МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ЛІНГВІСТИЧНОГО АНАЛІЗУ ТЕКСТУ

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувачка _____ **Софія КРИВА**

«__» _____ 2026 р.

Керівник роботи
канд. техн. наук,
доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«_____» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувачки

Криви Софії

1. Тема кваліфікаційної роботи «Вебзастосунок лінгвістичного аналізу тексту» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від « 26 » грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «____» _____ 2026 р.
3. Очікуваним результатом є створений вебзастосунок лінгвістичного аналізу тексту, що інтегрує модулі морфологічного та синтаксичного аналізу, автоматизує підготовку корпусів і забезпечує графічне відображення результатів.

4. Перелік питань, що підлягають розробці:

– провести аналіз предметної області та огляд існуючих програмних аналогів і сервісів для лінгвістичного аналізу тексту, визначити їхні переваги та недоліки;

– визначити функціональні та нефункціональні вимоги до системи, розробити сценарії використання для різних суб’єктів;

– спроєктувати архітектуру системи (обґрунтувати вибір клієнтської та серверної частин, підходи до масштабування, зберігання даних і безпеки);

– визначити набір сервісів для обробки тексту (токенізація, лематизація, POS-тегінг, NER, синтаксичний парсинг, сентимент-аналіз, класифікація);

– розробити структуру бази даних для зберігання інформації (корпусів, метаданих, анотацій) і механізм імпорту/експорту даних (CSV, JSON формати);

– реалізувати серверну частину з використанням сучасних технологій (REST API, інтеграція з NLP-модулями, контейнеризація);

– реалізувати клієнтську частину у вигляді інтуїтивно зрозумілого вебінтерфейсу з інструментами візуалізації результатів (графи залежностей, виділення сутностей, хмари слів);

– провести тестування розробленої платформи та оцінити точність і продуктивність реалізованих NLP-модулів на прикладних наборах даних.

5. Презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «25» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: **«Вебзастосунок лінгвістичного аналізу тексту»**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	15.12.2025	25.12.2025	*Виконано*
2.	Огляд літератури за темою роботи	26.12.2025	16.01.2026	*Виконано*
3.	Складання календарного плану КБР	02.02.2026	03.02.2026	*Виконано*
4.	Складання вимог: визначення ролей користувачів та аналіз існуючих платформ для обробки текстових даних	04.02.2026	11.02.2026	*Виконано*
5.	Розробка проєктних рішень та вибір стеку технологій	12.02.2026	09.03.2026	*Виконано*
6.	Моделювання та конструювання ПЗ	10.03.2026	26.03.2026	*Виконано*
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування	27.03.2026	15.05.2026	*Виконано*
8.	Оформлення КБР та презентації	18.05.2026	22.05.2026	*Виконано*
9.	Відгук керівника КБР	25.05.2026	26.05.2026	*Виконано*
10.	Попередній захист	27.05.2026	27.05.2026	*Виконано*
11.	Завершення оформлення КБР та презентації	28.05.2026	09.06.2026	*Виконано*
12.	Рецензування			
13.	Захист кваліфікаційної роботи			

Здобувачка _____ **Софія КРИВА**

«___» _____ 2026 р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«___» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Вебзастосунок лінгвістичного аналізу тексту»

Здобувачка 409 гр.: Крива Софія

Керівник: канд. техн. наук, доцент Давиденко Євген

Актуальність роботи полягає у потребі прискорення обробки великих масивів текстів, а також підвищенні якості лексико-граматичної та семантичної складових наряду зі зменшенням людського фактора в рутинних завданнях.

Метою є проєктування та розробка вебзастосунку лінгвістичного аналізу тексту для автоматизації опрацювання текстових даних у навчальних, дослідницьких та прикладних задачах.

Об’єктом кваліфікаційної роботи є процеси автоматичної лінгвістичної обробки текстів у вебсередовищі.

Предметом роботи є методи аналізу текстових даних та інтерпретації природної мови (модулі морфології та синтаксису).

Кваліфікаційна бакалаврська робота складається зі вступу, 4 розділів, висновків, переліку джерел посилання і додатків.

У вступі обґрунтовано актуальність обраної теми, визначено мету і проведено огляд поставлених завдань, зазначено предмет та об’єкт роботи, а також розкрито практичне значення створюваного вебзастосунку.

У першому розділі описано аналітичну частину, а саме огляд існуючих застосунків-аналогів для роботи із текстовою інформацією, визначено їх функціонал, переваги та недоліки; обґрунтовано план виконання завдання.

Другий розділ присвячено опису процесу розробки проєктних рішень, що забезпечують виконання пунктів специфікації вимог, тобто архітектури системи, а також функціональних та інформаційних моделей програмного забезпечення (сценарії використання). Крім того, описано обраний стек технологій (мови програмування та компоненти застосунку, тобто бібліотеки, фреймворки, плагіни тощо). Під час процесу розробки складено детальний алгоритм вирішення поставленої задачі. Наступною частиною розділу є

формування та опис специфікації вимог до програмного забезпечення, що розробляється.

У третьому розділі описано обсяг виконаної роботи з конструювання та моделювання програмного забезпечення, а саме розробку UML-діаграм та взаємодію між компонентами бази даних, в тому числі й створення сценаріїв використання системи. Також тут надано мокапи оформлення.

У четвертому розділі продемонстровано кодування і тестування інтерфейсу користувача та системи, оцінку отриманих результатів і налаштування злагодженої роботи модулів аналітики та компонентів шляхом контейнеризації.

У висновках проведено аналіз й узагальнено результати виконаної роботи, підтверджено виконання поставлених завдань та визначено шляхи подальшого розширення і вдосконалення функціоналу створеного вебзастосунку.

Кваліфікаційна робота пройшла апробацію на конференції «Могилянські читання – 2025». Тези було подано у секції «Філологія» у підсекції «Іншомовна підготовка студентів у контексті оновлення змісту освіти».

КБР викладена на 103 сторінки, вона містить 4 розділи, 63 ілюстрації, 18 таблиць, 32 джерела в переліку посилань і 2 додатки.

Ключові слова: *алгоритми обробки тексту, вебзастосунок, лематизація, лінгвістичний аналіз тексту, мовні моделі, обробка природної мови, текстовий корпус, токенізація, частиномовне тегування, REST API.*

ABSTRACT

of the Bachelor's Thesis

"Web application for linguistic text analysis"

Student of group 409: Sofiia Kryva

Supervisor: Candidate of Technical Sciences (Ph. D.), Associate Professor

Davydenko Yevhen

The relevance of this work lies in the need to accelerate the processing of large text corpora, as well as to improve the quality of lexical, grammatical, and semantic components while reducing the human factor in routine tasks.

The objective is to design and develop a web-based linguistic text analysis application to automate the processing of text data for educational, research, and applied purposes.

The object of this thesis is the processes of automatic linguistic text processing in a web environment.

The subject of the work is methods of text data analysis and natural language interpretation (morphology and syntax modules).

The bachelor's thesis consists of an introduction, 4 chapters, conclusions and a list of references.

The introduction justifies the relevance of the chosen topic, defines the objective and provides an overview of the tasks set, specifies the subject and the object of the work, and highlights the practical significance of the web application being developed.

The first chapter describes the analytical part, specifically an overview of existing similar applications for working with text information, identifying their functionality, advantages and disadvantages; it also justifies the plan for carrying out the task.

The second chapter is dedicated to description of the process of developing design solutions that ensure compliance with the requirements specification, namely the system architecture, as well as the functional and information models of the software (use cases). In addition, the chosen technology stack (programming languages and application components, i.e. libraries, frameworks, plugins, etc.) is described. During the development process, a detailed algorithm for solving the task at hand was drawn up. The

next part of the chapter involves the formulation and description of the requirements specification for the software being developed.

The third chapter describes the scope of work carried out on the design and modelling of the software, namely the development of UML diagrams and the interaction between database components, including the creation of system use cases in text form. Design mock-ups are also provided here.

The fourth chapter demonstrates the coding and testing of the user interface and the system, the evaluation of the results obtained, and the configuration of the coordinated operation of analytic modules and components through containerisation.

The conclusions analyse and summarise the results of the work accomplished, confirm the completion of the set tasks, and identify ways to further expand and improve the functionality of the created web application.

The thesis was presented at the ‘Mohyla Readings – 2025’ conference. The abstract was submitted to the ‘Philology’ section, under the sub-section ‘Foreign Language Learning in the Context of Educational Content Renewal’.

The bachelor’s thesis is presented on 103 pages, containing 4 chapters, 63 illustrations, 18 tables, 32 sources in the list of references and 2 appendices.

Keywords: *language models, lemmatisation, linguistic text analysis, NLP, POS-tagging, REST API*, *text corpus, text processing algorithms, tokenisation, web application.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

СКБД – система керування базами даних

ШІ – штучний інтелект

AI – Artificial Intelligence

API – Application Programming Interface (інтерфейс програмування застосунків)

CI/CD – Continuous Integration/Continuous Delivery (безперервна інтеграція та безперервне розгортання)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

CSRF – Cross-Site Request Forgery (підробка міжсайтових запитів)

CSV – Comma-Separated Values (текстовий формат подання табличних даних, де значення розділені комами)

GDPR – General Data Protection Regulation (Загальний регламент захисту даних ЄС)

JSONB – JavaScript Object Notation Binary (бінарний формат у PostgreSQL)

JWT – JSON Web Token (формат токена для автентифікації та авторизації)

NER – Named Entity Recognition (розпізнавання іменованих сутностей)

NLP – Natural Language Processing (обробка природної мови)

PMI – Pointwise Mutual Information (показник взаємної інформації)

POS – Part-of-Speech (частина мови)

PII – Personally Identifiable Information (інформація, за якою можна ідентифікувати особу)

TCP/IP – Transmission Control Protocol/Internet Protocol (протокол управління передачею/інтернет-протокол)

TLS – Transport Layer Security (протокол захисту транспортного рівня)

UI – User Interface (інтерфейс користувача)

XSS – Cross-Site Scripting (міжсайтове виконання шкідливих сценаріїв)

ВСТУП

Сьогодні обробка природної мови та автоматичний аналіз текстів стають необхідністю в багатьох галузях – від освіти й журналістики до бізнес-аналітики та систем підтримки прийняття рішень. Зростання обсягів текстової інформації в цифровому середовищі вимагає ефективних інструментів для автоматизованого вилучення змісту, виявлення тональності, класифікації, розпізнавання іменованих сутностей та синтаксичного аналізу. Існуючі рішення часто або орієнтовані на вузькі задачі, або потребують глибоких технічних навичок для налаштування, або ж комерційно дорогі. Розробка зручного вебзастосунку, доступного широкому колу користувачів (викладачам, дослідникам, журналістам, контент-менеджерам), має високий науково-практичний сенс.

Сучасні досягнення в області глибокого навчання (наприклад, трансформерні архітектури), а також наявність відкритих і комерційних сервісів для NLP дають змогу створювати потужні інструменти аналізу. Проте практична інтеграція передових моделей у зручний, безпечний і масштабований вебінтерфейс лишається нетривіальним інженерним завданням. У цьому контексті провідні розробники й дослідницькі центри, такі як OpenAI та Google, а також академічні осередки (наприклад, Stanford University) демонструють швидкий прогрес у створенні базових моделей і методів.

Актуальність обраної теми полягає у потребі прискорення обробки великих масивів текстів, а також підвищенні якості лексико-граматичної та семантичної складових наряду зі зменшенням людського фактора в рутинних завданнях.

Метою роботи є проєктування та розробка вебзастосунку лінгвістичного аналізу тексту для автоматизації опрацювання текстових даних у навчальних, дослідницьких та прикладних задачах.

Для досягнення мети необхідно вирішити наступні **завдання**:

– провести аналіз предметної області та існуючих програмних аналогів і сервісів для лінгвістичного аналізу тексту, визначити їхні переваги та недоліки;

– визначити функціональні та нефункціональні вимоги до системи, розробити сценарії використання для різних суб’єктів;

– спроєктувати архітектуру системи (обґрунтувати вибір клієнтської та серверної частин, підходи до масштабування, зберігання даних і безпеки);

– визначити набір сервісів для покрокової обробки тексту (токенізація, лематизація, POS-тегінг, NER, синтаксичний парсинг, класифікація);

– розробити структуру бази даних для зберігання інформації (корпусів, метаданих), механізм імпорту й експорту даних (CSV, JSON формати);

– реалізувати серверну частину з використанням сучасних технологій (REST API, інтеграція з NLP-модулями, контейнеризація);

– реалізувати клієнтську частину у вигляді інтуїтивно зрозумілого вебінтерфейсу з інструментами візуалізації (графи залежностей, виділення сутностей, хмари слів);

– провести тестування розробленої платформи та оцінити точність і продуктивність реалізованих NLP-модулів на прикладних наборах даних.

Об’єктом кваліфікаційної роботи є процеси автоматичної лінгвістичної обробки текстів у вебсередовищі.

Предметом роботи є методи аналізу текстових даних та інтерпретації природної мови (модулі морфології та синтаксису).

Практичне значення полягає у створенні вебзастосунку, який полегшить підготовку лінгвістичних корпусів та графічне представлення результатів; надасть можливість внесення користувачами змін до локальної системи правил (за необхідності); підвищить ефективність роботи редакцій, служби підтримки та дослідницьких груп, що працюють із джерелами інформації.

В існуючих вебзастосунках присутні наступні недоліки: обмежена підтримка мов (українська – неповноцінно) та надмірні рекомендації, які змінюють авторський стиль. Ще помітна некоректна робота з великими текстами і документами. Також значний негативний вплив має повільна обробка, а деякі модулі або візуалізації можуть здаватися занадто простими для певних аналітичних потреб. Окремим аспектом, вартим уваги, є те, що безкоштовна версія має ліміти на кількість перевірок.

Розроблена система дозволить полегшити роботу із текстом завдяки наявності редактора та функції перевірки правопису.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз та характеристика предметної області

Предметна область даної роботи охоплює процеси автоматизованого аналізу текстових даних, їх обробки та виявлення помилок. Не менш важливою складовою постає і візуальне представлення результатів. Текст є одним із головних носіїв інформації в таких сферах, як освіта, наука, медіа, професійне спілкування та редакторська діяльність. Щоденно створюється значна кількість цифрових текстових ресурсів, які потребують звичайного редагування, а іноді й більш глибокого опрацювання з точки зору лінгвістики [1].

Із позиції системного аналізу, окрім послідовності символів, текст можна розглядати як багаторівневий об’єкт обробки. На базовому рівні він містить графічні одиниці та пунктуацію, на наступному включає слова й речення. Далі вже з’являються морфологічні, синтаксичні та семантичні характеристики. Така багатошаровість тексту зумовлює складність його автоматизованого аналізу, а якісний результат вимагає виконання чіткої визначеної послідовності кроків [2].

Актуальність теми пояснюється тим, що у більшості наявних цифрових рішень ці завдання розв’язуються фрагментарно. Якщо умовно розділити системи на класи, то:

- один із них орієнтований переважно на перевірку граматики та правопису;
- інший спрямований саме на корпусний аналіз;
- деякі роблять акцент на стилістичному редагуванні та візуалізації даних.

Внаслідок такого розподілу задач користувач змушений переносити тексти між різними сервісами та повторно виконувати одні й ті самі дії. Розрізненість інструментів створює додаткове навантаження і знижує ефективність роботи.

У межах цієї роботи предметна область розглядається як сукупність взаємопов’язаних процесів. Сюди входять завантаження тексту, його попередня підготовка та аналіз, виявлення правописних і стилістичних помилок, порівняння варіантів, формування корпусу, подальша візуалізація й експорт. Визначено, що для практичного використання в редакторському середовищі ці процеси варто об’єднати в межах єдиного вебзастосунку [3].

Перш за все потрібно мати на увазі фактор різнорідності вхідних даних. Наприклад, тексти можуть надходити з локальних файлів, вебпосилань або бути введеними вручну. Ще однією важливою рисою предметної області є поєднання автоматичного й напівавтоматичного підходів. Повністю автоматичний аналіз корисний для швидкого отримання підказок. Проте у багатьох випадках користувач повинен мати можливість перевірити, уточнити або скоригувати інтерпретацію системи. Під цю вимогу підпадають інструменти ручної анотації, прийняття чи відхилення запропонованих виправлень і створення власних словників.

Користувачами системи є декілька основних ролей:

− гість має доступ до обмеженого функціоналу, може переглядати наявні інструменти;

− дослідник може завантажувати документ із текстом чи посилання, запускати аналіз, експортувати результати;

− редактор, який приймає або відхиляє пропозиції виправлень від користувачів, виправляє зауваження, анотує приклади помилок;

− адміністратор, який керує підключеними моделями, налаштуваннями правил, правами доступу.

Така рольова модель безпосередньо впливає на архітектуру застосунку, оскільки вимагає розмежування функціональних можливостей і доступу до даних.

Окремої уваги потребує питання безпеки та цілісності даних. Система працює з текстами, що можуть містити навчальні, наукові або редакторські матеріали, а також конфіденційні дані. Через це постає потреба в автентифікації користувачів, контролі доступу до ресурсів, захисті результатів аналізу та можливості відстеження змін. Зважаючи на те, що результати перевірки можуть змінюватися залежно від оновлення мовних ресурсів або алгоритмів, важливою також є версійність правил і моделей.

Отже, предметна область визначається комплексною проблемою об’єднання обробки, інтерпретації та представлення результатів у межах єдиного середовища. Вихідною точкою для проєктування вебзастосунку лінгвістичного аналізу тексту «LinguaInsight» є необхідність розробки сукупності існуючих мовних блоків.

1.2 Огляд існуючих програмних рішень

Аналіз існуючих програмних рішень дає змогу оцінити сучасний стан предметної області. Завдяки ньому можна виявити функціональні та архітектурні обмеження наявних інструментів. До того ж, ці властивості враховуються при формуванні вимог до системи, що розробляється.

Для порівняння обрано п’ять рішень, що відображають різні підходи до роботи з текстом: Grammarly як засіб корекції граматики та перевірки написання (табл. 1.1) [4], LanguageTool як багатонапрямлений сервіс (табл. 1.2) [5], DeepL Write – інструмент для написання текстів в єдиному стилі (табл. 1.3) [6], Google AI Grammar Checker (табл. 1.4) [7; 8] та Voyant Tools як інструмент частотного аналізу та графічного представлення (табл. 1.5) [9].

Grammarly

Першим аналогом є Grammarly – це одна з найвідоміших хмарних систем перевірки граматики, правопису й стилю. Вона орієнтована на практичне редагування текстів і використовує сучасні ШІ-моделі для формування рекомендацій. Сильними сторонами Grammarly є широкий набір інтеграцій та зручність використання в повсякденній роботі. Проте ця система не орієнтована на корпусний або поглиблений лінгвістичний аналіз. Крім того, її функціональність істотно залежить від хмарної інфраструктури та комерційної моделі використання.

Таблиця 1.1 – Опис хмарної системи Grammarly

Назва	Grammarly
Виробник	Grammarly Inc. (США)
Архітектура	Клієнт-серверна, хмарна з ШІ-моделями для обробки текстів
Мова реалізації	Python (TensorFlow, PyTorch), C++, JavaScript (фронтенд)
Основні функції	1) автоматична перевірка граматики та правопису; 2) виявлення стилістичних і семантичних помилок, плагіату; 3) пропозиції щодо перефразування; 4) виявлення плагіату; 5) адаптація стилю під контекст (формальний, неформальний).

Кінець таблиці 1.1

Переваги	1) високоточні рекомендації; 2) підтримка багатьох платформ; 3) інтеграція з MS Word, Google Docs, Gmail; 4) потужний ШІ; 5) постійні оновлення алгоритмів; 6) інтуїтивно зрозумілий інтерфейс.
Недоліки	1) платна повна версія; 2) підтримка обмеженої кількості мов; 3) надмірні рекомендації, які змінюють авторський стиль; 4) потребує постійного інтернет-з'єднання; 5) регіональне обмеження деяких функцій (перевірка на плагіат).

Інтерфейс Grammarly наведений на рис. 1.1.

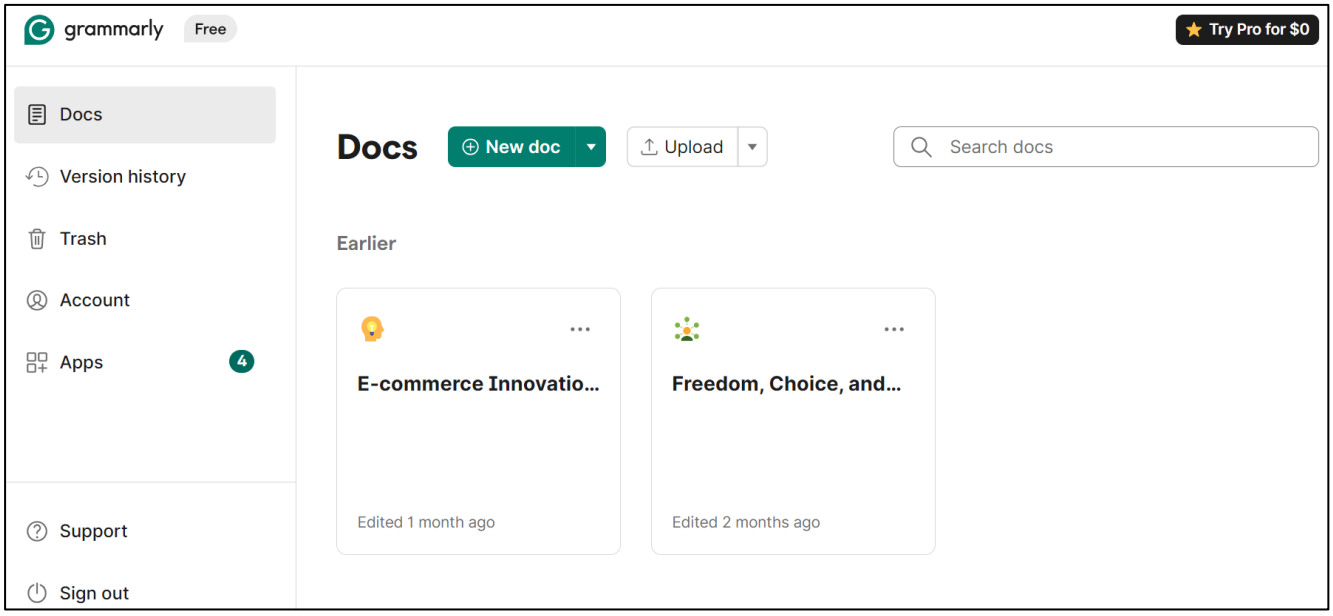

Рисунок 1.1 – Інтерфейс системи Grammarly

Цікаво, що 2025 року система запустила вісім ШІ-агентів:

1) Reader Reactions прогнозує реакцію читача, допомагає визначати непорозуміння та адаптувати текст під конкретну аудиторію;

2) AI Grader оцінює роботу за завантаженими критеріями, темою та інформацією про курс, даючи персоналізовані рекомендації;

3) Citation Finder підбирає релевантні джерела, автоматично форматує посилання;

4) Expert Review надає експертну оцінку тексту і поради для підвищення якості;

5) Proofreader працює як особистий асистент із письма, пропонуючи поради для чіткості та стилю (рис. 1.2);

6) AI Detector визначає, наскільки текст створений ШІ чи людиною, допомагаючи зберегти автентичність;

7) Plagiarism Checker перевіряє текст на відповідність академічним роботам, вебресурсам та базам даних;

8) Paraphraser адаптує текст під потрібний тон, аудиторію та стиль, оцінює поточний тон і допомагає створити власний стиль письма [10].

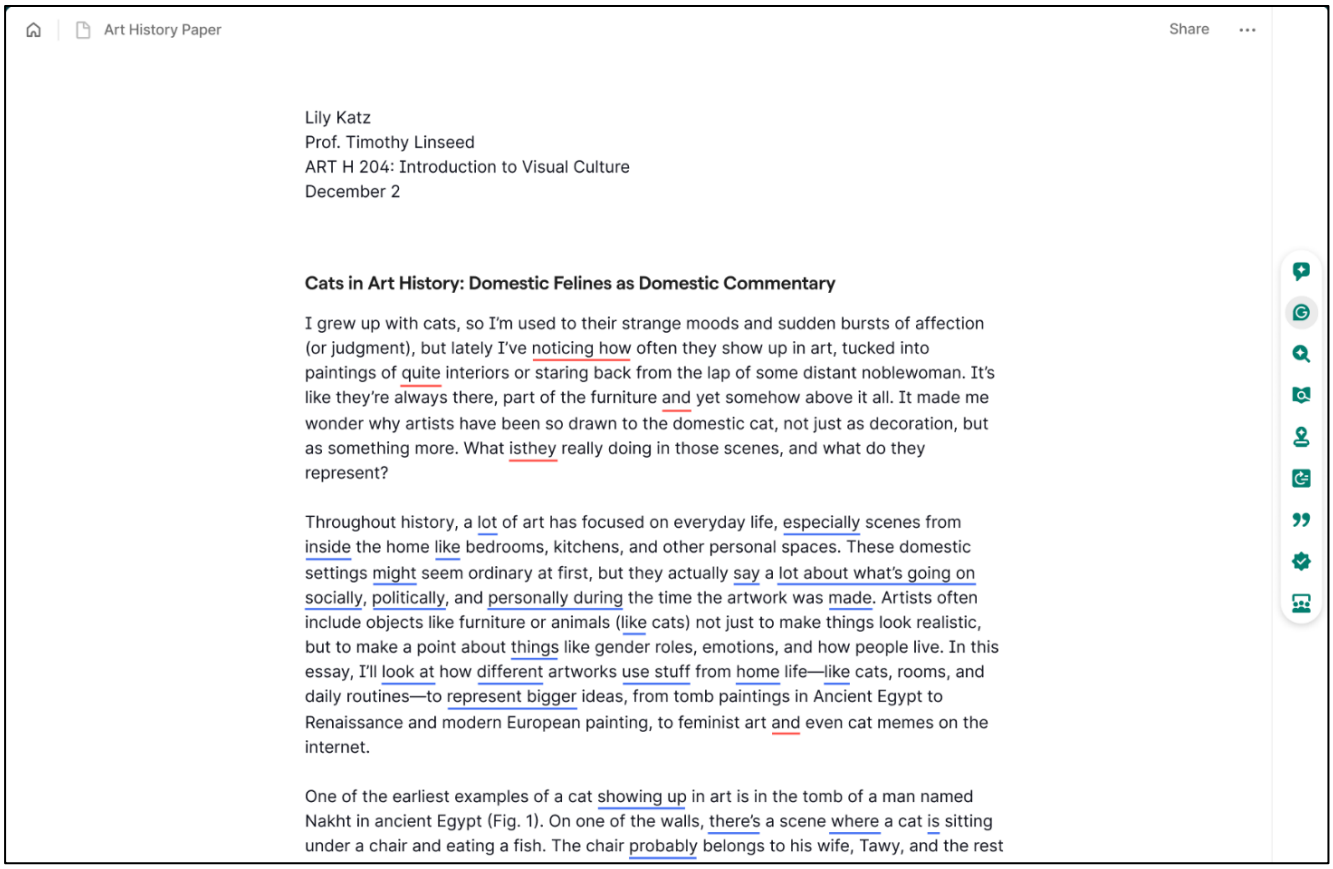

Рисунок 1.2 – Використання ШІ-агента Proofreader

LanguageTool

Другим аналогом є LanguageTool, який поєднує rule-based підхід із окремими машинними методами. На відміну від Grammarly, ця система є більш відкритою та

гнучкою і підтримує значну кількість мов. Її додатковим функціоналом є змога використовувати власні правила та словники. Для предметної області даної роботи це суттєва перевага, адже можливість персоналізації наближає систему до вимог наукового та освітнього середовища. Попри суттєві переваги, LanguageTool зосереджений переважно на перевірці правопису й граматики. Застосування на практиці показує, що глибокий аналіз та дослідження, в тому числі й семантичні, у ньому реалізовані неповноцінно. Також його інтерфейс (рис. 1.3) і масштабованість для великих обсягів тексту поступаються більш спеціалізованим рішенням.

Таблиця 1.2 – Опис LanguageTool

Назва	LanguageTool
Виробник	Open-source спільнота (LanguageTool Foundation)
Архітектура	Клієнт-серверна або локальна установка; rule-based та машинне навчання
Мова реалізації	Java, JavaScript, Python API
Основні функції	1) перевірка правопису та граматики для 30+ мов (українська включно); 2) виявлення стилістичних помилок; 3) інтеграція з браузерами та офісними редакторами; 4) API для розробників; 5) можливість створення власних правил, словників.
Переваги	1) безкоштовна і відкрита; 2) можлива персоналізація; 3) дозволяє працювати офлайн; 4) легка інтеграція через API у власні продукти; 5) є готові плагіни для LibreOffice, MS Word, браузерів; 6) спільнота постійно розширює функціонал.
Недоліки	1) менш точна перевірка стилістики (порівняно з Grammarly); 2) обмежені ШІ-можливості; 3) ліміти в онлайн-редакторі; 4) менш зручний і сучасний інтерфейс; 5) некоректна робота з великими текстами і документами, повільна обробка.

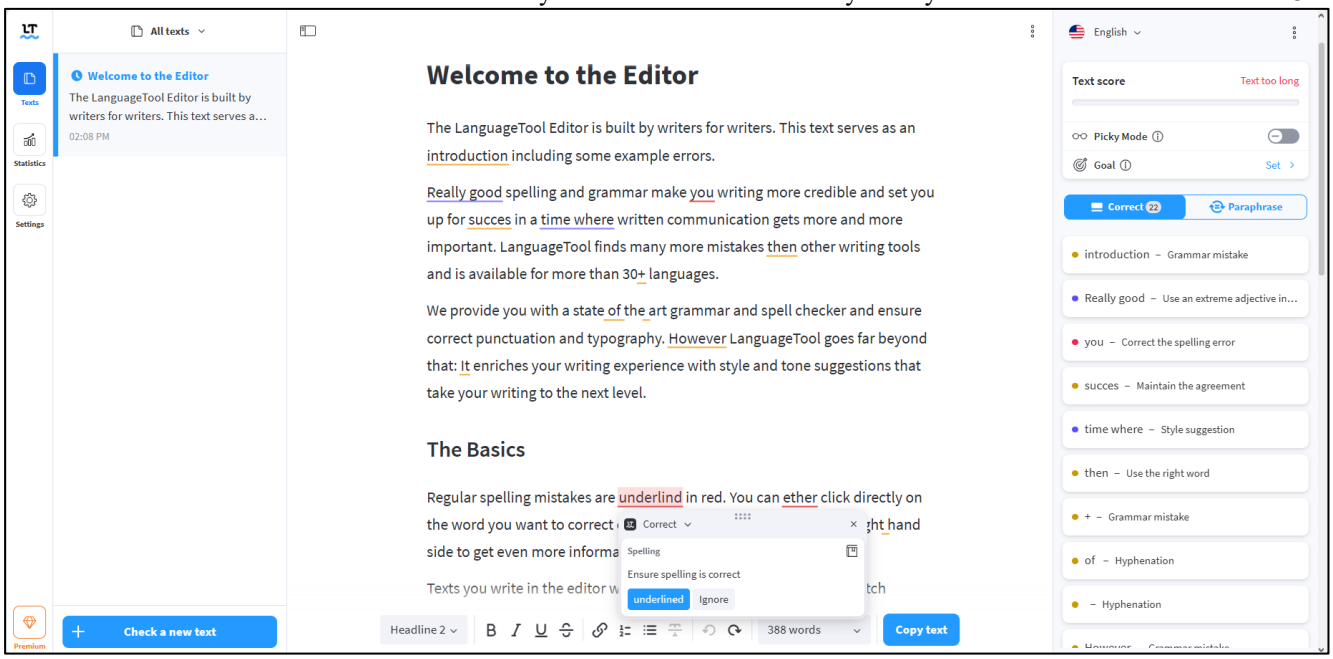

Рисунок 1.3 – Інтерфейс вебзастосунку LanguageTool

DeepL Write

Наступним розглянуто DeepL Write, що репрезентує підхід до інтелектуального редагування тексту на основі нейромережевих моделей. Це рішення особливо корисне у стилістичному вдосконаленні, перефразуванні та підвищенні природності мовлення. Однак функціонально DeepL Write є скоріше інструментом, ніж повноцінною системою. Він не розрахований безпосередньо на лінгвістичний аналіз. Додатковим обмеженням у контексті цієї роботи є відсутність повноцінної підтримки української мови. Через це аналог не підходить як основний орієнтир для розробки прикладного застосунку, спрямованого на ширше коло користувачів.

Таблиця 1.3 – Опис інструменту DeepL Write

Назва	DeepL Write
Виробник	DeepL GmbH (Німеччина)
Архітектура	Хмарна, нейромережі (transformer-based)
Мова реалізації	Python (TensorFlow, PyTorch), C++, JavaScript
Основні функції	1) автоматичне редагування текстів (стиль, зрозумілість); 2) синонімічні та стилістичні пропозиції; 3) перевірка правопису; 4) інтеграція з перекладачем DeepL; 5) швидка робота з великими обсягами тексту.

Кінець таблиці 1.3

Переваги	1) дуже природні стилістичні рекомендації; 2) потужна модель ШІ; 3) висока якість роботи з мовами ЄС; 4) можливість поєднання редагування і перекладу; 5) простий і сучасний інтерфейс; 6) можливість інтеграції у Chrome.
Недоліки	1) обмежена кількість мов (українська не підтримується); 2) безкоштовна версія має ліміти на кількість перевірок; 3) основний акцент на стилі, а не на граматичних помилках; 4) потребує постійного інтернет-з'єднання; 5) обмеженість інтеграцій.

Інтерфейс інструменту DeepL Write наведений на рис. 1.4.

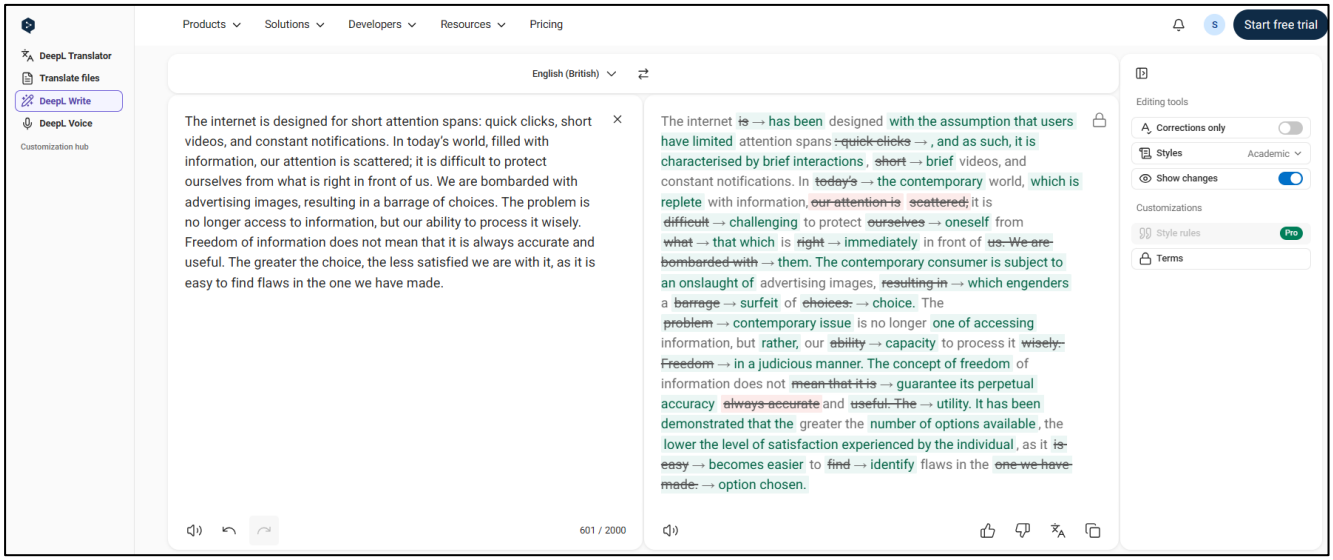

Рисунок 1.4 – Інтерфейс інструменту DeepL Write

Google AI Grammar Checker

Четвертим рішенням є Google AI Grammar Checker, інтегрований у середовище Google Docs. Його сильна сторона полягає у простоті доступу, швидкодії та можливості вбудовування в хмарний редактор документів. Для користувача це зручний інструмент миттєвої перевірки правопису та окремих граматичних і стилістичних помилок. Разом із позитивними характеристиками, помітним стає доволі обмежена функціональність. Він не надає можливостей для

формування складних вибірок, використання своїх словників і правил у потрібному обсязі. У системі відсутня підтримка розширеної аналітики й візуалізації результатів. Крім того, працездатність істотно залежить від екосистеми Google.

Таблиця 1.4 – Опис інструменту Google AI Grammar Checker (Google Docs)

Назва	Google AI Grammar Checker
Виробник	Google LLC.
Архітектура	Хмарна (Google Cloud AI)
Мова реалізації	Python (TensorFlow, PyTorch), C++, Java, JavaScript
Основні функції	1) автоперевірка правопису, граматики; 2) контекстна перевірка; 3) стильові підказки; 4) робота в реальному часі; 5) підтримка багатьох мов.
Переваги	1) безкоштовна інтеграція в Google Docs; 2) доступний на різних пристроях; 3) швидкість роботи та миттєва перевірка завдяки онлайн-редактору.
Недоліки	1) обмежена кількість мов (підтримку української реалізовано частково); 2) немає перевірки на плагіат; 3) потрібне стабільне підключення до інтернету; 4) залежність від екосистеми Google.

Інтерфейс інструменту Google AI Grammar Checker наведений на рис. 1.5.

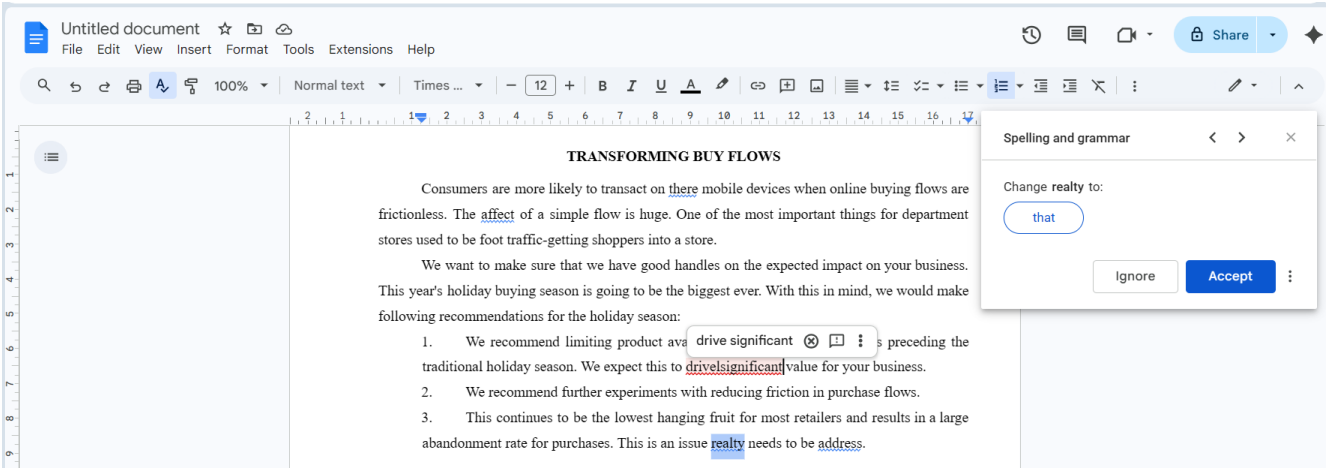

Рисунок 1.5 – Інтерфейс інструменту Google AI Grammar Checker

Voyant Tools

Окрему групу становить Voyant Tools – це відкрита платформа для корпусного та статистичного аналізу текстів. На відміну від попередніх рішень, цей інструмент орієнтований саме на дослідження отриманих даних. Voyant Tools підтримує побудову частотних словників, concordance, колокацій, графіків частот, хмар слів та інших інтерактивних візуалізацій. Це робить систему найближчою до дослідницького сегмента предметної області. Водночас Voyant Tools не вирішує наступні завдання: автоматичний контроль правопису, стилістичні рекомендації та інтегрована робота із сучасними NLP-модулями. Він демонструє потужний підхід до візуалізації та корпусної аналітики, але не охоплює повний цикл задач.

Таблиця 1.5 – Опис вебзастосунку Voyant Tools

Назва	Voyant Tools
Виробник	Stéfan Sinclair і Geoffrey Rockwell
Архітектура	Клієнт-серверна, існує окрема серверна збірка для локальної та/або офлайн-роботи – VoyantServer
Мова реалізації	Java (бекенд), клієнтський інтерфейс – JavaScript та HTML
Основні функції	1) побудова частотних словників і списків словоформ; 2) графіки частот (Trends) і тимчасові лінії; 3) KWIC та concordance (алфавітний покажчик слів із випадками їх вживання в певному тексті); 4) хмари слів (Cirrus), Bubblelines, Collocates; 5) інтерактивні візуалізації і мультипанельний інтерфейс; 6) завантаження текстів з файлів або URL, збереження та поширення колекцій; 7) розширення для програмного аналізу (Spyral/Voyant Notebooks).
Переваги	1) безкоштовна та відкрита платформа (open-source); 2) дуже швидка обробка даних; 3) інтерактивне представлення результатів, зручний інтерфейс для викладачів і дослідників; 4) можливість працювати локально через VoyantServer; 5) зрозуміла документація та навчальні посібники для гуманітаріїв; 6) підтримка багатьох мов як окремо, так і одночасно.

Кінець таблиці 1.5

Недоліки	1) обмеження при аналізі великих корпусів (масштабування може вимагати додаткових ресурсів або налаштувань); 2) менше функціональності «просунутого» NLP/машинного навчання порівняно з деякими спеціалізованими інструментами; 3) інтерактивний GUI підходить для дослідження, але для повністю автоматизованих пайплайнів потрібні додаткові інструменти або скрипти; 4) деякі модулі/візуалізації можуть здаватися занадто простими для певних аналітичних потреб; 5) можливі помилки при створенні корпусу із надто довгих або параметризованих URL (сервер відхиляє запит); 6) корпуси, створені за URL можуть містити інформацію з рекламних банерів, навігацію та HTML-теги, що спотворює результати; локальні документи зазвичай точніші для аналізу.

Інтерфейс вебзастосунку Voyant Tools наведений на рис. 1.6.

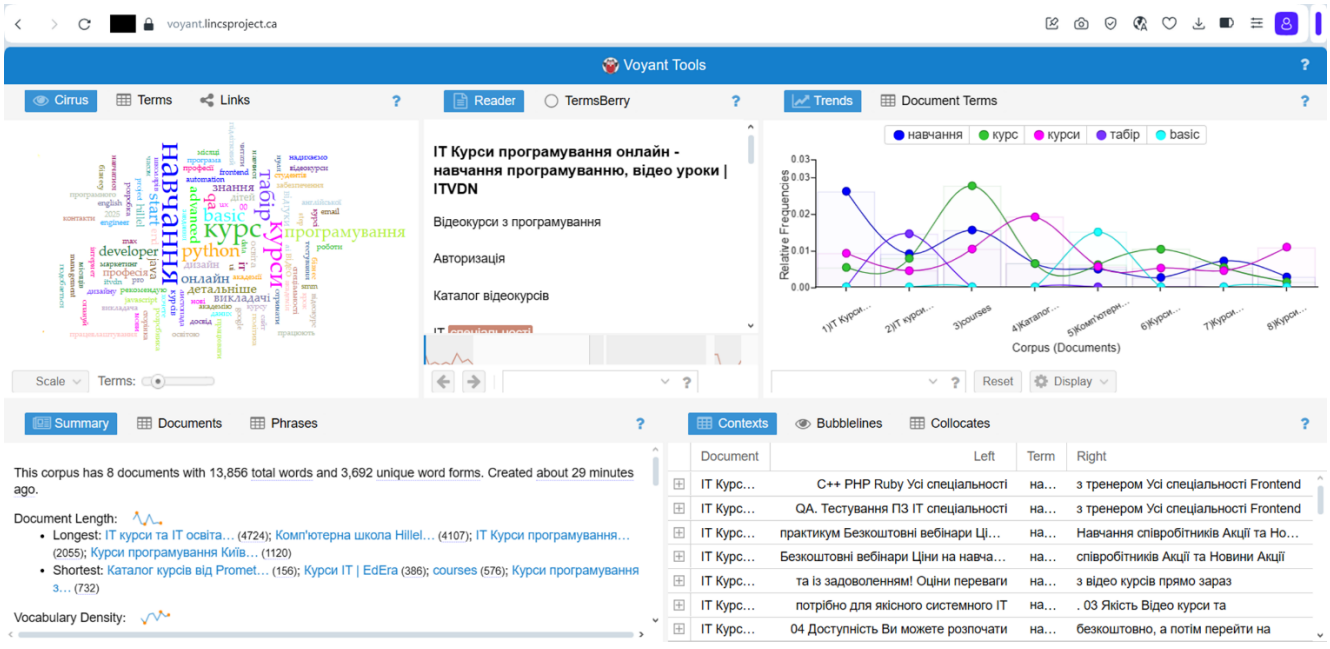

Рисунок 1.6 – Інтерфейс вебзастосунку Voyant Tools

Порівняльний аналіз наведених систем дозволяє зробити кілька важливих висновків. По-перше, наявні рішення можна умовно поділити на дві групи: системи інтелектуального редагування тексту та інструменти корпусного аналізу. По-друге, жодна із розглянутих платформ не об’єднує в межах єдиного середовища повний

набір функцій для проведення якісної обробки. По-третє, частина аналогів є потужною лише в межах певної мови або екосистеми, що знижує їх універсальність при застосуванні.

Отже, огляд наявних рішень підтверджує, що сучасне програмне забезпечення для роботи з текстом є доволі функціональним, але переважно вузькоспеціалізованим. Об’єднання декількох блоків стає необхідним для розроблення власного вебзастосунку.

1.3 Обґрунтування необхідності розроблення власного програмного продукту

Відповідно до проведеного аналізу, виявлено, що проблема роботи з текстом у цифровому середовищі полягає у відсутності цілісного програмного продукту, який об’єднував би повний цикл лінгвістичного опрацювання. Наявні аналоги вирішують лише такі задачі, як: перевірка правопису, стилістичне редагування, статистичний аналіз або візуалізація корпусних даних. Проте між цими блоками існує природний функціональний зв’язок, який у більшості наявних систем не реалізований на рівні архітектури.

З огляду на це, дослідник потребує конкретних статистичних графіків та морфологічної складової. Редактору потрібні не лише стилістичні підказки, а й пояснення, можливість працювати з прикладами. Поєднання цих потреб формує вимоги до нового програмного продукту.

У більшості аналогів користувач розглядається як окремий редактор або дослідник. Таке проєктне рішення робить застосунок інструментом аналізу та повноцінним цифровим середовищем для різногалузевої взаємодії [3].

Ще одним аргументом на користь власної розробки є потреба в адаптивності та розширюваності. Сфера лінгвістичного аналізу швидко розвивається: щороку змінюються мовні норми, правила правопису, з’являються нові моделі NLP, розширюються вимоги до якості аналітики.

У практичному сенсі розроблення власного вебзастосунку дозволяє вирішити одразу кілька проблем:

1) усунути необхідність використовувати кілька розрізнених сервісів;

2) реалізувати єдину модель даних, у якій текст, його анотація, статистика, правила перевірки та історія змін є пов’язаними сутностями;

3) адаптувати систему під конкретні сценарії, чого неможливо досягти за рахунок використання готових аналогів без суттєвих компромісів.

Таким чином, розроблення власного програмного продукту є концептуально обґрунтованим. Це поєднає переваги систем перевірки граматики, редагування стилістики та корпусного аналізу, шляхом виключення їхньої функціональної фрагментарності.

Висновки до розділу 1

У першому розділі проведено аналіз предметної області вебзастосунку лінгвістичного аналізу тексту. Встановлено, що сучасна робота з текстовими даними вимагає комплексного підходу. Він включає збір текстів; попередню обробку; морфологічний, синтаксичний і семантичний аналіз; формування корпусів; візуалізацію результатів та підтримку ручної корекції.

Під час огляду існуючих програмних рішень проаналізовано можливості Grammarly, LanguageTool, DeepL Write, Google AI Grammar Checker і Voyant Tools. Виявлено, що кожен із перерахованих інструментів має сильні сторони в межах окремого класу задач, однак жоден із них не забезпечує повний цикл інтегрованого лінгвістичного опрацювання тексту в межах єдиного середовища.

Проведений аналіз визначив необхідність розроблення власного вебзастосунку. В його основу покладено поєднання модулів морфології та синтаксису за допомогою автоматичного контролю правопису й стилістики, а також ручної анотації. Такий підхід дозволяє розглядати майбутню систему як цілісне цифрове середовище, здатне забезпечити ефективну роботу з текстами для різних категорій користувачів.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ РОБОТИ

2.1 Аналіз сучасного інструментарію та методів розробки вебзастосунків

Аналіз сучасного стану засобів розроблення систем лінгвістичного аналізу тексту дає змогу виділити кілька взаємопов'язаних напрямів:

1) архітектурні підходи до побудови інтерактивних вебзастосунків для роботи з текстами;
2) інструментарій автоматичної обробки природної мови;
3) засоби зберігання, індексації та пошуку текстових даних;
4) інструменти візуалізації та підготовки аналітичних звітів;
5) інфраструктурні рішення для розгортання багатокомпонентних систем.

Сучасні вебзастосунки, орієнтовані на аналіз тексту, будуються як багаторівневі клієнт-серверні системи. Користувацький інтерфейс, прикладна серверна логіка, аналітичний модуль і рівень зберігання даних виконують у них різні функції. Система має забезпечувати обробку тексту і підтримку мовних моделей, збереження результатів і швидкий пошук за вмістом (табл. 2.1) [1].

Взагалі реальні NLP-системи мають розглядатися як частина ширшого програмного продукту. Тут мають значення інтеграція з інтерфейсом, оцінювання результатів і подальше їх використання в конкретному прикладному сценарії [11].

Однією із провідних тенденцій є використання **SPA-підходу** для клієнтської частини. У застосунках, де користувач працює з текстом, важливо мінімізувати повні перезавантаження сторінок і забезпечити швидке оновлення окремих частин інтерфейсу. Через це в подібних системах поширеним є використання **React**, який підтримує компонентну модель побудови та гнучке керування станом сторінки [12]. Функція інтерфейсу – одночасне відображення усіх етапів обробки.

Для взаємодії між клієнтом і сервером у таких системах зазвичай використовуються HTTP API. У межах даного проєкту це узгоджується з вибором **Axios** як клієнтської бібліотеки для обміну даними та **React Router** як засобу організації навігації між сторінками й компонентами. Для побудови макету використано **Bootstrap**, що дозволяє створення адаптивного користувацького

інтерфейсу без надмірного ускладнення CSS-структури. Тому клієнтська частина набуває характеру інтерактивного аналітичного середовища.

У контексті серверної частини важливою тенденцією є розмежування прикладної логіки керування користувачами, документами та доступом до ресурсів від обчислювально складних NLP-процесів. Для системи, що проєктується, це обґрунтовує використання двох взаємодоповнювальних серверних компонентів: **Laravel** як прикладного вебрівня та **FastAPI** як аналітичного API-рівня [13]. Дана структура чудово підходить для лінгвістичної роботи.

Дослідник Хагівара М. розглядає NLP-застосунки через практичний цикл створення: підготовку текстових даних, побудову моделі, навчання, перевірку якості та впровадження у прикладне середовище. Така послідовність важлива для використання отриманих результатів у вебредакторі або аналітичному модулі [14].

Серед методів і моделей аналізу тексту в існуючих системах домінує **гібридний підхід**. У його основі лежить поєднання класичних алгоритмів NLP та нейромережевих рішень. Як показує наявність програмних засобів, сучасні бібліотеки обробки природної мови й глибинного навчання розвиваються в Python-екосистемі. За допомогою спеціалізованих NLP-інструментів стає можливим виконання етапів автоматизованого опрацювання тексту. Сюди входять нормалізація і токенізація, сегментація і визначення частин мови, лематизація та розпізнавання іменованих сутностей, а також синтаксичний аналіз. У даному проєкті таку роль виконують **spaCy** (у публікації [15] подано як industrial-strength – бібліотеку промислового рівня) та **Stanza**. Вони надають готові пайплайни для багаторівневого мовного аналізу і дозволяють працювати зі словом [2; 16].

Там, де потрібна глибша семантична інтерпретація тексту, доцільно використовувати **Hugging Face Transformers**, що відповідає сучасному стану розвитку NLP. Трансформерні моделі є основою для складніших задач, що виходять за межі традиційного rule-based або статистичного аналізу [1]. Виконання цих моделей у системі забезпечується через **PyTorch**, який виступає базою для глибинного навчання та роботи з нейромережевими компонентами [17].

Hugging Face Transformers застосовуються для класифікації текстів, розпізнавання сутностей і машинного перекладу, про що зазначено у відповідній статті-рев'ю [18]. Даний підхід спрямований на контекстно чутливе опрацювання тексту. Огляд також акцентує увагу на моделях BERT і GPT, ефективність яких пов'язана зі здатністю зменшувати вплив неоднозначності мовних конструкцій.

Використання попередньо навчених моделей дає змогу швидше створювати прикладні рішення й адаптувати їх до конкретних задач без повного навчання моделі з нуля. У цьому полягає роль transfer learning (передавального навчання) у NLP. Для вебзастосунку, що розробляється, можна використовувати готові мовні моделі та налаштовувати їх під конкретні корпуси [19].

Окремий напрям сучасного інструментарію стосується зберігання та пошуку результатів аналізу. Саме тому обґрунтованим є використання **PostgreSQL** із підтримкою **JSONB**. Це дає змогу зберігати не лише сам документ, а й пов'язані з ним лінгвістичні структури без втрати цілісності моделі даних [20].

Водночас для швидкого всеохоплюючого пошуку та побудови вибірок за текстовими й аналітичними ознаками доцільно використовувати окремий пошуковий шар. У межах проєкту таку роль виконує **OpenSearch**, який забезпечує індексацію текстових колекцій та інших елементів [21].

У класичній праці Меннінга К., Шютце Г. та Рагхавана П. інформаційний пошук розглядається як процес збирання, індексації та оцінювання документів у великих текстових колекціях. Це варто враховувати безпосередньо під час проєктування вебзастосунку, оскільки для роботи з корпусами необхідно забезпечити швидкий пошук за словами та лемами [22].

У задачах обробки природної мови та інформаційного пошуку широко використовуються графові підходи. Вони можуть бути корисними під час представлення зв'язків між словами або синтаксичних залежностей. При цьому графова модель дає змогу наочно показати відношення між елементами тексту [23].

Суттєвим компонентом сучасних систем лінгвістичного аналізу є також візуалізація даних. Окрім сирого JSON або звичайного табличного списку система повинна надавати користувачу зрозуміле представлення результатів [24]. Для

цього в проєкті використано **Plotly**, **matplotlib**, **pandas** та **d3-cloud**. У статті [25] порівнюються бібліотеки візуалізації Python за різними критеріями: функціональністю, гнучкістю, простотою використання та швидкодією, що впливає на вибір інструментів графічного представлення результатів у системі. Plotly доцільний для інтерактивних графіків, де користувач має змогу працювати з динамічними діаграмами, частотними розподілами та інтерактивними панелями. Matplotlib є зручним для побудови статичних графіків, які можуть бути включені до звітів або експортуватися в готовому вигляді. Pandas використовується як інструмент табличного відтворення та візуалізації. Практичні рекомендації щодо побудови графіків у Python та способи впровадження варіативності комбінацій наведено у статті Хана С. та Квака І.-Ю. [26]. Зокрема у ній подано інформацію про використання заголовків, назв осей, легенд та інших елементів оформлення. Даний матеріал може використовуватись під час розробки модуля аналітики.

Графіки і таблиці, формалізовані через feature models (моделі ознак або функцій), являють собою найкращі практики побудови візуалізацій. У статті [27] описано покрокову конфігурацію, яка дає змогу адаптувати графічне подання до різних режимів відображення результатів.

Не менш важливим є інфраструктурний аспект. Система поєднує технологічно різнорідні компоненти (React-клієнт, Laravel, FastAPI, PostgreSQL, OpenSearch), тому для розгортання варто використовувати контейнеризацію. У статті [28] контейнери розглядаються як ізольовані середовища, що інкапсулюють застосунок разом із його залежностями. Це впливає на сумісність, переносимість і відтворюваність програмного оточення. Для складних систем особливо важливими є засоби масштабованого розгортання контейнерів та керування ними. В такому випадку в нагоді стає оркестрація, яка спрощує підтримку й адміністрування великої кількості взаємопов’язаних сервісів.

У межах проєкту це реалізується за допомогою **Docker** і **Docker Compose**. У статті [29] запропоновано таксономію Docker-екосистеми та проаналізовано архітектурні підходи для її реалізації. Наведені у ній поради й настанови дають змогу відтворити середовище як єдину багатокомпонентну платформу.

Таблиця 2.1 – Основні напрями сучасного інструментарію для систем лінгвістичного аналізу тексту

Напрям	**Ключові інструменти та підходи**	**Значення для системи, що проєктується**
Клієнтські SPA-інтерфейси	React, Axios, React Router, Bootstrap	забезпечують інтерактивну роботу з текстом, фільтрами, таблицями та графіками
Серверна прикладна логіка	Laravel	відповідає за маршрути, користувачів, доступ, службові сценарії
Аналітичний API-рівень	FastAPI	забезпечує виклик NLP-модулів і повернення структурованих результатів
NLP-аналіз	spaCy, Stanza	токенізація, лематизація, POS-тегінг, NER, синтаксичний аналіз
Нейромережеві NLP-моделі	Hugging Face Transformers, PyTorch	контекстний аналіз, класифікація, семантичні задачі
Зберігання структурованих даних	PostgreSQL	збереження документів, користувачів, корпусів, сесій
Зберігання напівструктурованих результатів	JSONB	робота з вкладеними результатами анотації та метаданими
Пошук та індексація	OpenSearch	повнотекстовий пошук, вибірки за лінгвістичними ознаками
Візуалізація	Plotly, matplotlib, d3-cloud	інтерактивні та статичні графіки, хмара слів
Звітність і таблична аналітика	pandas, pdfmake	підготовка статистики, таблиць і зведених результатів
Інфраструктура розгортання	Docker, Docker Compose	стабільний запуск багатокомпонентної системи

Табл. 2.1 підкреслює, що для вебзастосунків лінгвістичного аналізу тексту найбільш доцільним є поєднання комбінованих засобів розробки. Саме така сукупність підходів і визначає технологічну основу системи «LinguaInsight», що проєктується.

2.2 Технологічний стек та архітектура системи

На основі результатів проведеного аналізу сформовано технологічний стек та визначено архітектуру взаємодії компонентів майбутньої системи. Відповідно до цього визначено, що вебзастосунок лінгвістичного аналізу тексту повинен забезпечувати низку можливостей. Вони включають збереження документів і відображення результатів, послідовне виконання аналітичного конвеєра: імпорт тексту, його обробку, індексацію в пошуковому шарі та представлення результатів у вигляді таблиць, графіків і текстових фрагментів із розміткою (табл. 2.2).

Система, що проєктується, реалізує багаторівневу клієнт-серверну архітектуру. Такий підхід дозволяє розмежувати відповідальність між складовими системи та спростити подальшу підтримку й розвиток застосунку.

Клієнтська частина системи виконується у браузері користувача у вигляді Single Page Application на базі React [12]. Для навігації між сторінками та функціональними модулями використовується React Router, а для обміну даними із сервером – Axios. Оформлення інтерфейсу та адаптивна сітка реалізуються через Bootstrap. Як результат, користувач працює з єдиним інтерактивним середовищем.

Серверна частина має комбіновану побудову. Laravel виконує роль прикладного вебрівня, що відповідає за маршрутизацію та роботу з користувачами і є частиною прикладної логіки. Завдяки наявності FastAPI реалізується окремий аналітичний API. Таке розділення є доцільним, оскільки Python-екосистема є базовою для сучасного NLP [13]. Разом із цим Laravel забезпечує зручну організацію вебсервісної частини застосунку. У підсумку система розподіляє усі завдання між спеціалізованими компонентами.

Аналітичне ядро системи реалізується засобами spaCy, Stanza, Hugging Face Transformers та PyTorch. Після отримання документа система виконує його нормалізацію, токенізацію, сегментацію на речення, морфологічну анотацію, лематизацію та розпізнавання іменованих сутностей [3]. Далі, залежно від обраного режиму, можуть виконуватися синтаксичний аналіз, виділення ключових слів, побудова *n*-грам, аналіз читабельності тощо. Результати аналізу повертаються у структурованому вигляді, що дозволяє відображати їх у різних форматах [15].

Рівень доступу до даних реалізовано на базі PostgreSQL. Це дозволяє системі зберігати відомості про користувачів, тексти і корпуси, також налаштування аналізу, історію та інші сутності. Особливістю моделі даних є поєднання класичних реляційних таблиць із полями типу JSONB. Основне їх призначення у даній системі – збереження списків токенів, морфологічних ознак та параметрів моделі [20]. Така побудова уникає чіткого розмежування даних без втрати зв’язків між ними.

Окремий компонент системи становить OpenSearch, який використовується для індексації текстів і результатів їх обробки. Завдяки цьому стає можливою реалізація швидкого пошуку за словами, граматичними ознаками, категоріями сутностей, підкорпусами тощо. Такий підхід особливо важливий для функцій concordance, вибірки контекстів і порівняння результатів між текстами [21].

Життєвий цикл запиту в системі виглядає наступним чином. Користувач завантажує текст, файл або посилання через React-інтерфейс. Після цього клієнтська частина надсилає HTTP-запит до серверного рівня. Laravel обробляє запит згідно з доступом, ідентифікацією документа та прикладною логікою, після чого передає текст і параметри аналізу до FastAPI. Аналітичний сервіс виконує NLP-конвеєр із використанням spaCy, Stanza або трансформерних моделей (в залежності від складності), формує структуровану відповідь та повертає результат. Далі результати зберігаються в PostgreSQL, індексуються в OpenSearch і надсилаються назад до клієнта. На завершальному етапі React відображає дані у вигляді текстової розмітки, таблиць, графіків та інтерактивних блоків аналізу.

Pandas застосовується для підготовки табличних наборів даних, обчислення статистики й формування зведених результатів. Matplotlib використовується для побудови статичних графіків, зручних для експорту до звітів. Plotly забезпечує інтерактивні візуалізації, які користувач може відстежувати безпосередньо в інтерфейсі вебзастосунку. Завдяки цьому результати аналізу подаються не лише як текстові анотації, а і як зрозумілі аналітичні представлення [24; 26].

Інфраструктурний рівень реалізується на базі Docker і Docker Compose. Це зменшує залежність від конкретної машини розробника. Впровадження цієї стратегії полегшує локальне тестування і дає змогу розглядати систему як узгоджений багатокомпонентний програмний продукт.

Таблиця 2.2 – Співставлення технологічних задач із вибором інструментів

Задача	Технологічний стек	Очікуваний ефект
Побудова інтерактивного інтерфейсу	React	компонентна архітектура, швидке оновлення інтерфейсу
HTTP-взаємодія між клієнтом і сервером	Axios	централізований обмін даними через API
Клієнтська маршрутизація	React Router	навігація між сторінками без перезавантаження
Адаптивне оформлення інтерфейсу	CSS3, Bootstrap	responsive-дизайн і швидке створення інтерфейсу користувача
Прикладна серверна логіка	Laravel	організація користувачів, доступу, маршрутів і службових сценаріїв
Аналітичний API-рівень	FastAPI	обробка запитів до NLP-конвеєра
Морфологічний і синтаксичний аналіз	spaCy, Stanza	токенізація, лематизація, POS-тегінг, NER, парсинг
Глибинні мовні моделі	Hugging Face Transformers, PyTorch	семантичний аналіз, класифікація, контекстне опрацювання
Зберігання реляційних даних	PostgreSQL	цілісність зв’язків між сутностями системи
Зберігання вкладених результатів аналізу	JSONB	гнучке подання анотацій і метаданих
Пошук та аналітичні вибірки	OpenSearch	швидкий повнотекстовий пошук і фільтрація
Таблична аналітика	pandas	формування звітів, статистики, підготовка наборів даних
Статичні графіки	matplotlib	побудова ілюстрацій для звітів
Інтерактивні графіки	Plotly	візуальне дослідження результатів в інтерфейсі
Контейнеризоване розгортання	Docker, Docker Compose	відтворюване й стабільне середовище запуску

Таким чином, обраний технологічний стек безпосередньо відповідає функціональним потребам системи. Він забезпечить узгоджену взаємодію основних складових вебзастосунку та створить основу для подальшого розширення.

2.3 Специфікація вимог до програмного забезпечення

1) ***Призначення та межі проєкту:***

1.1) призначення системи (застосунку), для якої розробляється програмне забезпечення: вебзастосунок «LinguaInsight» призначений для автоматизації лінгвістичного аналізу та опрацювання корпусних даних з метою дослідження мовних явищ під час навчання та редагування текстів;

1.2) погодження, що ухвалені в програмній документації: узгоджено мову програмування Python; FastAPI; клієнт – React з CSS3 та Bootstrap; PostgreSQL (для метаданих JSONB); NLP-бібліотеки – spaCy, Hugging Face Transformers, Stanza; перевірка правопису – LanguageTool або власні регулярні вирази; контейнеризація та оркестрація – Docker і Docker Compose; CI/CD – GitHub Actions або GitLab CI; візуалізація та звіти – Plotly, matplotlib і pandas;

1.3) межі проєкту ПЗ: охоплює розробку вебклієнта і серверної інфраструктури для аналізу; не передбачає машинний переклад, розпізнавання рукописного тексту й повну автоматичну точність без перевірки користувачем.

2) ***Загальний опис:***

2.1) сфера застосування: освітня, науково-дослідницька та редакційна діяльність; платформа об’єднує дослідників й аналітичний текстовий матеріал;

2.2) характеристики користувачів: базові знання лінгвістики та цифрової філології; вміння користуватись гаджетами (смартфоном, ноутбуком, ПК);

2.3) загальна структура і ***склад системи***:

2.3.1) **модуль авторизації та реєстрації користувачів** – вхід до системи, підтримка аутентифікації через електронну пошту та пароль;

2.3.2) **модуль адміністрування** – адмін-панель, включає створення / редагування / видалення облікових записів, призначення ролей і розмежування прав (гість, дослідник, редактор, адміністратор), скидання паролів, прийняття рішень щодо змін та налаштування правил правопису, керування моделями (підключення нових, версійність, відкат), журнали помилок;

2.3.3) **модуль імпорту тексту та збереження результатів аналізу** надає підтримку імпорту з URL, завантажених файлів (txt, pdf, docx),

введення тексту вручну через UI; автоматичне вилучення метаданих (назва, автор, джерело, дата тощо), збереження у БД із підтримкою структуризації, функції пакетної обробки корпусів і можливість відкладеного запуску;

2.3.4) **модуль передобробки (pipeline)** включає токенізацію, нормалізацію Unicode (лапки, апострофи, пробіли), приведення регістру, видалення зайвих символів; лематизацію, POS-тегінг; налаштування етапів pipeline (підключення різних бібліотек або моделей NLP для кожного етапу);

2.3.5) **модуль NLP-аналізу та перевірки правопису** виконує перевірку орфографії; виявлення стилістичних і граматичних помилок (детектори для комбінованої оцінки), невідповідного узгодження, дублювань слів; інтегрує швидкі production-інструменти для базової обробки (spaCy) і трансформерні моделі для більш точних задач (Hugging Face Transformers), реалізує механізм правил (регулярні вирази), генерує пояснення для підказок або виправлень (визначення, приклад, джерело);

2.3.6) **модуль побудови *n*-грам та пошуку частих фраз** реалізує збір статистики, побудову частотних списків; порівнює підкорпуси (наприклад, студентські тексти та наукові статті); є індексація та контекстний пошук;

2.3.7) **вбудований редактор та інструменти для анотації** представлений вебмодулем із підсвічуванням помилок, інтерактивними підказками, можливістю прийняти або відхилити виправлення; історією змін (версії документа) та відкатом; є інструменти для внесення пропозицій;

2.3.8) **модуль аналітики та візуалізації результатів** представляє побудову статистичних звітів (розподіл типів помилок, динаміка виправлень, частотність), візуалізації (графи залежностей, хмари слів), надає можливість експорту звітів у PDF/CSV для подальшої обробки та використання.

Загальна структура системи має наступний вигляд:

1. **Модуль авторизації користувачів**:

– форма логіну для існуючих користувачів та форма реєстрації;

– системне розмежування прав доступу за ролями: гість, дослідник, редактор, адміністратор.

2. **Модуль адміністрування (адмін-панель):**

– інтерфейс керування користувачами (створення, редагування, видалення облікових записів і призначення ролей);

– розмежування прав за ролями вручну (гість, дослідник, редактор);

– налаштування правил перевірки правопису та стилістики;

– керування моделями (підключення нових, версійність, відкат);

– перегляд журналів помилок, черг обробки та метрик продуктивності.

3. **Модуль імпорту тексту і збереження результатів аналізу:**

– імпорт із URL, завантажених файлів (txt, pdf, docx), введення тексту вручну через UI;

– автоматичне вилучення метаданих (назва, автор, джерело, дата тощо);

– збереження документів і результатів аналізу в БД із підтримкою структурованих метаданих (JSONB);

– пакетна обробка корпусів і відкладений або фоновий запуск аналізу.

4. **Модуль передобробки (pipeline):**

– токенізація, нормалізація Unicode (лапки, апострофи, пробіли), приведення регістру, видалення небажаних символів;

– лематизація та POS-тегінг;

– налаштовувані етапи pipeline (можливість підключати різні бібліотеки або моделі для кожного етапу).

5. **Модуль NLP-аналізу та перевірки правопису:**

– перевірка орфографії, виявлення стилістичних помилок, невідповідностей узгодження, простих граматичних помилок, дублювань слів;

– інтеграція production-інструментів для базової обробки (spaCy) і трансформерних моделей для складніших задач (Hugging Face Transformers);

– механізм правил (регулярні вирази) у поєднанні з детекторами машинного навчання для комбінованої оцінки помилок;

– генерація підказок із поясненням правила, прикладами та рекомендованим виправленням.

6. **Модуль побудови *n*-грам та пошуку частих фраз:**
   - збір статистики *n*-грам, колокацій та частотних виразів;
   - пошук частих фраз у корпусі;
   - порівняння підкорпусів із різними стилями;
   - індексація та швидкий пошук за контекстом.
7. **Вбудований редактор та інструменти для анотації:**
   - вебредактор із підсвічуванням помилок та інтерактивними підказками;
   - прийняття/відхилення пропозицій виправлень, внесення власних правок;
   - історія змін і версійність документів із можливістю відкату;
   - інструменти для ручної анотації помилок (за категоріями, позначки).
8. **Модуль аналітики та візуалізації результатів:**

– побудова статистичних звітів (розподіл типів помилок, динаміка виправлень, часті *n*-грами);

- візуалізації (графи залежностей, хмари слів);
- експорт звітів у PDF/CSV для подальшої обробки або презентацій.

2.4) загальні обмеження:

2.4.1) ліміт на кількіть символів, що обробляється за один запит;

2.4.2) залежність точності аналізу від якості та мови вхідного тексту;

2.4.3) підтримка обмеженої кількості форматів файлів (txt, pdf, docx);

2.4.4) стабільне інтернет-з’єднання для ефективної роботи;

2.4.5) продуктивність системи залежить від серверних ресурсів та складності NLP-моделей;

2.4.6) частина функціоналу (керування моделями, перегляд журналів) передбачена лише для користувачів із відповідними ролями.

3) ***Функції системи:***

3.1) функція автоматичного підвантаження текстів (посилання, файли):

3.1.1) опис функції: забезпечує завантаження текстового матеріалу із різних джерел: файли (TXT, DOCX, PDF), URL-адреси та пряме введення тексту в UI; попередня валідація і підготовка до подальшої обробки;

3.1.2) вхідна і вихідна інформація: (1) файл (TXT/DOCX/PDF/HTML), URL чи текст із відповідного поля; додаткові параметри (вказане кодування, мова); (2) документ збережений у тимчасовому або основному сховищі з метаданими (назва, мова, розмір, кількість токенів) і попереднім переглядом;

3.1.3) функціональні вимоги: користувачі повинні мати змогу:

– завантажувати документи форматів TXT, DOCX, PDF, HTML;

– працювати із автоматично визначеним кодуванням (UTF-8, Windows-1251 тощо);

– без перешкод використовувати тексти, очищені від HTML-тегів та вилучену інформацію зі структури документа;

– бачити у попередньому перегляді потенційні проблеми із підсвічуванням (непрочитані символи);

– відслідковувати операції імпорту з позначкою часу в системі.

3.1.4) нефункціональні вимоги:

– парсери повинні працювати коректно для файлів об’ємом до 10 МБ; використовувати часові обмеження і черги для більших файлів, встановлювати пріоритети;

– валідація і парсинг не мають блокувати UI (фонова обробка).

3.2) функція формування корпусу текстів і керування колекціями:

3.2.1) опис функції: дозволяє об’єднувати підвантажені документи в корпуси або підкорпуси зі спільними характеристиками; додавати деталі;

3.2.2) вхідна і вихідна інформація: (1) набір документів, метадані (тематичні теги, автор, джерело), параметри доступу; (2) сформований корпус з індексом, він доступний для аналізу та експорту;

3.2.3) функціональні вимоги: користувачі повинні мати змогу:

– створювати, редагувати або видаляти корпуси;

– керувати корпусами згідно з визначеними правами доступу;

– імпортувати/експортувати корпуси (ZIP із окремими файлами).

3.2.4) нефункціональні вимоги: система повинна підтримувати масштабованість зберігання (пакетна обробка для великих корпусів).

3.3) функція автоматичного контролю правопису та генерації підказок:

3.3.1) опис функції: виявляє орфографічні та прості граматичні помилки, пропонує виправлення з поясненням правила мови; перевірка орфографії через регулярні вирази;

3.3.2) вхідна і вихідна інформація: (1) нормалізований та проаналізований текст (POS/NER); (2) набір знахідок – тип помилки, позиція, запропоноване виправлення, пояснення, рівень упевненості, візуальне відображення у вебредакторі;

3.3.3) функціональні вимоги: користувачі повинні мати:

- змогу створювати словники і власні правила;
- можливість додавати локальні випадки слів у словник;
- змогу звертатись до журналу дій і застосованих виправлень.

3.3.4) нефункціональні вимоги:

- швидкість відгуку UI при локальних діях < 300 мс;
- наявність опції використання зовнішнього сервісу, компонента (LanguageTool) або локальної логіки.

3.4) функція аналітики, візуалізації та експорту результатів:

3.4.1) опис функції: формує інтерактивні звіти та графіки (розподіли помилок, динаміка виправлень, хмари слів), експортує їх у PDF/CSV/JSON;

3.4.2) вхідна і вихідна інформація: (1) результати аналізів, метрики моделей; (2) інтерактивні схеми, діаграми, PDF- та CSV-звіт.

3.4.3) функціональні вимоги: користувачі повинні мати змогу:

- бачити інтерактивні візуалізації з фільтруванням й експортом;
- генерувати звіти одноразово або періодично (cron).

3.4.4) нефункціональні вимоги: система має використовувати для побудови графіків бібліотеки для візуалізації (Plotly) і забезпечити швидке відтворення для великих обсягів.

3.5) функція визначення іменованих сутностей (NER):

3.5.1) опис функції: виділяє і класифікує іменовані сутності (особи, організації, локації, дати тощо);

3.5.2) вхідна і вихідна інформація: (1) токенізований або маркований текст; (2) список сутностей із типом та релевантністю;

3.5.3) функціональні вимоги: користувач має змогу:

- працювати із розпізнанням базових типів сутностей;
- експортувати результати у форматі CSV.

3.5.4) нефункціональні вимоги:

- високий рівень точності залежно від наявних підходящих моделей;
- можливість заміни моделі без простою сервісу.

4) ***Вимоги до технічного забезпечення:*** відповідність клієнтського пристрою мінімальним системним вимогам сучасного браузера (остання версія), увімкнений JavaScript і стабільне інтернет-з'єднання.

5) ***Вимоги до програмного забезпечення:***

5.1) архітектура програмної системи: клієнтська частина (frontend), серверний API (backend) та шар збереження або індексації (СКБД і пошуковий рушій);

5.2) системне програмне забезпечення: Nginx 1.24 (reverse-proxy, static), Node.js 18 LTS, фреймворк FastAPI; клієнтська частина – SPA-застосунок; PostgreSQL; контекстний пошук – Elasticsearch; задачі NLP – бібліотеки spaCy, Hugging Face Transformers і tokenizers; Plotly (візуалізація результатів аналізу) та опціонально – інтеграція LanguageTool (перевірка правопису);

5.3) програмне забезпечення ведення інформаційної бази: операції з даними – через API програмної реалізації (REST/HTTP); PostgreSQL 15 та міграції; швидкий контекстний пошук; резервне копіювання БД і збереження артефактів моделей у файловому або об'єктному сховищі;

5.4) мова і технологія розробки програмного забезпечення: Python 3.12, Laravel 13.11, React 19, Bootstrap 5.3, CSS 3.0.

6) ***Вимоги до зовнішніх інтерфейсів:***

6.1) інтерфейс користувача: інтуїтивно зрозумілий, адаптивний, доступний (прийнятне масштабування шрифту), єдина локалізація; чіткі підказки для прийняття рішень; інтерактивні блоки аналітики (фільтри, сортування);

6.2) апаратний інтерфейс: будь-який пристрій (стандартний ПК, ноутбук, мобільний пристрій), який може робити вихід в мережу Інтернет;

6.3) програмний інтерфейс: REST API для всіх основних функцій (імпорт, запуск аналізу, отримання результатів, керування моделями та користувачами); використання вебсокетів для оновлення статусу обробки в режимі реального часу.

7) ***Властивості програмного забезпечення:***

7.1) доступність: цілодобово за виключенням технічних робіт; програмна реалізація та сервіси розгортаються за допомогою контейнерів, що дозволяє виконувати rolling-оновлення без суттєвих простоїв;

7.2) супроводжуваність: модульна структура коду, читабельність для інших розробників; CI/CD-процеси для автоматичного тестування та розгортання (у конвеєрі GitHub Actions/GitLab CI); Docker для контейнеризації та уніфікації середовища;

7.3) переносимість: система має бути портативною – розгортання на іншому сервері або хмарі без змін у коді забезпечується завдяки контейнерам та декларативним конфігураціям; немає залежності від конкретної ОС сервера;

7.4) продуктивність: ПЗ повинно забезпечувати стабільну та швидку роботу при нормальному навантаженні.

Основні вимоги до продуктивності:

– час відповіді для інтерактивних дій (локальні операції UI) ≤ 300 мс;

– час базової перевірки тексту (орієнтовно до 10 000 символів) в межах кількох секунд, залежно від підключених моделей;

– пакетна обробка великих корпусів виконується у фоновому режимі з чергами та масштабуванням сервісу аналізу;

– індексація для пошуку реалізована через Elasticsearch.

7.5) надійність: забезпечується резервними копіями БД і артефактів моделей, моніторингом стану сервісів, логуванням помилок і автоматичними оповіщеннями про відмови; можливість відкату до попередньої стабільної версії моделі або правила; відновлення працездатності після збою ≤ 5 хв;

7.6) безпека на рівні інформації:

– розмежування ролей і прав доступу (гість, дослідник, редактор, адміністратор);

– захист API (HTTPS, авторизація JWT/OAuth2);

– захист від типових загроз (перевірка вхідних файлів, захист від XSS/CSRF/SQL-ін’єкцій, обмеження розміру завантажень);

– опціональне шифрування збережених документів та політика видалення даних за запитом (GDPR/локальні вимоги);

– можливість інтеграції зовнішнього інструмента для перевірки граматики (LanguageTool) як компонента із чіткою політикою передачі даних.

7.7) інші вимоги: GP відповідає чинним стандартам та рекомендаціям щодо розробки вебзастосунків, підтримується масштабоване розширення функціоналу (мікросервіси, плагіни).

Висновки до розділу 2

У даному розділі описано процес моделювання об’єкта та предмета роботи. Під час розгляду матеріалу проведено аналіз сучасного інструментарію та методів розробки вебзастосунків для лінгвістичного аналізу тексту. Як наслідок, визначено, що для реалізації такої системи необхідне поєднання кількох технологічних рівнів.

У межах розділу обґрунтовано доцільність вибору технологічного стеку, до складу якого входять React, Laravel, FastAPI, spaCy, Stanza, Hugging Face Transformers, PyTorch, PostgreSQL, OpenSearch, Docker, Plotly, matplotlib і pandas. Без цієї сукупності інструментів неможлива реалізація повного циклу обробки тексту. Завдяки такому підходу система зможе виконувати як прикладні завдання керування даними, так і складні процеси із опрацювання.

Також у розділі сформовано специфікацію вимог до програмного забезпечення. У ній визначено основні функції системи, обмеження та вимоги до технічного, програмного й зовнішнього забезпечення. Отже, отримані результати створюють основу для подальшого проєктування та практичної реалізації вебзастосунку «LinguaInsight».

3 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ЛІНГВІСТИЧНОГО АНАЛІЗУ ТЕКСТУ

3.1 Розробка UML-діаграм

Для майбутньої системи UML-діаграми слугують засобом наочного подання структури та логіки роботи. Це своєрідний спосіб узгодити розуміння основних функцій вебзастосунку, ролей користувачів та взаємодії між модулями. За допомогою мови графічного моделювання UML можна формально описати сценарії функціонування та простежити характер зв’язків. Крім того, такі діаграми допомагають поступово деталізувати систему із урахуванням усіх рівнів: від загальної архітектури клієнт-серверної організації до окремих компонентів аналітичного конвеєра. Завдяки цьому модель системи залишається цілісною на різних етапах проєктування.

Серед UML-діаграм, запланованих для «LinguaInsight», виділено наступні:

1) діаграма прецедентів (Use Case) – використовується з метою визначення основних сценаріїв та акторів, по 3-5 сценаріїв для кожного із них;

2) діаграма класів – для представлення статичної структури доменних сутностей і методів; охоплює більше 10 класів (User, Researcher, Editor, Administrator, Document, Chunk, Model, ModelManager і ряд сервісів);

3) діаграми взаємодії (Sequence) – з метою відобразити послідовність надходження й обробки повідомлень між об’єктами у сценаріях; у системі демонструють процеси завантаження документа, запуску аналізу тексту, перевірки правопису та стилю та побудови *n*-грам;

4) діаграми станів та переходів (Statechart) – використовуються для моделювання життєвих циклів (наприклад, загального функціонування системи як глобального процесу та одного із детальних підпроцесів);

5) діаграми діяльності (Activity) – з метою відображення алгоритмічних потоків операцій (аналізу тексту, побудови *n*-грам, прийняття або відхилення пропозицій виправлень);

6) діаграма компонентів – використовується для представлення логічних модулів та залежностей (MainComponent, Presentation, Services, Models, DBAccess);

7) діаграма пакетів – у системі представлена з метою розподілу класів на логічні модулі (WebApplication, Models, DataAccess);

8) діаграма розгортання (Deployment) або впровадження – для демонстрації фізичного розміщення вузлів і артефактів, сюди входять робоча станція користувача, вебсервер, блок обчислювально інтенсивних операцій, сервер бази даних.

Підсумовуючи інформацію про наведений вище набір діаграм, можна з упевненістю сказати, що вони охоплюють усі рівні абстракції – як сценарії використання, так і фізичне розгортання системи. Усе це потребує послідовного відтворення: спочатку визначаються актори та основні прецеденти, далі послідовності дій, структура класів і логіка роботи модулів. Після цього описуються стани ключових об’єктів, компоненти системи та особливості її середовища виконання. Визначений підхід забезпечує цілісне подання та спрощує подальше проєктування, реалізацію й супровід системи. Для створення діаграм використано застосунок StarUML та онлайн-платформу Visual Paradigm.

Діаграма прецедентів (варіантів) використання

У вебзастосунку лінгвістичного аналізу тексту передбачено взаємодію чотирьох основних типів користувачів: Гість, Дослідник, Редактор та Адміністратор. Для кожної із цих ролей визначено свої специфічні варіанти використання, що відображають їхні можливості в системі. Ключовим елементом діаграми є варіант використання «Авторизація», який є спільним для всіх акторів, окрім Гостя, і пов’язаний із ними через асоціацію (рис. 3.1). На зображенні показано, що спільним прецедентом для Гостя і Дослідника є «Завантаження документа», а обидва Дослідник і Редактор можуть виконувати «Перевірку правопису та стилю».

Гість має обмежені можливості, зокрема перегляд публічних корпусів і запуск демо-перевірки. Дослідник вже може працювати з документами та корпусами, тобто виконувати повний цикл обробки. Редактор додатково рецензує пропозиції виправлень (приймає або відхиляє їх) та анотує приклади помилок. Адміністратор має розширені права, що охоплюють керування користувачами та моделями.

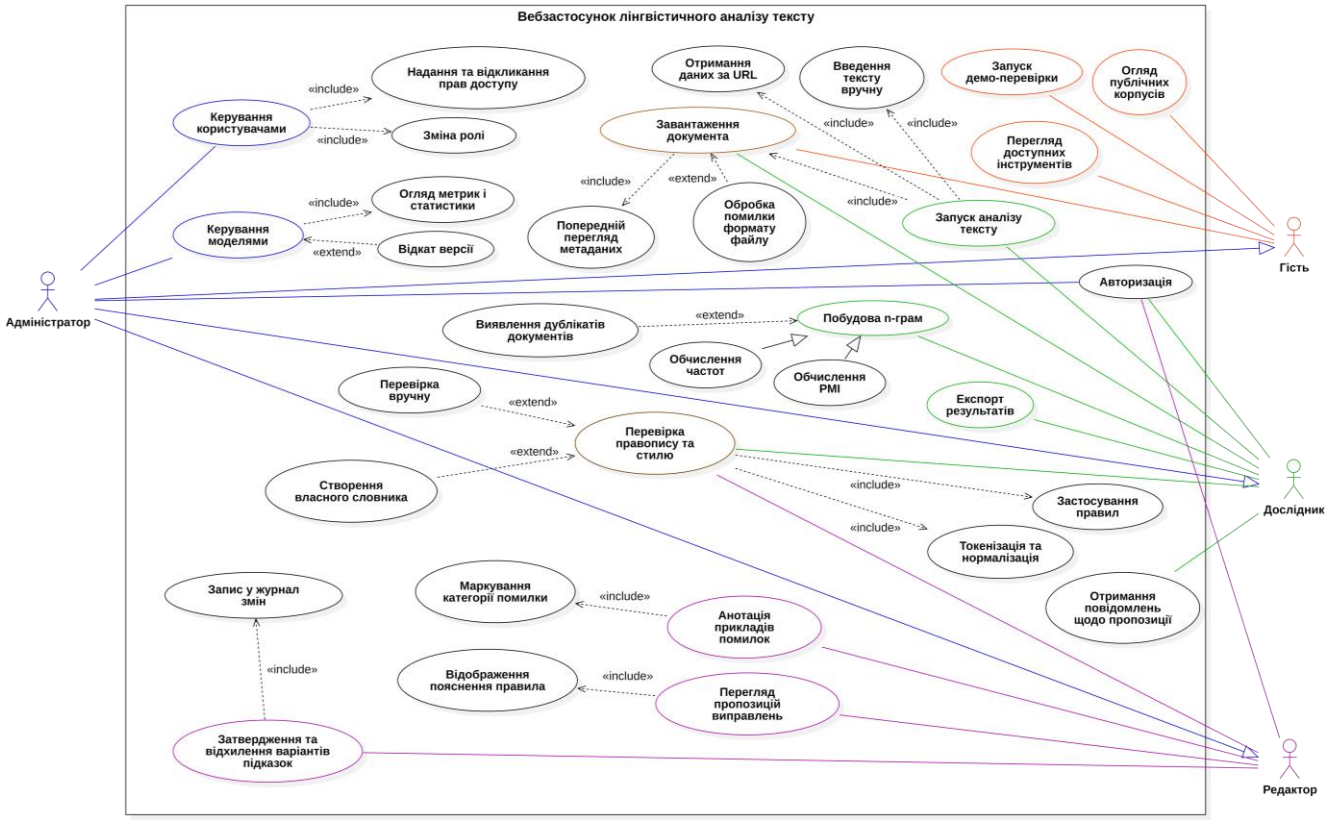

Рисунок 3.1 – Діаграма варіантів використання

Варіанти використання на діаграмі пов’язані зв’язками include, extend та generalization. Зв’язок include демонструє перегляд метаданих під час завантаження документа, токенізацію й нормалізацію під час перевірки правопису та стилю. Через цей зв’язок також реалізовано запис у журнал змін під час затвердження або відхилення підказок. Зв’язок extend використовується для таких варіантів, як: обробка помилки формату файлу, виявлення дублікатів документа та ручна перевірка результатів. Крім того, через extend представлено варіанти створення особистого словника та відкат версій моделі. Зв’язок generalization відображає розширення функціоналу усіх користувачів системи Адміністратором.

Діаграма класів

Зображена на рис. 3.2 діаграма класів відображає статичну структуру вебзастосунку. На ній представлені класи, їхні атрибути, методи та зв’язки між ними. Даний тип діаграм широко використовується в об’єктно-орієнтованому програмуванні та є основою для загального і детального проєктування структури системи. Також за їх допомогою відбувається інтерпретація моделей у програмний код. Розподіл класів на групи наведений у табл. 3.1.

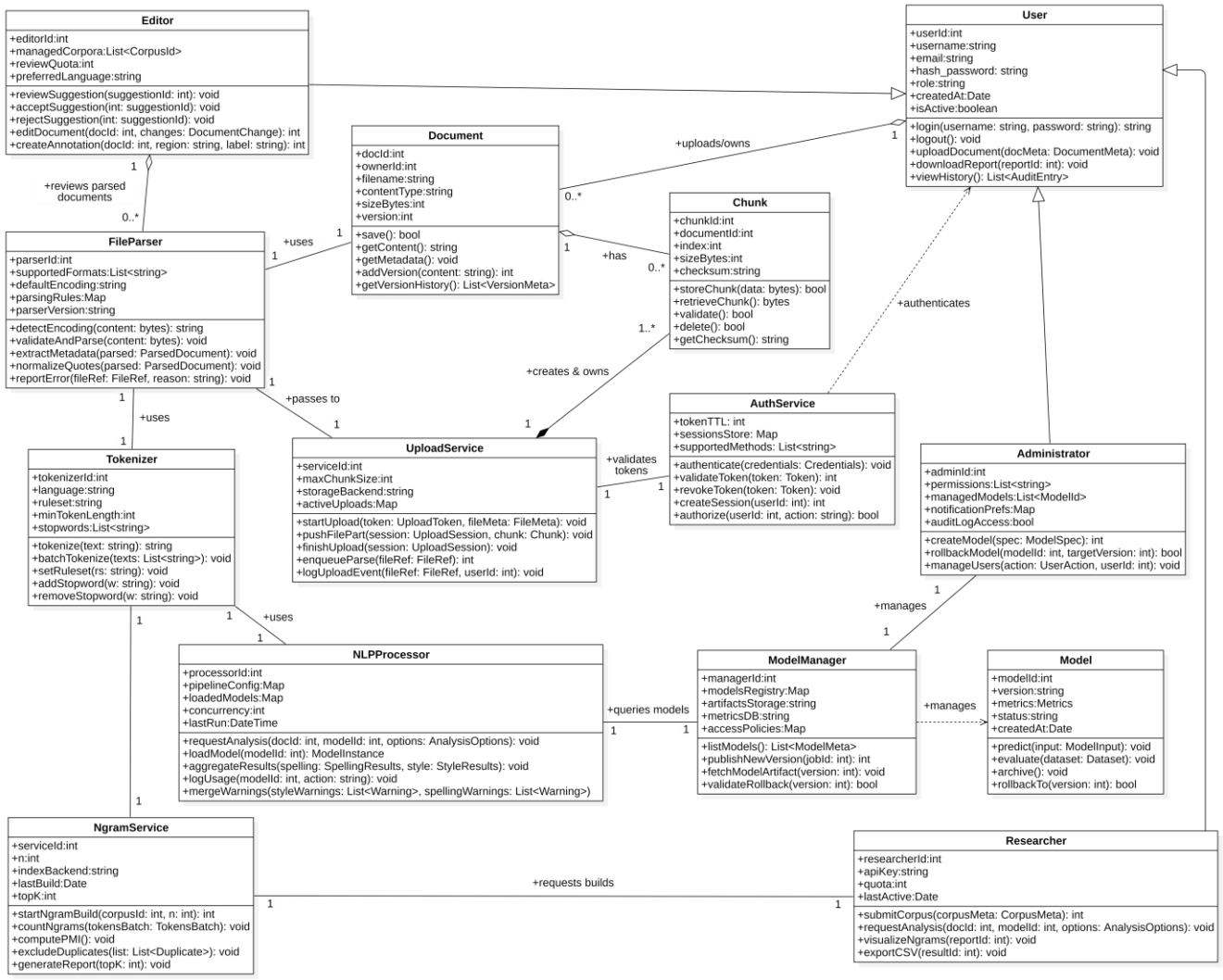

Рисунок 3.2 – Діаграма класів

Додатково у системі реалізовано перевірку орфографії (SpellChecker) і стилістики (StyleChecker), але їх не включено до діаграми класів, оскільки це не ключові компоненти архітектури.

Між класами використано такі типи зв’язків:

– наслідування (—▷) між User та ролями Researcher, Editor, Administrator;

– асоціація (—>) між сервісами, які взаємодіють під час виконання запитів;

– агрегація (◇—) між User і Document, оскільки користувач може завантажувати та мати кілька документів;

– композиція (◆—) представлена між UploadService і Chunk, оскільки частини файлу створюються й обробляються в межах процесу завантаження;

– залежність (-->) між ModelManager і Model, оскільки менеджер керує моделями, їхніми версіями та метриками.

Таблиця 3.1 – Групи класів та їх характеристика

Група	Класи, які віднесено	Призначення
Користувачі та ролі	User, Researcher, Editor, Administrator	Описують базового користувача системи та спеціалізовані ролі з різними правами доступу
Автентифікація та доступ	AuthService	Забезпечує вхід у систему, надання токенів, перевірку прав користувачів та керування сесіями
Документи та завантаження	Document, Chunk, UploadService, FileParser	Відповідають за збереження документів, поділ файлів на частини, завантаження, парсинг і витяг метаданих
Попередня обробка тексту	Tokenizer	Виконує токенізацію тексту, налаштування мови, правил орфографії
Аналітична обробка	NLPProcessor, NgramService	Запуск аналізу документа, агрегація результатів, побудова послідовностей, обчислення частот і PMI
Керування моделями	ModelManager, Model	Відповідають за список моделей, їхні версії, метрики, публікацію нових версій і відкат

Побудова діаграм послідовності

Під час проведення аналізу та моделювання системи важливим є розуміння точок контакту користувача будь-якого рівня із вебзастосунком. Тут у нагоді стають діаграми послідовності (Sequence diagrams), які відображають обмін повідомленнями між компонентами в межах окремого сценарію. В контексті системи «LinguaInsight», вагомим є шлях, який користувацький запит проходить через клієнтську частину; сервіси автентифікації, завантаження, парсингу файлів; базу даних та різні модулі. Завдяки хронологічному поданню розробник може визначити, які компоненти ініціюють виклики; дані, що передаються між ними та послідовність операцій. Не менш важливою постає потреба в ідентифікації етапу, який призводить до можливого виникнення помилок. Завдяки цьому ще під час проєктування уникають зайвих залежностей між сервісами та передбачають механізми обробки нестандартних ситуацій.

Діаграма № 1 (рис. 3.3) відповідає за безпечне та надійне завантаження файлу в систему. На ній показано основні компоненти цього процесу: вебклієнт, авторизація (тимчасовий токен) та завантаження, FileParser (передача файлу, визначення кодування, парсинг вмісту), збереження метаданих у БД та ErrorHandler.

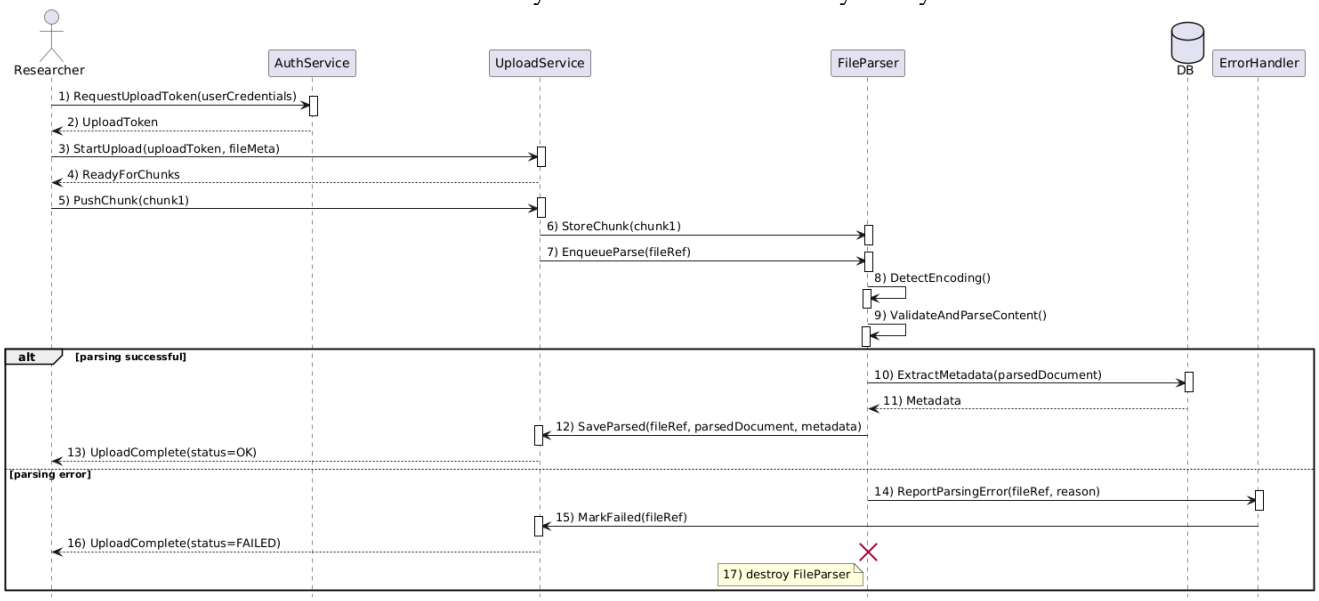

Рисунок 3.3 – Діаграма послідовності завантаження документа

Наступна діаграма № 2 (рис. 3.4) відображає процес виконання лінгвістичного аналізу тексту. Діаграма демонструє взаємодію Дослідника із системою під час запуску обробки документа. Вона показує послідовність викликів між вебклієнтом, NLP-процесором (запит на аналіз), менеджером (завантаження мовної моделі), стилем і правописом. Отримані результати об’єднуються в єдиний звіт. Діаграма також показує логування використання моделі для подальшого моніторингу роботи аналітичного модуля.

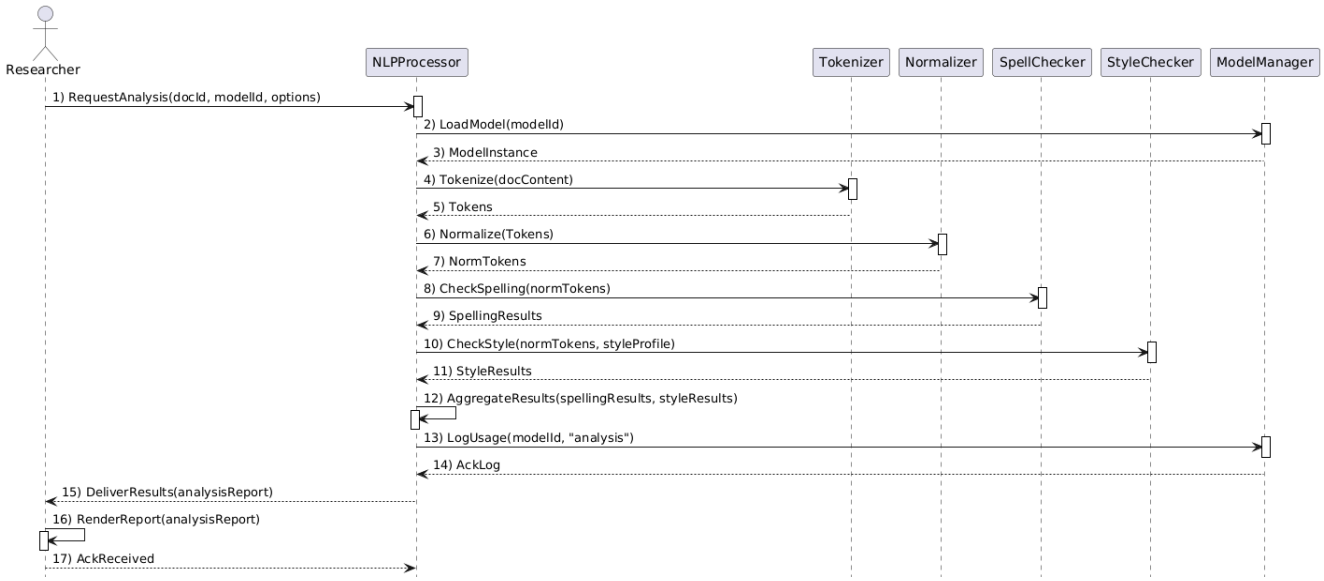

Рисунок 3.4 – Діаграма послідовності запуску аналізу тексту

Діаграма № 3 (рис. 3.5), де головними акторами виступають Гість і Дослідник, відповідає за демонстрацію процесу перевірки правопису та стилю в документі. Діаграма показує послідовність викликів, пов’язаних із автентифікацією, передачею документа на перевірку, визначенням токенів, аналізом правопису й стилю. Окремо на діаграмі відображено сценарій ручної верифікації результатів, який виникає при виявленні зауважень із низьким рівнем упевненості.

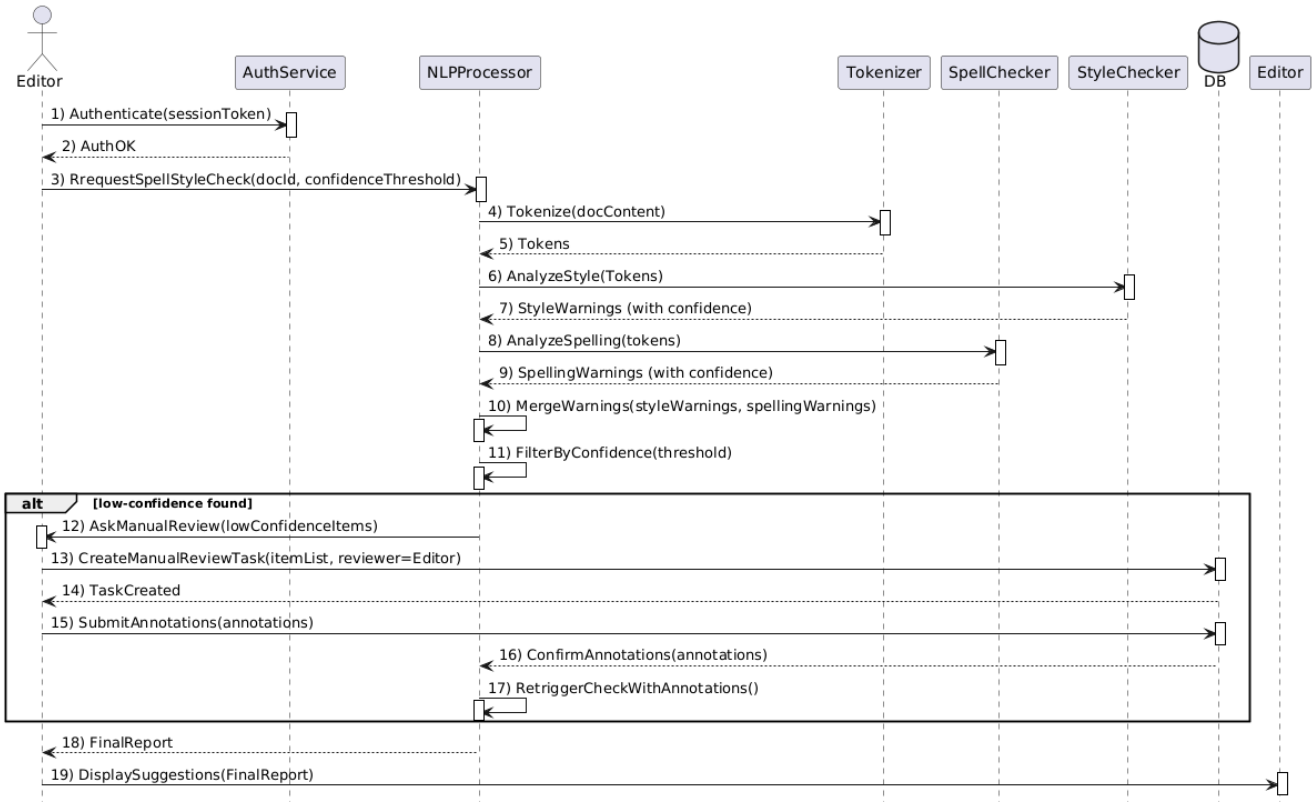

Рисунок 3.5 – Діаграма послідовності перевірки правопису та стилю

Ще одна діаграма послідовності № 4 (рис. 3.6) відповідає за демонстрацію дій Дослідника під час запиту на побудову *n*-грам у вибраному корпусі текстів. На ній показано кроки, пов’язані з отриманням текстів із набору, пакетним визначенням токенів у реченнях, підрахунком знайдених комбінацій, обчисленням частот і показників PMI. Також на діаграмі передбачено обробку ситуації з виявленням дублікатів у корпусі. Такий варіант розвитку подій являє собою виключення подібних входжень із подальшого аналізу та повторне виконання підрахунків. Мета цього процесу – більша точність у випадку їх трактування при корпусному розборі.

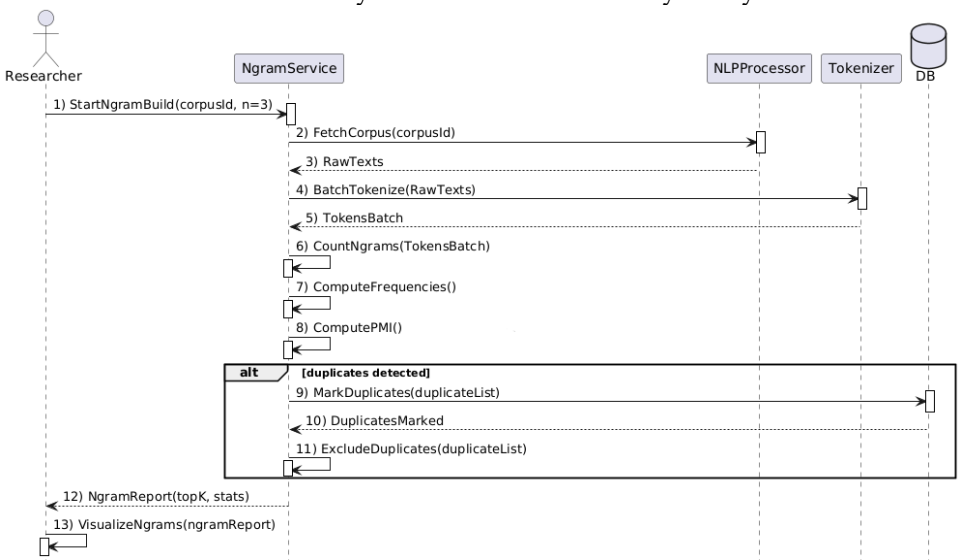

Рисунок 3.6 – Діаграма послідовності побудови *n*-грам

Діаграми станів та переходів

Statechart diagrams призначені для демонстрації певних сценаріїв, які будуть виконуватись під час функціонування системи [30]. З огляду на це, для вебзастосунку лінгвістичного аналізу тексту розроблено 2 такі діаграми:

1) загальна для всього застосунку (рис. 3.7), яка демонструє можливі дії користувача та поводження програмного забезпечення у відповідь на них;

2) для відтворення життєвого циклу документа в системі (рис. 3.8).

Ключова характеристика діаграм станів – це те, що у випадку перебування об’єкта чи системи, яка моделюється, в початковому стані, її поведінка може супроводжуватися реалізацією певних внутрішніх дій [31]. Після чого переходи між станами будуть можливими лише у 2-х випадках: після завершення дій, що виконуються, та при виникненні певних зовнішній подій.

На рис. 3.7 показано перехід системи зі стану готовності до прийому вхідних даних та постановку черги. Після цього йдуть кроки виконання аналізу та подальше опрацювання результатів. Зміни стану виконуються за подіями: запуск системи, завантаження файлу, надходження його частин, завершення аналізу чи виникнення помилки. Також передбачено історичний псевдостан Shallow (H) або інший його варіант – Deep (H*), які дають змогу повернутися до попередньої активної конфігурації вкладених станів. Це зручно для збереження контексту роботи під час повторного відкриття результатів аналізу або перегляду деталей попереднього запиту.

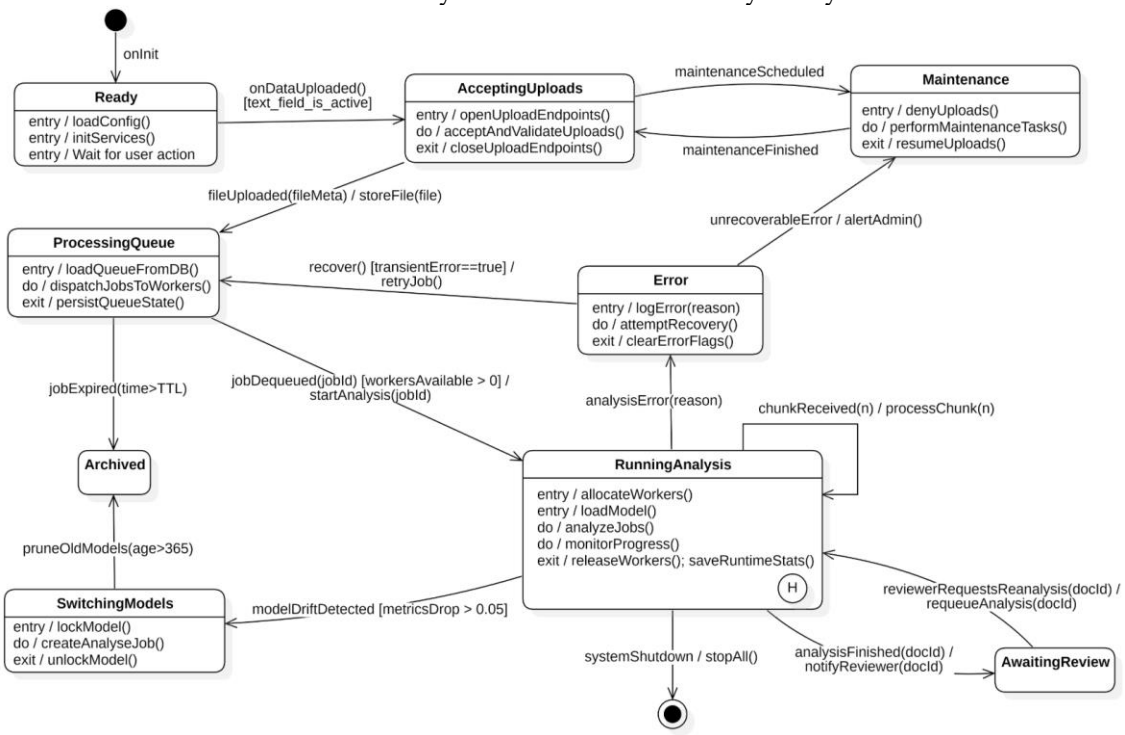

Рисунок 3.7 – Діаграма станів циклу обробки із отриманням звіту

На рис. 3.8 продемонстровано кроки, які проходить документ при потраплянні до вебзастосунку або після формування файлу. У стані Tokenizing передбачено поетапне опрацювання частин тексту, що важливо для роботи з великими обсягами даних. Якщо під час аналізу виявлено результати з низьким рівнем упевненості моделі, документ переходить до стану ManualReview. У разі успішної автоматичної обробки результати публікуються як нова версія документа.

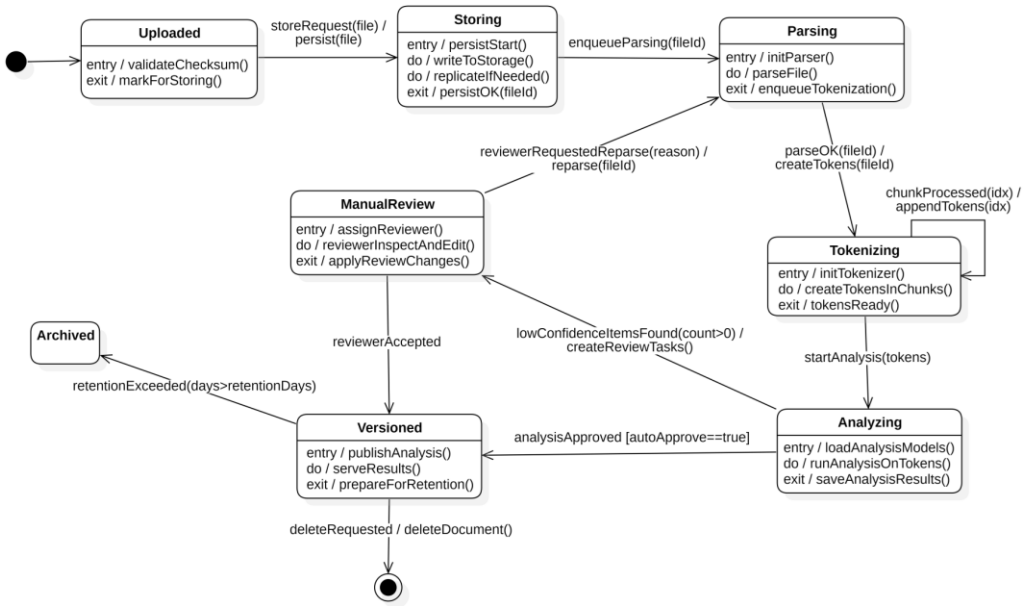

Рисунок 3.8 – Діаграма станів життєвого циклу документа в системі

Діаграма діяльності

Цей тип діаграм показує алгоритмічний потік операцій, необхідний для виконання певного процесу в «LinguaInsight». Вони відображають послідовності, розгалуження умов, паралельні гілки і точки синхронізації, дозволяючи виявити залежності та «вузькі місця» ще на етапі проєктування. За допомогою діаграми на рис. 3.9 відображено процес керування пропозиціями виправлень у 3-х варіантах.

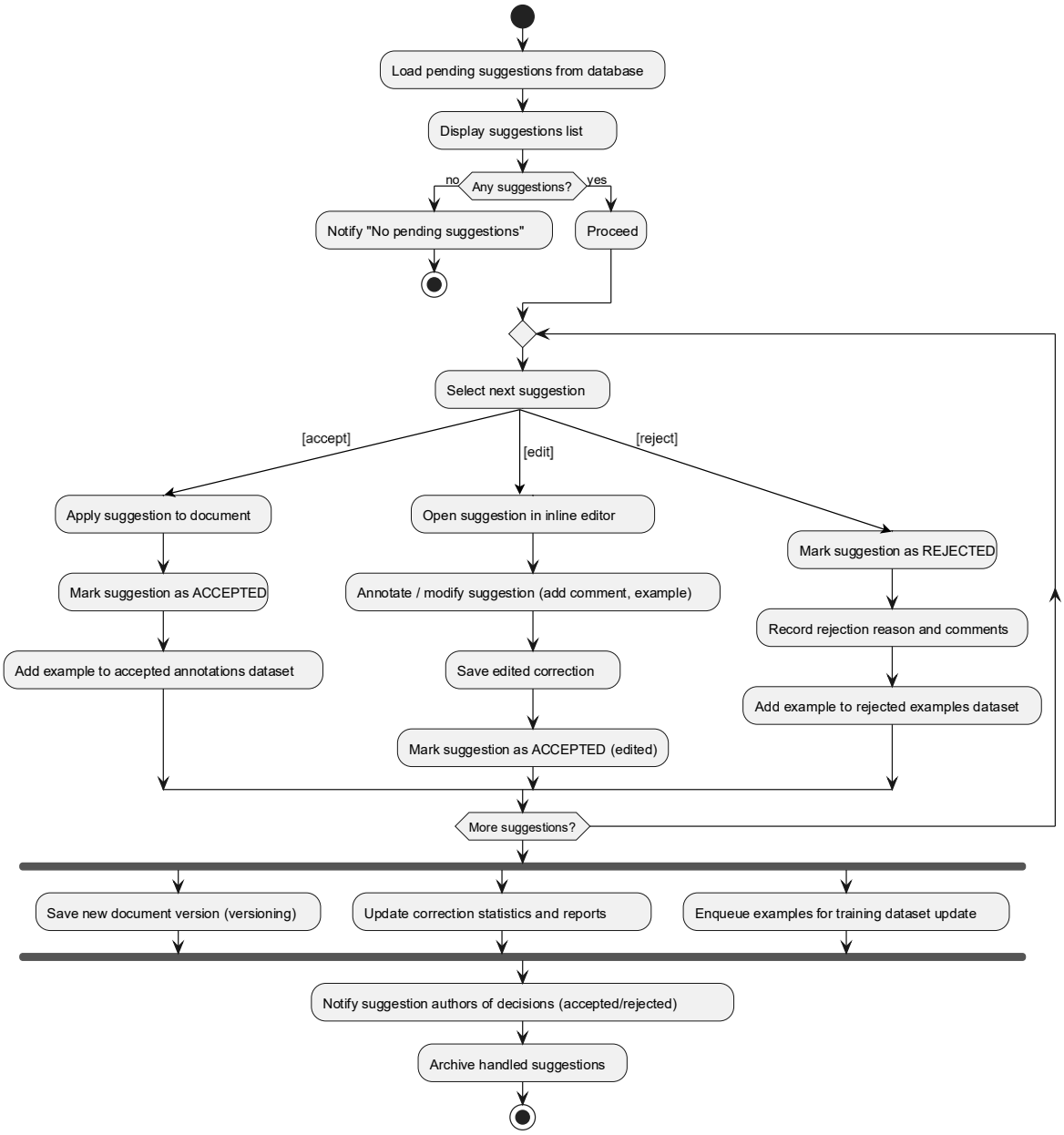

Рисунок 3.9 – Діаграма діяльності для прийняття/відхилення пропозицій виправлень

Дана діаграма демонструє послідовність дій, які виконує Редактор під час перегляду, прийняття або відхилення запропонованих змін. Використання decision-вузла дає змогу відобразити вибір дії над кожною пропозицією, а fork та join –

показати паралельні операції після завершення основного циклу обробки. Таке графічне подання полегшує комунікацію між командою розробки і слугує основою для написання юніт-тестів і сценаріїв автоматизації.

Розробка діаграми компонентів

Діаграма компонентів відображає логічну організацію вебзастосунку лінгвістичного аналізу тексту і показує, з яких основних модулів складається система. Вона дає змогу чітко визначити відповідальність кожного компонента й простежити залежності між ними (рис. 3.10).

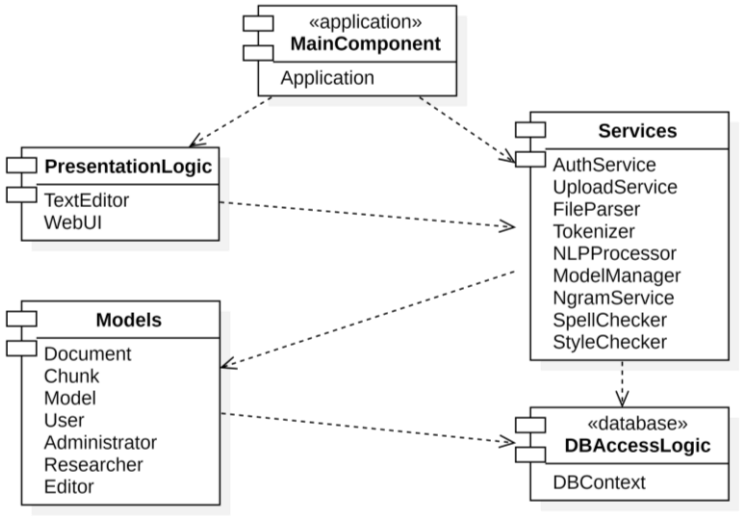

Рисунок 3.10 – Діаграма компонентів

Координуючим елементом системи є компонент MainComponent, який відповідає за запуск застосунку та узгодження роботи головних модулів через клас Application. Далі на рівні користувацького інтерфейсу розміщено компонент PresentationLogic. Він містить блоки WebUI і TextEditor, через які суб’єкт взаємодіє із системою. Основна логіка обробки зосереджена у блоці Services – функціональному ядрі, до якого входять сервіси авторизації, завантаження, аналітики (парсинг, токенізація, NLP-процесор та перевірки) і керування моделями.

Окремо на діаграмі виділено компонент Models, який містить доменні сутності системи: Document і Chunk, що відповідають за формування документа; Model – для аналізу; User, Administrator, Researcher та Editor, що репрезентують ролі. За збереження й отримання даних відповідає компонент DBAccessLogic, у межах якого розміщено DBContext. Цей компонент забезпечує взаємодію сервісів із БД.

Для ширшого розуміння та кращої структуризації блок Services можна розділити на окремі мікросервіси (підгрупи):

– роботи з файлами та документами (file and document processing), до яких входять UploadService, FileParser і Tokenizer;

– лінгвістичного аналізу (NLP core), що включають NLPProcessor, SpellChecker, StyleChecker та NgramService;

– керування моделями (model management), де відповідальний ModelManager;

– доступу та безпеки (authorization), що реалізується завдяки AuthService.

Діаграма пакетів

Створення Package diagram потрібно для групування основних елементів вебзастосунку, що розробляється, за їхнім призначенням. Пакетом у такому розподілі виступає інструмент, який використовується у конструкції UML і об’єднує її елементи в одиниці високого (вищого) рівня [32]. У межах цієї діаграми відображено основні рівні системи, а саме інтерфейс користувача, сервіси обробки, доменні моделі та доступ до даних.

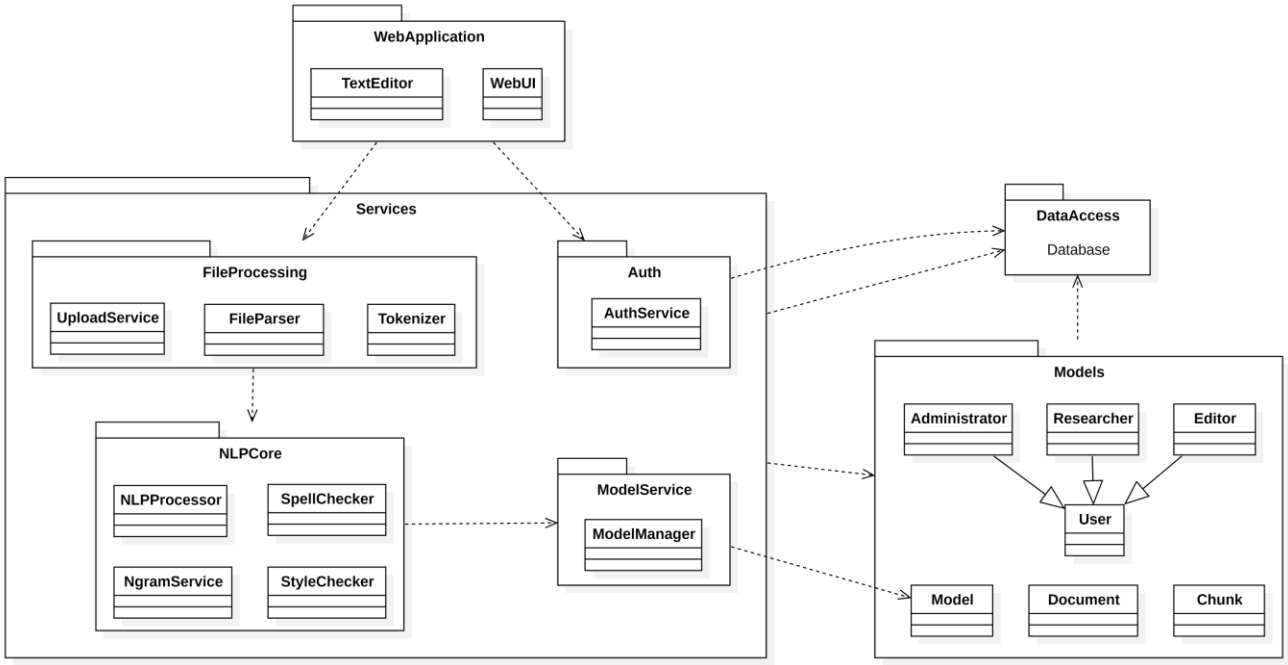

Рисунок 3.11 – Діаграма пакетів

До складу описаної на рис. 3.11 діаграми входять:

– пакет «WebApplication» формує клієнтський рівень системи та об’єднує елементи, через які користувач працює із вебзастосунком;

– пакет «Services» виступає проміжним рівнем між інтерфейсом, моделями та базою даних, забезпечуючи виконання основних операцій обробки;

– пакет «Models» задає структуру доменних об’єктів, які використовуються під час завантаження документів, аналізу тексту та керування ролями;

– пакет «DataAccess» зі стереотипом «database» відповідає за технічну взаємодію зі сховищем даних і забезпечує доступ до збережених сутностей.

Завдяки поділу стає можливим відокремлення інтерфейсної частини від бізнес-логіки, об’єктної моделі та рівня сховища.

Діаграма розгортання

Графічне представлення ПЗ, що розробляється, вимагає створення діаграми розгортання (рис. 3.12). Цей тип діаграм призначений для опису фізичного розміщення компонентів «LinguaInsight» у робочому середовищі.

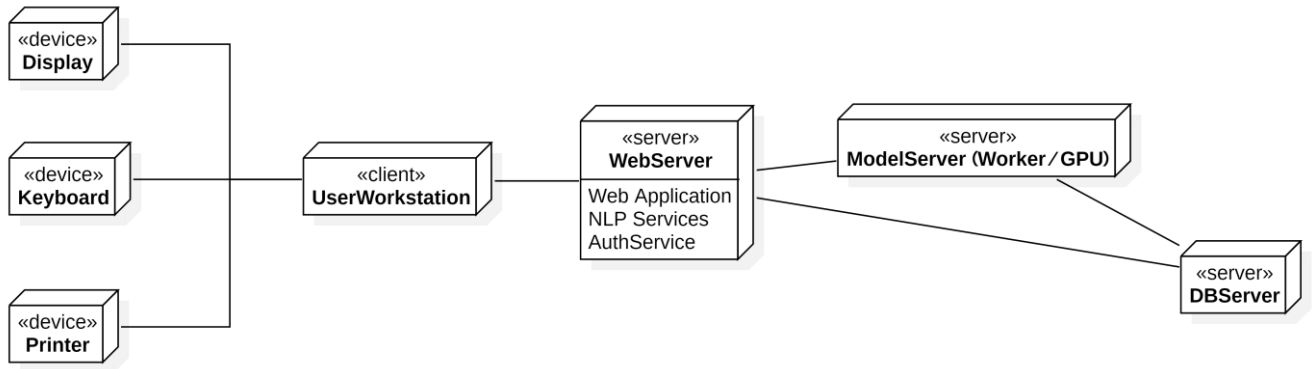

Рисунок 3.12 – Діаграма розгортання вебзастосунку лінгвістичного аналізу тексту

Архітектура системи має чотиришарову структуру (табл. 3.2). На боці користувача розміщується робоча станція, через яку відбувається взаємодія із вебінтерфейсом. До неї належать пристрої введення й виведення, зокрема клавіатура, дисплей і принтер. Виділення окремого сервера моделей для ресурсоємних операцій дає змогу не перевантажувати вебсервер і масштабувати аналітичні обчислення незалежно від клієнтської частини.

Таблиця 3.2 – Архітектура програмного комплексу

Шар	Атрибути, компоненти	Опис
1	Client – UserWorkstation. Devices: Display, Keyboard, Printer.	Робоча станція користувача, яка містить пристрої введення та виведення.

Кінець таблиці 3.2

2	Server – WebServer.	Вебсервер, який відповідає за виконання основної бізнес-логіки вебзастосунку.
3	Server – ModelServer.	Окремий сервер для ресурсоємних NLP-обчислень і роботи з мовними моделями.
4	Server – DBServer.	Сервер баз даних для збереження інформації про документи, користувачів, моделі, результати аналізу.

На діаграмі, зображеній на рис. 3.12, і WebServer, і ModelServer мають канали зв’язку із DBServer. Це пояснюється тим, що перший із них зберігає та читає інформацію про користувачів і результати, а другий може завантажувати або зберігати навчальні набори, додані моделі і записувати дані для статистики. Зв’язок між клієнтом і вебсервером доцільно організувати через захищений протокол HTTPS.

3.2 Моделювання інформаційних потоків під час обробки тексту

Після побудови діаграм взаємодії доцільно окремо розглянути інформаційні потоки, які виникають під час роботи вебзастосунку лінгвістичного аналізу тексту. Це робиться з метою продемонструвати, які саме дані передаються, модифікуються, зберігаються та повертаються користувачу на кожному етапі. Документи, токени, деякі статистичні показники та сформовані звіти у межах системи «LinguaInsight» є основними об’єктами обміну.

Для уникнення перевантаження процес моделювання інформаційних потоків поділено на кілька окремих блоків. Це дає змогу простежити рух даних від моменту надходження документа в систему до отримання готових результатів у вигляді звіту, таблиць або візуалізацій.

Перший інформаційний потік пов’язаний із завантаженням документа. На цьому етапі користувач передає файл або посилання, а система формує службові дані для подальшої обробки. До таких даних, що формуються у процесі, входять:

- uploadToken відповідає за тимчасове підтвердження права на завантаження;
- uploadId являє собою ідентифікатор сеансу завантаження;
- fileMeta включає назву, розмір, формат і MIME-тип файлу;
- chunks представляє частини документа для поетапного передавання;

- parsedContent містить розпізнаний текст після парсингу;
- metadata включає службові відомості про документ;
- uploadStatus представляє результат операції (успішно або з помилкою).

Після прийому даних необхідно перевірити формат, визначити кодування, виконати парсинг вмісту та зберегти інформацію в базі даних. У разі помилки система не переходить до наступних етапів, а фіксує причину збою, очищує тимчасові ресурси та позначає завантаження як невдале.

Другий потік описує перехід від збереженого документа безпосередньо до лінгвістичного аналізу. Після вибору документа користувач задає параметри обробки: мову, профіль аналізу тощо. Система завантажує відповідну мовну модель і виконує поетапну обробку. Результатом є узагальнений за кожним модулем звіт єдиної структури для відображення в інтерфейсі. У процесі аналізу тексту відбувається послідовне перетворення даних (рис. 3.13).

Рисунок 3.13 – Послідовність перетворення даних

Окрему роль відіграє інформаційний потік перевірки правопису та стилю. Він орієнтований на роботу редактора, задача якого отримати автоматичні підказки та оцінити їхню надійність. Система зіставляє рівень упевненості кожної рекомендації з установленим порогом. Підказки з достатнім рівнем упевненості можуть бути одразу відображені користувачу, а сумнівні результати передаються на ручну перевірку для уточнення. В останньому випадку перевірка може бути проведена повторно.

Наведена логіка взаємодії демонструє, що частина інформаційного потоку має зворотний характер: редактор отримує результати від системи, а також повертає більш якісні дані у вигляді анотацій. Надалі такі відомості можуть використовуватися для покращення правил, словників або мовних моделей.

Четвертий інформаційний потік пов’язаний із побудовою *n*-грам. На цьому етапі система працює вже з корпусом текстів. Вхідними даними є ідентифікатор корпусу, параметр *n*, список текстів, налаштування фільтрації тощо. Після обробки система формує набір комбінацій, обчислює їхні частоти і додаткові статистичні

показники. Якщо в корпусі виявлено дублікати, вони маркуються і виключаються з подальших підрахунків, щоб уникнути спотворення результатів.

Фінальні результати побудови можуть подаватися у кількох формах:

- таблиця найчастотніших *n*-грам;
- значення абсолютної та відносної частоти;
- показники PMI;
- список виключених дублікатів;
- підготовлені дані для графіків і візуалізацій.

Змодельовані інформаційні потоки показують, як у системі «LinguaInsight» текстові дані проходять повний шлях опрацювання. Кожен потік фіксує окремий фрагмент цього процесу, а разом вони демонструють цілісний механізм роботи вебзастосунку.

3.3 Написання варіантів використання системи (usecases)

Варіанти використання визначають функції високого рівня та описують взаємодію зовнішніх акторів із вебзастосунком через прецеденти. Вони дають змогу продемонструвати поведінку системи, не заглиблюючись у технічні деталі реалізації. Такий підхід дозволяє сформувати загальне уявлення про функціональні можливості майбутнього програмного забезпечення.

У контексті вебзастосунку, що розробляється, гість, дослідник, редактор та адміністратор є основними акторами. Вони, відповідно до своїх прав, запускають сценарії на зразок перегляду публічних матеріалів, автоматичного підвантаження тексту (табл. 3.3), проведення аналізу, керування користувачами (табл. 3.4) чи побудови *n*-грам. Після побудови загальної діаграми варіантів використання (див. рис. 3.1) виникає потреба в деталізації кожного важливого сценарію у вигляді usecase.

Таблиця 3.3 – Автоматичне підвантаження тексту

Usecase section	**Comment**
Use Case Name	Автоматичне підвантаження тексту
Scope	Вебзастосунок лінгвістичного аналізу тексту
Level	Імпорт та попередня обробка документів для подальшого аналізу

Продовження таблиці 3.3

Primary Actor	Дослідник
Stakeholders and interests	1. Користувач: швидке й коректне підвантаження документів, точне визначення формату чи кодування, зручний попередній перегляд; контроль збереження чи видалення імпортованих даних; 2. Адміністратор: підтримка різних парсерів, безпека завантажень; 3. Освітня установа/організація: можливість масового імпорту матеріалів для навчання та досліджень.
Preconditions	1. Наявність доступу до інтерфейсу імпорту (авторизований або гостьовий режим згідно з політикою); 2. Файл доступний локально або URL коректний, він відповідає серверу; 3. Парсер формату та модулі для декодування доступні на сервері; 4. Встановлена конфігурація обробки (максимальний розмір файлу, дозволені типи).
Success guarantee	1. Текст успішно підвантажено, Unicode-кодування правильне; 2. Система визначила формат і мову (або запропонувала вибір); 3. Текст розділений на токени і відображено попередній перегляд з метаданими (джерело, мова, кількість токенів).
Main Success Scenario	1. Користувач відкриває інтерфейс «Імпорт/Завантажити» й обирає файл або вводить URL; 2. Система приймає вхідні дані та верифікує тип файлу/доступність URL; 3. Виконується перевірка кодування; у разі невизначеного кодування – пробні варіанти (UTF-8, Windows-1251 тощо); 4. Система очищає HTML-теги, метадані, нормалізує Unicode; 5. Проводиться визначення мови та попередня класифікація вмісту (текстовий, сканований чи бінарний файл); 6. Якщо текст придатний для обробки, він зберігається в системі; 7. Користувач вносить зміни або підтверджує імпорт; 8. У кінці текст зберігається у тимчасовому сховищі або в базі даних (залежно від політики) та стає доступним для подальшого аналізу.

Кінець таблиці 3.3

Extensions	1. Некоректний формат або файл пошкоджений – система повідомляє користувача і пропонує завантажити інший або ввести текст вручну; 2. Багатомовний або неоднозначний вміст – виникає пропозиція розбити на піддокументи або вказати мову вручну; 3. Сканований документ (зображення/PDF) – система інформує про необхідність додаткового кроку; 4. Розмір файлу перевищує ліміт – пропозиція завантажити частинами; 5. Документ марковано як конфіденційний – не зберігати або зберігати у зашифрованому вигляді відповідно до налаштувань.
Special Requirements	1. Підтримка основних форматів: TXT, DOCX, PDF, HTML; 2. Надійні парсери для визначення кодування; 3. Політика збереження та видалення даних і шифрування для конфіденційних документів; 4. Обмеження на розмір файлу і часові рамки.
Technology and Data Variations List	1. Парсери: Apache Tika/python-docx/pdfminer; 2. Декодування: автоматичне або зазначення кодування вручну; 3. Збереження: тимчасове сховище, SQL/NoSQL або відправка в чергу для подальшого аналізу.
Frequency of Occurrence	80 %
Miscellaneous	1. Можливість відкладеної обробки. 2. Питання політики зберігання для анонімних користувачів. 3. Валідація на наявність чутливих даних до фінального збереження.

Одним із рушійних процесів у системі виступає автоматичне підвантаження тексту. Він є початковим етапом роботи з документом у системі та основою для усіх подальших кроків. До того ж, це впливає на якість отриманих результатів. Під час виконання цього сценарію користувач завантажує файл або вводить URL, а система перевіряє придатність матеріалу до обробки. Після успішного імпорту до тексту застосовується низка процедур, після чого він зберігається відповідно до політики системи. Цей прецедент забезпечує надійне введення текстових даних та узгоджується із наступними операціями лінгвістичного аналізу.

Таблиця 3.4 – Присвоєння ролі «Редактор» користувачу вручну

Usecase section	**Comment**
Use Case Name	Присвоєння ролі «Редактор» користувачу вручну
Scope	Вебзастосунок лінгвістичного аналізу тексту
Level	Управління доступом і правами користувачів
Primary Actor	Адміністратор
Stakeholders and interests	1. Адміністратор: коректне й безпечне призначення ролі; гарантія, що нові редактори відповідають політикам якості; 2. Користувач (потенційний редактор): отримання прав для редагування і рецензування.
Preconditions	1. Адміністратор автентифікований і може змінювати ролі; 2. Користувач існує в системі (акаунт активний); 3. Наявний інтерфейс для керування ролями; 4. Політика призначення ролей та бізнес-правила доступні.
Success guarantee	1. Користувачу присвоєно роль «Редактор», права набули чинності; 2. Дія задокументована в журналі (хто, коли, причина); 3. Користувач отримав повідомлення про зміну своєї ролі; 4. Можливе відкликання призначення або налаштування прав.
Main Success Scenario	1. Адміністратор переходить в «Керування користувачами» і знаходить потрібного користувача (через пошук або фільтр); 2. Коли користувача обрано, натискає «Змінити роль»; 3. У формі обирає роль «Редактор», за потреби вказує строк дії ролі; 4. Адміністратор натискає «Підтвердити», система перевіряє бізнес-правила (наприклад, чи користувач не заблокований); 5. Система оновлюється і надсилає внутрішнє повідомлення користувачу; 6. Користувач авторизується або оновлює сесію, бачить доданий функціонал редактора.
Extensions	1. Користувач не знайдений – система пропонує створити акаунт або перевірити введені дані; 2. Помилка збереження (БД або сервіс недоступні) – операція не виконується; адміністратору пропонується повторити або залишити запит у черзі; зміни не застосовано.

Кінець таблиці 3.4

Special Requirements	1. Ведення запису дій адміністратора з деталями (час, виконавець, користувач, причина); 2. Миттєве застосування прав або повідомлення про затримку; 3. Інтерфейс валідації бізнес-правил перед підтвердженням (пояснення причин відмови); 4. Можливість вказати строк дії ролі та автоматичне відкликання по закінченню терміну; 5. Забезпечити захищену передачу внутрішнього повідомлення про зміну ролі.
Technology and Data Variations List	Можливість призначення тимчасових ролей.
Frequency of Occurrence	10 %
Miscellaneous	1. Політика затвердження нових редакторів; 2. Механізм повідомлень: шаблони внутрішньої комунікації; 3. Процедура відкату та перевірку звітів для випадків зловживань; 4. Рекомендація: вести періодичний перегляд ролей.

Присвоєння нової ролі **дослідник**у розширює його права та відкриває доступ до функцій рецензування пропозицій виправлень. Перед зміною ролі система визначає, чи можливий взагалі такий розвиток подій. На цьому етапі перевіряються права адміністратора, стан облікового запису користувача та відповідність дії встановленим правилам доступу. Після підтвердження операції з’являється «новий» **редактор**, який входить до складу інших суб’єктів системи у цій категорії. У разі виникнення помилки (наприклад, якщо користувача не знайдено або є проблема із базою даних), зміни не застосовуються й адміністратору пропонується повторити дію або залишити запит у черзі.

3.4 Структура бази даних та взаємодія між її компонентами

Моделювання інформаційних потоків показало шлях текстової інформації в системі. Наступним кроком є проєктування логічної схеми бази даних, яка визначає, яким чином дані зберігатимуться, які зв’язки будуть між ними та як вони використовуватимуться різними модулями «LinguaInsight» (рис. 3.14).

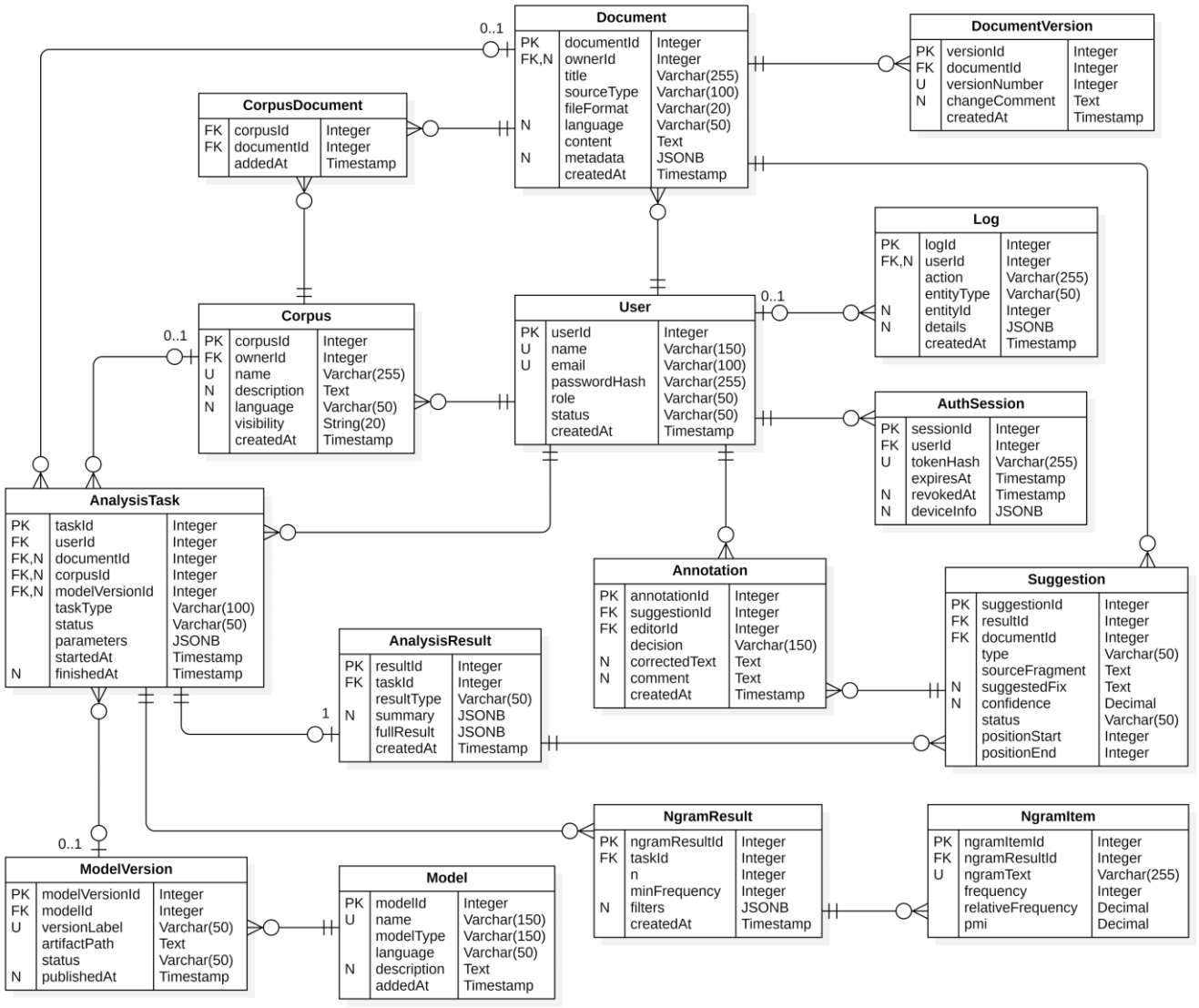

Рисунок 3.14 – ER-діаграма бази даних вебзастосунку

Спроєктована інформаційна модель містить різні елементи з метою забезпечення повного циклу роботи з текстом.

Таблиці бази даних доцільно поділити на кілька груп. Перша група пов’язана з користувачами та доступом до системи. Вона представлена таблицями User, що зберігає дані облікового запису, роль користувача і статус; та AuthSession, яка відповідає за активні сесії та їх відкликання, а також токени.

Друга група описує документи та корпуси. Основною є таблиця Document, у якій зберігаються відомості про завантажений текст, його формат, мову та власника. Для збереження історії змін існує таблиця DocumentVersion (фіксація окремих версій документа після редагування або аналізу). Корпуси текстів подано

таблицями Corpus і CorpusDocument (один корпус може містити багато документів, а один документ може входити до кількох корпусів).

Аналітична обробка представлена третьою групою. Таблиця AnalysisTask фіксує запити на аналіз документа або корпусу. Оскільки аналіз може виконуватися як для окремого документа, так і для корпусу, зв’язок із Document або Corpus у межах цієї таблиці може бути альтернативним. Підсумковий результат зберігається у таблиці AnalysisResult, яка пов’язана з AnalysisTask за принципом «одне завдання (декілька модулів) – один спільний звіт». У ній можуть міститися структуровані дані у форматі JSONB, такі як морфологічні ознаки або загальна статистика. Такий підхід дозволяє поєднати реляційні зв’язки з гнучким збереженням складних результатів NLP-обробки.

Окрема частина моделі призначена для перевірки правопису, стилю та ручної верифікації. Таблиця Suggestion зберігає знайдені системою пропозиції виправлень. Вона пов’язана як із документом, так і з результатом аналізу, що дає змогу швидко знаходити підказки для конкретного тексту. Таблиця Annotation використовується для збереження редакторських уточнень. Завдяки цьому система може накопичувати необхідні матеріали для подальшого покращення правил, словників або навіть мовних моделей.

Для корпусного аналізу передбачено таблиці NgramResult та NgramItem. Перша зберігає загальні параметри побудови *n*-грам і пов’язується із відповідним завданням. Друга містить конкретні знайдені послідовності слів та показник PMI. Такий поділ потрібний тому, що один запуск аналізу може створювати багато окремих статистичних записів.

Ще одна група таблиць пов’язана з мовними моделями. Таблиці Model і ModelVersion дають змогу зберігати інформацію про доступні моделі. Зв’язок ModelVersion з AnalysisTask є необов’язковим, оскільки перевірки можуть виконуватися лише на основі правил або сторонніх підключених сервісів.

Додатково в моделі передбачено таблицю Log, яка фіксує службові події: процеси завантаження документів, запуск аналізу, зміну ролей, редагування результатів і дії адміністратора. Окремі записи журналу можуть бути пов’язані з

конкретним користувачем, а системні події можуть зберігатися без такого зв’язку. Це забезпечує прозорість роботи системи та можливість подальшої перевірки дій користувачів.

Інформаційна модель поєднує класичні реляційні зв’язки та поля JSONB. Реляційна частина призначена для користувачів, документів, корпусів, моделей і журналів, оскільки ці дані потребують чіткої структури та цілісності. JSONB доцільно використовувати для результатів NLP-аналізу, параметрів запуску, метаданих документів і проміжних структур, оскільки їхній формат може змінюватися залежно від типу аналізу.

Таким чином, спроєктована база даних підтримує повний цикл роботи вебзастосунку «LinguaInsight». Вона дозволяє вебклієнту якісно взаємодіяти із серверною частиною завдяки формуванню запитів. Схема демонструє набір сутностей, які відповідають за можливість зберігати документи, об’єднувати їх у корпуси, запускати різні види аналізу, фіксувати підсумкові звіти, організовувати ручну перевірку та накопичувати матеріали для подальшого вдосконалення системи.

3.5 Мокапи інтерфейсів

Розробка мокапів передбачає візуалізацію ключових елементів вебзастосунку лінгвістичного аналізу тексту. Для цього створено декілька макетів, які функціонують як попередні прототипи і дають змогу перевірити логіку розміщення елементів інтерфейсу. Особливу увагу приділено сторінкам головної панелі й імпорту тексту, перевірці правопису і стилю, розгляду пропозицій виправлень і додавання випадку вживання слова до словника для врахування за необхідністю. За допомогою таких графічних представлень стає можливим оцінити структуру майбутнього інтерфейсу ще до повноцінної реалізації, а також узгодити його з функціональними вимогами системи. Надалі окремі блоки мокапів можуть бути перенесені у відповідні компоненти React без суттєвої зміни загальної логіки.

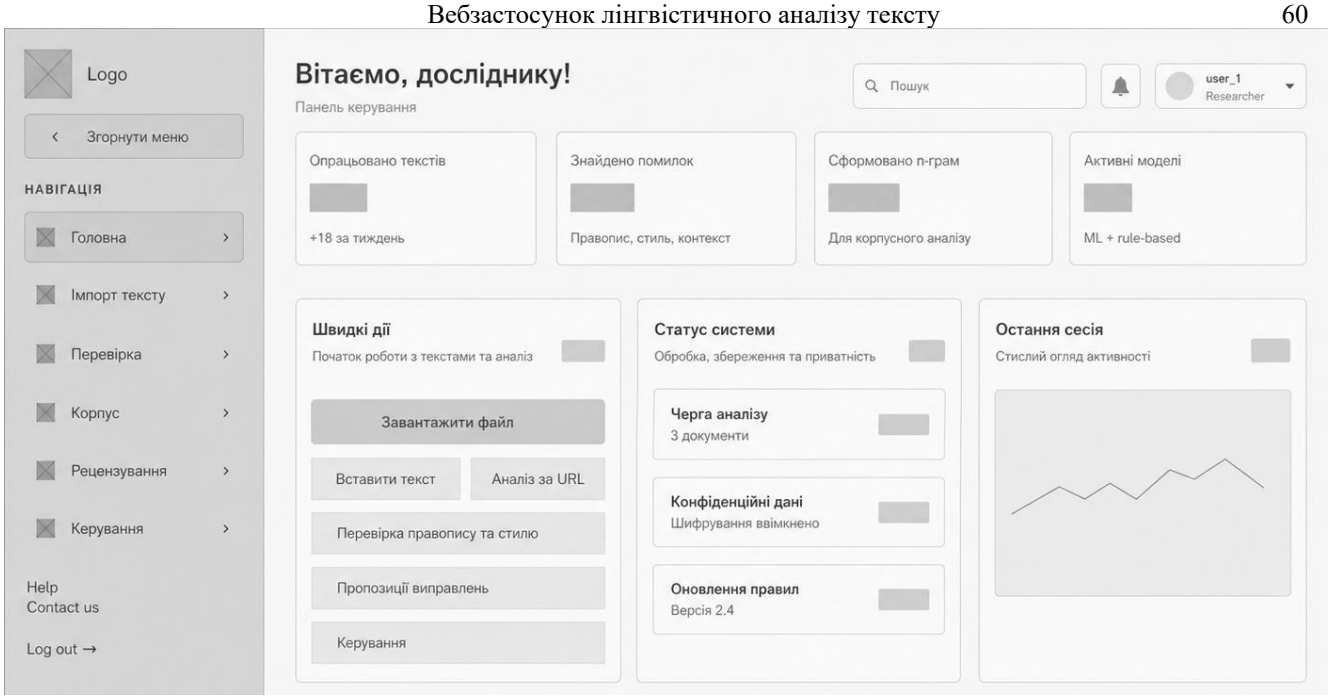

Рисунок 3.15 – Мокап інтерфейсу головної панелі

На рис. 3.15 відображено основні показники роботи вебзастосунку. У верхній частині сторінки розміщено цифрові блоки, ліворуч знаходиться навігаційне меню. Центральна частина містить блок швидких дій, статус системи й огляд останньої сесії. У нижній частині подано таблицю останніх проєктів.

Головна панель складається із наступних функціональних елементів:

- профіль користувача;
- статистичні картки з основними показниками;
- блоки швидких дій, статусу системи та аналітики помилок за категоріями;
- графік активності й таблиця останніх проєктів.

Сторінка імпорту призначена для завантаження та попередньої обробки текстових даних перед подальшим аналізом (рис. 3.16). Відповідно до оформлення вона містить блок вибору джерела даних, зону перетягування файлу, вибір мови документа й кодування. На ній відображаються метадані і статус приватності.

Визначені функціональні елементи:

- блок вибору джерела даних (зона перетягування та завантаження файлу, введення посилання або тексту);
- вибір мови документа й кодування;
- панель метаданих.

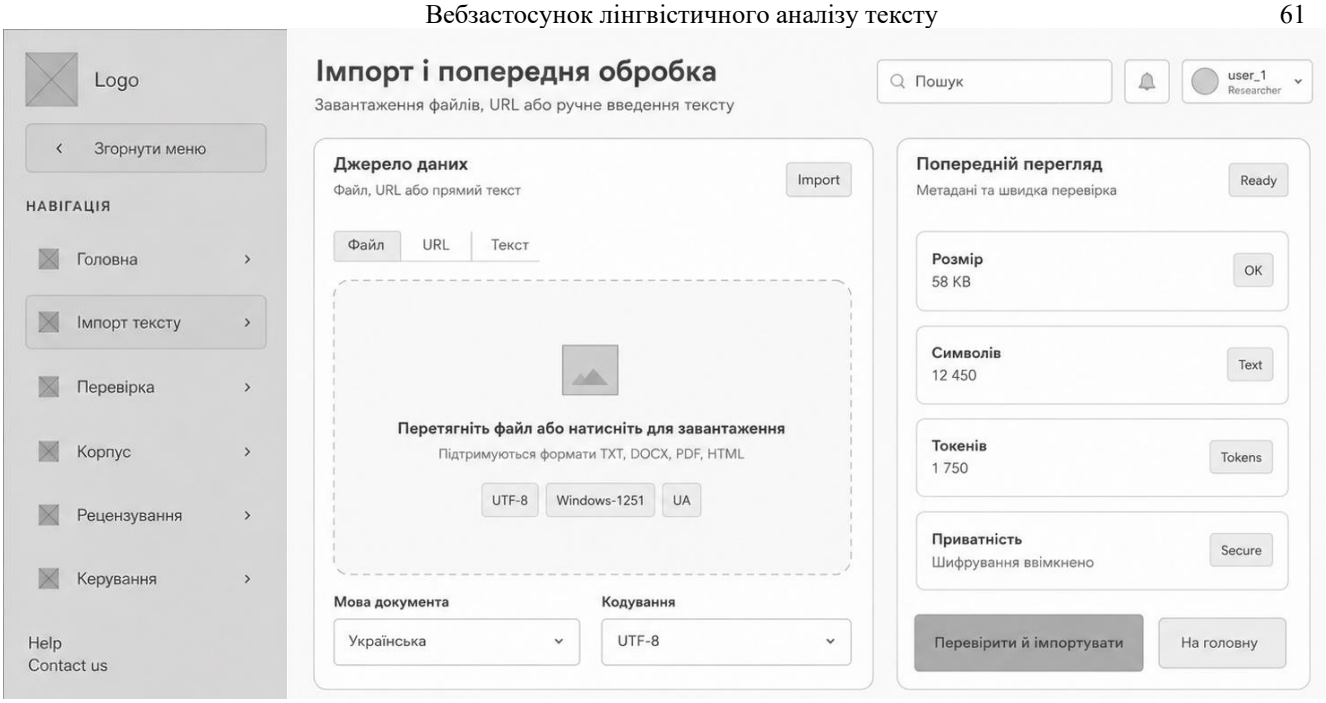

Рисунок 3.16 – Мокап інтерфейсу імпорту тексту

Функція сторінки рецензування на рис. 3.17 полягає у перевірці та затвердженні пропозицій виправлень. На ній подано загальну статистику; список активних із них, що потребують розгляду; фільтри за пріоритетом і рівнем довіри. Не менш важливою є наявність фрагменту тексту з контекстом для зручності редактора. Панель деталей рішення демонструє інформацію про обрану зі списку пропозицію та доступні для виконання дії.

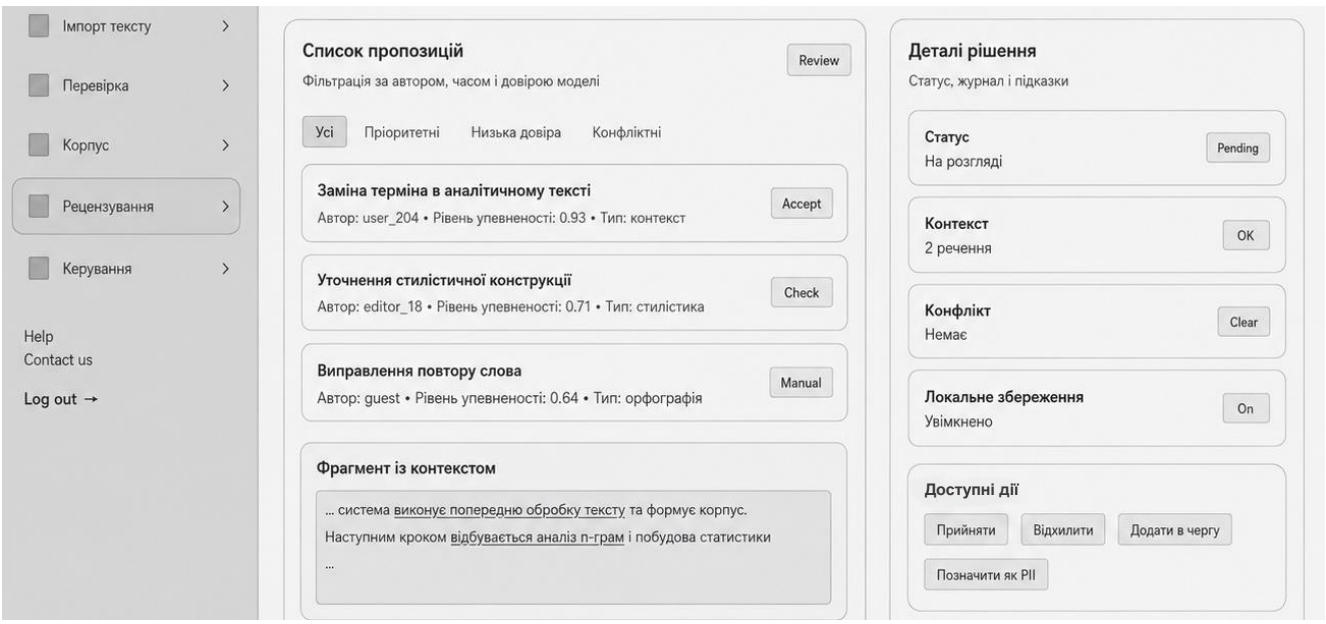

Рисунок 3.17 – Частина мокапу інтерфейсу рецензування пропозицій виправлень

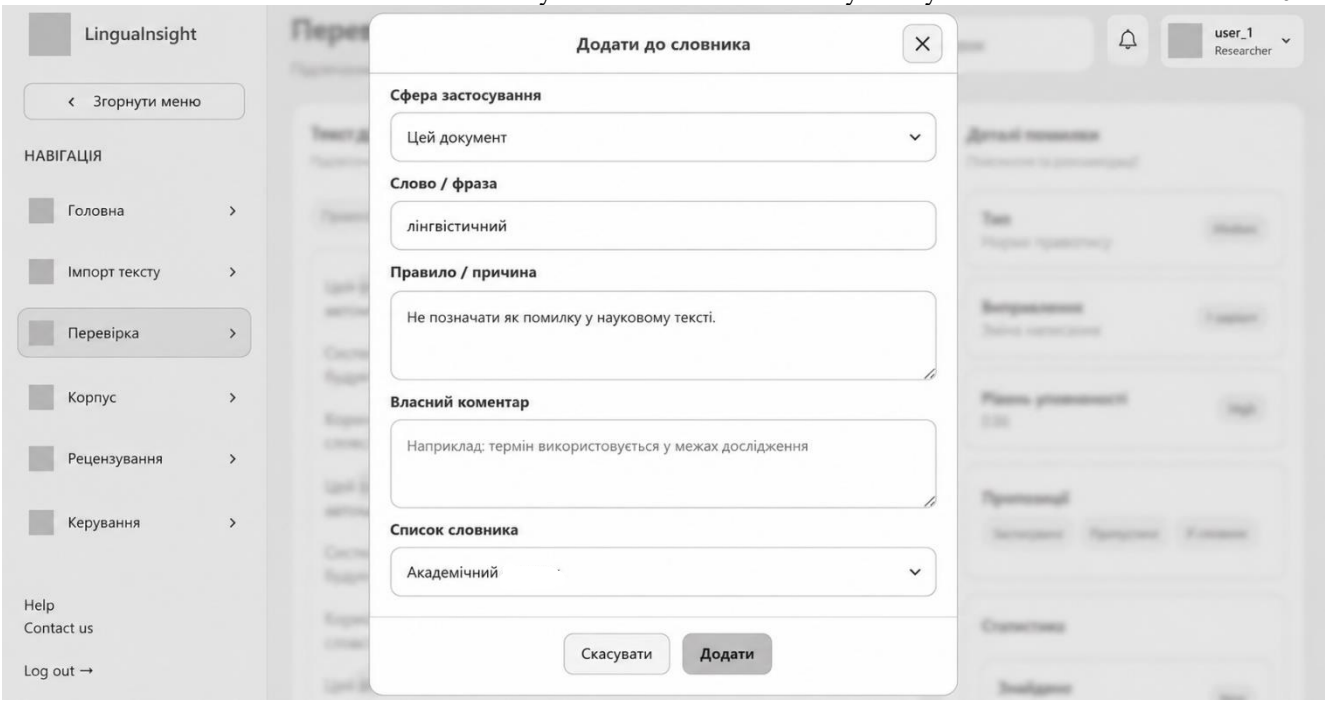

Рисунок 3.18 – Мокап інтерфейсу додавання випадку вживання до словника

У вебзастосунку реалізовано модальне вікно внесення окремих слів або фраз у користувацький чи загальний словник (рис. 3.18). Форма включає випадний список для вибору сфери застосування (документ або системний), поле введення, коментар користувача та вибір словника (академічний, технічний або індивідуальний).

Висновки до розділу 3

Даний розділ надає способи проєктування вебзастосунку лінгвістичного аналізу тексту під назвою «LinguaInsight». Для цього розроблено комплекс UML-діаграм, які відображають структуру системи та її фізичне розгортання.

Інформаційні потоки роботи з документом змодельовані для демонстрації кроків передачі даних. Важливі для подальшого вдосконалення правил і словників процеси спрощуються завдяки сценаріям ручної перевірки результатів.

Через варіанти використання системи деталізується робота гостя, дослідника, редактора й адміністратора. Також спроєктовано логічну схему бази даних, у якій поєднання реляційних зв’язків із полями JSONB забезпечує гнучкість для збереження складних результатів NLP-обробки. На завершальному етапі створено мокапи ключових інтерфейсів, що дає можливість подальшої практичної реалізації вебзастосунку та його програмних модулів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Вебплатформа для опрацювання текстових матеріалів розробляється як цифрове середовище для роботи з текстом. Усі ролі системи утворюють єдину логічну модель доступу. Така структура дозволяє чітко розмежувати права користувачів та організувати безпечну роботу.

4.1 Організація файлової складової проєкту

Програмна реалізація вебзастосунку складається із клієнтської частини, серверного API, моделей БД, модулів лінгвістичної обробки та службових файлів. Інтерфейс користувача відокремлено від бізнес-логіки, а обробка тексту й збереження результатів виконуються через окремі програмні блоки (табл. 4.1).

У корені проєкту розміщено ряд директорій. У frontend розташований React-застосунок, що відповідає за відображення сторінок. Директорія backend містить FastAPI-сервер, який приймає запити від клієнта і запускає основні операції аналізу. Окремо винесено директорію database із SQL-структурою логічної схеми бази даних. Файл docker-compose.yml описує розгортання вебзастосунку.

Таблиця 4.1 – Структура програмної реалізації вебзастосунку

Рівень	Основні директорії, файли	Призначення
Клієнтська частина	frontend/src/pages, frontend/src/components, frontend/src/api.js	Реалізація сторінок, спільних компонентів, маршрутизації та API-запитів
Серверна частина	backend/app/main.py, backend/app/models.py, backend/app/security.py	REST API, авторизація, робота із сутностями та виконання бізнес-логіки
NLP-обробка	backend/app/nlp.py	Нормалізація тексту, токенізація, перевірка правопису й стилю, побудова *n*-грам
База даних	database/schema.sql, backend/app/models.py	Опис таблиць, зв’язків, унікальних обмежень і моделей SQLAlchemy
Розгортання	docker-compose.yml, backend/Dockerfile	Запуск frontend, backend і PostgreSQL у контейнерах

Клієнтська частина реалізована на React із використанням React Router та Axios. Основні сторінки розміщено в директорії frontend/src/pages. Спільний каркас інтерфейсу винесено в компонент Layout.jsx, щоб уникнути дублювання розмітки.

API-запити клієнтської частини містяться у файл api.js. У ньому створено екземпляр Axios, який автоматично додає токен авторизації до запитів. Для більш контрольованої взаємодії frontend та backend окремі сторінки звертаються до підготовленого API-шару.

Серверна частина реалізована у директорії backend/app. Центральним файлом є main.py, де описано REST-точки доступу. Файл models.py містить ORM-моделі SQLAlchemy, які відображають структуру основних сутностей бази даних.

У файлі nlp.py описано необхідні функції лінгвістичної обробки. У межах поточної реалізації використано комбінований підхід. Він поєднує власні правила, зовнішню перевірку якості тексту та модулі автоматичної мовної обробки. Водночас структура модуля дозволяє надалі підключити spaCy, Hugging Face Transformers або окремий сервіс моделей без зміни загальної логіки frontend і API. Розподіл основних серверних методів наведено у табл. 4.2.

Таблиця 4.2 – Основні API-операції серверної частини

Група методів	**Точки доступу**	**Призначення**
Авторизація	/api/auth/register, /api/auth/login, /api/auth/me	Реєстрація, вхід і отримання поточного користувача
Документи	/api/documents, /api/documents/{id}, /api/documents/extract	Імпорт, зчитування, перегляд і збереження текстів
Корпуси	/api/corpora, /api/corpora/{id}/documents/{docId}	Створення корпусів і додавання документів
Публічні корпуси	/api/public/corpora, /api/corpora/{id}/visibility	Перегляд відкритих корпусів і керування видимістю
Аналіз	/api/analysis/run	Запуск перевірки та формування спільного звіту
N-грами	/api/ngrams/run	Побудова n-грам, частот, PMI та ключових фраз

Кінець таблиці 4.2

Рецензування	/api/suggestions, /api/suggestions/{id}/decision	Перегляд і прийняття або відхилення пропозицій
Моделі	/api/models	Отримання інформації про доступні мовні моделі
Адміністрування	/api/admin/users, /api/admin/users/{id}/role	Перегляд користувачів, зміна ролей і контроль доступу
Сповіщення	/api/notifications	Отримання повідомлень про рішення редактора та зміну ролей

Окремої уваги потребує реалізація доступу. У файлі security.py зосереджено логіку хешування пароля, створення токена сесії та перевірки поточного користувача. Залежність get_current_user використовується для захищених запитів. Вона перевіряє наявність токена, його чинність і статус користувача. Для дій адміністратора або редактора передбачено визначення ролі через require_role.

Реалізована структура програмного коду має модульний характер. Клієнтська частина відповідає за взаємодію користувача з інтерфейсом, серверна – за обробку запитів, контроль доступу й збереження даних, а модуль nlp.py – за текстову обробку. Прикладом розширення системи без повної перебудови проєкту може слугувати заміна rule-based перевірки на модельний аналіз.

4.2 Реалізація клієнтської частини

Клієнтська частина вебзастосунку «LinguaInsight» реалізована як Single Page Application на базі React. Для маршрутизації між функціональними модулями використано React Router, а для обміну даними із backend – Axios.

Ця частина складається із наступних груп файлів: сторінок frontend/src/pages, спільних компонентів у frontend/src/components, файлу API-шару frontend/src/api.js.

Реалізація загального каркасу інтерфейсу

Одним із ключових компонентів клієнтської частини є файл frontend/src/components/Layout.jsx. Він відповідає за повторювану структуру сторінок (рис. 4.1). У компоненті також реалізовано згортання бічної панелі.

```
const [collapsed, setCollapsed] = useState(false);
<aside className={`sidebar ${collapsed ? 'collapsed' : ''}`}>
    <button className="sidebar-toggle" type="button" onClick={() => setCollapsed((value) =>
!value)}>☰</button>
    <NavLink to="/dashboard">Панель</NavLink>
    <NavLink to="/import">Імпорт</NavLink>
    <NavLink to="/check">Перевірка</NavLink>
    <NavLink to="/corpus">Корпуси</NavLink>
    <NavLink to="/review">Рецензування</NavLink>
    <NavLink to="/models">Моделі</NavLink>
    <NavLink to="/admin">Адміністрування</NavLink>
</aside>
```

Рисунок 4.1 – Фрагмент коду бічної панелі Layout.jsx

На рис. 4.1 навігацію реалізовано через NavLink. Це дозволяє автоматично визначати активну сторінку й підсвічувати відповідний пункт меню. У порівнянні зі статичним HTML-макетом, де згортання меню виконувалося через DOM-скрипт, у React цей механізм реалізовано через стан компонента, що відповідає архітектурі.

Взаємодія з backend через API-шар

Для взаємодії клієнтської частини із сервером використано окремий файл api.js (рис. 4.2). У ньому створюється базовий екземпляр Axios, який містить адресу backend і автоматично додає токен авторизації до кожного запиту.

```
const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000/api';
export const api = axios.create({ baseURL: API_URL });
api.interceptors.request.use((config) => {
  const token = getStoredToken();
  if (token) { config.headers.Authorization = `Bearer ${token}`; }
  return config;
});
api.interceptors.response.use(
  (response) => response, (error) => {
    if (error.response?.status === 401) { clearAuth(); }
    return Promise.reject(error);
  }
);
export function saveAuth(data) {
  const token = data.token || data.accessToken;
  const user = data.user;
  if (!token || !user) { throw new Error('Backend не повернув token або user'); }
  localStorage.setItem('li_token', token);
  localStorage.setItem('li_user', JSON.stringify(user));
}
export function clearAuth() {
  localStorage.removeItem('li_token');
  localStorage.removeItem('li_user');
}
export function getStoredToken() {
  const token = localStorage.getItem('li_token');
  if (!token || token === 'undefined' || token === 'null') { return null; }
  return token;
}
```

Рисунок 4.2 – Фрагмент коду api.js

Головна панель користувача

Сторінка frontend/src/pages/Dashboard.jsx виконує роль стартового екрана після входу в систему. Хук useEffect, у якому створюється асинхронна функція, використовується для завантаження даних з точки доступу /dashboard (рис. 4.3).

```
useEffect(() => {
    async function loadDashboard() {
      try {
        const response = await api.get('/dashboard');
        setData(response.data);
      } catch (error) { console.error(error); }
    } loadDashboard();
  }, []);
```

Рисунок 4.3 – Фрагмент коду Dashboard.jsx

Реалізація імпорту тексту

Сторінка frontend/src/pages/ImportPage.jsx призначена для введення або завантаження текстових даних (рис. 4.4). У проєкті передбачено обмеження: кількість символів не має перевищувати 20 000, а розмір файлу – 10 MB. Це запобігає надмірному навантаженню на сервер і сприяє стабільній обробці документів.

```
const MAX_TEXT_CHARACTERS = 20000;
const MAX_FILE_SIZE_MB = 10;
const MAX_FILE_SIZE_BYTES = MAX_FILE_SIZE_MB * 1024 * 1024;

function validateTextLength(text) {
    const length = String(text || '').length;
    if (length > MAX_TEXT_CHARACTERS) {
      return `Текст перевищує допустимий обсяг: ${formatNumber(length)} символів із
${formatNumber(MAX_TEXT_CHARACTERS)}. Скоротіть текст або завантажте менший фрагмент.`;
    }
    return '';
  }
```

Рисунок 4.4 – Фрагмент коду перевірки завантажених даних ImportPage.jsx

Для зчитування файлів використовується точка входу /documents/extract. Frontend передає файл у форматі FormData, а backend повертає витягнутий текст, формат документа та метадані (рис. 4.5). Це важливо як для підтримки введення вручну, так і для імпорту TXT, DOCX і PDF. Після підготовки тексту система формує дані документа й надсилає їх на сервер, де вони зберігаються в базі даних.

```
        const { data } = await api.post('/documents/extract', formData, {
         headers: { 'Content-Type': 'multipart/form-data' }
       });
       const extractedText = data.content || '';
       const textLengthError = validateTextLength(extractedText);
       if (textLengthError) {
         setFileSizeBytes(file.size);
         setMeta(data.metadata || null);
         setMessage(textLengthError);
         event.target.value = '';
         return;
       }
       setForm((previous) => ({
         ...previous,
         title: data.title || file.name,
         content: extractedText,
         sourceType: 'file',
         fileFormat: data.fileFormat || file.name.split('.').pop()?.toUpperCase() || 'TXT'
       }));
       setMeta(data.metadata || null);
       setMessage('Файл успішно зчитано');
```

Рисунок 4.5 – Лістинг коду точки входу для зчитування файлів ImportPage.jsx

Блок перевірки правопису та стилю

Сторінка frontend/src/pages/CheckPage.jsx дає змогу вибрати документ або працювати з поточною версією документа. Після отримання відповіді від backend користувач бачить узагальнений звіт, список знайдених зауважень і деталі кожної пропозиції (рис. 4.6).

```
async function run() {
    setMsg('');
    setResult(null);
    setSelectedSuggestion(null);
    const textToAnalyze = editorText || finalVersionText || baseDocumentText || '';
    if (!textToAnalyze.trim()) {
      setMsg('Введіть або оберіть текст для перевірки'); return;
    }
    const currentVersion = versions[activeVersionIndex];
    if (currentVersion) { setDecisions(currentVersion.decisions || {}); } else {
setDecisions({}); }
    try { const { data } = await api.post('/analysis/run-text', {
        documentId: docId || null,
        text: textToAnalyze,
        language: currentDocument?.language || 'uk',
        taskType: 'spell_style',
        parameters: { confidenceThreshold: 0.7, useRealModels: true, detectPII: true }
      });
      setAnalysisText(textToAnalyze);
      setResult(data);
      setIsEditorDirty(false);
      setMsg('Аналіз виконано');
    } catch (error) { setMsg(error.response?.data?.detail || 'Помилка аналізу'); }
  }
```

Рисунок 4.6 – Лістинг коду запуску аналізу CheckPage.jsx

На рис. 4.6 продемонстровано розподіл відповідальності: інтерфейс відповідає за взаємодію з користувачем, в той час як аналітична логіка виконується на сервері. Окремо в клієнтській частині передбачено початкове виявлення чутливих даних у тексті.

Розпізнавання іменованих сутностей виконується на рівні NLP-модуля. У системі використовується rule-based механізм, який знаходить послідовності слів із великої літери та повертає їх як потенційні власні назви. Саме така структура зручна для передачі на frontend і збереження в JSON-полях.

Реалізація сторінки корпусного аналізу

Одним із найважливіших модулів клієнтської частини є сторінка frontend/src/pages/CorpusPage.jsx. Вона дозволяє якісно працювати з корпусами та переглядати результати у вигляді таблиць і графіків (рис. 4.7). Якщо користувач вибирає корпус, frontend надсилає на сервер corpusId, і backend аналізує весь вміст цього корпусу. Якщо корпус не вибрано, але вибрано окремий документ, на сервер передається documentId, і аналіз виконується лише для цього документа.

```
const basePayload = {
        n: Number(ngramSize),
        minFrequency: Number(minFrequency),
        language,
        stopWordsProfile,
        posMethod,
        maxResults: 500
      };
      const payload = corpusId
        ? { ...basePayload, corpusId }
        : { ...basePayload, documentId: docId
        };
      const { data } = await api.post('/ngrams/run', payload);
      setItems(data.items || []);
      setMsg(
        corpusId ? 'N-грами сформовано для всього корпусу' : 'N-грами сформовано для
вибраного документа'
      );
    } catch (error) { setMsg(error.response?.data?.detail || 'Помилка побудови n-грам');
}
```

Рисунок 4.7 – Лістинг коду визначення послідовностей слів CorpusPage.jsx

Для побудови графіка частот використовується Plotly.js. Графік показує найпоширеніші *n*-грами та їхню частоту (рис. 4.8). Таке представлення дозволяє користувачу візуально оцінити розподіл послідовностей у документі або корпусі.

```
<Plot
  data={[
    {
      x: chartItems.map((item) => item.ngramText),
      y: chartItems.map((item) => item.frequency),
      type: 'bar', text: chartItems.map((item) => String(item.frequency)),
      textposition: 'outside', hovertemplate: '<b>%{x}</b><br>Частота: %{y}<extra></extra>'
    }
  ]}
  layout={{
    height: 330,
    margin: { t: 18, r: 18, b: 92, l: 46 },
    paper_bgcolor: 'rgba(0,0,0,0)',
    plot_bgcolor: 'rgba(0,0,0,0)',
    xaxis: { tickangle: -35, automargin: true, title: { text: 'N-грами'} },
    yaxis: { title: { text: 'Частота'}, rangemode: 'tozero', gridcolor: '#dce4f1'
    }, showlegend: false
  }}
  config={{ responsive: true, displaylogo: false, modeBarButtonsToRemove: ['lasso2d',
'select2d'] }}
/>
```

Рисунок 4.8 – Лістинг коду побудови графіка послідовностей слів CorpusPage.jsx

Окрім графіка, у модулі корпусного аналізу реалізовано хмару ключових слів. Для цього використано бібліотеку d3-cloud, яка обчислює координати, розмір і поворот кожної одиниці (рис. 4.9). Розмір слова залежить від його ваги, а кількість слів і розмір полотна можуть змінюватися за необхідності.

```
function KeywordCloud({ words, width, height, requestedCount, onRenderedCount }) {
  const [layoutWords, setLayoutWords] = useState([]);
  const cloudRef = useRef(null);
  useEffect(() => {
    if (!words.length) { setLayoutWords([]); onRenderedCount?.(0); return; }
    const safeWidth = Math.max(280, Number(width) || 420);
    const safeHeight = Math.max(180, Number(height) || 300);
    const limit = Math.max(1, Number(requestedCount) || words.length);
    const selectedWords = words.slice(0, limit);
    const maxValue = Math.max( ...selectedWords.map((item) => Number(item.value || 1)), 1 );
    const minFont = limit > 40 ? 9 : limit > 25 ? 10 : 12;
    const maxFont = limit > 40 ? 20 : limit > 25 ? 26 : 36;
    const preparedWords = selectedWords.map((item, index) => {
      const value = Number(item.value || 1);
      const ratio = Math.sqrt(value) / Math.sqrt(maxValue);
      return { text: item.text, value, size: Math.round(minFont + ratio * (maxFont -
minFont)), color: WORD_CLOUD_COLORS[index % WORD_CLOUD_COLORS.length] };
    });
    d3Cloud()
      .size([safeWidth, safeHeight])
      .words(preparedWords)
      .padding(limit > 30 ? 1 : 2)
      .rotate((word, index) => { const directions = [0, 0, 0, 90, -90]; return
directions[index % directions.length]; })
      .font('Arial')
      .fontWeight('700')
      .fontSize((word) => word.size)
      .on('end', (result) => { setLayoutWords(result); onRenderedCount?.(result.length); })
      .start();
  }, [words, width, height, requestedCount, onRenderedCount]);
```

Рисунок 4.9 – Лістинг коду побудови хмари слів CorpusPage.jsx

Для полегшення інтерпретації частотних результатів сторінка корпусного аналізу містить як табличні дані, так й інтерактивні візуальні засоби. Таким чином можна побачити й оцінити загальну картину.

Експорт результатів корпусного аналізу

У CorpusPage.jsx передбачено експорт результатів у CSV, JSON, PDF та PNG. CSV використовується для табличного збереження всіх *n*-грам, JSON – для збереження даних разом із параметрами аналізу, а PDF – для формування структурованого звіту. Для PDF використано pdfmake, оскільки ця бібліотека створює файл з елементами, які можна виділяти й копіювати (рис. 4.10).

```
     const ngramRows = [
      [
        { text: '№', bold: true, alignment: 'center' },
        { text: 'N-грама', bold: true, alignment: 'center' },
        { text: 'Частота', bold: true, alignment: 'center' },
        { text: 'Відносна частота', bold: true, alignment: 'center' },
        { text: 'Зв’язність', bold: true, alignment: 'center' },
        { text: 'Профіль', bold: true, alignment: 'center' }
      ],
      ...frequencySortedSequences.map((item, index) => [
        { text: String(index + 1), alignment: 'center' },
        { text: item.ngramText, alignment: 'left' },
        { text: String(item.frequency), alignment: 'center' },
        { text: String(item.relativeFrequency), alignment: 'center' },
        { text: String(item.pmi), alignment: 'center' },
        { text: getProfileTag(item), alignment: 'center' }
      ])
    ];

    pdfMake
      .createPdf(docDefinition)
      .download( isCorpusReport ? 'linguainsight_corpus_report.pdf' :
'linguainsight_document_report.pdf');
    setMsg('PDF-звіт завантажено');
```

Рисунок 4.10 – Лістинг коду таблиці послідовностей та формування звіту

Реалізація сторінки рецензування пропозицій

Окремий модуль клієнтської частини призначений для роботи редактора. Сторінка frontend/src/pages/ReviewPage.jsx дозволяє прийняти або відхилити підказку та додати коментар. Після вибору рішення frontend надсилає відповідний запит на сервер (рис. 4.11).

```
async function load() {
    try {
      const response = await api.get('/suggestions');
      const sorted = sortSuggestionsByNewest(response.data || []);
      const manualItems = sorted.filter(needsManualReview);
      setAllItems(sorted);
      setItems(manualItems);
      setSelected((current) => {
        if (current) { return ( manualItems.find((item) => item.suggestionId ===
current.suggestionId) || sorted.find((item) => item.suggestionId === current.suggestionId) ||
            manualItems[0] || null
          ); } return manualItems[0] || null;
      });  } catch (error) { setMsg(getApiError(error, 'Не вдалося завантажити пропозиції')); }
  }
  async function decide(decision) { if (!selected) return;
    try { const { data } = await api.post(`/suggestions/${selected.suggestionId}/decision`, {
        decision,comment,correctedText:decision === 'accepted' ? selected.suggestedFix : '' });
      const journalItem = data.journalItem || { journalId: `${Date.now()}_${decision}`,
decision, comment, correctedText: decision === 'accepted' ? selected.suggestedFix : '',
authorName: 'Користувач', createdAt: new Date().toISOString() };
      const updateSuggestion = (item) => item.suggestionId === selected.suggestionId ? {
...item, status: decision, journal: [journalItem, ...(item.journal || [])] } : item;
      const nextAllItems = sortSuggestionsByNewest( allItems.map(updateSuggestion) );
      const nextManualItems = nextAllItems.filter(needsManualReview);
      setAllItems(nextAllItems);
      setItems(nextManualItems);
      setSelected((previous) => previous ? { ...previous,   status: decision, journal:
[journalItem, ...(previous.journal || [])] } : previous );
      setMsg('Рішення збережено');
      setComment('');
      await load();
    } catch (error) { setMsg(getApiError(error, 'Не вдалося зберегти рішення')); }
  }
```

Рисунок 4.11 – Лістинг коду завантаження пропозицій та прийняття рішень

Цей модуль забезпечує верифікацію результатів автоматичної перевірки вручну. Він дозволяє отримувати підказки і накопичувати редакторські рішення.

Реалізація профілю користувача

Сторінка frontend/src/pages/ProfilePage.jsx призначена для перегляду даних користувача, його ролі, документів, корпусів і словникових записів (рис. 4.12).

```
async function loadProfileData() {
    try {
      const [profileResponse, docsResponse, corporaResponse, dictionaryResponse] =
        await Promise.all([ api.get('/auth/me'), api.get('/documents'),
api.get('/corpora'), api.get('/dictionary') ]);
      setUser(profileResponse.data);
      setForm({username:profileResponse.data.username, email:profileResponse.data.email});
      setDocs(docsResponse.data || []);
      setCorpora(corporaResponse.data || []);
      setDictionary(dictionaryResponse.data || []);
      localStorage.setItem('li_user', JSON.stringify(profileResponse.data));
    } catch (error) { setMsg(getApiError(error, 'Не вдалося завантажити дані профілю')); }
  }
```

Рисунок 4.12 – Лістинг коду отримання даних користувача ProfilePage.jsx

Такий підхід дає змогу показати користувачу його облікові дані та робочу інформацію: документи, створені корпуси та власний словник.

Реалізація керування моделями

Завдяки сторінці frontend/src/pages/ModelsPage.jsx модуль демонструє підготовку системи до роботи з кількома типами мовних моделей. Це відповідає логіці системи, у якій моделі можуть оновлюватися, перевірятися й порівнюватися без зміни основного інтерфейсу.

Інтерфейс адмін-панелі

Адмін-панель функціонує для керування користувачами, ролями, статусами та журналом дій (рис. 4.13). Робота адміністратора полягає в перегляді суб'єктів, змінювати роль, фіксувати причину зміни та контролювати службові події. Це відповідає рольовій моделі системи, де дії дослідника, редактора й адміністратора мають різні рівні доступу.

```
const [users, setUsers] = useState([]);
const [journal, setJournal] = useState([]);
const [selectedUserId, setSelectedUserId] = useState('');
const [filters, setFilters] = useState({ search: '', role: 'all', status: 'all' });
const [roleForm, setRoleForm] = useState({ role:'Researcher',validUntil:'', reason:'' });
const current = getStoredUser();
const selectedUser = users.find((user) => user.userId === selectedUserId) || null;
const isSelfSelected = selectedUser?.userId === current?.userId;
async function load() {
  try {
    const usersResponse = await api.get('/admin/users');
    const logsResponse = await api.get('/admin/logs').catch(() => ({ data: [] }));
    const safeUsers = Array.isArray(usersResponse.data) ? usersResponse.data : [];
    setUsers(safeUsers);
    setJournal(Array.isArray(logsResponse.data) ? logsResponse.data : []);
    setSelectedUserId((previous) => { if (safeUsers.some((user) => user.userId ===
previous)) { return previous; }
      return safeUsers[0]?.userId || '';
    }); } catch (error) {setErr(error.response?.data?.detail || 'Доступно лише адміну');}
}
async function changeSelectedRole() {
  if (!selectedUser) { setErr('Оберіть користувача'); return; }
  if (isSelfSelected) { setErr('Адміністратор не може змінювати власну роль'); return; }
  try { await api.patch(`/admin/users/${selectedUser.userId}/role`, null, {
      params:{role:roleForm.role, reason:roleForm.reason, validUntil:roleForm.validUntil}
    });
    setMsg(`Роль користувача ${selectedUser.username} змінено`);
    setErr('');
    await load();
  } catch (error) { setErr(error.response?.data?.detail || 'Не вдалося змінити роль'); }
}
```

Рисунок 4.13 – Лістинг коду функцій керування користувачами AdminPage.jsx

Завдяки використанню CSS Grid і media-запитів інтерфейс адаптується до різної ширини екрана і це дозволяє зручно працювати на різних пристроях. Клієнтська частина «LinguaInsight» забезпечує повний цикл взаємодії користувача із системою на всіх її рівнях.

4.3 Розробка серверної частини та бізнес-логіки вебзастосунку лінгвістичного аналізу тексту

У межах проєкту серверна частина розміщується у директорії backend/app і побудована на FastAPI. Його основні модулі продемонстровано у табл. 4.3.

Таблиця 4.3 – Основні модулі серверної частини

Модуль	Призначення
main.py	Описує REST API для авторизації, документів, корпусів, аналізу, n-грам, рецензування, моделей, сповіщень і адміністрування
models.py	Містить ORM-класи SQLAlchemy, які відповідають сутностям бази даних
schemas.py	Визначає Pydantic-схеми для перевірки вхідних даних і формування відповідей
security.py	Реалізує хешування паролів, створення токенів, перевірку сесій і ролей
nlp.py	Містить функції нормалізації, токенізації, пошуку помилок, NER, PII, LanguageTool-перевірки та побудови n-грам
file_parser.py	Забезпечує зчитування тексту з TXT, DOCX, PDF та HTML
database.py	Налаштовує підключення до бази даних і створення сесій SQLAlchemy

Імпорт і попередня обробка файлів

Модуль file_parser.py відповідає за зчитування тексту з файлів (рис. 4.14). Можливості користувача розширяються завдяки підтримці форматів TXT, HTML, DOCX і PDF для документів, що завантажуються.

```
def extract_text_from_file( filename: str, content: bytes, encoding: str = "utf-8" ) ->
tuple[str, str]:
    extension = filename.rsplit(".", 1)[-1].lower() if "." in filename else "txt"
    if extension in {"txt", "csv"}: return read_txt(content, encoding), extension.upper()
    if extension in {"html", "htm"}: return read_html(content, encoding), "HTML"
    if extension == "docx": return read_docx(content), "DOCX"
    if extension == "pdf": return read_pdf(content), "PDF"
    return read_txt(content, encoding), extension.upper()
```

Рисунок 4.14 – Лістинг коду функції отримання текстової інформації file_parser.py

Для PDF додатково застосовується нормалізація витягнутого тексту, щоб прибрати зайві переноси, некоректні розриви рядків і наслідки PDF-екстракції. Точка входу (рис. 4.15) у main.py приймає файл, перевіряє його розмір, викликає extract_text_from_file, а вже після цього формує метадані документа.

```
@app.post("/api/documents/extract")
async def extract_document_text( file: UploadFile = File(...), encoding: str = Form("utf-8") ):
    content = await file.read()
    if len(content) > MAX_FILE_SIZE_BYTES: raise HTTPException( status_code=413,
detail="Файл завеликий. Максимальний дозволений розмір — 10 MB." )
    text, file_format = extract_text_from_file( filename=file.filename or "document.txt",
        content=content, encoding=encoding )
    is_pdf = ( (file_format or "").upper() == "PDF" or (file.filename or
"").lower().endswith(".pdf") )
    if is_pdf: text = normalize_pdf_extracted_text(text)
    if len(text) > MAX_TEXT_CHARACTERS:
        raise HTTPException( status_code=413, detail=f"Після зчитування файл містить понад
{MAX_TEXT_CHARACTERS} символів. Скоротіть текст або завантажте менший фрагмент." )
    metadata = build_metadata( text=text, source_type="file", file_format=file_format )
    metadata["fileSizeBytes"] = len(content)
    metadata["fileSize"] = len(content)
    return { "title": file.filename, "content": text, "fileFormat": file_format,
"metadata": metadata }
```

Рисунок 4.15 – Лістинг точки входу результату початкової обробки документа

Автентифікація та контроль доступу

Окремим модулем серверної частини є security.py. Для захищених точок доступу використовується залежність get_current_user (рис. 4.16). Також у цьому модулі оновлюється час останньої активності користувача, що використовується для визначення онлайн-статусу в адміністративній панелі.

```
def get_current_user(authorization: str | None = Header(default=None), db: Session =
Depends(get_db)) -> User:
    if not authorization or not authorization.startswith("Bearer "): raise
HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Потрібна авторизація")
    raw_token = authorization.replace("Bearer ", "", 1)
    session = db.query(AuthSession).filter(AuthSession.tokenHash ==
token_hash(raw_token)).first()
    if not session or session.revokedAt is not None or session.expiresAt < datetime.utcnow():
raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Сесія недійсна")
    session.deviceInfo = { **(session.deviceInfo or {}), "lastSeenAt":
datetime.utcnow().isoformat() }
    flag_modified(session, "deviceInfo")
    db.commit()
    user = db.get(User, session.userId)
    if not user or user.status != "active": raise
HTTPException(status_code=status.HTTP_403_FORBIDDEN, detail="Обліковий запис неактивний")
    return user
```

Рисунок 4.16 – Лістинг коду функції перевірки поточного користувача security.py

Захищені запити не працюють без токена. Для системи, де існують різні ролі (дослідник – редактор – адміністратор), доцільно визначати користувача за активною сесією. Для обмеження доступу до рецензування або адміністрування передбачено функцію require_role(); backend самостійно перевірить токен і статус облікового запису.

Робота з документами

Збереження документа реалізовано через точку входу /api/documents. Перед збереженням перевіряється якісні характеристики тексту, формуються метадані та створюється перша версія документа – оригінал (рис. 4.17). Це дозволяє надалі працювати з історією змін.

```
@app.post("/api/documents")
def create_document( payload: DocumentIn, user: User = Depends(get_current_user), db:
Session = Depends(get_db) ):
    content = payload.content or ""
    title = payload.title.strip() or "Новий документ"
    if payload.sourceType == "url" and content.startswith(("http://", "https://")):
        try:
            with urllib.request.urlopen(content, timeout=10) as response:
                raw_bytes = response.read(1_000_000)
                encoding = response.headers.get_content_charset() or "utf-8"
                raw = raw_bytes.decode(encoding, errors="ignore")
            content = extract_text_from_html_with_paragraphs(raw)
            if len(content) > MAX_TEXT_CHARACTERS:
                raise HTTPException( status_code=413, detail=f"Після зчитування URL текст
містить понад {MAX_TEXT_CHARACTERS} символів. Скоротіть матеріал або використайте інше
джерело." )
            title = title if title != "Новий документ" else payload.content[:120]
        except HTTPException: raise
        except Exception as exc: raise HTTPException( status_code=400, detail=f"Не
вдалося завантажити URL: {exc}" )
    if len(content) < 2: raise HTTPException(status_code=400, detail="Документ порожній")
    if len(content) > MAX_TEXT_CHARACTERS: raise HTTPException( status_code=413,
detail=f"Текст документа завеликий. Дозволено {MAX_TEXT_CHARACTERS} символів." )
    meta = build_metadata(content, payload.sourceType, payload.fileFormat)
    doc = Document( ownerId=user.userId, title=title, sourceType=payload.sourceType,
fileFormat=payload.fileFormat, language=payload.language or meta["languageDetected"],
content=content, docMeta=meta )
    db.add(doc)
    db.flush()
    db.add( DocumentVersion( documentId=doc.documentId, versionNumber=1,
content=doc.content, changeComment="Початковий імпорт" ) )
    log_event(db, "IMPORT_DOCUMENT", user.userId, "Document", doc.documentId, meta)
    db.commit()
    db.refresh(doc)
    return document_out(doc, include_content=True)
```

Рисунок 4.17 – Точка входу роботи із документами main.py

Запуск NLP-аналізу

Основна перевірка тексту реалізована через /api/analysis/run-text. Він приймає текст, передає його в NLP-модуль і повертає структурований результат (рис. 4.18).

```
@app.post("/api/analysis/run-text")
def run_text_analysis(payload: RunTextAnalysisRequest, db: Session = Depends(get_db)):
    active_tools = get_active_quality_tools(db)
    analysis = check_text( payload.text, active_tools=active_tools, language="uk" )
    return {
        "taskType": payload.taskType,
        "documentId": payload.documentId,
        "status": "completed",
        "summary": analysis.get("summary", {}),
        "fullResult": {
            "text": payload.text,
            "normalized": analysis.get("normalized", payload.text),
            "tokens": analysis.get("tokens", []),
            "entities": analysis.get("entities", []),
            "syntax": analysis.get("syntax", []),
            "suggestions": analysis.get("suggestions", []),
            "pii": analysis.get("pii"),
            "modelInfo": analysis.get("modelInfo")
        }
    }
```

Рисунок 4.18 – Точка входу запуску аналізу main.py

Реалізація NLP-модуля

Лінгвістична обробка винесена у файл nlp.py. Завдяки ньому можливий робочий процес опрацювання текстової інформації й інтеграція з LanguageTool. Функція отримання і формування метаданих зображена на рис. 4.19. Завдяки їй кожен документ отримує базові кількісні характеристики.

```
def build_metadata(text: str, source_type: str = "text", file_format: str = "TXT") ->
dict[str, Any]:
    tokens = tokenize(text)
    paragraphs = [
        paragraph.strip()
        for paragraph in re.split(r"\n\s*\n", text or "")
        if paragraph.strip()
    ]
    return {
        "sourceType": source_type,
        "fileFormat": file_format,
        "characters": len(text or ""),
        "tokens": len(tokens),
        "sentences": len([s for s in SENT_RE.findall(text or "") if s.strip()]),
        "paragraphs": len(paragraphs),
        "languageDetected": detect_language(text),
        "privacy": "Система",
    }
```

Рисунок 4.19 – Лістинг коду функції отримання даних файлу nlp.py

Перевірка правопису, граматики та стилістики

Система використовує комбінований підхід до перевірки тексту. Backend звертається до LanguageTool як до основного механізму перевірки. Локальні правила LinguaInsight працюють як доповнення (додаток А) і не конфліктують з іншими результатами (рис. 4.20).

```
def check_text( text: str, active_tools: list[dict] | None = None, language: str = "uk" )
-> dict:
    active_tools = active_tools or []
    languagetool_tool = next(
        ( tool for tool in active_tools if tool.get("modelType") == "languagetool-http"
            and tool.get("status") == "active"
        ), None
    )
    languagetool_suggestions = []
    if languagetool_tool:
        endpoint_url = languagetool_tool.get("artifactPath") or
"https://api.languagetool.org/v2/check"
        matches = call_languagetool( text=text, language=language or
languagetool_tool.get("language") or "uk", endpoint_url=endpoint_url )
        languagetool_suggestions = map_languagetool_suggestions(text, matches)
    local_result = check_text_linguainsight_rules(text)
    local_suggestions = local_result.get("suggestions", [])
    merged_suggestions = merge_suggestions( primary=languagetool_suggestions,
secondary=local_suggestions )
    summary = local_result.get("summary", {})
    summary["languageToolSuggestions"] = len(languagetool_suggestions)
    summary["linguaInsightSuggestions"] = len(local_suggestions)
    summary["totalSuggestions"] = len(merged_suggestions)
    return {
        **local_result,
        "summary": summary,
        "suggestions": merged_suggestions,
        "modelInfo": {
            "primary": "LanguageTool" if languagetool_tool else "LinguaInsight",
            "fallback": "LinguaInsight rules",
            "conflictPolicy": "LanguageTool priority, LinguaInsight complements only"
        }
    }
```

Рисунок 4.20 – Лістинг коду функції перевірки тексту nlp.py

Корпуси та вибір текстів для аналізу

У системі передбачено аналіз як окремого документа, так і цілого корпусу (рис. 4.21). Для цього в main.py використовується службова функція _get_texts_for_run(). Якщо користувач передає documentId, сервер повертає один документ. У випадку передачі corpusId, сервер об’єднує тексти документів відповідного корпусу.

```
def _get_texts_for_run(db: Session, user: User, document_id: str | None, corpus_id: str |
None) -> tuple[list[Document], str]:
    if document_id:
        doc = db.get(Document, document_id)
        if not doc or doc.ownerId != user.userId:
            raise HTTPException(status_code=404, detail="Документ не знайдено")
        return [doc], doc.content
    if corpus_id:
        corpus = db.get(Corpus, corpus_id)
        if not corpus or corpus.ownerId != user.userId:
            raise HTTPException(status_code=404, detail="Корпус не знайдено")
        docs = [cd.document for cd in corpus.documents]
        return docs, "\n".join(d.content for d in docs)
    raise HTTPException(status_code=400, detail="Потрібно вказати documentId або
corpusId")
```

Рисунок 4.21 – Лістинг коду функції вибору складових для аналізу main.py

Побудова *n*-грам і підготовка даних для візуалізацій

Точка входу /api/ngrams/run запускає побудову *n*-грам (додаток А) і повертає масив елементів із частотами (рис. 4.22). Саме ці дані потім використовуються для побудови таблиці, графіка частот, ключових фраз і PDF-звіту.

```
@app.post("/api/ngrams/run")
def run_ngrams(payload: NgramRunIn, user: User = Depends(get_current_user), db: Session =
Depends(get_db)):
    docs, _ = _get_texts_for_run(db, user, payload.documentId, payload.corpusId)
    task = AnalysisTask(userId=user.userId, documentId=payload.documentId,
corpusId=payload.corpusId, taskType="ngram", status="completed", parameters={"n": payload.n,
"minFrequency": payload.minFrequency}, startedAt=datetime.utcnow(), finishedAt=datetime.utcnow())
    db.add(task)
    db.flush()
    ng_result = NgramResult(taskId=task.taskId, n=payload.n,
minFrequency=payload.minFrequency, filters={})
    db.add(ng_result)
    db.flush()
    items = ngram_analysis( texts=[d.content for d in docs], n=payload.n,
min_frequency=payload.minFrequency,
        language=payload.language or "uk", stop_words_profile=payload.stopWordsProfile or
"academic",
        pos_method=payload.posMethod or "hybrid", max_results=payload.maxResults )
    for item in items:
        db.add(NgramItem( ngramResultId=ng_result.ngramResultId, ngramText=item["ngramText"],
            frequency=item["frequency"],
relativeFrequency=Decimal(str(item["relativeFrequency"])),
            pmi=Decimal(str(item["pmi"])), ))
    log_event(db, "BUILD_NGRAMS", user.userId, "NgramResult", ng_result.ngramResultId)
    db.commit()
    return {
        "ngramResultId": ng_result.ngramResultId,
        "parameters": { "n": payload.n, "minFrequency": payload.minFrequency, "language":
payload.language,
            "stopWordsProfile": payload.stopWordsProfile, "posMethod": payload.posMethod,
"maxResults": payload.maxResults },
        "items": items
    }
```

Рисунок 4.22 – Точка входу запуску визначення послідовностей main.py

Рецензування пропозицій виправлень

Контроль автоматичної перевірки тексту може виконуватись редактором або адміністратором. Точка входу /api/suggestions повертає список пропозицій, а /api/suggestions/{suggestion_id}/decision дозволяє прийняти або відхилити виправлення та зберегти коментар (рис. 4.23).

```
@app.post("/api/suggestions/{suggestion_id}/decision")
def decide_suggestion( suggestion_id: str, payload: SuggestionDecisionIn, user: User =
Depends(get_current_user), db: Session = Depends(get_db) ):
    if not is_editor_or_admin(user):
        raise HTTPException(status_code=403, detail="Доступ лише для редактора або
адміністратора")
    suggestion = db.get(Suggestion, suggestion_id)
    if not suggestion: raise HTTPException(status_code=404, detail="Пропозицію не
знайдено")
    suggestion.status = payload.decision
    annotation = Annotation( suggestionId=suggestion_id, editorId=user.userId,
decision=payload.decision, comment=payload.comment, correctedText=payload.correctedText )
    db.add(annotation)
    log_event( db, "REVIEW_SUGGESTION", user.userId, "Suggestion", suggestion_id,
        { "decision": payload.decision, "comment": payload.comment, "correctedText":
payload.correctedText } )
    create_suggestion_decision_notification( db=db, suggestion=suggestion, reviewer=user,
decision=payload.decision, comment=payload.comment )
    db.commit()
    db.refresh(suggestion)
    db.refresh(annotation)
    return {
        "message": "Рішення збережено",
        "status": suggestion.status,
        "journalItem": {
            "journalId": getattr(annotation, "annotationId", None) or
f"{suggestion_id}_{annotation.createdAt}",
            "decision": annotation.decision,
            "comment": annotation.comment,
            "correctedText": annotation.correctedText,
            "authorName": user.username,
           "createdAt": annotation.createdAt.isoformat() if annotation.createdAt else None
        }
    }
```

Рисунок 4.23 – Точка входу розгляду пропозицій main.py

Мовна модель у контексті вебзастосунку

У контексті даного вебзастосунку мовна модель виступає програмним компонентом або набіром файлів для автоматичної обробки природної мови. У системі «LinguaInsight» використовується модуль NLP. У поточній реалізації основною є LanguageTool-перевірка та збереження можливості розширення через spaCy, Stanza, Hugging Face Transformers або окремий сервіс.

Структура мовної моделі залежить від бібліотеки або платформи використання. Для трансформерних моделей із Hugging Face, типовою є структура з файлів конфігурації, ваг моделі, токенайзер-файлів і метаданих. Для зберігання моделей і датасетів працює платформа Hugging Face Hub. spaCy надає готові мовні pipeline-моделі для опрацювання. LanguageTool можна підключити через публічний HTTP API або запустити локально впроваджений HTTP сервер.

Серверна частина «LinguaInsight» виконує роль центрального шару. Вона забезпечує повний цикл роботи з текстом.

4.4 Оформлення інтерфейсу користувача

Інтерфейс вебзастосунку лінгвістичного аналізу тексту «LinguaInsight» побудовано таким чином, щоб користувач міг швидко перейти до основних функцій системи. Основна увага приділена візуальному поділу функціональних блоків і відображенню стану обробки текстових даних.

Цільова сторінка містить форму авторизації та реєстрації (рис. 4.24), а також представляє основні можливості вебзастосунку. Тут можна ознайомитись із функціоналом платформи до входу в систему. Окремо передбачено блок публічних корпусів, доступних для перегляду (рис. 4.25).

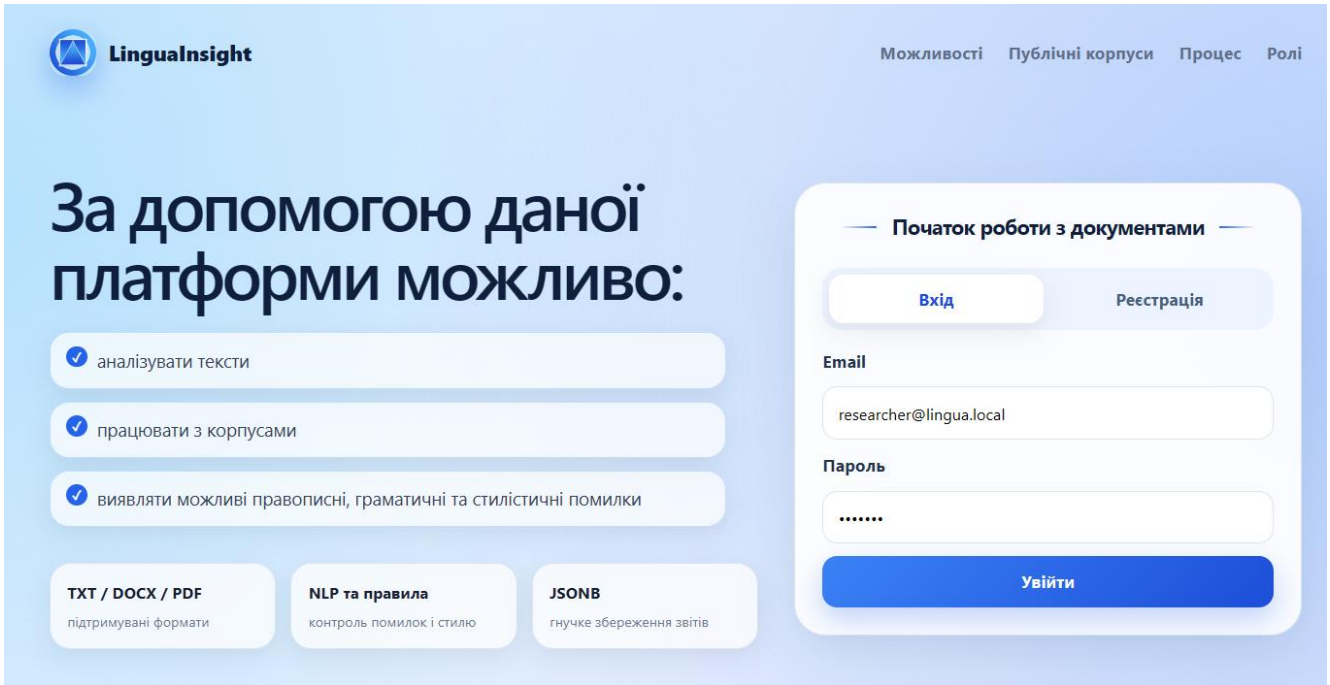

Рисунок 4.24 – Цільова сторінка вебзастосунку

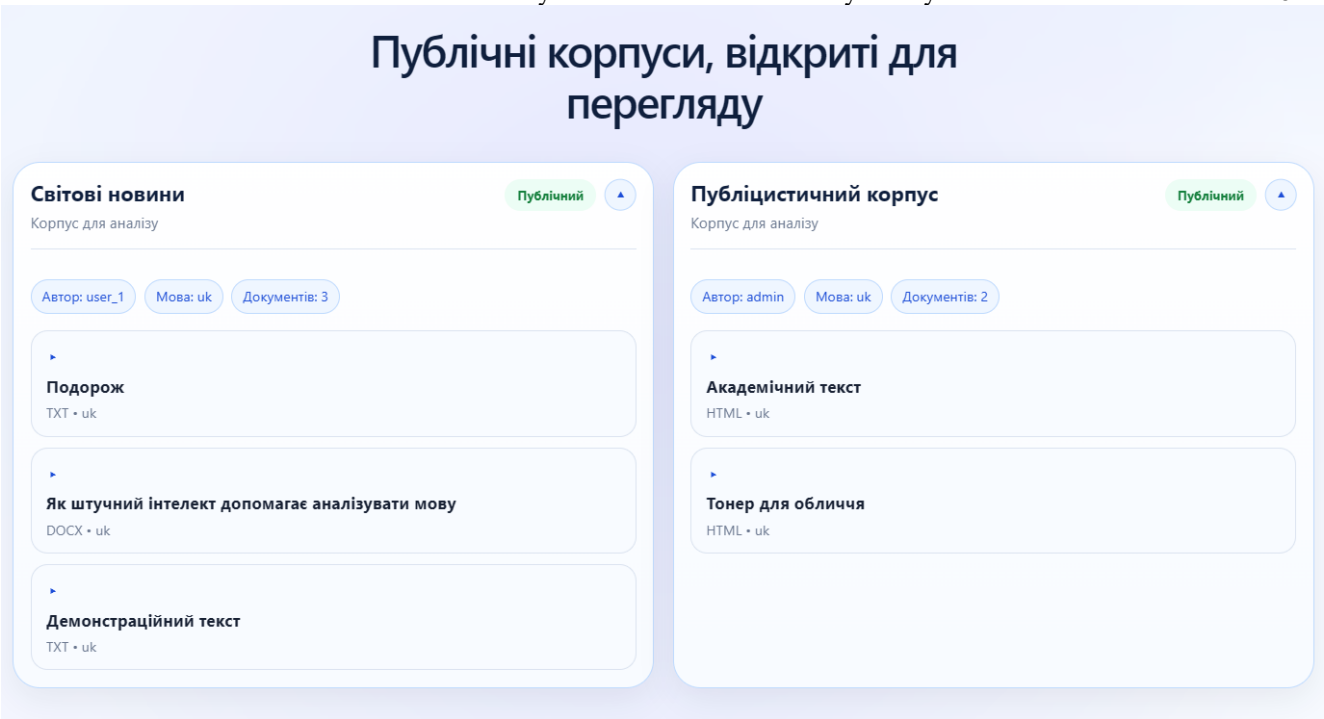

Рисунок 4.25 – Оформлення публічних корпусів у гостьовому режимі

Головна панель виконує роль робочого центру користувача, оскільки коротко показує поточний стан системи й надає швидкий доступ до найважливіших функцій (рис. 4.26). На сторінці представлено блок швидких дій, через який можна перейти до відповідної вкладки меню.

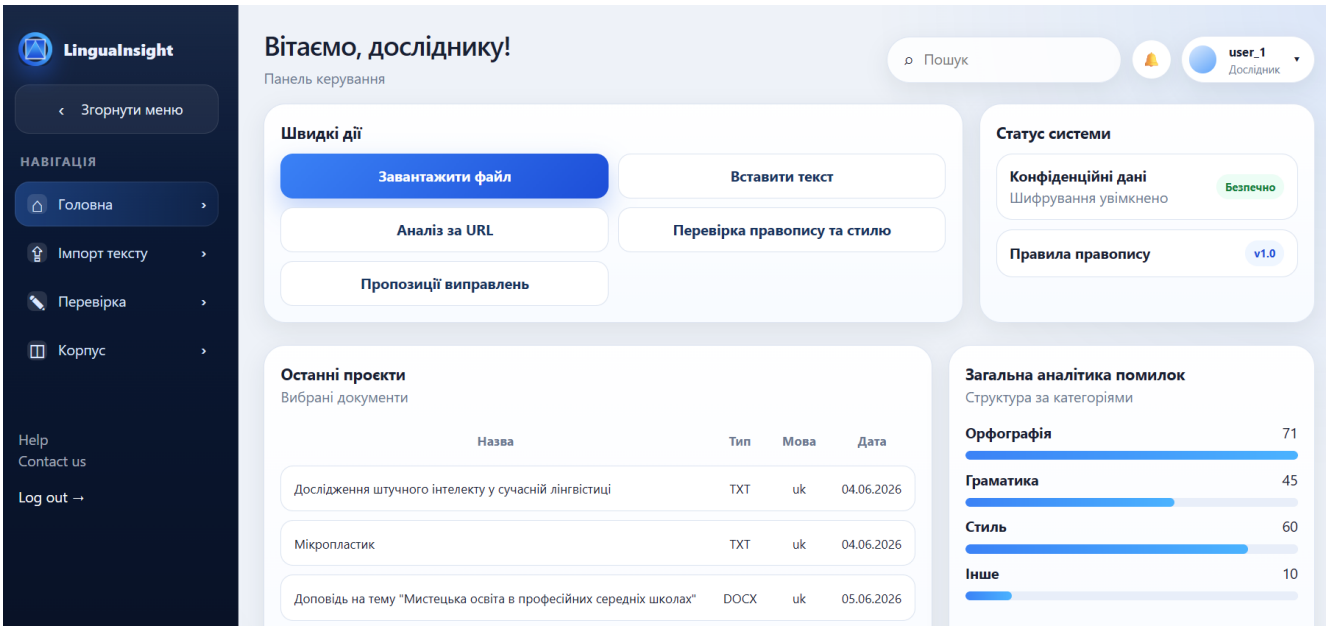

Рисунок 4.26 – Інтерфейс головної панелі

Однією з основних сторінок є сторінка імпорту тексту (рис. 4.27). Дані можна ввести вручну або завантажити файлом. Отримані в результаті зчитування метадані дозволяють ще до запуску повного аналізу оцінити структуру документа та підготувати його до подальшої обробки.

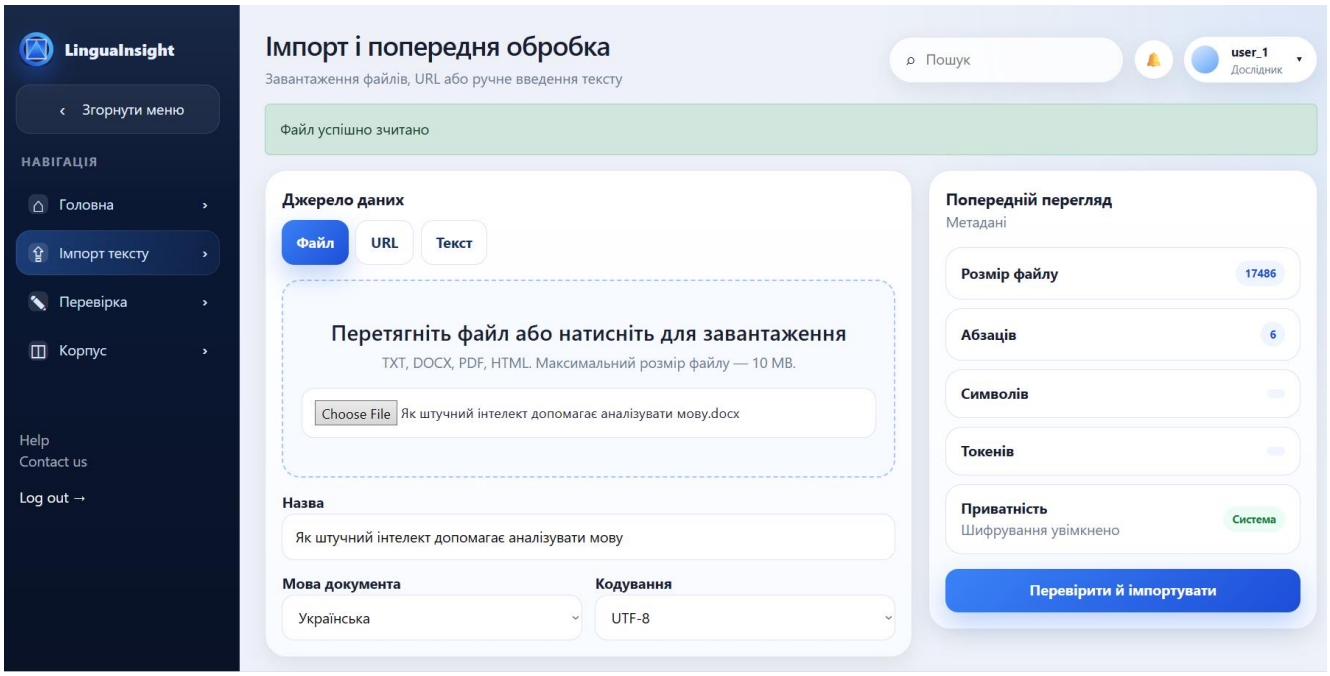

Рисунок 4.27 – Інтерфейс сторінки імпорту тексту

Перевірка правопису та стилю дає змогу користувачу обрати документ і запустити аналіз (рис. 4.28). Панель деталей надає зручності для комфортної роботи. На сторінці також реалізовано фільтрацію знайдених зауважень за типами: правопис, граматика, синтаксис і стиль (рис. 4.29).

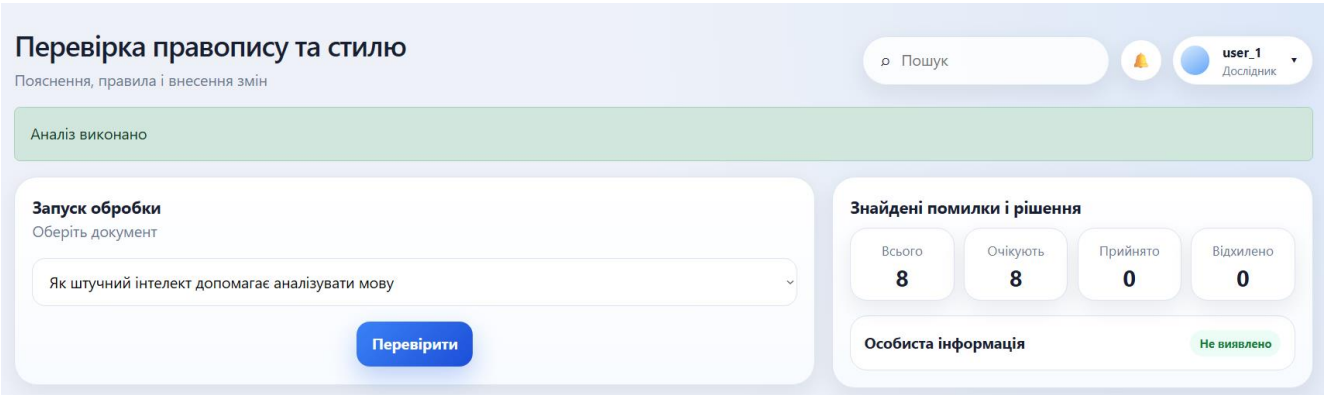

Рисунок 4.28 – Блоки вибору документа та цифрових показників

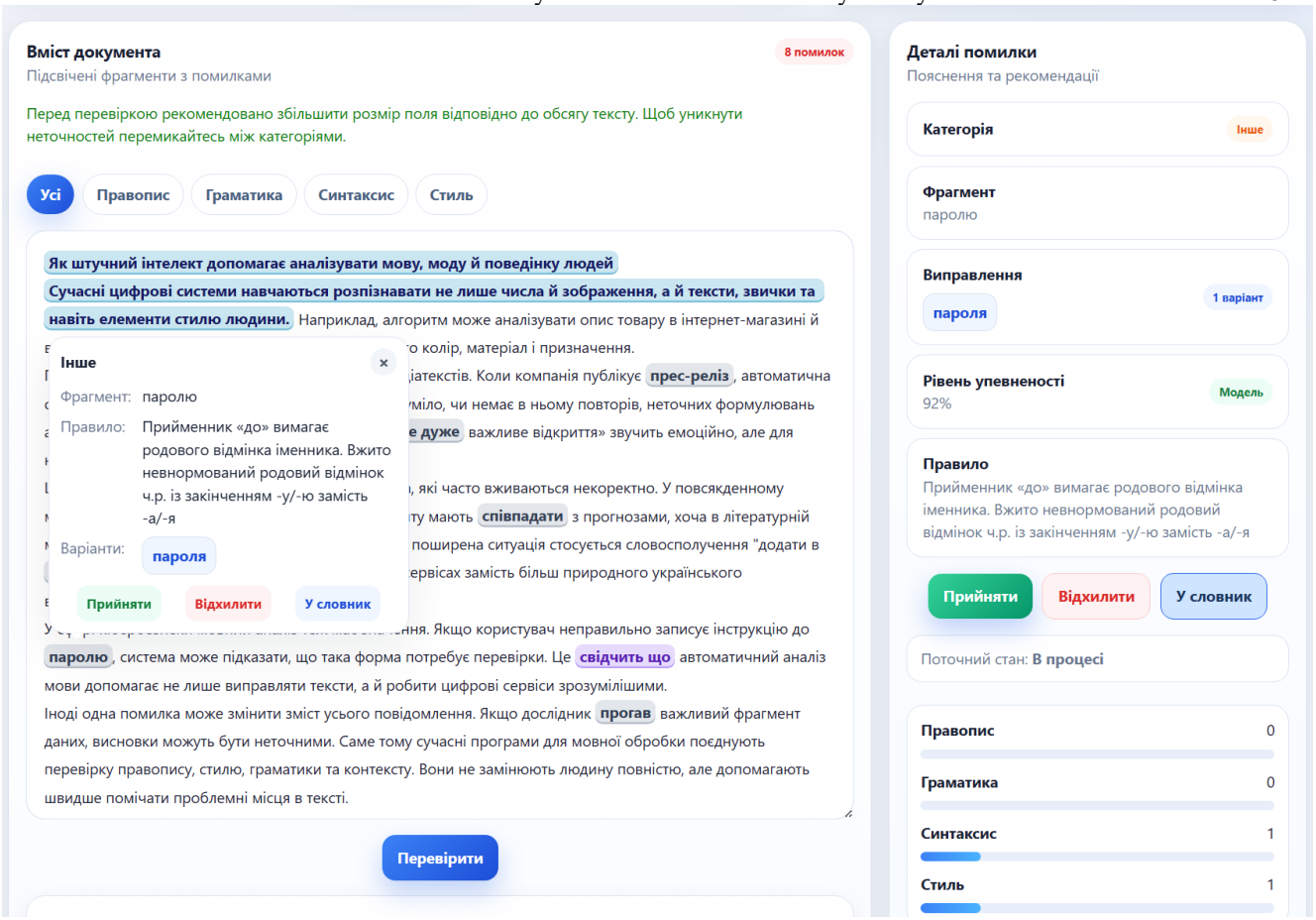

Рисунок 4.29 – Інтерфейс сторінки перевірки правопису та стилю

Функціонал створення корпусів, додавання до них документів, запуск визначення послідовностей надає окремий модуль (рис. 4.30). Результати подаються у вигляді таблиці з частотами.

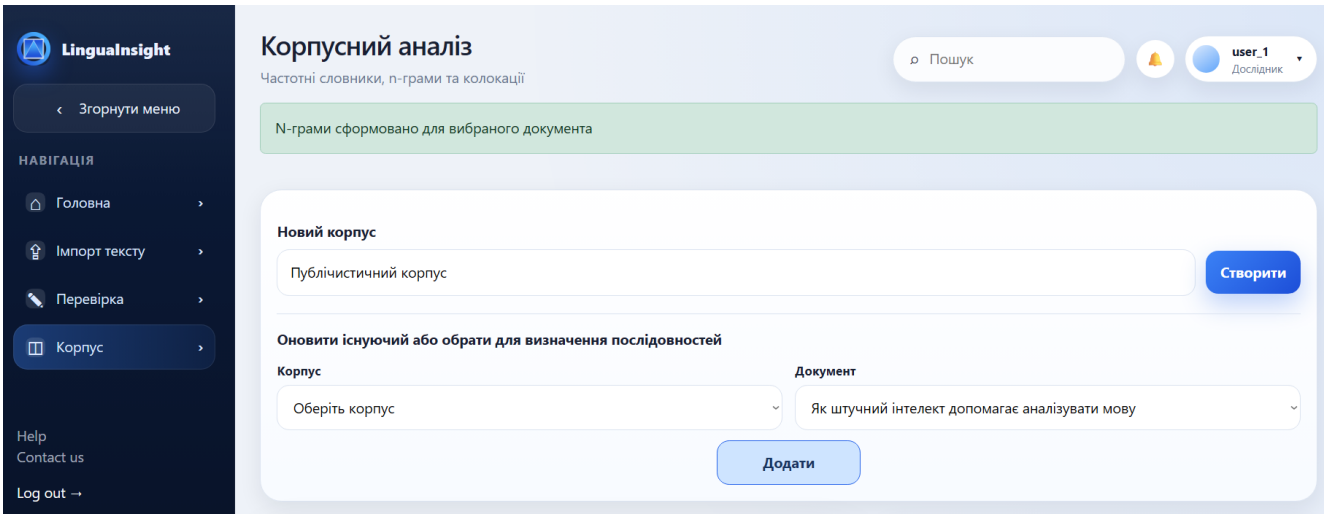

Рисунок 4.30 – Інтерфейс сторінки корпусного аналізу та побудови *n*-грам

Сторінка корпусного аналізу містить також візуальні елементи: графік частот і хмару ключових слів та фраз. Графік дає змогу швидко побачити найпоширеніші послідовності (рис. 4.31), а хмара слів візуально виділяє важливі одиниці тексту.

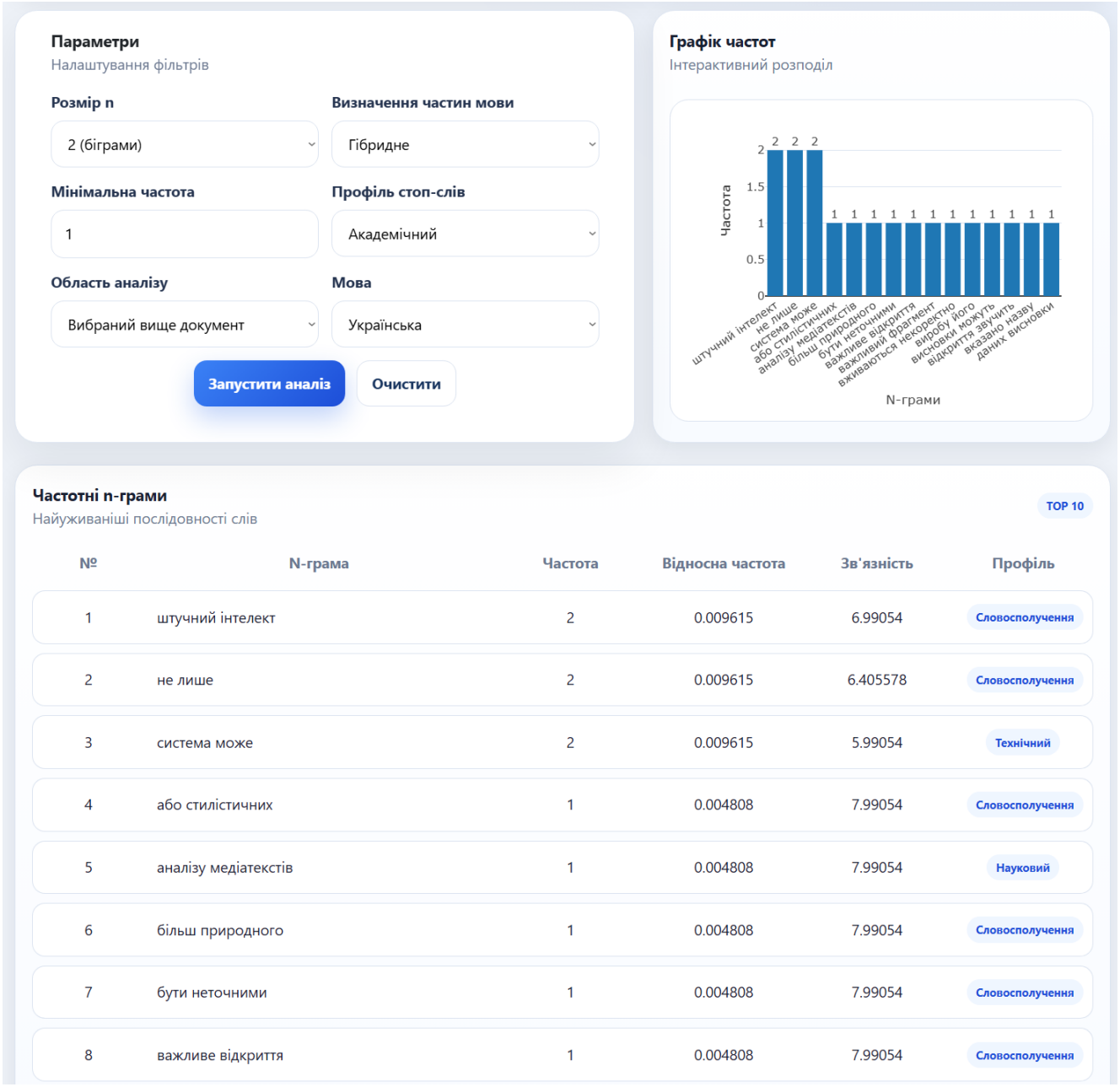

Рисунок 4.31 – Робота з частотними послідовностями

Окремо передбачено експорт результатів у CSV, JSON і PDF (рис. 4.32). PDF-звіт формується як структурований документ із параметрами аналізу, графіком, ключовими фразами та таблицею всіх сформованих *n*-грам (останнє наведено в додатку Б).

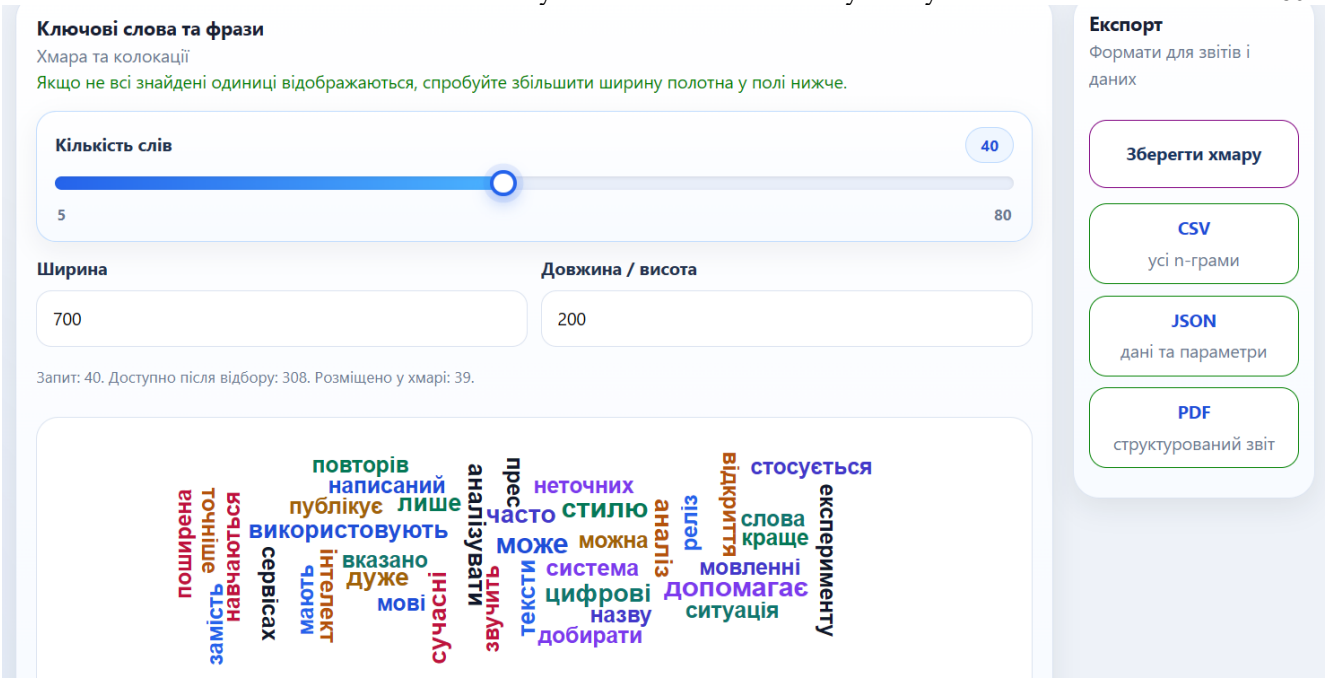

Рисунок 4.32 – Хмара слів та можливості експорту

Для редактора передбачено окрему сторінку рецензування (рис. 4.33). Вона містить список знайдених пропозицій, контекст і поле коментаря.

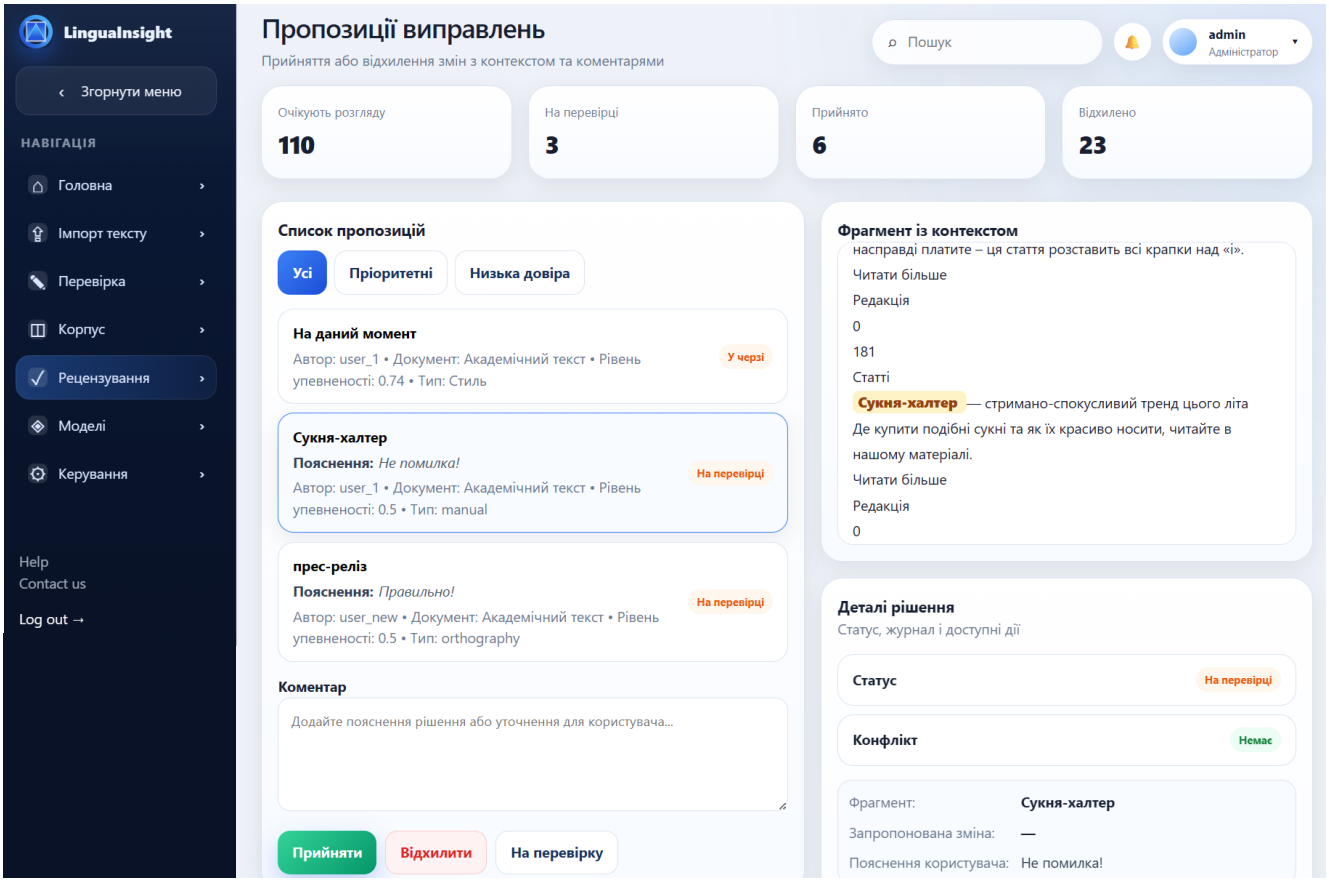

Рисунок 4.33 – Інтерфейс сторінки рецензування пропозицій виправлень

Праворуч на рис. 4.33 розміщено панель деталей рішення, де показано доступні дії та показники якості. Редактор може прийняти/відхилити виправлення та/або провести подальший розгляд.

Адмін-панель призначена для керування моделями та користувачами. Обсяг роботи адміністратора полягає у підключенні нових моделей або оновлення їх версій (рис. 4.34).

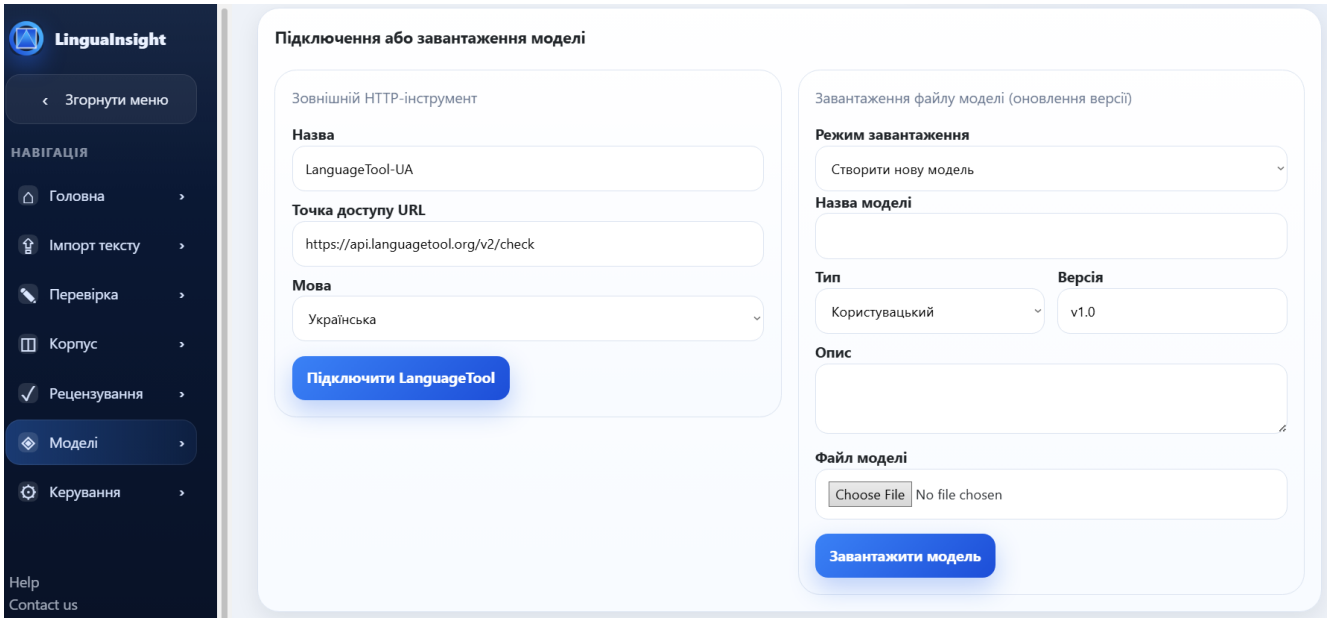

Рисунок 4.34 – Адміністративна панель вебзастосунку

Також адміністратору доступний список користувачів (рис. 4.35) та можливість змінювати їхні ролі. Завдяки цьому система підтримує розмежування прав та обов’язків.

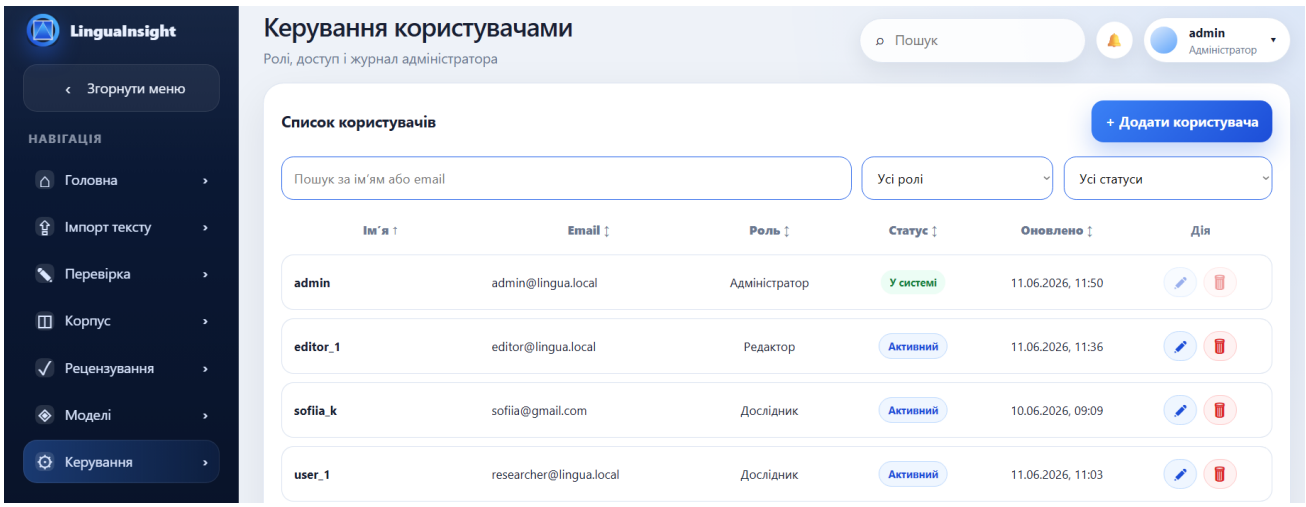

Рисунок 4.35 – Призначення користувачу іншої ролі в адмін-панелі

Інтерфейс призначення ролі користувачу наведений на рис. 4.36. Завдяки такому механізму процес адміністрування більш контрольований. Додатково система не дозволяє адміністратору змінювати власну роль.

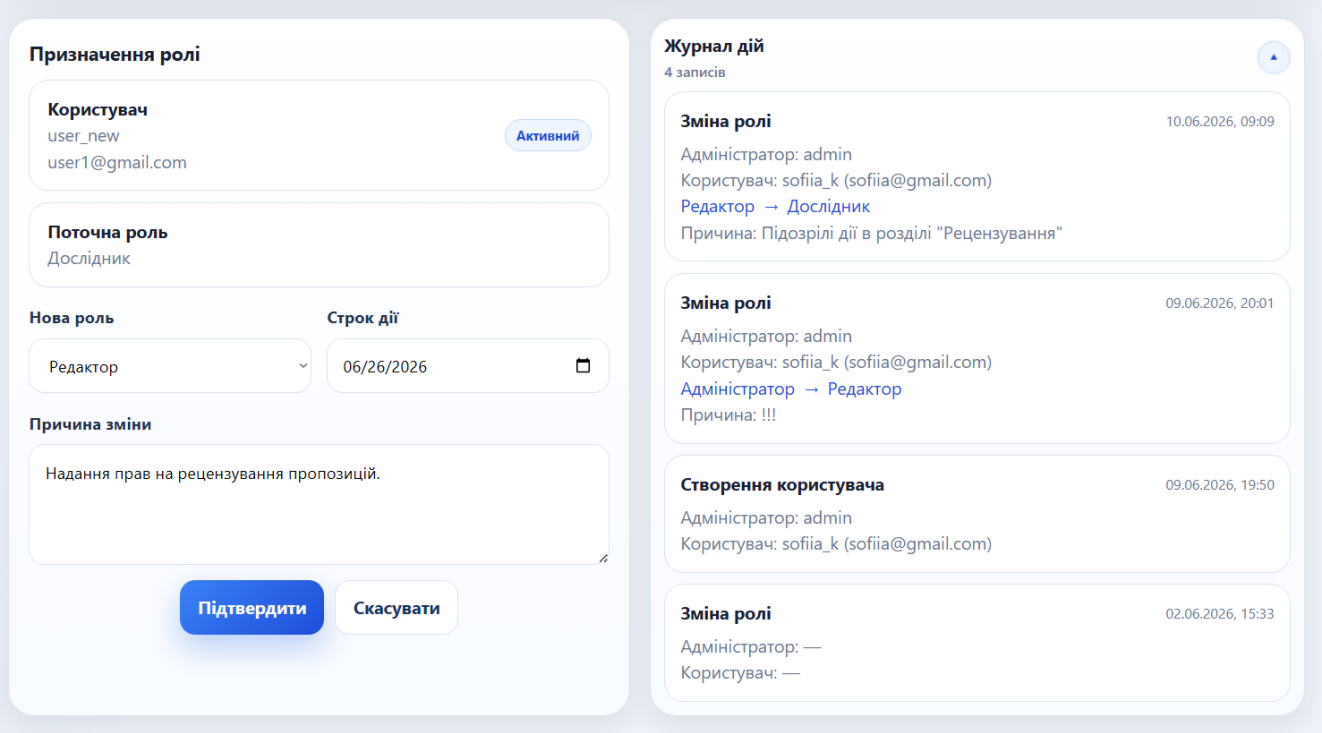

Рисунок 4.36 – Призначення ролі користувачу

У вебзастосунку також реалізовано внутрішню пошту. Це зроблено з метою інформування користувача про важливі події (зміну ролі або рішення щодо пропозиції). Завдяки цьому він може бачити службові сповіщення безпосередньо в інтерфейсі.

Окремим елементом інтерфейсу є профіль користувача з відображенням основних даних облікового запису (рис. 4.37). Тут можна переглядати власні матеріали, контролювати доступність корпусів і працювати з особистим словником. Записи, додані в ході перевірки текстів, можуть використовуватись при виборі профілю стоп-слів для корпусного аналізу. Наявність профілю робить систему більш персоналізованою.

Інтерфейс «LinguaInsight» підтримує повний шлях користувача під час роботи з текстом. Він не перевантажує його технічними деталями, але водночас показує важливі стани системи.

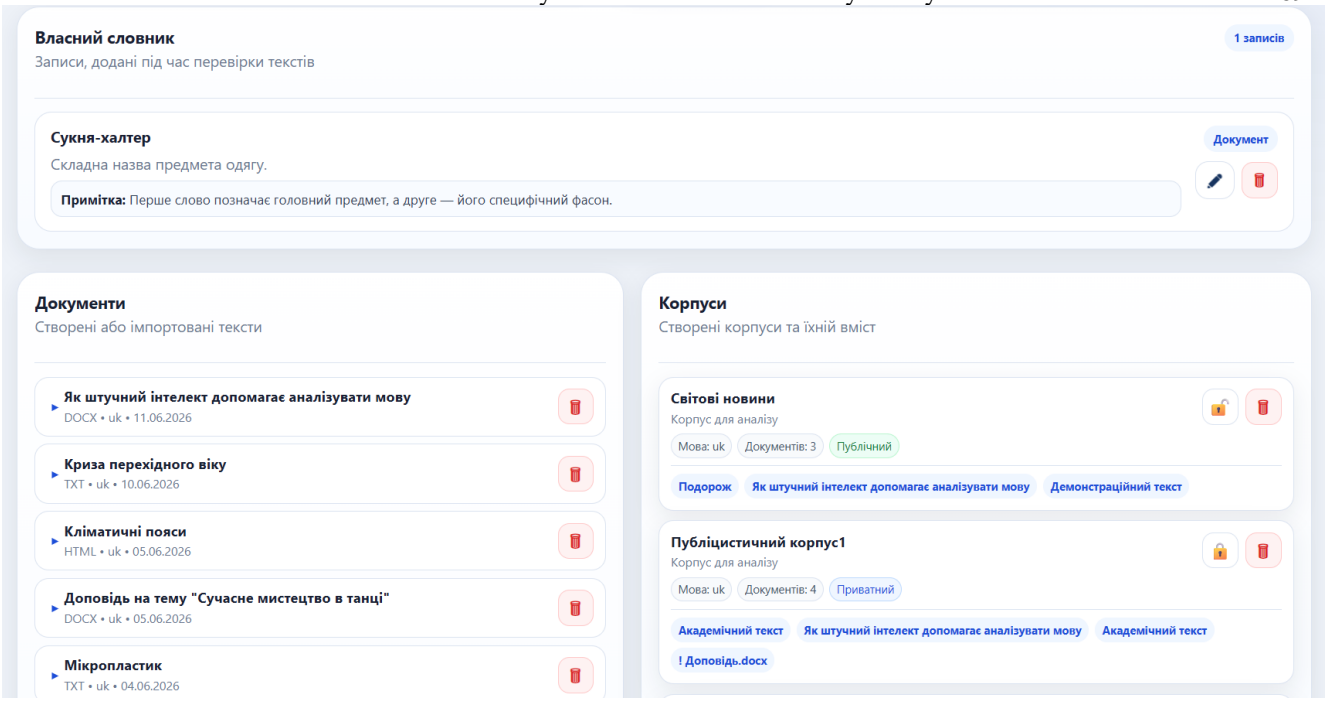

Рисунок 4.37 – Фрагмент профілю користувача

Отже, така побудова інтерфейсу робить вебзастосунок зручним як для дослідника, так і для редактора чи адміністратора. Позитивною особливістю є легка адаптація для широкого кола користувачів.

4.5 Тестування роботи вебзастосунку

Метою тестування вебзастосунку «LinguaInsight» є перевірка коректності роботи основних модулів системи. Оскільки вона працює з текстовими даними й має кілька ролей доступу, цей процес охоплює правильність обчислень і безпеку запитів. Також важливо перевірити стабільність роботи API.

Тестування виконувалося на кількох рівнях:

- службові методи опрацювання тексту перевірялися окремими функціями;
- основні точки входу backend перевірялися через API;
- повний шлях користувача перевірявся через інтерфейс.

Призначення консольного тестування полягає у перевірці роботи backend через послідовність автоматизованих сценаріїв, наближених до реальної роботи користувача. Фрагмент коду тестового скрипта наведено на рис. 4.38.

Негативне тестування дає змогу перевірити реакцію системи на відсутність певних вхідних даних. Це важливо, оскільки backend має виконувати правильні запити і коректно блокувати заборонені дії.

Після виконання тестів скрипт автоматично формує звіти у форматах JSON і Markdown у директорії test_reports.

```
load_scenarios = [
    {
        "name": "Список документів",
        "method": "GET",
        "url": f"{base_url}/documents",
        "kwargs": {"headers": headers}
    },
    {
        "name": "NLP-перевірка тексту",
        "method": "POST",
        "url": f"{base_url}/analysis/run-text",
        "kwargs": {"headers": headers, "json": analysis_payload}
    },
    {
        "name": "Побудова n-грам",
        "method": "POST",
        "url": f"{base_url}/ngrams/run",
        "kwargs": {"headers": headers, "json": ngram_payload}
    }
]

for scenario in load_scenarios:
    stat = run_load_scenario(
        name=scenario["name"],
        method=scenario["method"],
        url=scenario["url"],
        total_requests=args.requests,
        concurrency=args.concurrency,
        **scenario["kwargs"]
    )
```

Рисунок 4.38 – Лістинг коду скрипта навантажувального тестування

Наведений фрагмент демонструє принцип навантажувального тестування. Для кожного сценарію виконується задана кількість HTTP-запитів із визначеним рівнем паралельності. Після цього обчислюються середній час відповіді, p50 (медіанний час відповіді), p95 (час, у межах якого виконалися 95 % запитів) та інші показники. Серед параметрів консольного тестування: одночасні потоки – 10, запити на сценарій – 50, множник довжини тестового тексту – 3. Проведення функціонального консольного тестування показало результати, зазначені у табл. 4.4.

Таблиця 4.4 – Результати функціонального консольного тестування API

Сценарій	**Точка входу**	**Час, мс**	**Наявний результат**	**Примітка**
Перевірка доступності API	GET /health	8,69	Успішно	–
Реєстрація тестового користувача	POST /auth/register	34,67	Успішно	–
Вхід тестового користувача	POST /auth/login	34,20	Успішно	–
Отримання поточного користувача	GET /auth/me	12,14	Успішно	–
Зчитування текстового файлу	POST /documents/extract	2,95	Успішно	Токенів: 90
Створення документа	POST /documents	28,01	Успішно	–
Отримання списку документів	GET /documents	8,07	Успішно	–
NLP-перевірка тексту	POST /analysis/run-text	18475,16	Успішно	Зауважень: 9, сутностей: 12
Створення корпусу	POST /corpora	47,31	Успішно	–
Додавання документа до корпусу	POST /corpora/{id}/documents/{id}	39,35	Успішно	–
Зміна видимості корпусу	PATCH /corpora/{id}/visibility	35,35	Успішно	–
Перевірка публічних корпусів	GET /public/corpora	8,14	Успішно	–
Побудова *n*-грам	POST /ngrams/run	36,88	Успішно	*N*-грам: 19
Отримання головної панелі	GET /dashboard	54,49	Успішно	–

Відповідно до результатів функціонального тестування, всі основні сценарії backend виконалися успішно. Система коректно пройшла шлях від створення тестового користувача до формування корпусу україномовних файлів. Найбільший час виконання має NLP-перевірка тексту – 18475,16 мс. Це пояснюється більшою складністю мовної обробки порівняно зі звичайними CRUD-запитами.

Таблиця 4.5 – Якісні показники NLP та корпусного аналізу

Показник	**Значення**
Токени після зчитування	90
Знайдені зауваження	9
Знайдені іменовані сутності	12
PII-елементи	9
Мова аналізу	uk
Сформовані *n*-грами	19

Табл. 4.5 демонструє показники якості роботи NLP під час виконання поставлених задач. Отримані дані підтверджують, що NLP-модуль наряду із

загальним статусом виконання повертає ще і змістовні результати аналізу. У контрольному тексті (рис. 4.39) виявлено 9 мовних зауважень, 12 іменованих сутностей і 9 потенційних PII-елементів. Також система визначила мову як українську та сформувала 19 *n*-грам для корпусного аналізу.

```
test_text = (
        "На даний момент система працює стабільно. "
        "Це свідчить що аналіз виконано. "
        "данні користувача перевірено. "
        "Іван Франко є важливою постаттю української культури. "
        "Контакт: test@example.com, IP: 127.0.0.1. "
    ) * max(1, args.text_multiplier)
```

Рисунок 4.39 – Контрольний текст

Навантажувальне тестування показало (табл. 4.6), що звичайні API-запити працюють стабільно при 10 паралельних потоках і 50 запитах на сценарій. Для усіх сценаріїв, окрім перевірки тексту, відсоток помилок становив 0 %. Значення p95 для цих сценаріїв не перевищило 1 с, що є прийнятним для навчально-дослідницького вебзастосунку.

Таблиця 4.6 – Результати навантажувального тестування API

Сценарій	**Запитів**	**Потоків**	**Помилок**	**Error, %**	**Average, мс**	**p50, мс**	**p95, мс**	**Max, мс**	**Throughput, запитів/с**
Список документів	50	10	0	0,0	141,34	116,37	372,29	537,40	62,59
Головна панель	50	10	0	0,0	193,42	151,73	461,42	613,86	46,81
NLP-перевірка тексту	50	10	1	2,0	12575,83	12945,66	25093,75	30000,92	0,68
Побудова *n*-грам	50	10	0	0,0	200,85	155,97	592,79	907,17	44,77

Найважчим сценарієм стала NLP-перевірка тексту. При 50 запитах і 10 потоках лише 1 запит завершився помилкою. Значення p95 для цього сценарію становило 25093,75 мс, а максимальний час відповіді – 30 000,92 мс. Це свідчить про те, що мовна обробка потребує більше часу й ресурсів, ніж звичайні операції читання. Ширший спектр роботи з матеріалом пояснює цю особливість.

За результатами консольного тестування встановлено, що основні сценарії роботи backend виконуються коректно. Негативні сценарії підтверджують, що захищені точки входу не доступні без авторизації або без відповідної ролі.

Проведене програмне тестування доповнене дослідженням інтерфейсу вручну. Виявлено, що система добре розпізнає складнопідрядні речення та чудово виправляє граматичні помилки. Натомість розроблений вебзастосунок гірше реагує на стилістичні помилки, що більше властиве саме людині з її авторським стилем. Трапляються випадки віднесення граматичних помилок до категорії «Інше» і низька впевненість моделей при наданні деталей опису. Іноді спостерігається накладання підказок одна на одну або за змістом і поясненнями.

Отже, проведене тестування підтверджує працездатність основних модулів «LinguaInsight» і здебільшого правильну взаємодію між frontend, backend, базою даних та NLP-логікою. Система готова до тестової експлуатації в навчальному або дослідницькому середовищі, а отримані результати дозволяють визначити напрям подальшого вдосконалення – насамперед оптимізацію ресурсоємних NLP-запитів.

Висновки до розділу 4

У даному розділі описано програмну реалізацію вебзастосунку «LinguaInsight» та показано, як окремі частини системи об’єднуються в єдине середовище. Реалізована структура проєкту має модульний характер. Клієнтську частину реалізовано як React-застосунок із логічно поділеними сторінками покрокової роботи з текстом. Структура інтерфейсу максимально відповідає вимогам користувача і підкреслює зручність орієнтування. Візуальні елементи, в тому числі графіки частот і хмари слів, роблять результати більш наочними. Серверна частина реалізує основну бізнес-логіку вебзастосунку. Всі наявні операції виконуються через REST API. Окремо реалізовано контроль доступу за ролями. Проведене тестування підтвердило працездатність основних модулів системи, задані функціональні сценарії виконалися успішно. Навантажувальне тестування показало, що звичайні API-запити працюють стабільно, а найбільш ресурсоємким процесом є NLP-перевірка тексту.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи спроєктовано та реалізовано вебзастосунок лінгвістичного аналізу тексту «LinguaInsight», що повністю відповідає поставленій меті. Розроблена система поєднує імпорт документів, перевірку правопису й стилю, рецензування та візуалізацію результатів. Робота призначена для автоматизації опрацювання текстових даних у навчальних, дослідницьких і прикладних задачах.

У ході виконання вирішено всі заплановані завдання:

– проаналізовано предметну область та існуючі сервіси-аналоги;

– визначено вимоги, розроблено сценарії використання для основних суб’єктів системи;

– спроєктовано архітектуру системи;

– визначено набір сервісів для покрокової обробки тексту;

– розроблено структуру бази даних для зберігання корпусів, метаданих та анотацій, а також механізм імпорту й експорту отриманої інформації;

– налаштовано взаємодію об’єктів зі створеними NLP-модулями та контейнеризацію;

– у клієнтській частині реалізовано інструменти візуалізації результатів (графи залежностей, виділення сутностей і хмари слів);

– проведено тестування та оцінено продуктивність системи на прикладних наборах даних.

У першому розділі досліджено предметну область. Установлено, що більшість наявних рішень не поєднують увесь обсяг необхідних функцій в одному середовищі. Це підтвердило доцільність створення власного вебзастосунку, який включає редакторські, дослідницькі й аналітичні інструменти.

У другому розділі сформовано архітектуру системи та визначено технологічний стек. Для клієнтської частини використано React, React Router і Axios, для серверної частини – FastAPI та SQLAlchemy, а для зберігання інформації – реляційну базу даних. Окрему увагу приділено моделюванню основних процесів і структурі взаємодії між компонентами системи.

У третьому розділі виконано проєктування системи, де описано її функціональну логіку, структуру даних і сценарії взаємодії користувачів. Враховано, що вебзастосунок має підтримувати роботу кількох ролей: гостя, дослідника, редактора та адміністратора. Результати автоматичної обробки можуть контролюватись людиною. Така модель дала змогу розмежувати права доступу та передбачити перевірку результатів аналізу вручну.

У четвертому розділі реалізовано програмну частину вебзастосунку. Клієнтська частина забезпечує зручну навігацію між елементами. Багато уваги приділено серверній частині, яка відповідає за обробку звернень. NLP-модуль побудований через комбінований підхід: поєднання зовнішнього інструменту з локальними правилами.

Проведене тестування підтвердило працездатність основних модулів «LinguaInsight». Для цього створено функціональні сценарії, які виконалися успішно. Навантажувальне тестування показало, що звичайні API-запити працюють стабільно, а найбільш трудомістким компонентом є NLP-перевірка тексту.

Система полегшує покрокове опрацювання тексту. Подальший розвиток «LinguaInsight» може бути пов’язаний із оптимізацією ресурсоємних запитів і розширенням механізму користувацьких словників. Перспективним також є винесення NLP-обробки в окремий сервіс або чергу задач для стабільнішої роботи з великими текстами.

Таким чином, кваліфікаційну роботу виконано у повному обсязі. Для цього проведено досконалий аналіз предметної області і виконано необхідне покрокове формування архітектури проєкту відповідно до вимог. Наприкінці проведено тестування та підтверджено працездатність системи.

Результатом проведеної роботи є вебзастосунок лінгвістичного аналізу тексту «LinguaInsight», який забезпечує автоматизацію обробки текстових даних. Її розширювана архітектура дозволяє підключати нові мовні моделі, що сприяє ефективному дослідженню мовних явищ у навчальних, дослідницьких і прикладних задачах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jurafsky D., Martin J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3rd ed. 2026. 626 p.

2. Федорін І. В. Основи обробки природної мови. Частина 1. Основи обробки тексту [Електронний ресурс] : практикум : навч. посіб. для здобувачів ступеня бакалавра за спеціальністю F3 Комп'ютерні науки. КПІ ім. Ігоря Сікорського; І. В. Федорін; – Київ: КПІ ім. Ігоря Сікорського, 2025. – 178 с.

3. Згуровський М. З. Обробка природної мови [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра, магістра які навчаються за спеціальностями 122 Комп'ютерні науки (F3 Комп'ютерні науки) та 124 Системний аналіз (F4 Системний аналіз та наука про дані). Київ : КПІ ім. Ігоря Сікорського, 2025. 229 с.

4. Grammarly. *Product Features.* URL: https://www.grammarly.com/features (Accessed: 06.02.2026).

5. LanguageTool – Spell and Grammar Checker. URL: https://languagetool.org (Accessed: 06.02.2026).

6. DeepL. *DeepL Write.* URL: https://www.deepl.com/en/write (Accessed: 06.02.2026).

7. Google Docs' AI-based grammar checker. URL: https://www.neowin.net/news/google-docs-ai-based-grammar-checker-now-available-to-a-wider-set-of-g-suite-users/ (Accessed: 09.02.2026).

8. Як перевіряти граматику й правопис у Google Документах. URL: https://support.google.com/docs/answer/57859?hl=uk&sjid=1521065284396141706-EU (дата звернення: 09.02.2026).

9. Voyant Tools Help. URL: https://voyant-tools.org/docs/ (Accessed: 10.02.2026).

10. Grammarly запустила вісім ШІ-агентів. Що вони можуть і для кого призначені. URL: https://dou.ua/lenta/news/grammarly-presents-8-ai-agents/ (дата звернення: 10.02.2026).

11. Vajjala S., Majumder B., Gupta A., Surana H. Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems. O'Reilly Media, 2020. 456 p.

12. Lazuardy M. F. S., Anggraini D. Modern front end web architectures with React.js and Next.js. *Research Journal of Advanced Engineering and Science*. 2022. Vol. 7, No. 1. P. 132–141.

13. Peralta J. H. Microservice APIs: Using Python, Flask, FastAPI, OpenAPI and More. Simon and Schuster, 2023. 440 p.

14. Hagiwara M. Real-World Natural Language Processing: Practical Applications with Deep Learning. Manning Publications, 2021. 322 p.

15. Honnibal M., Montani I., Landeghem S. Van, Boyd A. spaCy: Industrial-strength natural language processing in Python. 2020. URL: https://doi.org/10.5281/zenodo.1212303 (Accessed: 16.02.2026).

16. Altinok D. Mastering spaCy: An end-to-end practical guide to implementing NLP applications using the Python ecosystem. Packt Publishing Ltd, 2021. 356 p.

17. Tunstall L., Werra L. Von, Wolf T. Natural language processing with transformers. "O'Reilly Media, Inc.", 2022. 408 p.

18. Țucudean G., Bucos M., Drăgulescu B., Căleanu C.-D. Natural language processing with transformers: a review. *PeerJ Computer Science*. 2024. URL: https://doi.org/10.7717/peerj-cs.2222 (Accessed: 24.03.2026).

19. Azunre P. Transfer Learning for Natural Language Processing. Manning Publications, 2021. 272 p.

20. Ferrari L., Pirozzi E. Learn PostgreSQL: Use, manage, and build secure and scalable databases with PostgreSQL 16. Packt Publishing Ltd, 2023. 744 p.

21. Kuppusamy S. Mastering OpenSearch: A Comprehensive Guide. Saravanan, 2025. 338 p.

22. Manning C. D., Raghavan P., Schütze H. Introduction to Information

Retrieval. Cambridge University Press, 2008. 506 p. URL: https://doi.org/10.1017/CBO9780511809071 (Accessed: 27.03.2026).

23. Mihalcea R., Radev D. Graph-Based Natural Language Processing and Information Retrieval. Cambridge University Press, 2011. 192 p. URL: https://doi.org/10.1017/CBO9780511976247 (Accessed: 31.03.2026).

24. Gupta P., Bagchi A. Data Visualization with Python BT – Essentials of Python for Artificial Intelligence and Machine Learning. *Data Visualization with Python*. Red. Gupta P., Bagchi A. Cham: Springer Nature Switzerland, 2024. P. 237–282. URL: https://doi.org/10.1007/978-3-031-43725-0_7 (Accessed: 08.04.2026).

25. Lavanya A., Gaurav L., Sindhuja S., Seam H., Joydeep M., Uppalapati V., Ali W., Sagar V. S. D. Assessing the performance of Python data visualization libraries: a review. *Int. J. Comput. Eng. Res. Trends*. 2023. Vol. 10, No. 1. P. 28–39. URL: https://doi.org/10.22362/ijcert/2023/v10/i01/v10i0104 (Accessed: 16.04.2026).

26. Han S., Kwak I.-Y. Mastering data visualization with Python: practical tips for researchers. *Journal of Minimally Invasive Surgery*. 2023. Vol. 26, No. 4. P. 167. URL: https://doi.org/10.7602/jmis.2023.26.4.167 (Accessed: 01.05.2026).

27. Romero-Organvidez D., Horcas J.-M., Galindo J. A., Benavides D. Data visualization guidance using a software product line approach. *Journal of Systems and Software*. 2024. Vol. 213. 28 p. URL: https://doi.org/10.1016/j.jss.2024.112029 (Accessed: 04.05.2026).

28. Zhou N., Zhou H., Hoppe D. Containerization for High Performance Computing Systems: Survey and Prospects. *IEEE Transactions on Software Engineering*. 2023. Vol. 49, No. 4. P. 2722–2740. URL: https://doi.org/10.1109/tse.2022.3229221 (Accessed: 05.05.2026).

29. Muzumdar P., Bhosale A., Basyal G. P., Kurian G. Navigating the Docker ecosystem: A comprehensive taxonomy and survey. *Asian Journal of Research in Computer Science*. 2024. Vol. 17, No. 1. P. 42–61. URL: https://doi.org/10.9734/AJRCOS/2024/v17i1411 (Accessed: 07.05.2026).

30. Sahoo R., Derbali M., Smieee H. J., Thang D., Kumar P., Sahoo S. Test Case Generation from UML-Diagrams Using Genetic Algorithm. *Computers, Materials &*

Continua. 2021. Vol. 67, No. 2. P. 2321–2336. URL: https://doi.org/10.32604/cmc.2021.013014 (Accessed: 11.05.2026).

31. André É., Liu S., Liu Y., Choppy C., Sun J., Dong J. 2023. Formalizing UML State Machines for Automated Verification – A Survey. *ACM Computing Surveys*. 2023. Vol. 55, No. 13s, Article No. 277. 47 p. URL: https://doi.org/10.1145/3579821 (Accessed: 13.05.2026).

32. Ozkaya M., Erata F. A survey on the practical use of UML for different software architecture viewpoints. *Information and Software Technology*. 2020. Vol. 121. Article 106275. 27 p. URL: https://doi.org/10.1016/j.infsof.2020.106275 (Accessed: 14.05.2026).

ДОДАТОК А

Програмна реалізація логіки об’єднання результатів та принципу формування *n*-грам

Лістинг коду логіки об’єднання результатів LanguageTool і локальних правил nlp.py:

```
def suggestions_overlap(first: dict, second: dict) -> bool:
    first_start = int(first.get("positionStart", -1))
    first_end = int(first.get("positionEnd", -1))
    second_start = int(second.get("positionStart", -1))
    second_end = int(second.get("positionEnd", -1))

    if first_start < 0 or second_start < 0:
        return False

    return first_start < second_end and second_start < first_end


def merge_suggestions(primary: list[dict], secondary: list[dict]) -> list[dict]:
    merged = list(primary)

    for item in secondary:
        has_conflict = any(
            suggestions_overlap(item, existing)
            for existing in merged
        )

        if not has_conflict:
            item["engine"] = item.get("engine") or "LinguaInsight"
            merged.append(item)

    return merged
```

Лістинг коду принципу формування *n*-грам із параметрами за замовчуванням:

```
def ngram_analysis(
    texts: list[str],
    n: int = 3,
    min_frequency: int = 1,
    language: str = "uk",
    stop_words_profile: str = "academic",
    pos_method: str = "hybrid",
    max_results: int = 300
) -> list[dict[str, Any]]:
    stop_words = _get_stop_words(language, stop_words_profile)

    sentence_tokens_by_doc: list[list[list[str]]] = []

    for text in texts: document_sentences = []
        for sentence in _split_sentences(text):
```

```
            tokens = [_normalize_token(token) for token in tokenize(sentence)]
            if tokens: document_sentences.append(tokens)

        sentence_tokens_by_doc.append(document_sentences)

    all_tokens = [
        token
        for document in sentence_tokens_by_doc
        for sentence in document
        for token in sentence
    ]

    if not all_tokens or n <= 0: return []

    unigram_counts = Counter(all_tokens)
    candidate_counts: Counter[tuple[str, ...]] = Counter()
    document_frequency: Counter[tuple[str, ...]] = Counter()

    for document in sentence_tokens_by_doc:
        document_seen: set[tuple[str, ...]] = set()

        for sentence in document:
            if len(sentence) < n: continue

            for index in range(0, len(sentence) - n + 1):
                gram = tuple(sentence[index:index + n])
                content_count = sum(1 for token in gram if
_is_content_token(token, stop_words))
                content_ratio = content_count / len(gram)
                if n == 1 and content_count == 0: continue
                if n > 1 and content_ratio < 0.5: continue
                if all(token in stop_words for token in gram): continue
                candidate_counts[gram] += 1
                document_seen.add(gram)

        for gram in document_seen: document_frequency[gram] += 1

    if not candidate_counts: return []

    effective_min_frequency = max(1, min_frequency)
    total_candidates = sum(candidate_counts.values()) or 1
    total_tokens = len(all_tokens) or 1

    results: list[dict[str, Any]] = []

    for gram, count in candidate_counts.items():
        if count < effective_min_frequency: continue

        prob_ngram = count / total_candidates
        prob_parts = 1.0
        for token in gram:
            prob_parts *= unigram_counts[token] / total_tokens

        pmi = math.log2(prob_ngram / prob_parts) if prob_parts > 0 and prob_ngram
> 0 else 0.0
```

```
        expected = total_candidates * prob_parts
        t_score = (count - expected) / math.sqrt(count) if count > 0 else 0.0

        first_freq = unigram_counts[gram[0]]
        last_freq = unigram_counts[gram[-1]]
        dice = (2 * count) / (first_freq + last_freq) if (first_freq + last_freq)
> 0 else 0.0
        content_count = sum(1 for token in gram if _is_content_token(token,
stop_words))
        content_ratio = content_count / len(gram)
        profile = _classify_sequence(list(gram))
        score = _sequence_score(
            count=count,
            pmi=pmi,
            t_score=t_score,
            dice=dice,
            content_ratio=content_ratio,
            document_frequency=document_frequency[gram]
        )

        results.append({
            "ngramText": " ".join(gram),
            "frequency": count,
            "relativeFrequency": round(count / total_candidates, 6),
            "pmi": round(pmi, 6),
            "tScore": round(t_score, 6),
            "dice": round(dice, 6),
            "score": score,
            "profile": profile,
            "contentRatio": round(content_ratio, 4),
            "documentFrequency": document_frequency[gram]
        })

    if not results and min_frequency > 1:
        return ngram_analysis(
            texts=texts,
            n=n,
            min_frequency=1,
            language=language,
            stop_words_profile=stop_words_profile,
            pos_method=pos_method,
            max_results=max_results
        )

    results.sort(
        key=lambda item: (
            item["score"],
            item["frequency"],
            item["pmi"],
            item["documentFrequency"]
        ),
        reverse=True
    )
    return results[:max_results]
```

ДОДАТОК Б

Приклади визначених біграм у корпусі текстів із PDF-звіту

Таблиця Б.1 – Приклади визначених біграм у корпусі текстів із PDF-звіту

№	N-грама	Частота	Відносна частота	Показник зв'язності	Профіль
1	аналізу тексту	5	0.002904	6.339326	Частотний
2	використання матеріалів	4	0.002323	8.039766	Освітній
3	текстових даних	4	0.002323	7.232411	Лінгвістичний
4	секрети макіяжу	3	0.001742	9.454804	Словосполучення
5	такий підхід	3	0.001742	9.454804	Науковий
6	матеріалів сайту	3	0.001742	9.039766	Освітній
7	читати більше	3	0.001742	9.039766	Словосполучення
8	макіяжу губ	3	0.001742	8.039766	Словосполучення
9	зробити губи	3	0.001742	7.624729	Словосполучення
10	лінгвістичного аналізу	3	0.001742	7.339326	Науковий
11	ваш сайт	2	0.001161	10.039766	Словосполучення
12	запропоновані виправлення	2	0.001161	9.454804	Словосполучення
13	прав доступу	2	0.001161	9.454804	Адміністративний
14	прикладних наборах	2	0.001161	9.454804	Словосполучення
15	природної мови	2	0.001161	9.454804	Словосполучення
16	скопіюйте цей	2	0.001161	9.454804	Словосполучення
17	цей код	2	0.001161	9.454804	Технічний
18	прийняття рішень	2	0.001161	9.039766	Словосполучення
19	автоматична перевірка	1	0.000581	11.039766	Словосполучення
20	автори події	1	0.000581	11.039766	Словосполучення
21	адаптувати під	1	0.000581	11.039766	Словосполучення
22	академічні осередки	1	0.000581	11.039766	Словосполучення
23	активного гіперпосилання	1	0.000581	11.039766	Словосполучення
24	актуальність теми	1	0.000581	11.039766	Словосполучення
25	аналітичних потреб	1	0.000581	11.039766	Словосполучення
26	багато хто	1	0.000581	11.039766	Словосполучення
27	бажаного ефекту	1	0.000581	11.039766	Словосполучення
28	великими текстами	1	0.000581	11.039766	Лінгвістичний