontStyle(BlkObj, j, k); //get word
font style
        //    res = NSOCR.Engine.Blk_GetWordQual(BlkObj, j, k); //get word quality
        //    res = NSOCR.Engine.Blk_IsWordInDictionary(BlkObj, j, k); //check if
this word is in dictionary
        //    NSOCR.Engine.Blk_GetWordText(BlkObj, j, k, txt); //get word text

        return prop;

    }
```

```
    public String findPos(int left, int up, int width, int height, int presize,
int fashion, int[] realPosition, int wordCount) {
        int right = left + width;
        int down = up + height;
        HBLK BlkObj = new NSOCR.HBLK();// id зони
        NSOCR.NSInt Xpos = new NSOCR.NSInt(0);
        NSOCR.NSInt Ypos = new NSOCR.NSInt(0);
        NSOCR.NSInt Wdth = new NSOCR.NSInt(0);
        NSOCR.NSInt Hght = new NSOCR.NSInt(0);
        engineObj.Img_GetSize(Wdth, Hght);
        int wBlank = Wdth.GetValue();
        int hBlank = Hght.GetValue(); // отримали размір бланку
        int znCnt = engineObj.Img_GetBlockCnt();  //число зон
        int zoneLeft = 0, zoneRight = 0, zoneUp = 0, zoneDown = 0;
        realPosition[0] = -1; //поки не знайшли зону
        for (int k = 0; k < znCnt; k++) {// шукаєм зону
            engineObj.Img_GetBlock(k, BlkObj); //отримали зону
            getEngine().Blk_GetTextRect(BlkObj, -1, -1, Xpos, Ypos, Wdth, Hght);
            zoneLeft = OCR_ENGINE.TRANSFORM_COEF * Xpos.GetValue() / wBlank;
            zoneRight = zoneLeft + OCR_ENGINE.TRANSFORM_COEF * Wdth.GetValue() /
wBlank;
            zoneUp = OCR_ENGINE.TRANSFORM_COEF * Ypos.GetValue() / hBlank;
            zoneDown = zoneUp + OCR_ENGINE.TRANSFORM_COEF * Hght.GetValue() /
hBlank;  //отримали позицію зони у % від розмірів бланку
            if ((zoneLeft <= (left + presize)) && (zoneUp <= (up + presize)) &&
((zoneRight + presize) >= right) && ((zoneDown + presize) >= down)) {
                realPosition[0] = k + 1;
                break;  // знайшли зону котра містить задану область
            }
        }
        if (realPosition[0] == -1) {
            return null;   //не знайшли зону котра містить задану область
        }
        realPosition[1] = -1; //поки не знайшли строку
        int lnCnt = getEngine().Blk_GetLineCnt(BlkObj);   // число строк
        for (int k = 0; k < lnCnt; k++) {// шукаєм строку
            getEngine().Blk_GetTextRect(BlkObj, k, -1, Xpos, Ypos, Wdth, Hght);
            zoneLeft = OCR_ENGINE.TRANSFORM_COEF * Xpos.GetValue() / wBlank;
            zoneRight = zoneLeft + OCR_ENGINE.TRANSFORM_COEF * Wdth.GetValue() /
wBlank;
            zoneUp = OCR_ENGINE.TRANSFORM_COEF * Ypos.GetValue() / hBlank;
            zoneDown = zoneUp + OCR_ENGINE.TRANSFORM_COEF * Hght.GetValue() /
hBlank;  //отримали позицію  строки у % від размерів бланку
            //if ((zoneLeft <= (left + presize)) && (zoneUp <= (up + presize)) &&
((zoneRight + presize) >= right) && ((zoneDown + presize) >= down)) {
            if ((zoneLeft <= (left + presize)) && (zoneUp <= (up + presize)) &&
((zoneDown + presize) >= down)) {
                realPosition[1] = k + 1;
                break;  // знайшли строку котра містить задану область
            }
        }
        if (realPosition[1] == -1) {
            return null;   // не знайшли строку котра містить задану область
        }
```

```
        realPosition[2] = -1; // пока не нашли слова
        int wdCnt = getEngine().Blk_GetWordCnt(BlkObj, realPosition[1] - 1);
//число слов
        for (int k = 0; k < wdCnt; k++) {// ищем слово
            getEngine().Blk_GetTextRect(BlkObj, realPosition[1] - 1, k, Xpos,
Ypos, Wdth, Hght);
            zoneLeft = OCR_ENGINE.TRANSFORM_COEF * Xpos.GetValue() / wBlank;
            zoneRight = zoneLeft + OCR_ENGINE.TRANSFORM_COEF * Wdth.GetValue() /
wBlank;
            zoneUp = OCR_ENGINE.TRANSFORM_COEF * Ypos.GetValue() / hBlank;
            zoneDown = zoneUp + OCR_ENGINE.TRANSFORM_COEF * Hght.GetValue() /
hBlank;//получили позицию  слова в % от размеров бланка
            if ((Math.abs(zoneLeft - left) < presize) && (Math.abs(zoneUp - up) <
presize)) { // начало совпадает в пределах точности
                if (fashion == -1) { // режим совпадения только начала
                    realPosition[2] = k + 1;
                    break;  // нашли слово которая содержит заданную область
                }
            }
            if ((Math.abs(zoneRight - right) <= presize) && (Math.abs(zoneDown -
down) <= presize)) { // конец совпадает в пределах точности
                if (fashion == 1) { // режим совпадения только конца
                    realPosition[2] = k + 1;
                    break;  // нашли слово которая содержит заданную область
                } else { // // режим совпадения обоих концов
                    if (realPosition[2] != -1) { // и начало  совпало
                        realPosition[2] = k + 1;
                        break;
                    }
                }
            }
        }
        if (realPosition[2] == -1) {
            for (int k = 0; k < wdCnt; k++) {// ищем слово
                getEngine().Blk_GetTextRect(BlkObj, realPosition[1] - 1, k, Xpos,
Ypos, Wdth, Hght);
                zoneLeft = OCR_ENGINE.TRANSFORM_COEF * Xpos.GetValue() / wBlank;
                zoneRight = zoneLeft + OCR_ENGINE.TRANSFORM_COEF * Wdth.GetValue()
/ wBlank;
                zoneUp = OCR_ENGINE.TRANSFORM_COEF * Ypos.GetValue() / hBlank;
                zoneDown = zoneUp + OCR_ENGINE.TRANSFORM_COEF * Hght.GetValue() /
hBlank;//получили позицию  слова в % от размеров бланка
                if ((Math.abs(zoneDown - down) <= presize) && (Math.abs(zoneUp -
up) < presize)) { // попали по высоте в пределах точности
                    if (Rect.intervalsAreCrossing(zoneLeft, zoneRight, left,
right)) {//пересеклись по горизонтали
                        realPosition[2] = k + 1;
                        break;
                    }
                }
            }
        }
        if (realPosition[2] == -1) {
            return null;   //не нашли слову в  заданную область
```

```
        }
        realPosition[3] = Xpos.GetValue();
        realPosition[4] = Ypos.GetValue();
        realPosition[5] = Wdth.GetValue();
        realPosition[6] = Hght.GetValue();
        realPosition[7] = zoneLeft;
        realPosition[8] = zoneUp;
        realPosition[9] = zoneRight - zoneLeft;
        realPosition[10] = zoneDown - zoneUp;
        // сформировали его позицию
        // выдали слово
        return getText(realPosition[0], realPosition[1], realPosition[2],
wordCount);
    }

    public String findPos2(int left, int up, int width, int height, int[]
realPosition) {
        //EngineObj engineObj2 = fwOcrAPP.getTextRecognizer().getEngineObj2();
        int res;
        Rect frame2 = new Rect();
        frame2.left = (int) ((float) left * (float) getFullSizeBmp().getWidth() /
(float) TRANSFORM_COEF);
        frame2.right = (int) (((float) left * (float) getFullSizeBmp().getWidth()
/ (float) OCR_ENGINE.TRANSFORM_COEF + (float) width * (float)
getFullSizeBmp().getWidth() / (float) OCR_ENGINE.TRANSFORM_COEF - 1)) + 1;
        frame2.top = (int) ((float) up * (float) getFullSizeBmp().getHeight() /
(float) TRANSFORM_COEF);
        frame2.bottom = (int) (((float) up * (float) getFullSizeBmp().getHeight()
/ (float) OCR_ENGINE.TRANSFORM_COEF + (float) height * (float)
getFullSizeBmp().getHeight() / (float) OCR_ENGINE.TRANSFORM_COEF - 1)) + 1;
        if (frame2.getWidth() > 0 && frame2.getHeight() > 0) {
            int[] resArr;
            resArr = getFullSizeBmp().getRGB(frame2.left, frame2.top,
frame2.getWidth(), frame2.getHeight(), null, 0, frame2.getWidth());
            res = getEngineObj2().Img_LoadBmpData(resArr, frame2.getWidth(),
frame2.getHeight(), NSOCR.Constant.BMP_32BIT);
            if (res > NSOCR.Error.ERROR_FIRST) {
                //JOptionPane.showMessageDialog(this, "Error Img_LoadBmpData " +
Integer.toHexString(res));
                Logger.getLogger(OCR_ENGINE.class.getName()).log(Level.SEVERE,
"Error Img_LoadBmpData {0}", Integer.toHexString(res));
                return null;
            }
            res = getEngineObj2().Img_OCR(Constant.OCRSTEP_FIRST,
Constant.OCRSTEP_LAST, Constant.OCRFLAG_NONE);
            //res = engineObj2.Img_OCR(0, 0, Constant.OCRFLAG_NONE);
            if (res > NSOCR.Error.ERROR_FIRST) {
                Logger.getLogger(OCR_ENGINE.class.getName()).log(Level.SEVERE,
"error Img_OCR {0}", Integer.toHexString(res));
                return null;
            }
            //Logger.getLogger(DocImagePanel.class.getName()).log(Level.INFO,
"engineObj2.Img_GetBlockCnt() = {0}", getEngineObj2().Img_GetBlockCnt());
            StringBuffer text = new StringBuffer();
```

```
            getEngineObj2().Img_GetImgText(text, Constant.FMT_EXACTCOPY);
            return text.toString().trim();
            //curItem.setRegex(text.toString());  // записали текущее слово в
шаблон
        }
        return null;
    }

    public int[] getWordPos(int zone, int line, int word) {
        int[] prop = new int[16];// список свойств
        HBLK BlkObj = new NSOCR.HBLK();// ид зоны
        NSOCR.NSInt Xpos = new NSOCR.NSInt(0);
        NSOCR.NSInt Ypos = new NSOCR.NSInt(0);
        NSOCR.NSInt Width = new NSOCR.NSInt(0);
        NSOCR.NSInt Height = new NSOCR.NSInt(0);

        int znCnt = engineObj.Img_GetBlockCnt();  //число зон
        if ((zone <= znCnt) & (zone > 0)) {  // номер зоны в нужном диапазоне
            engineObj.Img_GetSize(Width, Height);
            int wBlank = Width.GetValue();
            int hBlank = Height.GetValue(); // получили размер бланка
            engineObj.Img_GetBlock(zone - 1, BlkObj); //получили зону
            int lnCnt = getEngine().Blk_GetLineCnt(BlkObj);   // число строк
            getEngine().Blk_GetTextRect(BlkObj, -1, -1, Xpos, Ypos, Width,
Height); //get zone position and size
            prop[12] = Xpos.GetValue();
            prop[13] = Ypos.GetValue();
            prop[14] = Width.GetValue();
            prop[15] = Height.GetValue();
            if ((line <= lnCnt) & (line > 0)) {  // номер линиы в нужном
диапазоне
                int wdCnt = getEngine().Blk_GetWordCnt(BlkObj, line - 1); //число
слов
                if ((word <= wdCnt) & (word > 0)) {  // номер линиы в нужном
диапазоне
                    getEngine().Blk_GetTextRect(BlkObj, line - 1, word - 1, Xpos,
Ypos, Width, Height); //get word position and size
                    prop[4] = Xpos.GetValue();
                    prop[5] = Ypos.GetValue();
                    prop[6] = Width.GetValue();
                    prop[7] = Height.GetValue();
                    prop[0] = OCR_ENGINE.TRANSFORM_COEF * prop[4] / wBlank;
                    prop[1] = OCR_ENGINE.TRANSFORM_COEF * prop[5] / hBlank;
                    prop[2] = OCR_ENGINE.TRANSFORM_COEF * prop[6] / wBlank;
                    prop[3] = OCR_ENGINE.TRANSFORM_COEF * prop[7] / hBlank;
                    prop[8] = prop[4] - prop[12];
                    prop[9] = prop[5] - prop[13];
                    prop[10] = prop[6];
                    prop[11] = prop[7];
                    wBlank = prop[14];
                    hBlank = prop[15];
                    prop[12] = OCR_ENGINE.TRANSFORM_COEF * prop[8] / wBlank;
                    prop[13] = OCR_ENGINE.TRANSFORM_COEF * prop[9] / hBlank;
                    prop[14] = OCR_ENGINE.TRANSFORM_COEF * prop[6] / wBlank;
```

```
                    prop[15] = OCR_ENGINE.TRANSFORM_COEF * prop[7] / hBlank;

                } else {
                    prop[0] = -3;//номер слова вне диапазона
                }
            } else {
                prop[0] = -2;//номер линии вне диапазона
            }
        } else {
            prop[0] = -1;//номер зоны вне диапазона
        }
        return prop;
    }

    public int[] movePos(int lineShift, int wordShift, int lineOut, int wordOut,
int wordShiftType) {
        // TODO: implement
        return null;
    }

    public BufferedImage getImage() {
        return getImage(isDrawBinary());
    }

    public BufferedImage getImage(boolean drawBinary) {
        if (!isImgLoaded()) {
            return null;
        }
        NSInt width = new NSInt(0);
        NSInt height = new NSInt(0);
        if (engineObj.Img_GetSize(width, height) > NSOCR.Error.ERROR_FIRST) {
            return null;
        }
        return getImage(width.Value, height.Value);
    }

    public BufferedImage getImage(int newWidth, int newHeight) {
        return getImage(newWidth, newHeight, isDrawBinary());
    }

    public BufferedImage getImage(int newWidth, int newHeight, boolean drawBinary)
{
        if (!isImgLoaded()) {
            return null;
        }
        NSInt width = new NSInt(newWidth);
        NSInt height = new NSInt(newHeight);
        if (width.Value <= 0 || height.Value <= 0) {
            return null;
        }
        int bmpdata[] = new int[width.Value * height.Value];
        if (drawBinary) {
            engineObj.Img_GetBmpData(bmpdata, width, height,
NSOCR.Constant.DRAW_BINARY);
```

```
        } else {
            engineObj.Img_GetBmpData(bmpdata, width, height,
NSOCR.Constant.DRAW_NORMAL);
        }
        if (width.Value == 0 || height.Value == 0) {
            return null;
        }
        BufferedImage bmp = new BufferedImage(width.Value, height.Value,
BufferedImage.TYPE_INT_ARGB);
        bmp.setRGB(0, 0, width.GetValue(), height.GetValue(), bmpdata, 0,
width.GetValue());
        return bmp;
    }

    public int[] getImgSize() {
        NSInt width = new NSInt(0);
        NSInt height = new NSInt(0);
        int[] size = new int[2];
        int err = engineObj.Img_GetSize(width, height);
        // if (err > NSOCR.Error.ERROR_FIRST) { не пущает т.к
NSOCR.Error.ERROR_FIRST=1879048192 что гораздо более 0 при нормальном возврате
        if (err == 0) { // успешно получила размер
            size[0] = width.Value;
            size[1] = height.Value;
        }
        return size;
    }

    private boolean isImgLoaded() {
        NSInt width = new NSInt(0);
        NSInt height = new NSInt(0);

        if (engineObj.Img_GetSize(width, height) > NSOCR.Error.ERROR_FIRST) {
            return false;
        }
        return (width.Value > 0) && (height.Value > 0);
    }

    public int getZonesNumber() {
        int nz = 0;
        if (Constant.OCRSTEP_LAST == getStage()) {
            nz = engineObj.Img_GetBlockCnt(); //get number of zones
            setnBlocks(nz);
        }
        return nz;
    }

    public int getnBlocks() {
        return nBlocks;
    }

    public void setnBlocks(int nBlocks) {
        this.nBlocks = nBlocks;
    }
```

```
    public float getCurDocScale(int w, int h) {
        if (!isImgLoaded()) {
            return 1.0f;
        }
        NSInt width = new NSInt(0);
        NSInt height = new NSInt(0);
        engineObj.Img_GetSize(width, height);
        float kX = (float) w / width.Value;
        float kY = (float) h / height.Value;
        float k;
        if (kX > kY) {
            k = kY;
        } else {
            k = kX;
        }
        return k;
    }
    public boolean isTemplateApplicable(DocTemplate dt) {
        boolean isApplicable=true;
        StringBuffer text = new StringBuffer();
        engineObj.Img_GetImgText(text, Constant.FMT_EXACTCOPY);
        for (DocMeta dm : dt.getData()) {
            if (dm.getType() == DocMetaType.BLANK_KEY_WORD){
                String keyWord = dm.getRegex();
                if (text.indexOf(keyWord)==-1) {
                    return false;
                }
            }
        }
        return isApplicable;
    }


    public List<Integer> findTemplate() {
        List<Integer> templPull;
        Supplier supplier=getDocProcessed().getSupplier();
        if (supplier == null)supplier = new
Supplier(OCR_APPLICATION.fwOcrAPP.getDocDb());
        int supplierId=supplier.getIdentifier();
        DocTemplate dt=new DocTemplate(OCR_APPLICATION.fwOcrAPP.getDocDb());
        if (supplierId == DbObject.NOT_EXIST){

            if
(OCR_APPLICATION.fwOcrAPP.getDocRecognizer().getTmplPull().loadNextPull()>0){

templPull=OCR_APPLICATION.fwOcrAPP.getDocRecognizer().getTmplPull().getPull();
                for (Integer id:templPull) {
                   dt.loadObject(id);
                   if (isTemplateApplicable(dt)){
                      supplier=dt.getSupplier();
                      break;
                   }
                }
```

```
        }else {
            return null;
        }
    }
    return supplier.getTemplList(DocType.INVOICE);

}

public boolean isDrawBinary() {
    return drawBinary;
}

public void setDrawBinary(boolean drawBinary) {
    this.drawBinary = drawBinary;
}

private Engine getEngine() {
    return engineObj.getEngine();
}


public int Img_GetSize(NSInt Width, NSInt Height) {
    return engineObj.Img_GetSize(Width, Height);
}

public EngineObj getEngineObj() {
    return engineObj;
}

public EngineObj getEngineObj2() {
    if (engineObj2 == null) {
        engineObj2 = new EngineObj();
    }
    return engineObj2;
}

public BufferedImage getFullSizeBmp() {
    if (fullSizeBmp != null) {
        return fullSizeBmp;
    } else {
        fullSizeBmp = fwOcrAPP.getTextRecognizer().getImage(true);
    }
    return fullSizeBmp;
}

Rectangle getKeyRect(DocMeta recvDef) {
   String[] keywords;    // ключевое слово
   Rectangle keyRect=null;   // область найденного совпадения
   int dist,oldDist;            // различия в расположении
   int fConsidence; // 1е совпадение под слов
   int nConsidence,oldConsidence; // число совп слов
   int iNextSubWord;
   int iNextWord;
   int iPreviousSubWord;
```

```
        int iPreviousWord;
        StringBuffer TextStr=new StringBuffer ();
        HBLK BlkObj = new NSOCR.HBLK();// ид зоны
        NSOCR.NSInt Xpos = new NSOCR.NSInt(0);
        NSOCR.NSInt Ypos = new NSOCR.NSInt(0);
        NSOCR.NSInt Wdth = new NSOCR.NSInt(0);
        NSOCR.NSInt Hght = new NSOCR.NSInt(0);
        keywords =split(recvDef.getRegex()," "); // слова в ключевой фразе
        int nKsubWords=keywords.length;
        if (recvDef.getWord()==0){// реквизит не был распознан в исходном документе
         return null;
        }
        engineObj.Img_GetSize(Wdth, Hght);
        int wBlank = Wdth.GetValue();
        int hBlank = Hght.GetValue(); // получили размер бланка
        int znCnt = engineObj.Img_GetBlockCnt();  //число зон
        int zoneLeft = 0, zoneRight = 0, zoneUp = 0, zoneDown = 0;
         //  все строки в них и все слова на предмет совпадения с ключевым
         oldDist=-1;
         oldConsidence=-1;
         boolean isFind;
         for (int k = 0; k < znCnt; k++) {//просмотреть все зоны
             engineObj.Img_GetBlock(k, BlkObj); //получили зону
             getEngine().Blk_GetTextRect(BlkObj, -1, -1, Xpos, Ypos, Wdth, Hght);
             int lnCnt = getEngine().Blk_GetLineCnt(BlkObj);   // число строк в
этой зоне
             for (int l = 0; l < lnCnt; l++) {// все строки в ней
                 int wdCnt = getEngine().Blk_GetWordCnt(BlkObj, l); //число слов
                 for (int w= 0; w < wdCnt; w++) {// ищем слово
                     engineObj.getEngine().Blk_GetWordText(BlkObj, l, w, TextStr) ;
// выдрали слово
                     if (fConsidence>=0){// есь совпадение с 1 словом в ключевой
фразе
                         isFind=true;
                         nConsidence++;
                         getEngine().Blk_GetTextRect(BlkObj, l, w, Xpos, Ypos,
Wdth, Hght); //  определили прямоугольник содержащий это слово
                         zoneLeft = OCR_ENGINE.TRANSFORM_COEF * Xpos.GetValue() /
wBlank;
                         zoneRight = zoneLeft + OCR_ENGINE.TRANSFORM_COEF *
Wdth.GetValue() / wBlank;
                         zoneUp = OCR_ENGINE.TRANSFORM_COEF * Ypos.GetValue() /
hBlank;
                         zoneDown = zoneUp + OCR_ENGINE.TRANSFORM_COEF *
Hght.GetValue() / hBlank;  //получили позицию зоны в % от размеров бланка
                       //   if (fConsidence>0){  // не первое слово
                       //       iPreviousSubWord=fConsidence-1;
                       //       iPreviousWord=w-1;
                       //       while(iPreviousSubWord>=0 && iPreviousWord>=0 ){ //
пристыковываем к нему все слова слева из движка
                       //           iPreviousSubWord--;
                       //           iPreviousWord--;
                       //       }
```

```
                    //       getEngine().Blk_GetTextRect(BlkObj, l,
iPreviousWord+1, Xpos, Ypos, Wdth, Hght); //  определили прямоугольник содержащий
это слово
                    //       zoneLeft = OCR_ENGINE.TRANSFORM_COEF *
Xpos.GetValue() / wBlank;
                    //   }
                       if (nKsubWords>1){// не последнее подслово
                           iNextSubWord=1;
                           iNextWord=w+1;
                           while(iNextSubWord< nKsubWords && iNextWord<wdCnt ){
// определяем правое крайнее
                              engineObj.getEngine().Blk_GetWordText(BlkObj, l,
iNextWord, TextStr) ; // выдрали слово
                              int indx=TextStr.indexOf(keywords[iNextSubWord]);
                              if (indx>=0){// есь совпадение со след  словом в
ключевой фразе
                                nConsidence++; //увеличиваем число совпавших
                              } else {   // подслово в ключевой фразе не то
                                 isFind=false;
                                 break;
                              }
                              iNextSubWord++;
                              iNextWord++;
                           }
                           getEngine().Blk_GetTextRect(BlkObj, l, iNextWord-1,
Xpos, Ypos, Wdth, Hght); //  рихтуем правую границу
                           zoneRight = OCR_ENGINE.TRANSFORM_COEF *(
Xpos.GetValue()  +  Wdth.GetValue()) / wBlank;
                            // w=iNextWord-1; рихтуем счётчик слов (w) с учётом
проверенных подслов
                       }

                       if (isFind){
                           dist=abs(recvDef.getUp()-zoneUp)+
abs(recvDef.getLeft()-zoneLeft);
                           if ((oldDist==-1) ||( dist <oldDist)){// среди всех
найденых совпадений выбрать ближайший к шаблону по совпадению ирасположению
                             if (keyRect==null )keyRect = new Rectangle();
                               keyRect.setLocation(zoneLeft,zoneUp);
                               keyRect.setSize(zoneRight-zoneLeft,zoneDown-
zoneUp);
                               oldDist=dist;
                              // oldConsidence=nConsidence;
                           }
                       }
                   }
               }
           }
           return keyRect;
       }

       boolean findDataByPos(DocRecv recv, Dimension shift,int presize) {
           Rectangle dataRect=null;   // область поиска
```

```
        Rectangle findedRect=null;// область найденного текста
        DocMeta dataDef=recv.getRecvDef();
        HBLK BlkObj = new NSOCR.HBLK();// ид зоны
        NSOCR.NSInt Xpos = new NSOCR.NSInt(0);
        NSOCR.NSInt Ypos = new NSOCR.NSInt(0);
        NSOCR.NSInt Wdth = new NSOCR.NSInt(0);
        NSOCR.NSInt Hght = new NSOCR.NSInt(0);
        engineObj.Img_GetSize(Wdth, Hght);
        StringBuffer lineText;
        int wBlank = Wdth.GetValue();
        int hBlank = Hght.GetValue(); // получили размер бланка
        int znCnt = engineObj.Img_GetBlockCnt();  //число зон
        int zoneLeft = 0, zoneRight = 0, zoneUp = 0, zoneDown = 0;
        Rectangle inters;
        int sizeOfintersection;
        int oldSize;
        dataRect=new Rectangle(dataDef.getLeft()+shift.width,
dataDef.getUp()+shift.height, dataDef.getWidth(), dataDef.getHeight());   //
область поиска
        recv.setZone(-1); //пока не нешли зону
        oldSize=-1;
        for (int k = 0; k < znCnt; k++) {// ищем зону
            engineObj.Img_GetBlock(k, BlkObj); //получили зону
            lineText=this.getText(k+1);
            getEngine().Blk_GetTextRect(BlkObj, -1, -1, Xpos, Ypos, Wdth, Hght);
            Rectangle zoneRect=new Rectangle(OCR_ENGINE.TRANSFORM_COEF *
Xpos.GetValue() / wBlank,
                                             OCR_ENGINE.TRANSFORM_COEF *
Ypos.GetValue() / hBlank,
                                             OCR_ENGINE.TRANSFORM_COEF *
Wdth.GetValue() / wBlank,
                                             OCR_ENGINE.TRANSFORM_COEF *
Hght.GetValue() / hBlank );
              //получили позицию зоны в % от размеров бланка
            inters= zoneRect.intersection(dataRect); // пересечение области с
зоной
            if ((!inters.isEmpty()) && (!lineText.toString().trim().isEmpty()))
{// нашли зону которая пересекается с заданной область
                sizeOfintersection=inters.height+inters.width;
                if ((oldSize==-1) || (sizeOfintersection > oldSize) ) {// она
певрая или лучше уже найденной
                    recv.setZone(k + 1);
                    oldSize=sizeOfintersection;
                }
            }
        }
        if (recv.getZone()>=0) {//нашли зону которая содержит заданную область
продолжаем поиск
            oldSize=-1;
            engineObj.Img_GetBlock(recv.getZone()-1, BlkObj); //получили зону
            recv.setLine(-1); //пока не нешли строку
            int lnCnt = getEngine().Blk_GetLineCnt(BlkObj);   // число строк
            for (int k = 0; k < lnCnt; k++) {// ищем строку
```

```
                getEngine().Blk_GetTextRect(BlkObj, k, -1, Xpos, Ypos, Wdth,
Hght);
                lineText=new StringBuffer();
                getEngine().Blk_GetLineText(BlkObj, k, lineText);
                Rectangle lineRect=new Rectangle(OCR_ENGINE.TRANSFORM_COEF *
Xpos.GetValue() / wBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Ypos.GetValue() / hBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Wdth.GetValue() / wBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Hght.GetValue() / hBlank );
                //получили позицию строки  в % от размеров бланка
                inters= lineRect.intersection(dataRect); // пересечение области с
линией
                if (!inters.isEmpty()) {// нашли строку которая пересекается с
заданной областью
                    sizeOfintersection=inters.height+inters.width;
                    if ((oldSize==-1) || (sizeOfintersection > oldSize) ) {// она
певрая или лучше уже найденной
                        recv.setLine(k + 1);
                        oldSize=sizeOfintersection;
                    }
                }
            }
            if (recv.getLine()>=0) {//нашли строку которая содержит заданную
область продолжаем поиск
                int nLine=recv.getLine()-1;
                int wdCnt = getEngine().Blk_GetWordCnt(BlkObj, nLine);
                oldSize=-1;
                recv.setWord(-1); //пока не нашли слово
                recv.setNwords(0);
                for (int k = 0; k < wdCnt; k++) {// ищем слово
                   getEngine().Blk_GetTextRect(BlkObj, nLine, k, Xpos, Ypos, Wdth,
Hght);
                   Rectangle wordRect=new Rectangle(OCR_ENGINE.TRANSFORM_COEF *
Xpos.GetValue() / wBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Ypos.GetValue() / hBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Wdth.GetValue() / wBlank,
                                                 OCR_ENGINE.TRANSFORM_COEF *
Hght.GetValue() / hBlank );

                    inters= wordRect.intersection(dataRect); // пересечение
области с словой
                    if (!inters.isEmpty()) {// нашли слову которая пересекается с
заданной областью

                        if (recv.getNwords()==0) {//  певрая слова
                            recv.setWord(k + 1);
                            recv.setNwords(1);
```

```
                        } else{
                            recv.setNwords(recv.getNwords()+1);

                        }
                    }
                }
                if (recv.getNwords()!=0){ // нашлось слово
                    recv.setRecvValue(getText(recv.getZone(), recv.getLine(),
recv.getWord(), recv.getNwords()));
                    recv.setStatus(0);
                } else{// слова не нашлась
                   recv.setStatus(3);
                }
            } else{ // линия не нашлась
                recv.setStatus(2);
            }

        }  else { // зона не нашлась
            recv.setStatus(1);
        }
        if (recv.getStatus()==0){//выделилась реквизита
            return true;
        } else{
            return false;
        }
        }
    boolean findData(DocRecv recv, Dimension shift,int method,int precision) {
        switch (method){
            case 1:
               return findDataByPos(recv,shift,precision);
            case 2:
               return  findDataByEngine(recv,shift,precision);
            default:
              return false;
        }


    }
    private boolean findDataByEngine(DocRecv recv, Dimension shift, int precision)
{

         Rectangle dataRect=null;   // область поиска
         DocMeta dataDef=recv.getRecvDef();
         dataRect=new Rectangle(dataDef.getLeft()+shift.width,
dataDef.getUp()+shift.height, dataDef.getWidth(), dataDef.getHeight());   //
область поиска

         int res=0;
         int nStep=5;
         int iStep=0;
         String protocol="";

        if (dataRect.width > 0 && dataRect.height > 0) {
            while (iStep<nStep){
```

```
                int left= (int) ( (float)dataRect.x * (float)
getFullSizeBmp().getWidth() / (float) TRANSFORM_COEF);
                int  top=(int) ((float)dataRect.y * (float)
getFullSizeBmp().getHeight() / (float) TRANSFORM_COEF);
                int  hght=(int)((float)dataRect.height * (float)
getFullSizeBmp().getHeight() / (float) OCR_ENGINE.TRANSFORM_COEF );
                int  wdth=(int)((float)dataRect.width * (float)
getFullSizeBmp().getWidth() / (float) OCR_ENGINE.TRANSFORM_COEF );
                int[] resArr;
                try {
                    resArr = getFullSizeBmp().getRGB(left, top, wdth, hght, null,
0, wdth);
                    res = getEngineObj2().Img_LoadBmpData(resArr, wdth, hght,
NSOCR.Constant.BMP_32BIT);
                } catch (Exception ex){
                     recv.setStatus(9);
                    protocol=protocol+"Load rectangle image failed for
"+recv.getRecvDef().getName().name()+" step"+Integer.toString(iStep);

protocol=protocol+"("+Integer.toString(left)+","+Integer.toString(top)+","+Integer
.toString(wdth)+","+Integer.toString(hght)+")";
                    break;
                }


                if (res <= NSOCR.Error.ERROR_FIRST) {   // образ загружен в движок
успешно
                    res = getEngineObj2().Img_OCR(Constant.OCRSTEP_FIRST,
Constant.OCRSTEP_LAST, Constant.OCRFLAG_NONE);
                    if (res <= NSOCR.Error.ERROR_FIRST) {// образ распознан
успешно
                        StringBuffer text = new StringBuffer();
                        getEngineObj2().Img_GetImgText(text,
Constant.FMT_EXACTCOPY);
                        recv.setRecvValue(text.toString().trim());
                        if (!recv.getRecvValue().isEmpty()){
                            recv.setStatus(0);
                            break;
                         //    return true;
                        } else {
                           // OCR_APPLICATION.logger.log(Level.SEVERE, "extracted
text is empty {0}", Integer.toHexString(res));
                            protocol=protocol+"extracted text is empty
"+Integer.toString(iStep);
                            recv.setStatus(10);
                           // return false;
                        }
                    } else {
                        //OCR_APPLICATION.logger.log(Level.SEVERE, "error Img_OCR
{0}", Integer.toHexString(res));
                        protocol=protocol+"error
Img_OCR"+Integer.toHexString(res)+" on step "+Integer.toString(iStep);
                        recv.setStatus(9);
                        //return false;
```

```
                    }
                } else {
                    //JOptionPane.showMessageDialog(this, "Error Img_LoadBmpData "
+ Integer.toHexString(res));
                    //OCR_APPLICATION.logger.log(Level.SEVERE, "Error
Img_LoadBmpData {0}", Integer.toHexString(res));
                    protocol=protocol+"Error Img_LoadBmpData on step
"+Integer.toString(iStep);
                    recv.setStatus(8);
                  //  return false;
                }
                dataRect.grow( precision,  precision);
                iStep++;
            }


            //curItem.setRegex(text.toString());  // записали текущее слово в
шаблон
        }else {
            recv.setStatus(11);
            protocol=protocol+"data rectangle has zero size ";
        }
        if (!protocol.isEmpty()){
          OCR_APPLICATION.logger.log(Level.SEVERE, protocol);
        }
        if (recv.getStatus()==0) {
            return true;
        }else{
            return false;
        }
    }
}
```