МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram**

Виконав: студент 4 курсу, групи 6.1212-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
І.С. Кобіков
(ініціали та прізвище)
Керівник асистент, доктор філософії (PhD)
О. О. Жуков
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ім. Ю.М. Потебні ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кобікову Ігорю Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram,
керівник роботи Жуков Олександр Олексійович, д. ф. (PhD), асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ЗНУ від 23.12.2025 р. № 1830-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2026
3. Вихідні дані бакалаврської роботи
   - комплект нормативних документів ;
   - технічне завдання до роботи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   - огляд та збір літератури стосовно теми кваліфікаційної роботи;
   - аналіз предметної області та існуючих програмних рішень;
   - формулювання вимог до системи автоматичного анотування наукових статей;
   - проєктування архітектури, бази даних і модулів обробки PDF;
   - реалізація TG-бота, backend, панелі адміна та сервісів аналізу;
   - тестування системи, аналіз отриманих результатів і формування висновків.
5. Перелік графічного матеріалу (презентація до захисту: схема архітектури системи, схема послідовності обробки PDF-документа, схема бази даних)
14 слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	01.03–05.03.2026	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	06.03–10.03.2026	виконано
3	Огляд існуючих рішень для роботи з науковими статтями	11.03–17.03.2026	виконано
4	Проєктування структури бази даних і ролей користувачів	18.03–24.03.2026	виконано
5	Проєктування архітектури backend, Telegram-бота та web-панелі	25.03–31.03.2026	виконано
6	Реалізація backend core, моделей бази даних і міграцій	01.04–07.04.2026	виконано
7	Реалізація Telegram-бота, головного меню та авторизації користувачів	08.04–13.04.2026	виконано
8	Реалізація завантаження PDF і технічного кешування файлів	14.04–20.04.2026	виконано
9	Реалізація DOI-пошуку, отримання метаданих і локального аналізу тексту	21.04–25.04.2026	виконано
10	Реалізація бібліотеки, категорій, пошуку та картки статті	26.04–28.04.2026	виконано
11	Розробка адміністративної web-панелі та статистики	29.04–02.05.2026	виконано
12	Підключення шару LLM/API-аналізу для повторної перевірки результатів	03.05–05.05.2026	виконано
13	Тестування системи та усунення помилок	06.05–08.05.2026	виконано
14	Підготовка до подальшого перенесення на хостинг	09.05–10.05.2026	виконано
15	Створення презентації та демонстраційного сценарію	11.05–12.05.2026	виконано
16	Оформлення кваліфікаційної роботи	13.05–17.05.2026	виконано

Студент _____ _____І.С. Кобіков

Керівник роботи _____ _____О. О. Жуков

Нормоконтроль пройдено

Нормоконтролер _____ _____І.А. Скрипник

АНОТАЦІЯ

Сторінок — 65.

Рисунків — 8.

Таблиць — 10.

Джерел — 18.

Кобіков І. Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник О.О. Жуков. Запоріжжя : ЗНУ, 2026. 65 с.

У кваліфікаційній роботі розглянуто проблему організації особистої бібліотеки наукових статей і отримання структурованої інформації з PDF-документів. Актуальність роботи зумовлена тим, що звичайне збереження файлів не забезпечує зручної навігації, пошуку за метаданими, категоризації та повторного використання результатів аналізу.

Метою роботи є розробка інформаційної системи, що поєднує Telegram-бот, backend, базу даних, локальний кеш PDF, модулі DOI/метаданих, локального аналізу тексту, LLM/API-шар повторної перевірки, статистику, журналювання та адміністративну web-панель. У роботі проаналізовано предметну область, розглянуто аналоги, обґрунтовано Python-стек і сформульовано вимоги до системи.

Практичне значення полягає у створенні працездатної основи сервісу для роботи з науковими матеріалами. Система дає змогу завантажувати PDF через Telegram, переглядати власну бібліотеку, отримувати DOI, метадані, категорії, короткий і розширений описи, орієнтовний УДК із поясненням та посилання за ДСТУ. Адміністратор може контролювати статті, користувачів, журнали подій і статистику обробок.

Ключові слова: *Telegram-бот, наукова стаття, PDF, анотування, DOI, метадані, УДК, ДСТУ, FastAPI, aiogram, PostgreSQL, SQLAlchemy, LLM, адміністративна панель.*

ABSTRACT

Pages — 65.

Drawings — 8.

Tables — 10.

Sources — 18.

Kobikov I. Development of a system for automatic annotation of scientific articles with integration into Telegram messenger: bachelor qualification work in specialty 121 “Software Engineering” / scientific supervisor O.O. Zhukov. Zaporizhzhia : ZNU, 2026. 65 p.

The qualification work considers the problem of organizing a personal library of scientific articles and obtaining structured information from PDF documents. The relevance of the work is determined by the fact that ordinary file storage does not provide convenient navigation, metadata search, categorization and reuse of analysis results.

The purpose of the work is to develop an information system that combines a Telegram bot, backend, database, local PDF cache, DOI/metadata modules, local text analysis, an LLM/API layer for rechecking, statistics, logging and an administrative web panel. The subject area was analyzed, existing solutions were reviewed, the Python stack was justified and system requirements were formulated.

The practical value lies in creating a working basis for a service for scientific materials. The system allows users to upload PDF articles via Telegram, view a personal library, receive DOI, metadata, categories, short and extended summaries, an approximate UDC code with explanation and a DSTU reference. The administrator can monitor articles, users, event logs and processing statistics.

Keywords: *Telegram bot, scientific article, PDF, annotation, DOI, metadata, UDC, DSTU, FastAPI, aiogram, PostgreSQL, SQLAlchemy, LLM, admin panel.*

ЗМІСТ

ВСТУП 8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 13
1.1 Опис проблеми автоматичного анотування наукових статей 13
1.2 Огляд літературних джерел і технологічних основ 15
1.3 Огляд існуючих рішень 17
1.4 Постановка задачі 20
1.5 Висновки до розділу 22
2 ДОСЛІДЖЕННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ФОРМУВАННЯ ВИМОГ 24
2.1 Вимоги до апаратного забезпечення 24
2.2 Вимоги до програмного забезпечення 25
2.3 Вимоги до користувачів 25
2.4 Організаційні вимоги 26
2.5 Архітектура системи 28
2.5.1 Загальна архітектурна модель 28
2.5.2 Взаємодія основних компонентів 28
2.5.3 Сервісний шар системи 29
2.5.4 Адміністративна частина 29
2.6 Специфікація даних 31
2.6.1 Опис форматів і структури даних 31
2.6.2 Основні сутності системи 32
2.6.3 Зв’язки між сутностями 33
2.6.4 Обмеження цілісності даних 34
2.7 Функціональні вимоги 36
2.7.1 Вимоги до користувацьких функцій 36
2.7.2 Вимоги до функцій адміністратора 37
2.7.3 Вимоги до обробки статті 38
2.8 Вимоги до інтерфейсу 39
2.8.1 Telegram-інтерфейс користувача 39

2.8.2 Картка статті та навігація 41
2.8.3 Web-панель адміністратора 41
2.9 Інші вимоги 44
2.9.1 Вимоги до надійності 44
2.9.2 Вимоги до продуктивності 44
2.9.3 Вимоги до безпеки і журналювання 44
2.9.4 Вимоги до супроводжуваності 45
2.10 Висновки до розділу 46
3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ 47
3.1 Засоби реалізації 47
3.2 Модулі і алгоритми 47
3.2.1 Деталізація алгоритму обробки PDF-документа 48
3.2.2 Логіка розмежування доступу користувачів 50
3.3 Структури даних 51
3.4 Проєкт інтерфейсу 51
3.5 Реалізація програмних модулів 54
3.5.1 Реалізація backend і бази даних 54
3.5.2 Реалізація Telegram-бота 54
3.5.3 Реалізація модулів аналізу 55
3.5.4 Реалізація адміністративної панелі 56
3.6 Висновки до розділу 56
4 ТЕСТУВАННЯ, РЕЗУЛЬТАТИ ТА ОЦІНКА ПРАЦЕЗДАТНОСТІ 57
4.1 Методика локального тестування 59
4.2 Аналіз результатів тестування 61
4.3 Висновки до розділу 62
ВИСНОВКИ 63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

ВСТУП

Робота з науковими статтями є важливою складовою навчальної, дослідницької та професійної діяльності студентів, викладачів і фахівців у сфері інформаційних технологій. На практиці вона часто зводиться до збереження PDF-файлів у локальних папках, хмарних сховищах або повідомленнях месенджера. Однак зі збільшенням кількості матеріалів виникає потреба у швидкому пошуку потрібної статті, розумінні її змісту, виділенні ключових понять, формуванні бібліографічного посилання та впорядкуванні документів за темами.

Актуальність теми зумовлена тим, що звичайне накопичення PDF-файлів не вирішує задачу систематизації знань. Користувач може мати доступ до самого документа, але не мати зручного способу швидко визначити, про що йдеться у статті, хто є її авторами, який DOI зазначено, до якої категорії належить матеріал, який орієнтовний УДК може бути запропонований і як сформувати посилання за ДСТУ. У таких умовах доцільною є розробка інформаційної системи, яка не лише зберігає технічні дані про документ, а й формує структурований опис наукового матеріалу.

Особливістю розробленої системи є використання Telegram не просто як каналу сповіщень, а як основного користувацького інтерфейсу. Telegram-бот дає змогу працювати з системою без окремого мобільного або настільного клієнтського застосунку. Користувач надсилає PDF-файл у чат, отримує службові повідомлення про стан аналізу, переглядає власну бібліотеку, відкриває картку статті, переходить до категорій, виконує пошук і завантажує початковий файл. Такий підхід робить систему доступною з телефона, планшета або комп’ютера [1].

Водночас Telegram у межах цієї роботи не розглядається як єдине сховище інформації. Файл може залишатися доступним у Telegram-повідомленні, але база даних системи зберігає користувачів, статті, метадані, категорії, результати аналізу, журнали подій, статистику обробок і дані про

використання LLM/API-аналізу. Локальна копія PDF використовується як технічний кеш для обробки, а не як єдина довгострокова форма збереження файлу. Це відповідає практичній логіці системи: зберігати лише ті дані, які потрібні для навігації, аналізу та повторного використання результатів.

Метою кваліфікаційної роботи є розробка інформаційної системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram, яка забезпечує завантаження PDF-документів, отримання метаданих, формування структурованих результатів аналізу, організацію особистої бібліотеки користувача, категоризацію матеріалів, пошук, статистику, журналювання та базове адміністрування.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати предметну область роботи з науковими статтями та проблеми організації особистої бібліотеки PDF-документів;

– розглянути існуючі програмні рішення для зберігання, аналізу та анотування наукових матеріалів;

– сформулювати вимоги до інформаційної системи, Telegram-інтерфейсу, backend-логіки та адміністративної частини;

– обґрунтувати вибір технологічного стеку для локального запуску і подальшої підготовки до хостингу;

– спроєктувати структуру бази даних для зберігання користувачів, статей, метаданих, категорій, результатів аналізу, статистики та журналів;

– реалізувати Telegram-бота з головним меню, бібліотекою, завантаженням PDF, пошуком, категоріями та карткою статті;

– реалізувати backend на FastAPI, service layer для PDF, DOI, метаданих, локального аналізу та LLM/API-шару;

– реалізувати адміністративну web-панель для перегляду користувачів, статей, статусів аналізу, категорій, журналів і статистики використання;

– перевірити працездатність системи локально та підготувати її до подальшого розгортання.

Об’єктом дослідження є процес збереження, пошуку, аналізу та впорядкування наукових статей у цифровому середовищі. Об’єкт охоплює взаємодію користувача з власною бібліотекою матеріалів, отримання метаданих, повторне використання результатів аналізу та адміністративний контроль роботи системи.

Предметом дослідження є програмні засоби, архітектурні рішення та алгоритмічні підходи до створення системи автоматичного анотування наукових статей з Telegram-інтерфейсом. До предмета дослідження належать структура бази даних, взаємодія Telegram-бота з backend, обробка PDF, пошук DOI, отримання метаданих із зовнішніх джерел, локальне витягнення тексту, формування короткого змісту, розширеного опису, орієнтовного УДК, посилання за ДСТУ, категорій і ключових слів.

У процесі виконання роботи використано методи аналізу, порівняння, системного підходу, моделювання, проєктування баз даних і практичного програмування. Метод аналізу застосовано для вивчення предметної області та вимог до системи. Метод порівняння використано для зіставлення існуючих рішень. Системний підхід застосовано для поділу програмного продукту на Telegram-інтерфейс, backend, базу даних, локальне сховище, модулі аналізу та web-панель адміністратора. Моделювання використано під час визначення сутностей і сценаріїв використання. Практичне програмування застосовано для реалізації й перевірки працездатності створеної системи.

Елементи новизни роботи полягають у подальшому розвитку підходу до побудови прикладної інформаційної системи для роботи з науковими статтями, у якій Telegram використовується як повноцінний інтерфейс доступу до бібліотеки, а аналіз документа виконується поетапно: спочатку перевіряється DOI та готові метадані, потім використовується локальне витягнення тексту, а LLM/API-аналіз підключається лише за потреби. Такий підхід відрізняється від простого завантаження файлів у чат тим, що система формує окремі сутності для метаданих, короткого змісту, розширеного опису, УДК, ДСТУ-посилання, категорій і статистики обробки.

Практичне значення роботи полягає у створенні працездатної основи сервісу для організації особистої бібліотеки наукових матеріалів. Система може бути використана студентом, викладачем або дослідником для швидкого додавання PDF-статей, перегляду їхніх метаданих, отримання короткого змісту, пошуку за назвою, DOI, автором або категорією, а також для контролю власної статистики. Адміністративна панель дає змогу контролювати користувачів, статті, статуси аналізу, журнали подій і потенційні витрати або кількість запусків LLM/API-аналізу.

Особистий внесок автора полягає у формуванні вимог до системи, проєктуванні її структури, реалізації основних програмних модулів, налаштуванні бази даних, Telegram-інтерфейсу, сервісів аналізу, адміністративної панелі та перевірці працездатності системи у локальному середовищі. Окремо було враховано зауваження щодо доцільності використання Telegram як довгострокового доступу до файлу та необхідності підключення LLM саме для короткого змісту, УДК і бібліографічного посилання.

Апробація результатів роботи здійснена шляхом підготовки тез «Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram», поданих до збірника університетської конференції «Молода наука-2026» [18]. У тезах обґрунтовано актуальність системи, її функціональні можливості та практичну цінність Telegram-інтерфейсу.

Структура роботи відповідає логіці розробки програмного забезпечення. У вступі обґрунтовано актуальність теми, визначено мету, завдання, об’єкт, предмет, методи дослідження, елементи новизни та практичне значення. У першому розділі подано аналіз предметної області, огляд літературних джерел і програмних продуктів-аналогів. У другому розділі досліджено засоби реалізації та сформовано вимоги до системи. У третьому розділі описано проєктування та реалізацію програмної системи. У четвертому розділі наведено тестування, результати та оцінку працездатності розробленої системи.

Глосарій

API — програмний інтерфейс взаємодії між компонентами системи або зовнішніми сервісами.

Backend — серверна частина інформаційної системи, яка виконує бізнес-логіку, роботу з базою даних і службові операції.

Bot API — інтерфейс Telegram для створення ботів і роботи з повідомленнями, файлами та callback-запитами.

DOI — цифровий ідентифікатор об’єкта, який використовується для пошуку та отримання метаданих наукової публікації.

LLM — локальна мовна модель або сумісний локальний сервіс, що використовується у системі для повторної перевірки та уточнення структурованого опису.

ORM — підхід до роботи з базою даних через об’єктні моделі мовою програмування.

PDF — формат електронного документа, у якому користувач завантажує наукові статті.

SQL — мова структурованих запитів до реляційної бази даних.

UML — уніфікована мова моделювання для опису структури і поведінки програмних систем.

УДК — універсальна десяткова класифікація, що використовується для тематичного позначення наукового матеріалу.

ДСТУ — державний стандарт України, у межах роботи використовується для формування бібліографічного посилання.

Кеш PDF — локальна технічна копія PDF-файлу, потрібна для аналізу документа в локальному середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис проблеми автоматичного анотування наукових статей

Предметна область кваліфікаційної роботи пов’язана з організацією роботи користувача з науковими PDF-статтями. У межах цієї області взаємодіють користувачі, які завантажують статті до власної бібліотеки, програмні модулі, які виконують аналіз документів, зовнішні сервіси метаданих, локальне файлове сховище, база даних і адміністратор системи. Основна цінність такої системи полягає не у простому збереженні PDF-файлу, а у перетворенні неструктурованого документа на набір зручних для пошуку та перегляду даних.

Наукова стаття як документ має кілька рівнів інформації. Перший рівень становлять формальні метадані: назва, автори, журнал, рік публікації, DOI, сторінки та видавець. Другий рівень становить змістова інформація: короткий зміст, розширений опис, ключові слова, категорії та орієнтовна тематика. Третій рівень пов’язаний із бібліографічним оформленням, зокрема з посиланням за ДСТУ 8302:2015. У ручному режимі користувачеві доводиться самостійно відкривати PDF, шукати DOI, копіювати дані, формувати короткий опис і створювати бібліографічний запис. Це потребує часу та часто призводить до неповноти або неузгодженості результатів.

Важливою проблемою є те, що готові метадані не завжди доступні в самому PDF або можуть бути подані неповно. Якщо DOI знайдено, система може звернутися до зовнішнього джерела, наприклад Crossref, і отримати частину відомостей без дорогого аналізу всього тексту [8]. Якщо DOI немає або метаданих недостатньо, необхідно виконати локальне витягнення тексту. Якщо якість локально витягнутого тексту низька або отриманий результат потребує уточнення, доцільно підключати LLM/API-аналіз. Така послідовність є важливою, оскільки дає змогу зменшити кількість зовнішніх запитів і повторно використовувати вже збережені результати.

Окремим аспектом є багатокористувацький режим. Кожен користувач повинен працювати тільки зі своєю бібліотекою статей. Це означає, що всі сутності — статті, результати аналізу, категорії, дії, статистика та налаштування — мають бути прив’язані до Telegram user id або до внутрішнього запису користувача, створеного на його основі. Такий підхід забезпечує логічне розмежування даних і не дозволяє одному користувачу бачити матеріали іншого.

Telegram-інтерфейс вирішує задачу доступності системи. Користувачу не потрібно встановлювати окрему програму або відкривати складний web-кабінет для базових дій. Він може надіслати PDF-файл у чат, отримати повідомлення «Статтю завантажено», побачити статус «Аналіз триває», перейти до власної бібліотеки, відкрити картку статті, переглянути короткий зміст, УДК або ДСТУ-посилання. Водночас для адміністратора доцільним є окремий web-інтерфейс, оскільки перегляд великої кількості користувачів, журналів і статистики зручніше виконувати через таблиці та сторінки адмін-панелі.

Отже, проблема полягає у відсутності простого прикладного середовища, яке поєднує Telegram-доступ, власну бібліотеку статей, пошук DOI, отримання метаданих, локальне витягнення тексту, структурований аналіз, категоризацію, статистику використання, журналювання та адміністративний контроль. Саме ця проблема визначає необхідність розробки власної інформаційної системи.

Загальну архітектуру системи автоматичного анотування наукових статей подано на рисунку 1.



Рисунок *1 – Загальна архітектура системи автоматичного анотування наукових статей*

1.2 Огляд літературних джерел і технологічних основ

Під час аналізу предметної області було враховано літературні та електронні джерела, пов’язані з Telegram Bot API, Python-розробкою, backend-фреймворками, базами даних, обробкою PDF, отриманням метаданих і бібліографічним оформленням. Оскільки робота має прикладний характер, основну увагу приділено тим джерелам, які безпосередньо впливають на архітектуру та реалізацію системи.

Основним інтерфейсом користувача є Telegram-бот. Telegram Bot API надає можливість отримувати повідомлення, документи, callback-запити, працювати з кнопками та надсилати відповіді користувачу [1]. Для реалізації бота використано aiogram, оскільки ця бібліотека орієнтована на асинхронну роботу з Telegram API, підтримує маршрутизатори, фільтри, FSM-стани та зручну структуру handlers [2]. У межах розробленої системи aiogram використовується для команд /start, головного меню, завантаження PDF, перегляду бібліотеки, пошуку, категорій і картки статті.

Базовою мовою програмування обрано Python. Офіційна документація Python описує мову як універсальну та придатну для створення різних типів програмного забезпечення [3]. Для дипломного проєкту Python є доцільним

вибором, оскільки дає змогу в межах одного стеку реалізувати Telegram-бота, backend, роботу з базою даних, обробку PDF і службові модулі аналізу.

Backend-частину реалізовано з використанням FastAPI. Цей фреймворк підтримує створення API, автоматичну документацію OpenAPI, валідацію даних і асинхронну обробку запитів [4]. У розробленій системі FastAPI використовується для службових API-маршрутів, healthcheck, підключення адміністративної web-панелі та загальної організації backend-застосунку.

Для роботи з базою даних використано SQLAlchemy та Alembic. SQLAlchemy надає можливості ORM і побудови SQL-запитів у Python [5], а Alembic використовується для керування міграціями схеми бази даних [7]. Це важливо для дипломного проєкту, оскільки структура даних поступово розширюється: спочатку додаються користувачі та статті, потім метадані, категорії, результати аналізу, журнали подій, статистика, Telegram-посилання на файл і ознаки використання зовнішнього аналізу.

Системою керування базами даних обрано PostgreSQL. PostgreSQL є потужною об’єктно-реляційною системою з відкритим кодом [6]. У межах цієї роботи PostgreSQL використовується для зберігання користувачів, статей, технічних даних про PDF, DOI, метаданих, авторів, категорій, ключових слів, коротких і розширених описів, УДК, ДСТУ-посилань, запусків аналізу та журналів подій.

Для отримання готових метаданих використовується Crossref REST API. Crossref надає можливість отримувати метадані публікацій за DOI [8]. Це відповідає логіці системи: перш ніж виконувати дорожчий змістовний аналіз, доцільно перевірити наявність DOI і спробувати отримати вже підготовлені дані про статтю.

Для локального витягнення тексту з PDF використовується pypdf. Такий підхід дає змогу отримувати текст із документа без звернення до зовнішніх сервісів і використовувати його для первинного формування короткого змісту, розширеного опису та орієнтовного УДК [10]. Якщо локальний результат недостатній, система може перейти до LLM/API-аналізу.

Для перспективного або локального LLM-аналізу передбачено локальний API-сервер, сумісний із форматом chat/completions підхід через LM Studio або інший сумісний сервер. LM Studio підтримує запуск локального сервера для роботи з моделями через API [11]. У системі цей шар не замінює всі інші методи, а використовується тільки за потреби: якщо DOI не дав достатніх метаданих або локальний аналіз сформував результат із низькою якістю.

Для адміністративної частини використано Jinja2-шаблони. Jinja2 є шаблонізатором для Python, який дає змогу формувати HTML-сторінки на основі переданих даних [17]. У розробленій системі web-панель адміністратора забезпечує перегляд користувачів, статей, статусів аналізу, категорій, журналів, запусків аналізу та статистики використання.

Послідовність обробки PDF-документа в системі показано на рисунку 2.

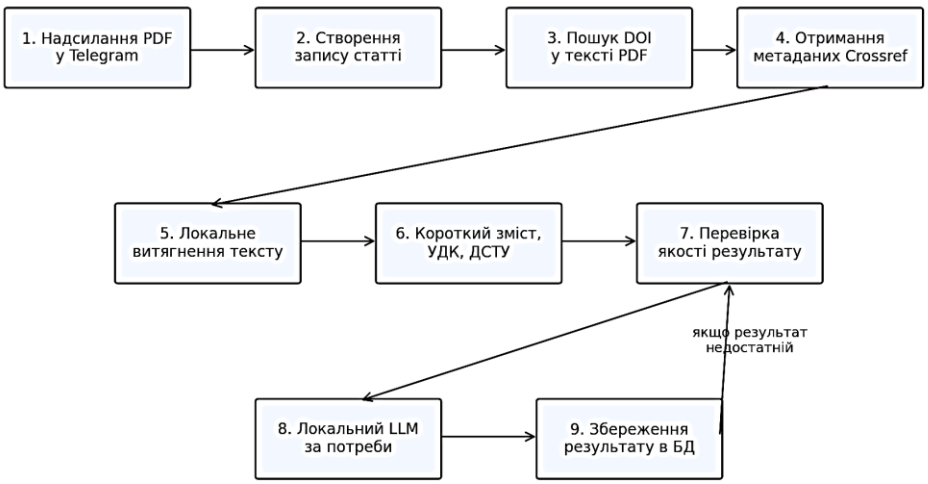


Рисунок *2 – Послідовність обробки PDF-документа в системі*

1.3 Огляд існуючих рішень

Оскільки завдання роботи пов’язане зі створенням інформаційної системи для зберігання, навігації та автоматичного анотування наукових

статей, було розглянуто існуючі рішення, які частково виконують подібні функції. До аналізу включено системи керування бібліографією, сервіси короткого опрацювання наукових текстів, інструменти аналізу PDF і програмне забезпечення для витягнення структурованих даних із наукових документів.

Zotero — система керування бібліографічними джерелами та особистою бібліотекою. Вона дає змогу зберігати джерела, організовувати колекції, працювати з PDF-файлами та формувати бібліографічні записи [12]. Перевагами Zotero є зрілість продукту, підтримка стилів цитування, інтеграція з браузером і текстовими редакторами. Водночас Zotero не орієнтований на Telegram-інтерфейс, не формує автоматично короткий зрозумілий зміст статті для користувача в чаті та не має вбудованої логіки поетапного використання DOI, локального аналізу й LLM/API-перевірки саме в межах Telegram-сценарію.

Mendeley Reference Manager — інструмент для зберігання наукових джерел і керування бібліотекою публікацій [13]. Він корисний для дослідників, які працюють із великою кількістю джерел і потребують синхронізації бібліотеки. Перевагами є підтримка PDF, бібліографічних записів і академічного робочого процесу. Недоліком у контексті цієї роботи є те, що Mendeley є готовим зовнішнім продуктом, який не надає Telegram-інтерфейсу для власної системи, не дозволяє реалізувати власну бізнес-логіку обробки, обліку вартості аналізу, журналювання та адміністративного контролю.

Scholarcy — сервіс, який допомагає опрацьовувати наукові статті та отримувати короткі виклади текстів [14]. Це рішення близьке до задачі автоматичного формування короткого змісту. Його перевагою є орієнтація на стислий виклад і роботу з академічними матеріалами. Водночас сервіс не є власною багатокористувацькою інформаційною системою з Telegram-інтерфейсом, базою даних, власними категоріями, журналами, статистикою використання та адміністративною панеллю.

ChatPDF — сервіс для взаємодії з PDF-документами через чат [15]. Він дає змогу ставити питання до документа та отримувати відповіді. Сильна сторона такого підходу полягає у швидкому діалоговому доступі до змісту. Однак для поставленої задачі цього недостатньо, оскільки система повинна не лише відповідати на питання, а й зберігати статтю у власній бібліотеці користувача, формувати структуровані сутності, виконувати пошук за DOI, авторами та категоріями, зберігати результати аналізу й вести адміністративний облік.

GROBID — програмне забезпечення з відкритим кодом для витягнення та реструктурування інформації з наукових документів [9]. Воно може бути корисним для майбутнього розширення системи, оскільки орієнтоване саме на структуру наукових текстів. Водночас GROBID є інструментом обробки документів, а не готовою користувацькою системою з Telegram-інтерфейсом, бібліотекою, категоріями, статистикою, журналами та admin-панеллю.

Порівняння розглянутих рішень з вимогами до розробленої системи наведено у таблиці 1.

Таблиця *1 – Порівняння існуючих рішень з вимогами до системи*

Рішення	**Тип рішення**	**Сильні сторони**	**Обмеження щодо задачі роботи**
Zotero	Менеджер бібліографії та PDF-бібліотеки	Колекції, бібліографічні записи, робота з PDF	Немає Telegram-інтерфейсу, власної логіки LLM-перевірки, журналів і admin-панелі
Mendeley	Менеджер наукових джерел	Синхронізація бібліотеки, зберігання джерел, академічний робочий процес	Готовий зовнішній продукт без можливості реалізації власної архітектури та обліку аналізу
Scholarcy	Сервіс короткого опрацювання статей	Орієнтація на стислий виклад академічного тексту	Не є багатокористувацькою Telegram-системою з власною бібліотекою і статистикою
ChatPDF	Чат-інтерфейс до PDF-документів	Швидка діалогова взаємодія з PDF	Не забезпечує повну бібліотеку, категорії, DOI-метадані, ДСТУ, журнали та admin-панель
GROBID	Інструмент витягнення структури з наукових PDF	Потужна обробка наукових документів, open source	Є інструментом обробки, а не готовою інформаційною системою з Telegram-доступом

Порівняльний аналіз показав, що розглянуті рішення вирішують лише окремі частини поставленої задачі. Менеджери бібліографії зручні для довгострокового зберігання джерел, але не забезпечують Telegram-доступу і власного сценарію автоматичного аналізу. Сервіси для спілкування з PDF корисні для швидких відповідей, але не формують повноцінну бібліотеку з метаданими, категоріями, УДК, ДСТУ-посиланнями та адміністративним контролем. Інструменти на кшталт GROBID можуть бути використані як додатковий модуль, але не замінюють прикладну систему.

Отже, створення власної системи є доцільним. Розробка дає змогу поєднати Telegram-інтерфейс, власну бібліотеку користувача, backend-логіку, PostgreSQL-базу даних, локальний PDF-кеш, DOI/Crossref-рівень, локальне витягнення тексту, LLM/API-перевірку, категорії, пошук, статистику та адміністративну web-панель у межах єдиного програмного продукту.

1.4 Постановка задачі

На основі аналізу предметної області та існуючих рішень сформульовано задачу розробки інформаційної системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram. Система має працювати як багатокористувацький програмний продукт із Telegram-ботом, backend-застосунком, PostgreSQL базою даних, локальним технічним сховищем PDF, service layer для аналізу статей і web-панеллю адміністратора.

Система повинна дозволяти користувачу починати роботу через команду /start, переглядати головне меню, надсилати PDF-файл у чат, отримувати повідомлення про завантаження та аналіз, відкривати власну бібліотеку, переглядати категорії, виконувати пошук і відкривати картку конкретної статті. Усі повідомлення, кнопки, статуси, помилки та системні тексти користувацького інтерфейсу мають бути українською мовою.

Для кожної статті система повинна зберігати технічні дані про файл, посилання на Telegram-повідомлення, локальний шлях технічного кешу,

розмір, контрольну суму, статус обробки, дату створення та належність до конкретного користувача. Один користувач не повинен бачити статті іншого користувача, тому всі запити до бібліотеки мають фільтруватися за власником.

Логіка аналізу статті повинна бути поетапною. Спочатку система шукає DOI у перших сторінках PDF і, якщо DOI знайдено, намагається отримати готові метадані з Crossref. Якщо метаданих достатньо, вони використовуються для заповнення картки статті без дорогого зовнішнього аналізу. Якщо DOI не знайдено або метадані неповні, виконується локальне витягнення тексту з PDF. Якщо локальний результат має недостатню якість або стаття потребує повторної перевірки, система може запустити LLM/API-аналіз.

Результат аналізу не повинен зберігатися як один загальний текст. Для кожної статті мають формуватися окремі сутності: метадані, автори, категорії, ключові слова, короткий зміст, розширений опис, орієнтовний УДК із поясненням, посилання за ДСТУ, текстовий кеш PDF, запуски аналізу та журнали подій. Такий поділ спрощує перегляд, пошук, повторне використання результатів і подальше розширення системи.

Telegram-картка статті повинна містити назву, файл, статус, DOI, авторів, категорії, ключові слова, журнал, рік, кількість сторінок і кнопки для перегляду короткого змісту, розширеного опису, УДК, ДСТУ-посилання, завантаження файлу та видалення статті. Список бібліотеки має бути зручним для перегляду останніх матеріалів, а пошук повинен працювати за назвою, DOI, іменем PDF-файлу, автором, ключовими словами та категоріями.

Адміністративна частина повинна бути реалізована як web-панель. Через неї адміністратор має переглядати користувачів, статті, статуси аналізу, помилки, категорії, журнали подій, запуски аналізу, ознаки використання зовнішнього API, орієнтовну вартість обробки та розмір локального PDF-кешу. Web-панель не повинна бути перевантаженою, однак має демонструвати, що система є повноцінною інформаційною системою з адмініструванням, а не лише Telegram-ботом.

З погляду технологічного стеку система повинна бути зручною для локального запуску на Windows 11 у VS Code. Backend реалізується мовою Python із використанням FastAPI, Telegram-бот — на aiogram, база даних — PostgreSQL, ORM — SQLAlchemy, міграції — Alembic, HTML-адмінка — через Jinja2. На першому етапі Docker не є обов'язковим, оскільки пріоритетом є зрозумілий локальний запуск, але структура проєкту повинна дозволяти подальшу підготовку до хостингу.

Отже, поставлена задача передбачає повну реалізацію прикладної інформаційної системи, яка поєднує Telegram-інтерфейс, backend, базу даних, локальну PDF-обробку, DOI/metadata layer, LLM/API-заділ, статистику, журналювання та адміністративну панель.

1.5 Висновки до розділу

1. У першому розділі проаналізовано предметну область автоматичного анотування наукових статей і організації особистої бібліотеки PDF-документів. Встановлено, що звичайне зберігання файлів у папках або повідомленнях не забезпечує повноцінного пошуку, категоризації, формування короткого змісту, УДК, бібліографічного посилання та повторного використання результатів аналізу.

2. Огляд літературних джерел і технологічних основ показав доцільність використання Python-стеку з FastAPI, aiogram, PostgreSQL, SQLAlchemy, Alembic, Jinja2 та окремими сервісними модулями для PDF, DOI, метаданих і LLM/API-аналізу. Такий підхід відповідає задачі створення зрозумілої локальної системи, яку надалі можна підготувати до хостингу.

3. Аналіз аналогів показав, що Zotero і Mendeley зручні для керування бібліографією, Scholarcy і ChatPDF допомагають працювати зі змістом документа, а GROBID може бути корисним як інструмент обробки наукових PDF. Проте жодне з розглянутих рішень не поєднує у потрібному для роботи вигляді Telegram-інтерфейс, власну бібліотеку користувача, DOI-

метадані, структуровані результати аналізу, категорії, пошук, статистику, журнали й адміністративну web-панель.

4. На підставі виконаного аналізу сформульовано задачу розробки системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram. У наступних розділах роботи доцільно перейти до опису вимог до оточення, архітектури системи, структури даних, функціональних вимог, реалізації та тестування програмного продукту.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ФОРМУВАННЯ ВИМОГ

2.1 Вимоги до апаратного забезпечення

Розроблена система призначена для локального запуску на персональному комп'ютері або ноутбуці. Основні обчислювальні операції пов'язані з роботою backend-застосунку, Telegram-бота, PostgreSQL і локальним витягненням тексту з PDF-документів. Таке навантаження не потребує спеціалізованого серверного обладнання, що відповідає навчальному та прикладному характеру роботи.

Мінімально достатнім є комп'ютер із багатоядерним процесором, 8 ГБ оперативної пам'яті та вільним дисковим простором від 2 ГБ для встановлення залежностей, бази даних, журналів і технічного кешу PDF. Для комфортної роботи з локальним LLM-шаром бажаним є 16 ГБ оперативної пам'яті та дискретний графічний адаптер, однак базова функціональність системи може працювати без нього.

Оскільки система на етапі кваліфікаційної роботи не розгортається на хостингу, вимоги до мережевої інфраструктури обмежуються наявністю стабільного доступу до Інтернету для роботи Telegram-бота, звернення до Crossref і, за потреби, завантаження моделей для локального LLM-сервера.

Вимоги до апаратного та програмного оточення системи наведено у таблиці 2.

Таблиця *2 – Вимоги до апаратного та програмного оточення*

Група вимог	**Мінімальне значення**	**Рекомендоване значення**
Процесор	2 ядра	4 і більше ядер
Оперативна пам'ять	8 ГБ	16 ГБ
Вільне місце на диску	2 ГБ	5 ГБ і більше
Операційна система	Windows 11	Windows 11
Мережа	доступ до Інтернету	стабільне підключення до Інтернету

2.2 Вимоги до програмного забезпечення

Для запуску програмної системи використовується Python 3.12. Саме ця версія забезпечує сумісність із застосованими бібліотеками FastAPI, aiogram, SQLAlchemy, Alembic, pypdf, Jinja2 та pydantic-settings. Усі залежності зібрані у файлі requirements.txt, що спрощує повторне встановлення проєкту на іншому комп'ютері.

Системою керування базами даних є PostgreSQL. Підключення до бази даних задається через файл .env, що дає змогу змінювати локальні параметри без редагування програмного коду. Міграції схеми бази даних виконуються через Alembic командою alembic upgrade head.

Для користувацької частини потрібен Telegram-акаунт і доступ до створеного бота. Для адміністративної частини потрібен web-браузер, оскільки admin-панель працює як HTML-інтерфейс у складі FastAPI-застосунку. Для повторної перевірки через LLM використовується локальний локальний LLM-сервер із HTTP API сервер, наприклад LM Studio, якщо відповідний режим увімкнено у конфігурації.

2.3 Вимоги до користувачів

Користувач системи повинен мати базові навички роботи з Telegram і PDF-документами. Для виконання основних дій не потрібно знати структуру backend-застосунку або бази даних. Взаємодія здійснюється через команди, reply keyboard та inline-кнопки.

Адміністратор системи повинен уміти запускати backend і Telegram-бота локально, перевіряти healthcheck, виконувати міграції бази даних, відкривати admin-панель і переглядати журнали подій. Такі вимоги є достатніми для експлуатації локальної версії системи в межах кваліфікаційної роботи.

2.4 Організаційні вимоги

Конфігураційні дані системи повинні зберігатися у файлі .env. До таких даних належать токен Telegram-бота, URL підключення до PostgreSQL, пароль адміністративної панелі, параметри локального LLM-сервера і шлях до каталогів storage та logs. Файл .env не повинен передаватися разом із відкритим кодом, оскільки може містити службові секрети.

PDF-файли на етапі локального запуску зберігаються як технічний кеш у каталозі storage/pdfs. При цьому Telegram-повідомлення залишається джерелом доступу до завантаженого файла. Такий підхід зменшує потребу в окремому довгостроковому файловому сховищі та відповідає вимогам керівника щодо практичного використання можливостей Telegram.

Для локального запуску важливо не тільки мати встановлені компоненти, а й забезпечити передбачуваний порядок їх увімкнення. Спочатку має бути доступна база даних PostgreSQL, потім застосовуються міграції, після цього запускається backend і лише потім Telegram-бот. Така послідовність зменшує кількість помилок, оскільки бот під час першої взаємодії одразу звертається до бази даних для створення або оновлення користувача.

Окремою вимогою є наявність стабільного доступу до мережі. Telegram-бот працює через Telegram API, а DOI/metadata layer може звертатися до Crossref. Якщо Інтернет тимчасово недоступний, локальна частина системи не втрачає дані, але завантаження нових файлів через Telegram і отримання метаданих із зовнішнього джерела не можуть виконуватися повноцінно.

Для збереження керованості середовища всі залежності повинні встановлюватися у віртуальне середовище Python. Це дозволяє уникнути конфліктів з іншими навчальними або робочими проєктами на тому самому комп’ютері. Усі залежності зафіксовано у файлі requirements.txt, тому інсталяція на іншій машині може бути повторена без ручного добору бібліотек.

Під час демонстрації системи доцільно використовувати два окремі вікна PowerShell. Перше призначене для backend-застосунку, друге — для Telegram-бота. Такий спосіб запуску спрощує контроль помилок, оскільки повідомлення backend і повідомлення бота не змішуються в одному потоці виведення.

Середовище локального LLM не є обов'язковою умовою базової працездатності, але є важливим для повної демонстрації можливостей повторної перевірки. Якщо локальний сервер вимкнений, система повинна коректно працювати з локальним аналізом і показувати користувачу сформовані результати без аварійного завершення.

Організаційно система повинна мати зрозумілу структуру каталогів. Каталоги logs і storage створюються автоматично або під час запуску, а локальний кеш PDF не повинен змішуватися з програмним кодом. Це спрощує очищення тестових файлів, перенесення проєкту і подальшу підготовку до розгортання.

Контрольні умови локального запуску системи наведено у таблиці 3.

Таблиця *3 – Контрольні умови локального запуску*

Умова	**Перевірка**	**Очікуваний стан**
PostgreSQL запущено	Підключення через DATABASE_URL	База приймає запити
Міграції застосовано	alembic upgrade head	Усі таблиці створено
Backend активний	GET /api/v1/health	status ok, database ok
Telegram-бот активний	python -m app.bot.main	Polling запущено
LM Studio за потреби	GET/POST до локального API	Модель приймає запити

Проміжний підсумок щодо вимог до оточення

У другому розділі визначено вимоги до апаратного, програмного, користувацького та організаційного оточення системи. Встановлено, що проєкт може працювати локально на комп'ютері з Windows 11, Python 3.12 і

PostgreSQL. Окремо визначено роль Telegram, локального PDF-кешу, файлу .env і локального LLM-шару повторної перевірки.

2.5 Архітектура системи

2.5.1 Загальна архітектурна модель

Архітектура системи побудована за модульним принципом. Кожен компонент виконує окрему роль: Telegram-бот забезпечує взаємодію з користувачем, backend-застосунок координує службову логіку, PostgreSQL зберігає структуровані дані, локальний PDF-кеш використовується для обробки документів, а адміністративна web-панель забезпечує контроль стану системи.

У програмному коді реалізовано поділ на рівні: рівень інтерфейсу, рівень сервісів, рівень доступу до даних і рівень зберігання. Такий поділ дозволяє не змішувати Telegram-обробники з бізнес-логікою аналізу статей. Обробники бота приймають дії користувача, а основні операції виконуються у сервісному шарі.

2.5.2 Взаємодія основних компонентів

Після надсилання PDF-файлу Telegram-бот отримує документ і перевіряє його тип. Далі файл завантажується як локальна технічна копія для аналізу, а у базі даних створюється запис статті з прив’язкою до користувача. Усі подальші запити до бібліотеки фільтруються за внутрішнім ідентифікатором користувача, який пов’язаний із Telegram user id.

Backend не використовується як окремий публічний користувацький API для основної роботи користувача. Його роль полягає у запуску FastAPI-застосунку, роботі healthcheck, підключенні admin-панелі, конфігурації і загальній організації сервісів. Основний користувацький інтерфейс залишається в Telegram.

2.5.3 Сервісний шар системи

Сервісний шар містить модулі для користувачів, статей, PDF-сховища, пошуку DOI, отримання метаданих, локального структурованого аналізу, формування УДК, бібліографічного посилання, пошуку, категорій, статистики та LLM-перевірки. Завдяки цьому кожна частина логіки має окрему відповідальність і може змінюватися без повного переписування системи.

Під час аналізу статті система спочатку виконує дешевші операції: пошук DOI, звернення до Crossref і локальне витягнення тексту. Локальний LLM-шар використовується лише тоді, коли результат локального аналізу потребує повторної перевірки або коли користувач чи адміністратор запускає такий аналіз вручну.

2.5.4 Адміністративна частина

Адміністративна web-панель реалізована у складі FastAPI-застосунку з використанням Jinja2-шаблонів. Вона не замінює Telegram-інтерфейс користувача, а виконує службову роль: показує користувачів, статті, категорії, журнали, запуски аналізу, статуси обробки та базову статистику.

Наявність admin-панелі є важливою ознакою інформаційної системи. Вона дає змогу контролювати не лише результат для конкретного користувача, а й загальний стан системи, кількість обробок, помилки та ознаки використання зовнішнього або локального LLM-шару.

В архітектурі системи принциповим є розділення каналу взаємодії та обробки даних. Telegram забезпечує зручний доступ, але не виконує роль бази знань. Основні результати аналізу зберігаються у PostgreSQL, тому користувач може повторно відкривати картку статті без повторної обробки PDF.

Використання модульної архітектури дозволяє ізолювати можливі помилки. Якщо не працює Crossref, це не повинно зупиняти Telegram-бота або admin-панель. Якщо локальний LLM-сервер повернув помилку, система зберігає локальний результат і фіксує подію у журналах. Такий підхід робить поведінку системи передбачуваною.

Сервісний шар виступає посередником між інтерфейсом і базою даних. Обробник Telegram-повідомлення не повинен напряму знати деталі створення УДК, ДСТУ або AnalysisRun. Це дає змогу перевикористовувати логіку з різних місць: наприклад, повторний аналіз можна запустити як з Telegram, так і з admin-панелі.

Архітектурно система ближча до модульного моноліту. Усі частини розміщені в одному проєкті, але відповідальність розділена за пакетами. Для дипломної роботи це практичніше, ніж повноцінна мікросервісна схема, оскільки не потребує окремих серверів, черг повідомлень і складного DevOps-рівня.

У разі майбутнього розвитку окремі частини можуть бути винесені у фонові задачі. Найбільш очевидним кандидатом є аналіз PDF і LLM-перевірка, оскільки ці операції довші за звичайний перегляд бібліотеки. Проте в локальній версії їх виконання в межах сервісного шару є достатнім і зрозумілим.

Адміністративна частина не повинна містити окрему бізнес-логіку, відмінну від Telegram-бота. Її маршрути звертаються до тих самих сервісів, що й користувацькі сценарії. Це важливо, щоб повторний аналіз у Telegram і повторний аналіз в admin-панелі давали однаковий результат і однаково фіксувалися в історії.

Сценарій повторної перевірки результату аналізу наведено на рисунку 3.

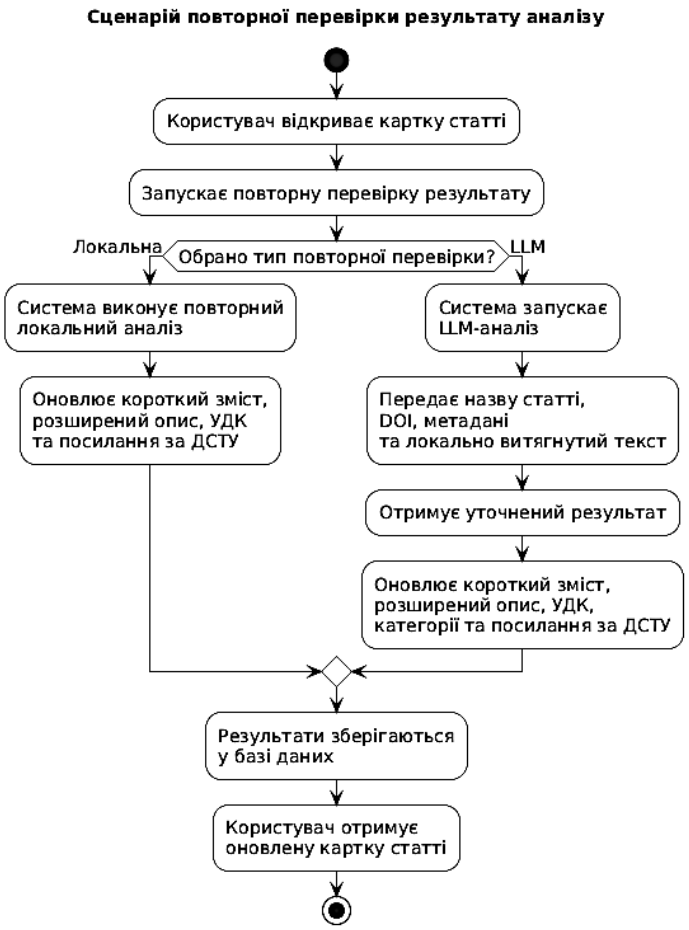


Рисунок *3 – Сценарій повторної перевірки результату аналізу*

Проміжний підсумок щодо архітектури системи

У третьому розділі описано архітектуру системи, що складається з Telegram-інтерфейсу, backend-застосунку, PostgreSQL, локального PDF-кешу, сервісного шару, модуля аналізу та адміністративної web-панелі. Така структура відповідає вимогам багатокористувацької інформаційної системи та не зводить розробку лише до простого Telegram-бота.

2.6 Специфікація даних

2.6.1 Опис форматів і структури даних

Основним вхідним форматом є PDF-документ, який користувач надсилає через Telegram. У системі зберігаються службові відомості про цей

документ: оригінальна назва файла, розмір, контрольна сума, локальний шлях технічного кешу, Telegram file id, Telegram file unique id і message id.

Структуровані дані зберігаються у PostgreSQL. До них належать користувачі, статті, метадані, автори, категорії, ключові слова, кеш витягнутого тексту, короткі описи, розширені описи, УДК, ДСТУ-посилання, запуски аналізу та журнали подій. Розділення результату на окремі сутності дозволяє не змішувати різні типи інформації в одному текстовому полі.

Узагальнену схему основних сутностей системи наведено на рисунку 4.

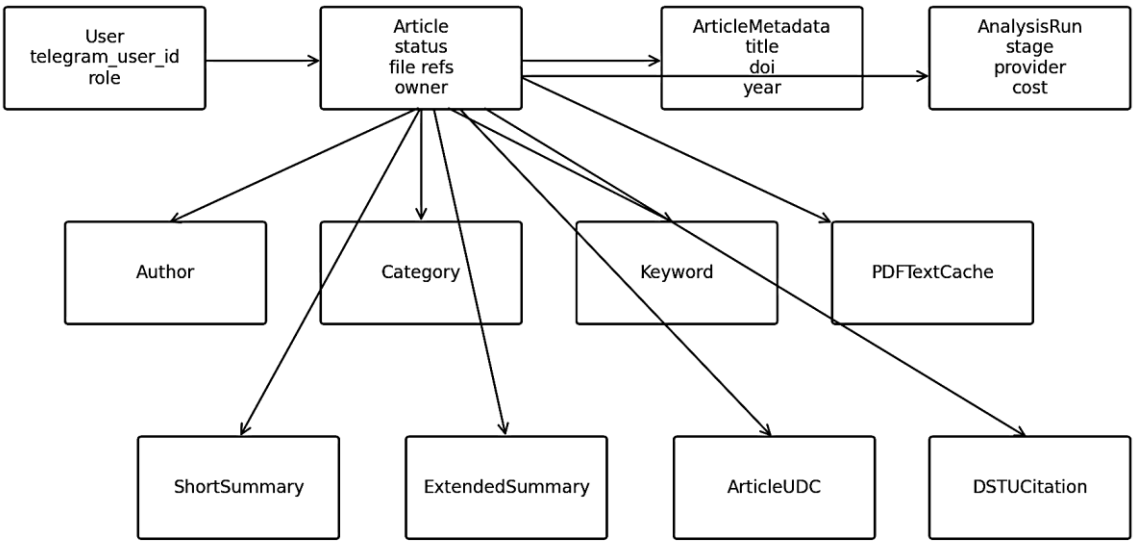


Рисунок *4 – Узагальнена схема основних сутностей системи*

2.6.2 Основні сутності системи

Основні сутності бази даних наведено у таблиці 4.

Таблиця *4 – Основні сутності бази даних*

Сутність	**Призначення**	**Ключові атрибути**
User	Зберігає дані користувача Telegram	id, telegram_user_id, username, role, is_active
Article	Описує завантажену статтю користувача	id, user_id, original_filename, status, file_path, telegram_file_id
ArticleMetadata	Зберігає формальні метадані статті	article_id, title, doi, journal, publisher, year, language
Author	Зберігає авторів наукових матеріалів	id, full_name, given_name, family_name
Category	Описує тематичну категорію статті	id, name, slug, description
Keyword	Зберігає ключові слова	id, name, normalized_name
PDFTextCache	Містить витягнутий текст PDF	article_id, extracted_text, pages_count, quality_score
ArticleSummary	Зберігає короткий зміст	article_id, text, source, confidence
ArticleExtendedSummary	Зберігає розширений опис	article_id, text, source, confidence
ArticleUDC	Зберігає орієнтовний УДК	article_id, code, rationale, source, confidence
DSTUCitation	Зберігає бібліографічне посилання	article_id, citation_text, source
AnalysisRun	Фіксує запуск аналізу	article_id, stage, status, provider, used_external_api, estimated_cost
EventLog	Фіксує події та помилки	user_id, article_id, level, action, message

2.6.3 Зв’язки між сутностями

Користувач може мати багато статей, але кожна стаття належить лише одному користувачу. Такий зв’язок забезпечує логічне розмежування бібліотек і не дозволяє користувачу бачити чужі матеріали. Стаття може мати один набір метаданих, один текстовий кеш і кілька результатів службової історії аналізу.

Зв’язок статей з авторами, категоріями та ключовими словами реалізується через проміжні таблиці. Це дає змогу одній статті мати кількох авторів і кілька категорій, а одну категорію або ключове слово

використовувати для різних статей. Запуски аналізу та журнали подій прив’язуються до конкретної статті, користувача і дати події.

2.6.4 Обмеження цілісності даних

Основним обмеженням є обов’язкова прив’язка статті до користувача. Також унікальність Telegram user id у таблиці користувачів унеможливлює створення кількох незалежних облікових записів для одного Telegram-користувача. Для статей зберігається контрольна сума файла, що дозволяє відстежувати дублювання або повторне завантаження однакових PDF-документів.

Для статусів статей, етапів аналізу та ролей користувачів застосовуються фіксовані значення. Це зменшує ризик помилок у записах і забезпечує передбачувану логіку обробки. Дані про LLM-аналіз зберігаються окремо від локального аналізу, що дає змогу відрізняти дешевшу локальну обробку від уточнення через локальний LLM-шар.

Дані системи можна умовно поділити на чотири рівні: технічний, бібліографічний, змістовий і службовий. Технічний рівень описує сам файл: назву, розмір, контрольну суму, Telegram-посилання і локальний кеш. Бібліографічний рівень містить DOI, авторів, журнал, рік, видавця і сторінки. Змістовий рівень містить короткий опис, розширений опис, УДК, категорії та ключові слова. Службовий рівень містить статуси, журнали та запуски аналізу.

Розділення цих рівнів підвищує якість даних. Наприклад, помилка у формуванні короткого опису не повинна перезаписувати DOI або авторів, отриманих із Crossref. Так само повторна LLM-перевірка може уточнити категорії і УДК, але не повинна знищувати технічні дані файла.

Зберігання історії AnalysisRun є важливим не лише для статистики, а й для пояснюваності роботи системи. Якщо користувач бачить, що стаття потребує повторної перевірки, адміністратор може відкрити історію і з’ясувати, на якому етапі виникла проблема: DOI не знайдено, локальний текст низької якості або локальний LLM повернув некоректну відповідь.

Для ключових слів і категорій застосовано окремі сутності, а не прості текстові поля у статті. Це дозволяє будувати навігацію за категоріями і виконувати пошук більш структуровано. Якщо категорія використовується багатьма статтями, вона зберігається один раз і зв’язується зі статтями через проміжну таблицю.

Сутність PDFTextCache потрібна для того, щоб не виконувати витягнення тексту повторно без потреби. Текстовий шар PDF може бути досить великим, тому збереження кешу зменшує навантаження під час повторних переглядів або повторної перевірки.

Службові статуси статті повинні використовуватися послідовно. Стаття не може одразу вважатися проаналізованою після простого завантаження файла. Спочатку вона має пройти хоча б базову обробку, після чого отримує статус, що відображає реальний стан результату.

Рівні даних статті у системі подано на рисунку 5.



Рисунок *5 – Рівні даних статті у системі*

Проміжний підсумок щодо структури даних

У четвертому розділі описано структуру даних системи, основні сутності, їхні атрибути та зв’язки. Визначено, що дані організовано навколо користувача, статті та результатів її аналізу. Такий підхід забезпечує багатокористувацький режим, пошук, категоризацію, повторне використання результатів і журналювання обробки.

2.7 Функціональні вимоги

2.7.1 Вимоги до користувацьких функцій

Функціональні вимоги сформульовано з урахуванням основних сценаріїв Telegram-інтерфейсу. Користувач повинен мати можливість

працювати із системою без окремого web-кабінету, використовуючи кнопки та повідомлення бота.

Основні функціональні вимоги до системи наведено у таблиці 5.

2.7.2 Вимоги до функцій адміністратора

Система повинна надавати адміністратору доступ до web-панелі через пароль, визначений у конфігурації. Адміністратор повинен мати можливість переглядати користувачів, статті, категорії, запуски аналізу і журнали подій. Також адміністратор повинен мати змогу відкрити детальну картку статті та вручну запустити повторний локальний аналіз або локальний LLM-аналіз.

Адміністративна частина не повинна містити складної системи ролей або платіжних механізмів на цьому етапі, оскільки метою роботи є створення працездатної основи інформаційної системи.

Таблиця *5 – Основні функціональні вимоги до системи*

Код	**Функціональна вимога**
FR-01	Система повинна реєструвати користувача за Telegram user id під час команди /start.
FR-02	Система повинна відображати головне меню користувача українською мовою.
FR-03	Система повинна дозволяти користувачу завантажувати PDF-файл через Telegram.
FR-04	Система повинна відхиляти файли, які не є PDF-документами.
FR-05	Система повинна створювати запис статті у бібліотеці користувача.
FR-06	Система повинна зберігати Telegram-посилання на файл і локальний технічний кеш.
FR-07	Система повинна шукати DOI у тексті PDF.
FR-08	Система повинна отримувати метадані через Crossref, якщо DOI знайдено.
FR-09	Система повинна виконувати локальне витягнення тексту з PDF.
FR-10	Система повинна формувати короткий зміст, розширений опис, УДК і ДСТУ-посилання.
FR-11	Система повинна запускати локальний LLM-шар повторної перевірки лише за потреби або вручну.
FR-12	Система повинна дозволяти переглядати власну бібліотеку статей.
FR-13	Система повинна дозволяти відкривати картку конкретної статті.
FR-14	Система повинна дозволяти переглядати категорії та статті всередині категорії.
FR-15	Система повинна дозволяти пошук за назвою, DOI, автором, файлом, категорією або ключовим словом.

Код	Функціональна вимога
FR-16	Система повинна дозволяти користувачу видаляти статтю зі своєї бібліотеки.
FR-17	Система повинна показувати статистику бібліотеки користувача.

2.7.3 Вимоги до обробки статті

Система повинна обробляти статтю поетапно. На першому етапі виконується пошук DOI. На другому етапі за наявності DOI виконується отримання метаданих. На третьому етапі виконується локальне витягнення тексту. На четвертому етапі формується структурований опис. На п’ятому етапі за потреби виконується локальна LLM-перевірка.

Короткий зміст і розширений опис повинні зберігатися як різні сутності. УДК повинен зберігатися разом із поясненням вибору. Бібліографічне посилання за ДСТУ повинно зберігатися окремо, щоб його можна було переглядати і використовувати незалежно від опису статті.

Функціональні вимоги системи мають бути пов’язані з реальними сценаріями користувача. Користувач не формулює SQL-запити і не працює напряму з файловою системою. Його дії зводяться до надсилання PDF, вибору кнопок, перегляду бібліотеки, пошуку і читання результатів. Тому всі складні операції повинні бути приховані за простим Telegram-інтерфейсом.

Окремою функціональною вимогою є відображення статусу аналізу. Без статусу користувач не розумів би, чи завершилася обробка, чи стаття ще в процесі, чи результат потребує уточнення. Саме тому статус є видимим і в Telegram-картці, і в admin-панелі.

Система повинна підтримувати не тільки автоматичний аналіз після завантаження, а й повторну обробку. Це важливо для практичного використання, оскільки результат може бути неповним через якість PDF, відсутність DOI або помилку локального LLM-сервера. Повторний запуск дозволяє оновити результат без повторного завантаження файла.

Функція видалення статті також є необхідною. Користувач повинен мати можливість прибрати матеріал зі своєї бібліотеки. При цьому видалення

має стосуватися саме його запису статті, а не чужих даних. Для цього кожна дія з карткою перевіряє належність статті поточному користувачу.

Для адміністратора важлива можливість бачити не лише фінальний результат, а і службову історію. Саме тому функціональні вимоги admin-панелі охоплюють AnalysisRun і SystemEventLog. Без цих даних неможливо оцінити, скільки разів запускалася обробка і де виникали помилки.

Пріоритети функціональних вимог наведено у таблиці 6.

Таблиця *6 – Пріоритети функціональних вимог*

Пріоритет	**Група вимог**	**Причина**
Високий	Завантаження PDF і створення статті	Без цього система не виконує основне призначення
Високий	Пошук DOI і метаданих	Зменшує потребу в повторній обробці
Високий	Бібліотека і картка статті	Забезпечує практичну навігацію користувача
Середній	Категорії і пошук	Підвищує зручність роботи з великою бібліотекою
Середній	LLM-перевірка	Уточнює результат у складних випадках
Службовий	Admin-панель і журнали	Потрібні для контролю і супроводу

Проміжний підсумок щодо функціональних вимог

У цьому розділі сформульовано основні функціональні вимоги до користувацької, адміністративної та аналітичної частини системи. Вимоги підтверджують, що програмний продукт повинен не лише приймати PDF-файли, а й забезпечувати повний цикл роботи з бібліотекою наукових матеріалів.

2.8 Вимоги до інтерфейсу

2.8.1 Telegram-інтерфейс користувача

Telegram-інтерфейс є основним інтерфейсом користувача. Він повинен бути простим, зрозумілим і не перевантаженим технічними деталями. Усі

кнопки, повідомлення, системні статуси та помилки мають бути українською мовою. Основна взаємодія відбувається через reply keyboard і inline keyboard.

Головне меню повинно містити кнопки «Моя бібліотека», «Завантажити статтю», «Категорії», «Пошук», «Статистика» та «Налаштування». Така структура відповідає основним сценаріям роботи користувача і дає змогу швидко перейти до потрібного розділу.

Основні сценарії взаємодії користувача з Telegram-ботом показано на рисунку 6.

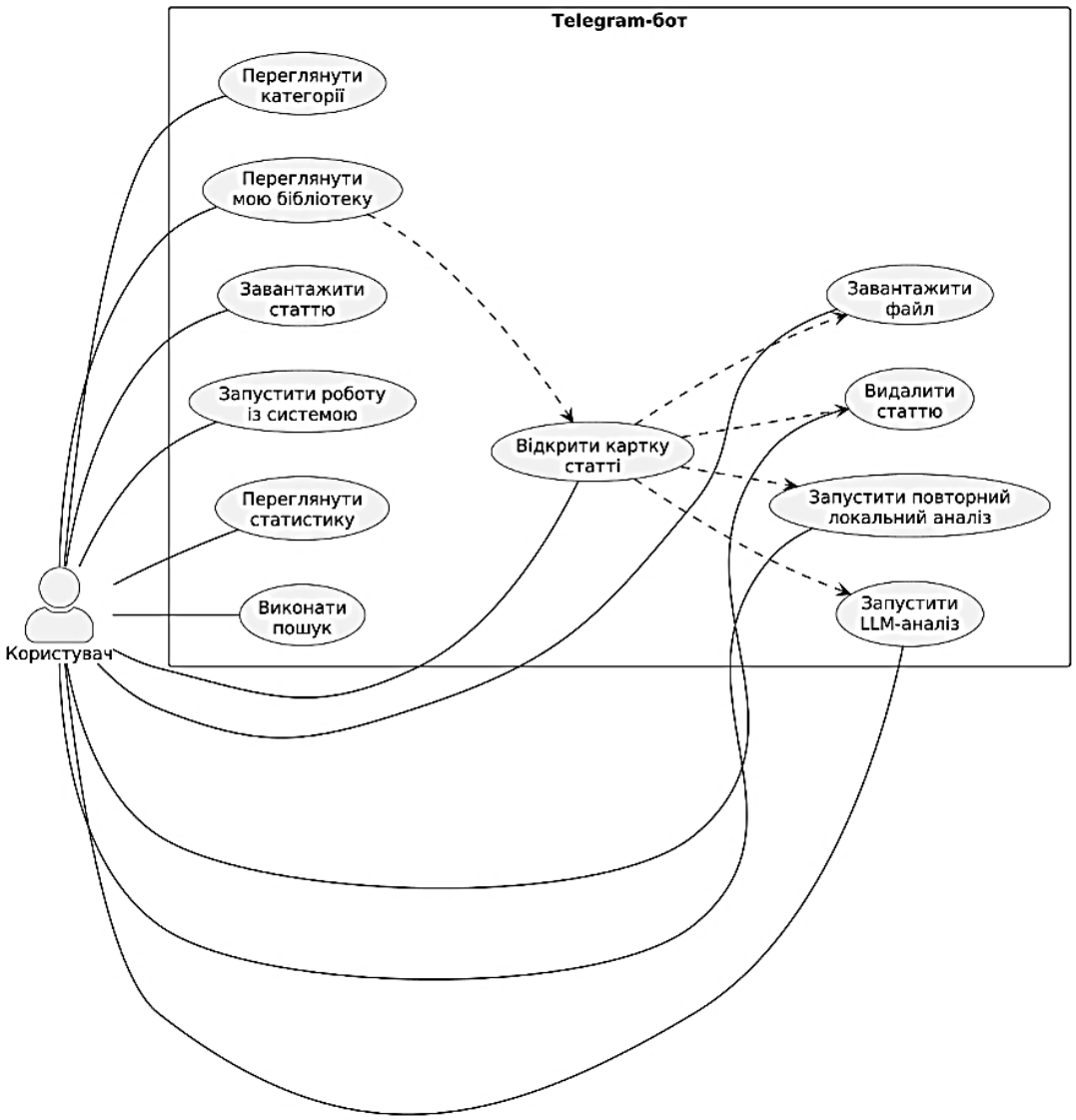


Рисунок *6 – Основні сценарії взаємодії користувача з Telegram-ботом*

2.8.2 Картка статті та навігація

Картка статті повинна відображати основні відомості про документ: назву, файл, статус, DOI, авторів, категорії, ключові слова, журнал, рік і кількість сторінок. Під карткою розміщуються inline-кнопки для перегляду короткого змісту, розширеного опису, УДК, ДСТУ-посилання, завантаження файла, повторного аналізу або видалення статті.

Для уникнення перевантаження одного повідомлення структурований результат не виводиться одразу повністю. Користувач сам обирає потрібний блок: короткий зміст, розширений опис, УДК або посилання за ДСТУ. Це робить інтерфейс зручним для роботи в месенджері.

2.8.3 Web-панель адміністратора

Інтерфейс адміністратора реалізовано як web-панель. На відміну від Telegram-інтерфейсу, admin-панель орієнтована на перегляд таблиць і службових записів. Вона містить навігацію між сторінками користувачів, статей, категорій, запусків аналізу, журналів і API-документації.

Детальна сторінка статті в admin-панелі повинна показувати службову інформацію, недоступну або зайву для звичайного користувача: шлях локального кешу, Telegram message id, контрольну суму файла, історію запусків аналізу, ознаку використання зовнішнього API і журнал подій по конкретній статті.

Користувацький інтерфейс повинен бути лаконічним, оскільки Telegram не призначений для виведення великих таблиць. Тому головна інформація подається у картці статті, а деталі відкриваються окремими повідомленнями. Такий підхід зменшує перевантаження і робить роботу з ботом зрозумілішою.

Повідомлення мають бути сформульовані так, щоб користувач розумів результат без технічних пояснень. Наприклад, повідомлення «DOI не знайдено» не повинно виглядати як помилка всієї системи. Воно має пояснювати, що стаття збережена і далі буде виконано локальний аналіз.

Inline-кнопки є доречними для дій, пов’язаних із конкретною статтею. Reply-клавіатура використовується для постійних пунктів головного меню. Таке розділення відповідає логіці Telegram: постійні розділи доступні знизу, а контекстні дії розташовані під конкретним повідомленням.

Admin-панель має інший характер інтерфейсу. Вона може містити таблиці, довгі рядки, службові ідентифікатори, шляхи до файлів, статуси та журнали. Саме тому вона реалізована як web-сторінки, а не як набір Telegram-команд. Сторінка детальної картки статті в admin-панелі. Вона демонструє всю сутність системи: файл, користувача, Telegram-дані, локальний кеш, DOI, метадані, результати аналізу, категорії, ключові слова, запуски аналізу і журнали. Склад детальної картки статті в адміністративній панелі наведено на рисунку 7.

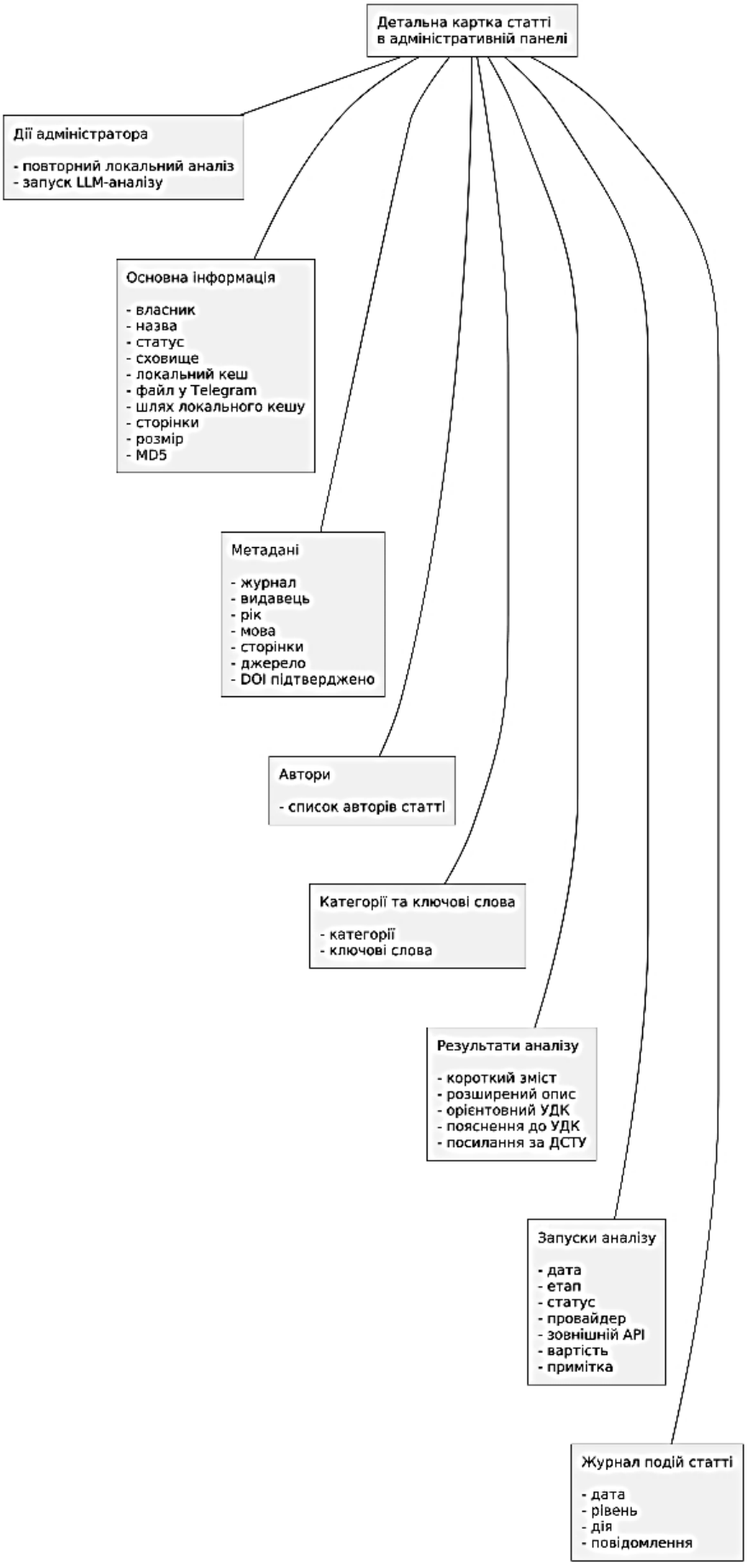


Рисунок *7 – Склад детальної картки статті в адміністративній панелі*

Проміжний підсумок щодо інтерфейсу

У шостому розділі визначено вимоги до Telegram-інтерфейсу та web-панелі адміністратора. Telegram використовується як зручний інтерфейс для користувача, а admin-панель — як службовий інструмент контролю системи, журналів, статей і результатів аналізу.

2.9 Інші вимоги

2.9.1 Вимоги до надійності

Система повинна коректно обробляти помилки завантаження файлів, відсутність DOI, недоступність Crossref, помилки локального LLM-сервера і невалідні відповіді аналізу. У таких випадках стаття не повинна зникати з бібліотеки користувача, а її статус має відображати поточний стан обробки.

У разі помилки аналізу користувач повинен отримувати зрозуміле повідомлення українською мовою. У базі даних має зберігатися відповідний запис у журналі подій, щоб адміністратор міг побачити причину збою.

2.9.2 Вимоги до продуктивності

У локальному режимі система розрахована на послідовну або помірну паралельну роботу кількох користувачів. Найдовшим етапом є LLM-перевірка, тому вона не повинна запускатися для кожної статті без потреби. Локальне витягнення тексту та DOI-пошук виконуються швидше і є основними етапами попередньої обробки.

Повторне використання збережених результатів зменшує навантаження на систему. Якщо метадані, короткий зміст або УДК уже сформовано, користувач отримує їх із бази даних без повторного виконання дорогих операцій.

2.9.3 Вимоги до безпеки і журналювання

Базове розмежування доступу забезпечується прив’язкою статей до користувача. Telegram user id використовується як стабільний зовнішній

ідентифікатор, а внутрішній user id застосовується для зв’язків у базі даних. Запити до бібліотеки повинні повертати лише матеріали поточного користувача.

Секрети системи, зокрема токен Telegram-бота і пароль admin-панелі, повинні зберігатися у файлі .env. Файл .env не повинен потрапляти до репозиторію. Журналювання має охоплювати ключові події: завантаження статті, початок аналізу, знайдений або відсутній DOI, завершення структурованого опису, запуск і завершення LLM-перевірки.

2.9.4 Вимоги до супроводжуваності

Код системи повинен бути розділений на логічні модулі. Telegram-handlers не повинні містити складної бізнес-логіки, а повинні звертатися до сервісів. Моделі бази даних мають бути згруповані за змістом: користувачі, бібліотека, аналіз. Міграції повинні виконуватися через Alembic.

Така організація спрощує подальше доопрацювання: можна замінити локальне сховище PDF на хмарне, змінити LLM-провайдера, додати фонову чергу обробки або розширити admin-панель без повного переписування всієї системи.

Надійність системи також пов’язана з тим, що окремі етапи обробки не повинні руйнувати вже отриманий результат. Якщо локальний LLM не повернув коректної відповіді, система не повинна стирати короткий опис, сформований локально. Замість цього фіксується помилка, а користувачеві повідомляється, що стаття залишилася з попередніми результатами.

Продуктивність у межах локальної системи залежить не тільки від швидкості Python-коду, а й від розміру PDF, якості текстового шару, роботи PostgreSQL і параметрів локальної моделі. Саме тому в роботі не наводяться перебільшені показники навантажувального тестування. Оцінювання виконується через реальні сценарії локального використання.

З погляду безпеки важливим є те, що Telegram user id використовується як стабільний зовнішній ідентифікатор, але в базі даних додатково

використовується внутрішній id користувача. Це дозволяє зручно будувати зв’язки між таблицями і водночас не втрачати прив’язку до Telegram-акаунта.

Журналювання подій робить систему прозорішою. У разі помилки адміністратор може побачити не лише те, що стаття має статус помилки, а і конкретну дію, на якій сталася проблема. Це важливо для майбутнього супроводу і для демонстрації серйозності реалізації.

Супроводжуваність забезпечується тим, що проєкт не перевантажений зайвими технологіями. Відсутність окремого frontend-фреймворку для admin-панелі, відмова від Docker на першому етапі та використання одного Python-стеку роблять систему зрозумілою для локального запуску і захисту.

Основні ризики локальної експлуатації та реакцію системи наведено у таблиці 7.

Таблиця *7 – Ризики локальної експлуатації та реакція системи*

Ризик	**Можливий наслідок**	**Передбачена реакція**
Недоступний Crossref	Метадані не отримано	Перехід до локального аналізу
Пошкоджений PDF	Текст не витягнуто	Статус помилки і запис у журналі
LLM API не відповідає	Повторна перевірка не виконана	Збереження локального результату
Надіслано не-PDF	Файл не може бути статтею	Повідомлення користувачу про формат
Дублі категорій	Погіршення відображення	Нормалізація і фільтрація назв

2.10 Висновки до розділу

У другому розділі визначено нефункціональні вимоги до надійності, продуктивності, безпеки, журналювання і супроводжуваності. Ці вимоги забезпечують стабільну локальну роботу системи та формують основу для подальшої підготовки до розгортання.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Засоби реалізації

Основною мовою реалізації обрано Python, оскільки вона дозволяє в одному стеку реалізувати Telegram-бота, backend, роботу з базою даних, обробку PDF і сервісний шар аналізу. Для backend використано FastAPI, який забезпечує запуск web-застосунку, healthcheck, інтеграцію admin-панелі та автоматичну OpenAPI-документацію [4].

Telegram-бот реалізовано з використанням aiogram [2]. Для зберігання даних використано PostgreSQL [6], для ORM — SQLAlchemy [5], для міграцій — Alembic [7]. Для admin-панелі застосовано Jinja2 [17]. Для роботи з PDF використано pypdf [10]. Локальний LLM-шар працює через локальний LLM-сервер із HTTP API, який може надаватися LM Studio [11].

Приклад відповідного фрагмента реалізації наведено у лістингу 1.

Лістинг 1 – Фрагмент конфігурації системи через файл .env

```
class Settings(BaseSettings):
    database_url: str
    telegram_bot_token: str
    admin_panel_password: str
    external_ai_enabled: bool = False

settings = Settings(_env_file='.env')
```

Такий підхід дозволяє змінювати параметри запуску без редагування програмного коду і не зберігати службові секрети у відкритому репозиторії.

3.2 Модулі і алгоритми

Призначення основних модулів програмної системи наведено у таблиці 8.

Таблиця *8 – Призначення основних модулів програмної системи*

Модуль	**Призначення**
app.bot.handlers.upload	Приймає PDF, створює статтю, запускає обробку.
app.bot.handlers.library	Показує бібліотеку, картку статті, результати аналізу та дії над статтею.
app.services.metadata	Шукає DOI, звертається до Crossref і зберігає метадані.
app.services.pdf	Витягує текст із PDF-документа.
app.services.analysis	Формує локальний короткий зміст, розширений опис, УДК і ДСТУ.
app.services.llm	Запускає локальний LLM-шар повторної перевірки та зберігає уточнений результат.
app.services.category_service	Формує категорії та показує статті за категоріями.
app.services.search_service	Виконує пошук у бібліотеці користувача.
app.admin	Забезпечує web-панель адміністратора.
app.models	Описує структуру таблиць бази даних.

Алгоритм завантаження статті починається з перевірки документа. Якщо файл не є PDF, користувач отримує повідомлення про помилку. Якщо файл коректний, система створює локальний технічний кеш, записує статтю в базу даних і змінює статус обробки.

Алгоритм аналізу побудований за принципом поступового уточнення. Система не переходить одразу до LLM-перевірки, а спочатку використовує DOI, готові метадані та локальне витягнення тексту. Це зменшує кількість дорогих або ресурсоємних операцій і робить результат відтворюваним.

3.2.1 Деталізація алгоритму обробки PDF-документа

Алгоритм обробки PDF-документа у розробленій системі побудовано як послідовність окремих перевірок і дій, кожна з яких виконує власне завдання та не дублює відповідальність інших модулів. Такий підхід обрано для того, щоб система не виконувала дорогий аналіз одразу після завантаження файла, а спочатку використовувала простіші та дешевші способи отримання інформації. На практиці це означає, що PDF спочатку приймається Telegram-

ботом, потім реєструється у базі даних як стаття користувача, після чого система переходить до технічної перевірки файла і пошуку DOI.

Першим кроком після збереження технічної інформації є перевірка формату. Система приймає тільки PDF-документи, оскільки інші типи файлів не відповідають предметній області кваліфікаційної роботи. Якщо користувач надсилає інший файл, бот повертає зрозуміле повідомлення українською мовою і не створює зайвих записів аналізу. У разі коректного PDF формується локальний технічний кеш, який потрібний саме для обробки документа. При цьому Telegram-повідомлення з файлом також залишається важливим джерелом доступу до початкового документа.

Після технічного приймання документа виконується пошук DOI. Для цього з перших сторінок PDF витягується текст, у якому за допомогою регулярних виразів визначається можливий DOI. Якщо DOI знайдено, система створює окремий запис про відповідний етап аналізу і переходить до отримання метаданих. Якщо DOI не знайдено, стаття не відкидається: система зберігає її у бібліотеці, фіксує подію в журналі та переходить до локального текстового аналізу. Це важливо, оскільки значна частина навчальних або локальних матеріалів може не мати DOI, але все одно повинна бути доступною для користувача.

Коли DOI доступний, система звертається до шару metadata. Цей шар ізольовано від Telegram-обробників і від моделей бази даних. Його завдання полягає в отриманні вже підготовлених відомостей про публікацію: назви, авторів, журналу, року, видавця та інших доступних полів. Якщо такі дані отримано, вони зберігаються окремо від результатів змістового аналізу. Це дозволяє не змішувати метадані з коротким змістом, УДК або бібліографічним посиланням, що відповідає загальній логіці інформаційної системи.

Наступним етапом є локальне витягнення тексту. Воно потрібне для формування короткого змісту, розширеного опису, категорій і ключових слів у тих випадках, коли метаданих недостатньо. Локальний аналіз працює без звернення до зовнішніх API і тому може використовуватися як базовий спосіб

обробки. Він не замінює повноцінного змістового аналізу складних статей, але дає змогу отримати первинний результат і не залишати статтю без опису.

Окремим етапом передбачено повторну перевірку через локальний LLM-шар. Цей механізм не запускається як обов'язковий для кожної статті. Він використовується тоді, коли локальний аналіз має недостатню якість, коли стаття отримала статус «потребує повторної перевірки», або коли адміністратор чи користувач запускає повторний аналіз вручну. Завдяки цьому система демонструє практичний підхід до керування ресурсами: прості випадки обробляються локально, а складні — уточнюються додатковим модулем.

Завершальним кроком алгоритму є оновлення картки статті. Після кожного етапу система змінює статус, додає запис до журналу подій, створює запис AnalysisRun і, за наявності результатів, оновлює метадані, категорії, ключові слова, короткий зміст, розширений опис, УДК та ДСТУ-посилання. Така послідовність робить роботу системи прозорою для адміністратора і зрозумілою для користувача Telegram-бота.

3.2.2 Логіка розмежування доступу користувачів

Багатокористувацький режим реалізовано через прив'язку кожної статті до внутрішнього користувача, створеного на основі Telegram user id. Це означає, що бібліотека, категорії, результати аналізу і статистика не є загальними для всіх користувачів. Кожен запит до бібліотеки або пошуку виконується з урахуванням власника запису. Такий підхід є простішим за складну систему ролей для звичайних користувачів, але достатнім для гарантування логічного розділення даних.

Telegram user id використовується як стабільний зовнішній ідентифікатор. Після команди /start система перевіряє, чи існує користувач у базі даних. Якщо запис відсутній, він створюється автоматично. Якщо користувач уже існує, оновлюються службові дані, зокрема ім'я користувача, username і дата останньої активності. Це спрощує початок роботи: користувачу

не потрібно проходити окрему реєстрацію, оскільки Telegram уже надає необхідний ідентифікатор.

Адміністратор має ширший доступ до даних, але тільки через web-панель. У Telegram-інтерфейсі звичайний користувач не отримує адміністративних функцій. Такий поділ дозволяє зберегти простоту бота і водночас забезпечити технічний контроль системи через браузер. У разі перенесення на хостинг цей підхід можна розширити додатковою автентифікацією, HTTPS і обмеженням доступу до адміністративної частини.

3.3 Структури даних

Структури даних проєкту відповідають сутностям предметної області. Центральною сутністю є Article, що зберігає технічну інформацію про PDF і статус обробки. До статті прив’язуються метадані, автори, категорії, ключові слова, короткий зміст, розширений опис, УДК, ДСТУ-посилання і журнали аналізу.

Для збереження історії обробки використовується AnalysisRun. Він фіксує етап обробки, статус, провайдера, ознаку використання зовнішнього API, ознаку кешу, орієнтовну вартість і дату запуску. Це дає змогу аналізувати, які операції виконувалися для конкретної статті.

3.4 Проєкт інтерфейсу

Проєкт інтерфейсу орієнтований на два типи користувачів: звичайного користувача Telegram і адміністратора. Звичайний користувач працює через кнопки бота, не бачить службових таблиць і не має доступу до чужих статей. Адміністратор працює через web-панель і бачить службову інформацію по всій системі.

У Telegram-інтерфейсі результат подається структуровано. Картка статті містить коротку службову інформацію, а детальні результати відкриваються

окремими кнопками. Це враховує обмеження месенджера і зменшує перевантаження повідомлень.

Проєктування модулів виконано так, щоб основні сценарії не залежали від конкретного інтерфейсу. Наприклад, сервіс повторного аналізу можна викликати як з Telegram, так і з admin-панелі. Це підтверджує, що реалізація не прив’язана жорстко до одного handler і може розширюватися.

Алгоритм формування ДСТУ-посилання використовує доступні метадані. Якщо є автори, назва, журнал і рік, посилання формується повніше. Якщо частина даних відсутня, сервіс формує допустимий скорочений варіант на основі назви файла або назви статті. Це краще, ніж залишати поле порожнім.

Проєкт УДК-сервісу враховує, що визначення УДК у системі є орієнтовним. Тому разом із кодом зберігається пояснення. Це важливо для академічної коректності: система не видає результат за офіційно затверджену класифікацію, а допомагає користувачу швидко зорієнтуватися у тематиці матеріалу.

Проєкт LLM-шару передбачає валідацію відповіді. Локальна модель може повернути неповний або неструктурований текст, тому результат не повинен автоматично зберігатися без перевірки. Наявність окремого клієнта і схеми відповіді зменшує ризик пошкодження даних статті.

Проєкт admin-панелі побудовано на шаблонах. Такий підхід дозволяє швидко додавати нові сторінки без складного frontend-проєкту. У майбутньому, якщо система буде розвиватися, admin-панель можна розширити фільтрами, редагуванням метаданих і експортом статистики.

Можливий напрям підготовки системи до розгортання показано на рисунку 8.

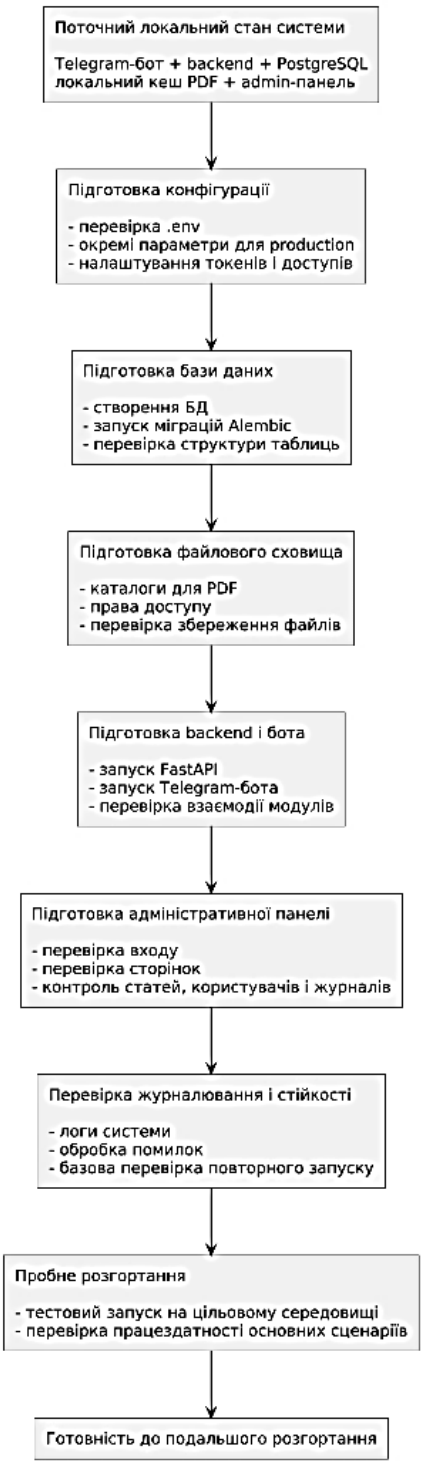


Рисунок *8 – Можливий напрям підготовки системи до розгортання*

Проміжний підсумок щодо проєкту програмної системи

У третьому розділі описано проєкт програмної системи, засоби реалізації, основні модулі, алгоритми, структури даних та інтерфейс.

Запропонований проєкт відповідає вимогам локальної багатокористувацької системи з Telegram-інтерфейсом і адміністративним контролем.

3.5 Реалізація програмних модулів

3.5.1 Реалізація backend і бази даних

Backend реалізовано як FastAPI-застосунок. Під час запуску система створює необхідні каталоги storage, storage/pdfs і logs, підключає API-маршрути, admin-панель і middleware для сесій. Для перевірки працездатності реалізовано маршрут /api/v1/health, який повертає стан backend, бази даних і локальних каталогів.

Моделі бази даних реалізовано через SQLAlchemy. Схема бази створюється міграціями Alembic. Початкова міграція формує основні таблиці системи, а наступна міграція додає поля для збереження Telegram-посилань на файл. Це дозволяє поєднати доступність файлу у Telegram із локальним технічним кешем для аналізу.

Backend-застосунок запускається з файла app/main.py. Підключення до бази даних винесено у app/core/db.py, а налаштування середовища — у app/core/config.py. Telegram-бот запускається через app/bot/main.py, а основні сценарії користувача розподілено між обробниками upload.py, library.py, categories.py і search.py.

3.5.2 Реалізація Telegram-бота

Telegram-бот реалізовано на aiogram. Під час команди /start система створює або оновлює користувача в базі даних. Далі користувачу відображається головне меню. Обробники бота поділено за призначенням: старт, меню, завантаження, бібліотека, категорії та пошук.

Під час завантаження PDF бот перевіряє тип файла, повідомляє користувачу про успішне завантаження, запускає аналіз і надсилає проміжні

повідомлення: «Аналіз триває», «DOI знайдено» або «DOI не знайдено», «Структурований опис сформовано», «LLM-аналіз завершено».

Приклад відповідного фрагмента реалізації наведено у лістингу 2.

Лістинг *2 – Спрощений фрагмент обробки PDF-файлу в Telegram-боті*

```
@router.message(F.document)
async def upload_pdf_handler(message: Message):
    if not
message.document.file_name.lower().endswith('.pdf'):
        await message.answer('Система приймає тільки PDF-
файли.')
        return

    article = await
create_article_from_telegram_file(message)
    await run_article_analysis(article.id)
```

Фрагмент показує загальну ідею обробки: спочатку перевіряється формат документа, після чого створюється запис статті та запускається аналіз.

3.5.3 Реалізація модулів аналізу

Модуль метаданих виконує пошук DOI у тексті PDF. Якщо DOI знайдено, система звертається до Crossref і зберігає отримані метадані. Якщо DOI відсутній, стаття все одно залишається у бібліотеці і переходить до локального аналізу.

Локальний структурований аналіз використовує витягнутий текст PDF для формування короткого змісту, розширеного опису, орієнтовного УДК і посилання за ДСТУ. Якщо якість результату недостатня, стаття отримує статус «потребує повторної перевірки». У такому випадку може бути запущено локальний LLM-шар, який уточнює результат.

Приклад відповідного фрагмента реалізації наведено у лістингу 3.

Лістинг *3 – Спрощений фрагмент формування посилання за ДСТУ з метаданих*

```
def build_dstu_citation(metadata):
    authors = format_authors(metadata.authors)
```

```
    title = metadata.title or metadata.file_name
    journal = metadata.journal or '[Електронний ресурс]'
    year = metadata.year or 'Б. р.'
    pages = metadata.pages or ''
    doi = f' DOI: {metadata.doi}.' if metadata.doi else ''
    return f'{authors} {title} // {journal}. — {year}. —
{pages}.{doi}'
```

У реалізації ДСТУ-посилання формується на основі доступних метаданих. Якщо частина відомостей відсутня, система створює допустимий неповний запис і залишає результат для перевірки користувачем.

3.5.4 Реалізація адміністративної панелі

Адміністративну панель реалізовано через FastAPI router і Jinja2-шаблони. Доступ до неї здійснюється через сторінку входу. Після авторизації адміністратор бачить dashboard, сторінки користувачів, статей, категорій, запусків аналізу і журналів.

На сторінці детальної картки статті адміністратор може переглянути власника, назву, DOI, статус, локальний шлях кешу, Telegram message id, сторінки PDF, контрольну суму, авторів, категорії, ключові слова, короткий зміст, розширений опис, УДК, ДСТУ-посилання, запуски аналізу та журнал подій.

3.6 Висновки до розділу

У третьому розділі описано проєктування і реалізацію програмної системи: засоби реалізації, основні модулі, алгоритм обробки PDF, логіку розмежування доступу, структури даних, інтерфейс користувача та адміністративну частину. Описані рішення підтверджують, що система побудована як модульний програмний продукт, у якому Telegram-інтерфейс, backend, база даних, сервіси аналізу і web-панель адміністратора мають окремі зони відповідальності.

4 ТЕСТУВАННЯ, РЕЗУЛЬТАТИ ТА ОЦІНКА ПРАЦЕЗДАТНОСТІ

Тестування виконувалося у локальному середовищі Windows 11. Перевірено запуск backend, Telegram-бота, healthcheck, створення бази даних, міграції, завантаження PDF, пошук DOI, локальний аналіз, LLM-перевірку, бібліотеку, категорії, пошук і admin-панель. Навантажувальне тестування та хмарне розгортання не виконувалися, оскільки вони не є предметом цієї кваліфікаційної роботи.

Результати локального тестування системи наведено у таблиці 9.

Таблиця *9 – Результати локального тестування системи*

№	Перевірка	Очікуваний результат	Фактичний результат
1	Запуск backend	FastAPI запускається без помилок	виконано
2	Healthcheck	База даних і каталоги доступні	виконано
3	Команда /start	Користувач створюється або оновлюється	виконано
4	Завантаження PDF	Створюється запис статті і PDF-кеш	виконано
5	PDF з DOI	DOI знаходиться, метадані отримуються з Crossref	виконано
6	PDF без DOI	Стаття переходить до локального аналізу	виконано
7	Локальний аналіз	Формується короткий зміст, УДК і ДСТУ	виконано
8	Локальний LLM-шар	За потреби уточнює результат аналізу	виконано
9	Бібліотека	Користувач бачить власні статті	виконано
10	Пошук	Повертаються статті за запитом	виконано
11	Admin-панель	Адміністратор бачить користувачів, статті, логи	виконано

Результати тестування показали, що система виконує основні сценарії використання: реєструє користувача, приймає PDF, створює запис статті, виконує поетапний аналіз, показує результат у Telegram, дозволяє виконувати пошук і надає адміністратору засоби контролю.

Основним обмеженням поточної реалізації є локальний режим роботи. Система підготовлена до подальшого розгортання, але у межах роботи не

переносилася на хостинг. Також LLM-шар залежить від локального сервера і доступної моделі, тому його продуктивність може відрізнятися на різних комп’ютерах.

Під час реалізації було враховано типові помилки локального запуску на Windows. Зокрема, psql може бути недоступним у PATH, тому для створення бази можна використовувати повний шлях до psql.exe. Після створення бази основним критерієм готовності backend є успішний healthcheck із відповіддю database: ok.

Реалізація Telegram-бота перевірялася через реальні натискання кнопок. Було перевірено, що кнопки головного меню працюють, користувач створюється в базі, бібліотека відображає тільки його статті, а статистика показує кількість матеріалів і запусків аналізу.

Під час перевірки PDF без DOI система коректно повідомила, що DOI не знайдено, але не зупинила обробку. Це важливо, оскільки багато навчальних або технічних PDF не мають DOI, але все одно повинні залишатися доступними у бібліотеці і проходити локальний аналіз.

Під час перевірки PDF з DOI система знайшла ідентифікатор, звернулася до Crossref і відобразила метадані. Цей сценарій підтверджує, що логіка «спочатку DOI і готові метадані» реалізована практично, а не лише описана в вимогах.

Локальна LLM-перевірка спочатку виявила проблему перевищення контексту. Після скорочення вхідного тексту запит став коректним, а система змогла отримати структурований результат. Цей випадок показав, що для локальних моделей важливо не передавати весь PDF без обмеження.

Admin-панель перевірялася через сторінки dashboard, users, articles, categories, analysis-runs і logs. Особливо важливо, що детальна картка статті показує як результат для користувача, так і службову історію. Це дозволяє підтвердити, що система веде облік обробки і може пояснити свої дії.

Тестування не включало штучне навантаження сотнями користувачів, оскільки така перевірка не відповідала локальному характеру роботи.

Натомість були перевірені повні сценарії, які демонструють основну цінність системи: завантаження, аналіз, повторна перевірка, бібліотека, пошук і адміністрування.

Під час локального тестування використовувалися два PDF-файли: стаття з DOI та документ без DOI. У першому випадку система знаходила DOI і отримувала метадані через Crossref, у другому — виконувала локальне витягнення тексту і позначала результат як такий, що потребує повторної перевірки.

Перевірені сторінки адміністративної панелі наведено у таблиці 10.

Таблиця *10 – Перевірені сторінки адміністративної панелі*

Сторінка	**Що перевірялося**	**Результат**
Dashboard	Зведена статистика	Дані відображаються
Users	Список користувачів Telegram	Користувачі доступні для перегляду
Articles	Список статей і статусів	Статті відображаються
Article detail	Результати аналізу і повторний запуск	Картка відкривається
Categories	Категорії і кількість статей	Дані відображаються
Analysis runs	Історія запусків	Локальні і LLM-запуски видимі
Logs	Журнал подій	Події сортуються і переглядаються

4.1 Методика локального тестування

Тестування програмної системи виконувалося у локальному середовищі Windows 11 з використанням ручної перевірки основних сценаріїв. Такий підхід є доцільним для кваліфікаційної роботи, оскільки система має кілька взаємопов’язаних інтерфейсів: Telegram-бот, backend, база даних, локальний PDF-кеш і web-панель адміністратора. Автоматичні модульні тести можуть бути додані на наступних етапах розвитку, однак для поточної версії основним завданням було підтвердити працездатність повного користувацького циклу.

Перевірка здійснювалася за методом «чорної скриньки», коли оцінюється поведінка системи з позиції користувача, а не внутрішня реалізація окремих функцій. Додатково застосовувалася часткова перевірка за методом «білої скриньки» для контролю записів у базі даних, журналів подій, статусів статей і запусків аналізу. Поєднання цих двох підходів дозволило перевірити не лише зовнішню роботу інтерфейсу, а й коректність збереження результатів обробки.

Перший блок тестування стосувався запуску інфраструктури. Перевірялося створення віртуального середовища Python, встановлення залежностей, підключення до PostgreSQL, застосування міграцій Alembic і запуск FastAPI-застосунку. Після запуску backend відкривалася службова адреса healthcheck. Коректною вважалася відповідь, у якій стан сервісу, бази даних і локальних каталогів позначено як успішний.

Другий блок тестування стосувався Telegram-бота. Перевірялася команда /start, повторний запуск бота для вже зареєстрованого користувача, робота головного меню, обробка кнопок «Моя бібліотека», «Завантажити статтю», «Категорії», «Пошук», «Статистика» і «Налаштування». Окремо перевірялося, що бот не завершує роботу з помилкою після натискання inline-кнопок і коректно обробляє ситуації, коли повідомлення не потребує оновлення.

Третій блок тестування стосувався завантаження PDF. Для перевірки використовувалися два типи документів: PDF із DOI та PDF без DOI. У першому випадку система повинна була знайти DOI, отримати метадані та сформувати картку статті з назвою, авторами, журналом і роком, якщо такі дані доступні. У другому випадку система повинна була не зупиняти обробку, а перейти до локального витягнення тексту і сформувати базовий структурований опис.

Четвертий блок тестування стосувався локального LLM-шару повторної перевірки. Перевірялося, що LLM-аналіз запускається не для кожної статті автоматично, а лише у випадках, коли система вважає результат локального

аналізу недостатнім або коли повторну перевірку запускає користувач чи адміністратор. Також перевірялося, що після успішного LLM-аналізу статус статті змінюється на «проаналізовано», а категорії, ключові слова, короткий зміст, розширений опис, УДК і ДСТУ-посилання оновлюються в базі даних.

П’ятий блок тестування стосувався адміністративної web-панелі. Перевірявся вхід адміністратора, dashboard, сторінка користувачів, список статей, детальна картка статті, категорії, запуски аналізу і журнали. Особлива увага приділялася тому, щоб у web-панелі були видимі не лише статті, а й службові дані: статуси, помилки, ознака використання локального LLM, орієнтовна вартість, Telegram message id, шлях локального кешу і записи журналу.

4.2 Аналіз результатів тестування

Результати локального тестування підтвердили, що система виконує основні функції, визначені у вимогах. Backend запускається локально, healthcheck повертає коректний стан, база даних приймає міграції, Telegram-бот реєструє користувача і працює з його власною бібліотекою. Завантажені PDF-документи створюють записи статей, а результати аналізу зберігаються у відповідних таблицях.

Під час перевірки було підтверджено, що користувач бачить тільки власні статті. Це є важливою умовою багатокористувацької системи, оскільки бібліотека не повинна бути спільною для всіх Telegram-користувачів. Пошук, категорії, статистика і картка статті також працюють у межах конкретного власника. Адміністратор, на відміну від звичайного користувача, може переглядати загальну інформацію через web-панель.

Окремо перевірено роботу з документами без DOI. Такі файли не відкидаються і не залишаються без результату. Система формує локальний структурований опис, визначає попередні категорії, створює орієнтовний УДК і ДСТУ-посилання. Якщо якість результату недостатня, стаття отримує статус

«потребує повторної перевірки», що дозволяє запускати локальний LLM-шар. Саме така поведінка відповідає практичній логіці системи, оскільки не всі корисні матеріали мають повні академічні метадані.

Виявлені під час розробки проблеми були пов’язані переважно з асинхронною роботою SQLAlchemy, оновленням Telegram-повідомлень без зміни тексту, розміром контексту локальної LLM-моделі та повторним додаванням категорій. Ці помилки було враховано у фінальній версії: звернення до пов’язаних даних виконуються коректно, TelegramBadRequest при однаковому повідомленні обробляється без аварійного завершення, вхідний текст для LLM обмежується, а категорії нормалізуються перед збереженням.

Система не проходила промислове навантажувальне тестування і не розгорталася на публічному хостингу, тому в роботі не наводяться штучні показники серверного навантаження. Водночас локальна перевірка підтвердила достатню швидкодію для демонстраційного та навчального сценарію: Telegram-бот відповідає на команди, backend стабільно обробляє службові маршрути, а аналіз одного PDF-документа виконується у прийнятний для локального використання час. Подальша оцінка продуктивності доцільна після перенесення системи на серверне середовище.

4.3 Висновки до розділу

У четвертому розділі описано реалізацію backend, Telegram-бота, модулів аналізу, адміністративної панелі та локальне тестування. Перевірка підтвердила працездатність основних сценаріїв і відповідність розробленої системи поставленій задачі.

ВИСНОВКИ

1. У кваліфікаційній роботі розроблено інформаційну систему автоматичного анотування наукових статей з інтеграцією у месенджер Telegram. Система призначена для роботи з особистою бібліотекою PDF-матеріалів і формує структурований результат аналізу.

2. Проаналізовано предметну область, розглянуто існуючі рішення, визначено їхні обмеження та сформульовано вимоги до системи. Обґрунтовано використання Python-стеку з FastAPI, aiogram, PostgreSQL, SQLAlchemy, Alembic, Jinja2, pypdf і локальним LLM-шаром.

3. Розроблено архітектуру системи, що містить Telegram-інтерфейс, backend, базу даних, локальний PDF-кеш, сервісний шар, модулі DOI/метаданих, локального аналізу, LLM-перевірки, статистики, журналювання та адміністративну web-панель. Усі сутності користувача прив’язано до Telegram user id, що забезпечує розмежування бібліотек.

4. Реалізовано поетапну логіку обробки статті: пошук DOI, отримання метаданих через Crossref, локальне витягнення тексту, формування короткого змісту, розширеного опису, орієнтовного УДК із поясненням і посилання за ДСТУ. Локальний LLM-шар використовується як засіб повторної перевірки або уточнення результату.

5. Telegram-частина дає змогу користувачу завантажувати PDF, переглядати власну бібліотеку, відкривати картку статті, працювати з категоріями, пошуком, статистикою, початковим файлом і повторним аналізом. Адміністративна панель забезпечує перегляд користувачів, статей, категорій, запусків аналізу, журналів і базової статистики використання.

6. Локальне тестування підтвердило працездатність основних сценаріїв. Система запускається на Windows 11, працює з PostgreSQL, обробляє PDF-документи та надає доступ до результатів через Telegram і web-панель. Розгортання на хостингу не виконувалося, але проєкт підготовлено до подальшого перенесення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API. Telegram. URL: https://core.telegram.org/bots/api (дата звернення: 07.04.2026).

2. Aiogram Documentation. Aiogram. URL: https://docs.aiogram.dev/ (дата звернення: 07.04.2026).

3. Python Documentation. Python Software Foundation. URL: https://docs.python.org/3/ (дата звернення: 07.04.2026).

4. FastAPI Documentation. FastAPI. URL: https://fastapi.tiangolo.com/ (дата звернення: 01.05.2026).

5. SQLAlchemy Documentation. SQLAlchemy. URL: https://docs.sqlalchemy.org/ (дата звернення: 01.05.2026).

6. PostgreSQL Documentation. PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/ (дата звернення: 01.05.2026).

7. Alembic Documentation. SQLAlchemy. URL: https://alembic.sqlalchemy.org/ (дата звернення: 01.05.2026).

8. Crossref. Metadata Retrieval via REST API. URL: https://www.crossref.org/documentation/retrieve-metadata/rest-api/ (дата звернення: 07.04.2026).

9. GROBID. Machine learning software for extracting and restructuring scholarly documents. GitHub. URL: https://github.com/grobidOrg/grobid (дата звернення: 07.04.2026).

10. pypdf Documentation. pypdf. URL: https://pypdf.readthedocs.io/ (дата звернення: 01.05.2026).

11. LM Studio Local Server Documentation. LM Studio. URL: https://lmstudio.ai/docs/developer/core/server (дата звернення: 01.05.2026).

12. Zotero Documentation. Zotero. URL: https://www.zotero.org/support/ (дата звернення: 01.05.2026).

13. Mendeley Reference Manager. Elsevier. URL: https://www.mendeley.com/reference-management/reference-manager (дата звернення: 01.05.2026).

14. Scholarcy. AI-powered article summarizer. URL: https://www.scholarcy.com/ (дата звернення: 01.05.2026).

15. ChatPDF. Chat with any PDF. URL: https://www.chatpdf.com/ (дата звернення: 01.05.2026).

16. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 16 с.

17. Jinja Documentation. Pallets. URL: https://jinja.palletsprojects.com/ (дата звернення: 25.04.2026).

18. Кобіков І. Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram. Збірник наукових праць студентів, здобувачів PhD, докторантів і молодих вчених, університетська конференція «МОЛОДА НАУКА-2026» Том III «Секції Інженерного навчально-наукового інституту ім. Ю.М. Потебні Запорізького національного університету» : м. Запоріжжя, 20–24 квітня 2026 р. Запоріжжя, 2026. С. 97–98.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Кобіков Ігор Сергійович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми «Програмне забезпечення систем», адреса електронної пошти ihorkobikov5@gmail.com, — підтверджую, що написана мною кваліфікаційна робота на тему «Розробка системи автоматичного анотування наукових статей з інтеграцією у месенджер Telegram» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 06.06.2026 _____ Кобіков Ігор Сергійович
(студент)

Дата 06.06.2026 _____ Жуков Олександр Олексійович
(керівник)