

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Веб-технології»)

на тему:

«Вебсервіс перевірки та корекції граматики української мови»

Виконав студент групи ВТ-22-2
ФОМЕНКО В'ячеслав Андрійович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ФАНТ Микола Олександрович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28 лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ФОМЕНКО В'ячеслав Андрійович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Вебсервіс перевірки та корекції граматики української мови»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: Об'єктом дослідження є процес автоматизованої обробки та виправлення україномовних текстів із застосуванням методів машинного навчання та лінгвістичного аналізу. Предметом дослідження є методи та алгоритми гібридної перевірки граматики, що включають нейромережеві моделі типу seq2seq та систему на основі правил LanguageTool, а також стратегії об'єднання результатів кількох моделей.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	1.03.2026	1.03.2026
2	Савіцький Р.С.	1.03.2026	1.03.2026
3	Савіцький Р.С.	1.03.2026	1.03.2026

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з розробленого вебсервісу перевірки та корекції граматики тексту українською мовою та пояснювальної записки. Пояснювальна записка до дипломної роботи містить 70 сторінок, 24 ілюстрації, 22 таблиці та 38 джерел.

Метою кваліфікаційної роботи є проєктування та розробка вебсервісу для автоматизованої перевірки та корекції тексту українською мовою, що поєднує нейронні мовні моделі.

У роботі визначено ключові завдання проєкту, проведено аналіз існуючих рішень у сфері перевірки граматики тексту, наведено сценарії використання системи. Обґрунтовано вибір архітектури та технологій, описано компоненти системи і ланцюжок гібридної корекції тексту.

У процесі розробки реалізовано вебінтерфейс для перевірки та корекції тексту українською мовою, програмний інтерфейс REST API для взаємодії із зовнішніми системами та систему автентифікації користувачів. Передбачено ведення журналу перевірок, управління API-ключами та формування звітів у форматі PDF.

Для обробки тексту використано гібридний підхід, який поєднує нейромережеві моделі корекції тексту та перевірку на основі правил LanguageTool. Реалізовано кілька режимів перевірки, що відрізняються набором застосованих алгоритмів та часом виконання обробки.

Результатом роботи є працездатний вебсервіс, розгорнутий у хмарному середовищі та доступний через браузер і REST API. Проведене тестування підтвердило коректність роботи основних функцій системи та можливість її використання для автоматизованої перевірки україномовних текстів.

**КЛЮЧОВІ СЛОВА: ВЕБСЕРВІС, ГРАМАТИКА, УКРАЇНСЬКА
МОВА, НЕЙРОННА МЕРЕЖА, LANGUAGE TOOL, NESTJS, REACT,
POSTGRES SQL, REST API.**

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		3

ABSTRACT

The Bachelor's qualification thesis consists of a developed web service for Ukrainian grammar checking and text correction, as well as an explanatory note. The explanatory note contains 70 pages, 24 figures, 22 tables, and 38 references.

The purpose of the qualification thesis is to design and develop a web service for automated checking and correction of Ukrainian-language texts using neural language models.

The thesis identifies the key objectives of the project, analyzes existing solutions in the field of grammar checking, and presents the system use cases. The choice of architecture and technologies is justified, and the system components and the hybrid text correction pipeline are described.

During the development process, a web interface for Ukrainian text checking and correction, a REST API for integration with external systems, and a user authentication subsystem were implemented. The system also provides a history of text checks, API key management, and PDF report generation.

A hybrid text-processing approach was applied, combining neural text correction models with rule-based analysis provided by LanguageTool. Several checking modes were implemented, differing in the set of algorithms used and processing time.

The result of the work is a functional web service deployed in a cloud environment and accessible through both a web browser and a REST API. Testing confirmed the correctness of the main system functions and demonstrated the possibility of using the service for automated checking and correction of Ukrainian-language texts.

KEYWORDS: WEB SERVICE, GRAMMAR, UKRAINIAN LANGUAGE, NEURAL NETWORK, LANGUAGE TOOL, NESTJS, REACT, POSTGRES SQL, REST API.

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT.....	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	10
1.1 Аналіз аналогів програмного продукту.....	10
1.2 Постановка задачі.....	14
1.3 Вибір архітектури системи.....	17
1.4 Обґрунтування вибору інструментальних засобів.....	21
Висновки до першого розділу.....	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСЕРВІСУ ПЕРЕВІРКИ ТА КОРЕКЦІЇ ГРАМАТИКИ УКРАЇНСЬКОЇ МОВИ	26
2.1 Аналіз вимог та варіанти використання.....	26
2.2 Проєктування архітектури та бази даних	31
2.3 Алгоритм гібридної корекції тексту.....	37
2.4 Реалізація функціоналу.....	42
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСЕРВІСОМ ПЕРЕВІРКИ ТА КОРЕКЦІЇ ГРАМАТИКИ УКРАЇНСЬКОЇ МОВИ	49
3.1 Порядок встановлення та налаштування параметрів системи.....	49
3.2 Інтерфейс вебкористувача.....	53
3.3 Тестування системи.....	58
Висновки до третього розділу.....	64
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	67

ДОДАТКИ	71
Додаток А	72
Додаток Б	77
Додаток В	83
Додаток Г	93
Додаток Д	100

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіицький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ПЗ – програмне забезпечення;

AWS (Amazon Web Services) – хмарна платформа Amazon;

API (Application Programming Interface) – інтерфейс прикладного програмування;

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

CSS (Cascading Style Sheets) – каскадні таблиці стилів;

DI (Dependency Injection) – впровадження залежностей;

DTO (Data Transfer Object) – об'єкт передачі даних;

HTML (HyperText Markup Language) – мова розмітки гіпертексту;

HTTP/HTTPS (HyperText Transfer Protocol / Secure) – протокол передачі гіпертексту та його захищена версія;

JSON (JavaScript Object Notation) – об'єктна нотація JavaScript, формат обміну даними;

JWT (JSON Web Token) – токен для автентифікації та авторизації;

ML (Machine Learning) – машинне навчання;

NLP (Natural Language Processing) – обробка природної мови;

OAuth (Open Authorization) – відкритий протокол делегованої авторизації;

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

PDF (Portable Document Format) – формат портативного документа;

REST (Representational State Transfer) – архітектурний стиль побудови мережових сервісів;

seq2seq (sequence-to-sequence) – архітектура нейронної мережі «послідовність-до-послідовності»;

SPA (Single Page Application) – односторінковий вебзастосунок;

UI/UX (User Interface / User Experience) – інтерфейс та досвід користувача.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Потреба в автоматизованій перевірці україномовних текстів зростає в міру того, як українська мова набуває більш ширшого вжитку в офіційному, діловому та освітньому середовищі [1]. Повномасштабне вторгнення прискорило процес переходу значної частини населення на українську як основну мову спілкування, що підвищило попит на інструменти, здатні допомагати у написанні грамотних текстів.

Вебсервіси для перевірки граматики орієнтовані переважно на англійську мову, тоді як підтримка української залишається або відсутньою, або реалізованою на недостатньому рівні [2]. Існуючі комерційні сервіси розроблені для масштабних міжнародних ринків і не враховують особливостей морфології та синтаксису української мови, а відкриті рішення підтримують українську лише частково. Це змушує користувачів поєднувати кілька інструментів або покладатись на вбудовані засоби текстових редакторів, які не забезпечують повноцінного граматичного аналізу.

Метою кваліфікаційної роботи є проєктування та розробка вебсервісу автоматичної перевірки та виправлення граматики україномовних текстів із використанням гібридного підходу, що поєднує нейромережеві моделі та систему граматичних правил.

Для реалізації поставленої мети визначено наступні **завдання**:

1. Проведення аналізу аналогів сервісів для перевірки та корекції граматики з метою дослідження функціоналу, визначення переваг і недоліків.
2. Формування вимог до системи та визначення її архітектури.
3. Обґрунтування вибору технологій та інструментальних засобів реалізації.
4. Розробка алгоритму гібридної корекції тексту на основі поєднання нейронних моделей та аналізу заснованого на правилах.
5. Проєктування бази даних та програмного інтерфейсу.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

6. Реалізація функціоналу системи, включаючи сервіс машинного навчання, шлюз API та вебклієнт.

Об'єктом дослідження є процес автоматизованої обробки та виправлення україномовних текстів із застосуванням методів машинного навчання та лінгвістичного аналізу.

Предметом дослідження є методи та алгоритми гібридної перевірки граматики, що включають нейромережеві моделі типу seq2seq та систему на основі правил LanguageTool.

Практичне застосування. Розроблений сервіс може використовуватись як самостійний вебзастосунок для перевірки текстів, так і як складова частина сторонніх систем через публічний програмний інтерфейс.

Серед потенційних напрямів використання є інтеграція у редактори документів, освітні платформи, системи керування контентом та інші застосунки, що потребують автоматичної обробки україномовних текстів. У редакторах документів сервіс може виступати як зовнішній модуль перевірки, що доповнює вбудовані засоби орфографії повноцінним граматичним і стилістичним аналізом.

В освітньому середовищі сервіс може застосовуватись для автоматичної перевірки письмових робіт студентів, що знижує навантаження на викладачів і забезпечує миттєвий зворотний зв'язок. У системах керування контентом інтеграція через REST API дозволяє автоматично перевіряти публікації перед їх виходом, що підвищує загальну якість україномовного контенту. Відсутність обмежень на кількість запитів для авторизованих користувачів робить сервіс придатним для використання у промислових системах з інтенсивним потоком текстових даних.

За тематикою теми дипломної роботи були опубліковані тези [3, 4].

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Аналіз аналогів програмного продукту

Аналіз існуючих програмних рішень у сфері автоматизованої перевірки граматики тексту проведено з метою визначення місця розроблюваного сервісу на ринку. Розглянуто три найбільш поширені інструменти з підтримкою або частковою підтримкою української мови.

Порівняльний аналіз охоплює представників трьох категорій – комерційний сервіс із широкою аудиторією (Grammarly), відкритий багатомовний інструмент (LanguageTool) та спеціалізований україномовний сервіс (UkrainianCorrection.com). Такий вибір дозволяє охопити різні підходи до реалізації сервісів граматичної корекції та виявити прогалини, що може заповнити розроблюваний сервіс.

Кожен аналог оцінювався за такими параметрами: підтримка української мови, наявність нейронних моделей корекції, кількість режимів перевірки, доступність публічного REST API, вимога реєстрації, можливість перегляду історії перевірок, експорт результатів у PDF та управління API-ключами.

Grammarly є одним із найвідоміших комерційних сервісів перевірки та покращення тексту [5]. Платформа використовує нейронні мовні моделі для виявлення граматичних, орфографічних, пунктуаційних і стилістичних помилок. Сервіс доступний у вигляді вебдодатку, розширення для браузера та десктопного застосунку. Для використання Grammarly обов'язкова реєстрація облікового запису, без якої доступ до будь-яких функцій перевірки неможливий. Зовнішній вигляд сервісу наведено на рис. 1.1.

Станом на 2026 рік Grammarly додав підтримку української мови у форматі бета-версії. Це означає, що якість корекції україномовних текстів залишається нижчою порівняно з англійською та може містити похибки. Просунуті функції AI-корекції, включаючи перефразування речень і тональний аналіз, доступні лише в платному тарифі Premium. Публічного REST API для зовнішніх

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

розробників Grammarly не надає, API доступний лише в корпоративному плані Grammarly Business.

Інтерфейс Grammarly побудований як розширення браузера та вебредактор. Серед ключових можливостей – пропозиції щодо стилю, тону та чіткості тексту, перевірка на плагіат, а також інтеграція з популярними платформами.

З точки зору позиціонування, Grammarly є повнофункціональним комерційним продуктом, що охоплює широкий спектр мов. Відсутні режими перевірки з різним балансом між швидкістю та точністю, а також функції збереження історії перевірок та експорту результатів у PDF.

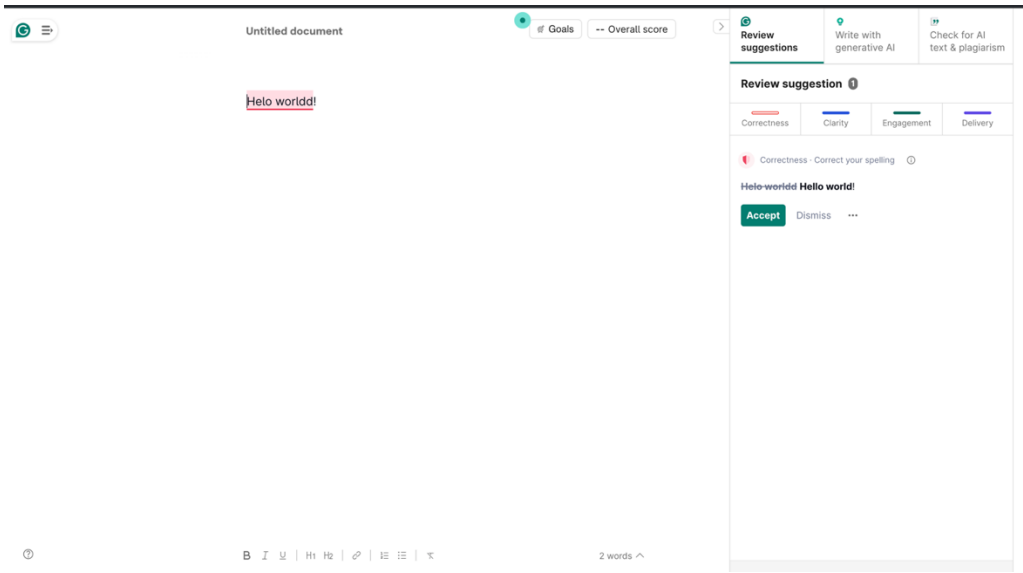


Рис. 1.1 Вигляд інтерфейсу Grammarly

LanguageTool є відкритим багатомовним інструментом перевірки граматики та орфографії, що підтримує понад 30 мов, зокрема українську [6]. В основі системи лежить підхід виявлення помилок здійснюється через перевірку тексту за набором мовних правил, які підтримує спільнота розробників. У преміальному тарифі LanguageTool Premium доповнено нейронними AI-моделями для виявлення більш складних помилок, зокрема контекстних замін слів і стилістичних неточностей. Вигляд вебінтерфейсу сервісу наведено на рис. 1.2.

Безкоштовний публічний API LanguageTool має суттєві обмеження. Не більше 20 запитів за хвилину з однієї IP-адреси та до 75 000 символів за хвилину,

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

а також обмеження на довжину тексту одного запиту. Для зняття лімітів необхідна платна підписка або розгортання власного сервера LanguageTool.

LanguageTool підтримує понад 30 мов, у тому числі українську, та може розгортатися локально. Серед його переваг – відкритий вихідний код, гнучка система правил на основі XML та інтеграція з популярними текстовими редакторами через розширення.

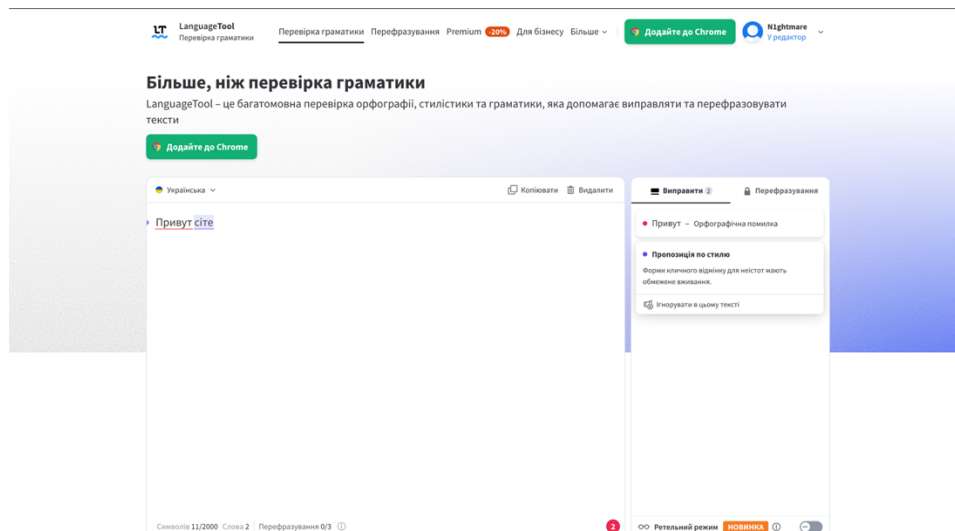


Рис.1.2. Вигляд вебінтерфейсу LanguageTool

UkrainianCorrection.com є безкоштовним онлайн-сервісом для перевірки правопису та граматики текстів виключно українською мовою. Інструмент орієнтований на носіїв мови та осіб, що вивчають українську, й дозволяє виявляти орфографічні та пунктуаційні помилки без реєстрації облікового запису [7].

Сервіс реалізований як простий вебзастосунок без підтримки REST API, нейронних моделей корекції чи кількох режимів перевірки. Відсутні функції перегляду історії перевірок, експорту результатів у PDF та управління API-ключами. Незважаючи на це, UkrainianCorrection.com залишається корисним інструментом для базової орфографічної перевірки україномовних текстів.

Порівняно з іншими аналогами, UkrainianCorrection.com є найбільш вузькоспеціалізованим сервісом, який орієнтований виключно на перевірку правопису та пунктуації в україномовних текстах, без будь-яких функцій нейронної корекції чи аналітики. Сервіс не надає програмного інтерфейсу, що

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

виключає можливість його інтеграції у сторонні застосунки або автоматизовані ланцюжки обробки тексту. Вигляд вебінтерфейсу сервісу наведено на рис. 1.3.

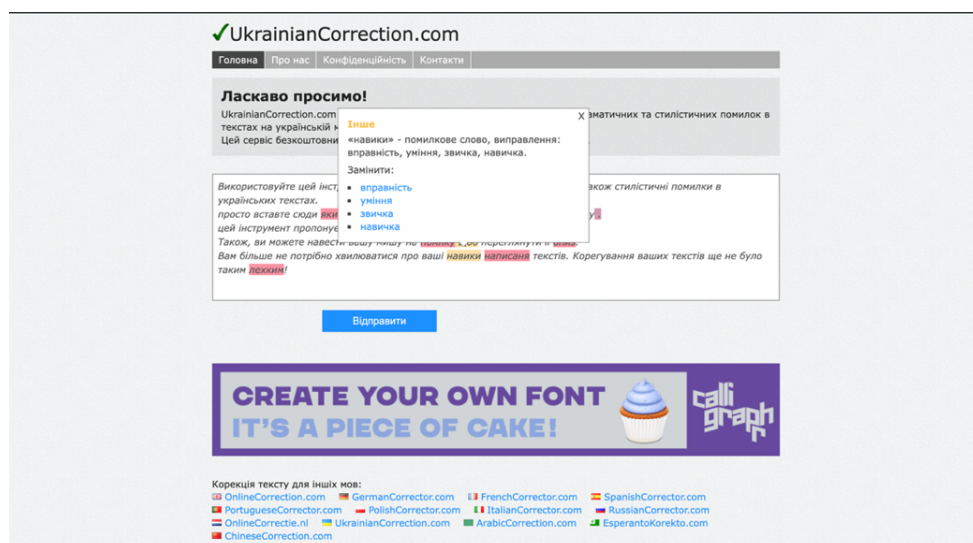


Рис. 1.3. Вигляд вебінтерфейсу UkrainianCorrection.com

У таблиці 1.1 наведено порівняння характеристик описаних аналогів із розроблюваним сервісом.

Таблиця 1.1

Порівняння аналогів із розроблюваним сервісом

Характеристика	Grammarly	LanguageTool	UkrainianCorrection.com	Perom
Підтримка української мови	—	+	+	+
Орієнтація на українську мову	—	—	+	+
Нейронна корекція тексту	+	+	—	+
Кілька режимів перевірки	—	—	—	+
Публічний REST API	—	+	—	+
Обов'язкова реєстрація	+	—	—	—
Перегляд історії перевірок	+	—	—	+
Експорт результатів	—	—	—	+
Управління API-ключами	—	—	—	+

Окрім розглянутих інструментів, варто зазначити вбудовані засоби перевірки правопису в текстових редакторах, зокрема Microsoft Word та Google Docs [8, 9]. Обидва продукти реалізують орфографічну перевірку на основі словникового підходу та підкреслюють слова, відсутні у вбудованому словнику. Якість перевірки для української мови в цих редакторах суттєво поступається англійській і зводиться переважно до пошуку орфографічних помилок без аналізу граматичного зв'язку між словами у реченні. Жоден із цих інструментів не надає зовнішнього API для інтеграції.

Результати аналізу аналогів показують, що жоден із розглянутих інструментів не надає повноцінної нейронної корекції тексту українською мовою у рамках безкоштовного публічного API. Grammarly підтримує українську мову лише в бета-режимі та не надає API для зовнішніх розробників. LanguageTool забезпечує кращу підтримку українського правопису завдяки базі правил, проте безкоштовний API обмежено 20 запитами на хвилину, а нейронна корекція доступна лише в платному тарифі. Жоден із аналогів не підтримує експорт результатів у PDF, систему управління API-ключами чи персональний журнал перевірок.

Таким чином, розробка власного вебсервісу з гібридним підходом, що поєднує дві нейронні моделі зі спеціалізацією на українській мові та аналіз на основі правил LanguageTool, є обґрунтованою і усуває виявлені обмеження існуючих рішень.

1.2 Постановка задачі

У межах роботи необхідно розробити вебсервіс автоматизованої перевірки та корекції тексту українською мовою, що забезпечує виявлення орфографічних, граматичних, пунктуаційних та стилістичних помилок із подальшим наданням рекомендацій щодо їх виправлення. Основною задачею системи є підвищення якості написання українськомовних текстів та спрощення процесу їх редагування як для звичайних користувачів, так і для програмних систем, що потребують автоматичної перевірки контенту. Сервіс має бути доступним через

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

вебінтерфейс для кінцевих користувачів, а також через REST API для інтеграції зі сторонніми застосунками, платформами та інформаційними системами.

Вебсервіс реалізує щонайменше три режими обробки тексту, що різняться швидкістю роботи та точністю перевірки, орієнтований на швидку базову перевірку орфографії та пунктуації, другий забезпечує поглиблений граматичний аналіз, а третій використовує максимальний набір алгоритмів для найбільш точної стилістичної та граматичної корекції. Такий підхід дозволяє користувачам обирати оптимальний режим залежно від обсягу тексту та вимог до якості перевірки. Додатково сервіс повинен підтримувати систему реєстрації та автентифікації користувачів, управління API-ключами для програмного доступу, а також ведення персоналізованого журналу перевірок із можливістю перегляду історії оброблених текстів.

Першим етапом розробки є проведення детального аналізу існуючих рішень у сфері автоматизованої перевірки граматики та орфографії. Аналіз охоплює популярні онлайн-сервіси, програмні бібліотеки та нейромережеві рішення, що використовуються для перевірки тексту різними мовами. Метою цього етапу є визначення основних переваг і недоліків наявних систем, оцінка рівня підтримки української мови. На основі отриманих результатів формуються функціональні та нефункціональні вимоги до розроблюваного сервісу, визначаються ключові сценарії використання та архітектурні обмеження.

Другим етапом є проектування архітектури системи з урахуванням вимог масштабованості, відмовостійкості та можливості незалежного розвитку окремих компонентів. Система поділяється на три основні незалежні сервіси: сервіс машинного навчання, шлюз API та вебклієнт. Сервіс машинного навчання відповідає за обробку тексту та виконання перевірок, API-шлюз забезпечує взаємодію між клієнтами та внутрішніми сервісами, а вебклієнт реалізує користувацький інтерфейс. Таке розділення дозволяє незалежно масштабувати ML-сервіс який є найбільш ресурсоємним, без впливу на роботу інших частин системи. Крім того, модульна архітектура спрощує подальше розширення функціональності та підтримку сервісу.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Третім етапом є реалізація сервісу машинного навчання, який поєднує декілька підходів до аналізу тексту. Основу системи складають дві нейронні моделі для автоматичної корекції тексту, доповнені аналізатором на основі правил LanguageTool. Нейронні моделі використовуються для контекстного аналізу речень і виявлення складних помилок, тоді як LanguageTool забезпечує перевірку правил граматики, орфографії та пунктуації. Використання гібридного підходу дозволяє підвищити точність виправлень, зменшити кількість хибних спрацювань і забезпечити кращу якість аналізу українськомовного тексту порівняно з використанням лише одного методу перевірки.

Четвертим етапом є реалізація програмного інтерфейсу на платформі NestJS. Цей компонент відповідає за централізовану обробку запитів, маршрутизацію даних між сервісами та забезпечення безпеки системи. У межах API реалізується управління користувачами, JWT-автентифікація, система API-ключів для зовнішніх клієнтів, механізм журналювання запитів, контроль доступу та обмеження кількості запитів. Додатково додаток надає можливість формування PDF-звітів із результатами перевірки тексту, що дозволяє користувачам експортувати результати аналізу у зручному форматі для подальшого використання.

П'ятим етапом є розробка вебклієнта з використанням бібліотеки React та мови TypeScript. Вебінтерфейс повинен забезпечувати зручну взаємодію користувача із системою та підтримувати сучасний адаптивний дизайн. Інтерфейс включає текстовий редактор із підсвічуванням помилок, можливість перегляду детальної інформації про кожну знайдену помилку, рекомендації щодо виправлення та інструменти швидкого редагування тексту. Крім цього, користувачі отримують доступ до налаштувань акаунту, історії перевірок, управління API-ключами та статистики використання сервісу. Особлива увага приділяється швидкодії клієнтської частини та зручності користування сервісом на різних типах пристроїв.

Результатом виконання поставленої задачі є створеновальноцінного вебсервісу перевірки та корекції граматики української мови, що забезпечує

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

якісну обробку текстів, підтримує різні режими перевірки та надає зручний API для інтеграції зі сторонніми системами. Розроблений сервіс дозволяє автоматизувати процес перевірки тексту та забезпечити користувачів сучасним інструментом для редагування та аналізу текстової інформації.

1.3 Вибір архітектури системи

Монолітна архітектура передбачає розміщення всієї функціональності в одному процесі та є найпростішою з точки зору розгортання. Проте застосування такого підходу в даній системі є неможливим: компонент машинного навчання потребує Python з бібліотеками PyTorch та Transformers, тоді як шлюз API реалізовано засобами Node.js. Оскільки ці технологічні стеки не можуть виконуватись в одному процесі, монолітна архітектура була відхилена.

Альтернативою є виокремлення ML-частини в окремий сервіс. Перевагою такого підходу є незалежне масштабування ML-сервісу. При зростанні навантаження, можна запустити додаткові екземпляри ML-сервісу без змін у решті системи. Саме цей підхід обрано для розроблюваного вебсервісу.

Окрім поділу системи на процеси, розглянуто також варіант повністю мікросервісної архітектури, коли кожен функціональний блок виноситься в окремий сервіс із власною базою даних. Такий підхід надає максимальну гнучкість, проте суттєво ускладнює розгортання, моніторинг та узгодження даних між сервісами. Для навчального проєкту з обмеженими ресурсами така складність є невиправданою. Тому обрано проміжний варіант, за якого окремо виноситься лише ресурсомісткий ML-сервіс, а решта бізнес-логіки залишається в одному процесі. Це поєднує переваги незалежного масштабування найважливого компонента з простотою підтримки решти системи.

Обрана архітектура складається з трьох незалежних компонентів. ML-сервіс виконує обчислювально інтенсивну роботу з нейронними моделями і розгортається на спеціалізованій платформі Hugging Face Spaces [10], яка надає GPU-прискорення та автоматичне кешування моделей. Шлюз API реалізує бізнес-логіку, автентифікацію, маршрутизацію та зберігання даних і розгортається на хмарних платформах Railway [11]. Вебклієнт виконується

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

безпосередньо у браузері користувача і розгортається на платформі Vercel як статичний застосунок.

Такий розподіл також дозволяє підібрати для кожного компонента окрему платформу розгортання, що найкраще відповідає його призначенню. ML-сервіс потребує середовища з підтримкою контейнерів і можливістю GPU-прискорення, тому його розміщено на Hugging Face Spaces. Шлюз API є тривало працюючим процесом із підключенням до бази даних, для чого добре підходить Railway. Вебклієнт складається зі статичних файлів, які найефективніше роздавати через мережу доставки контенту, що забезпечує Vercel. Кожен компонент оновлюється окремо, тому зміна однієї частини системи не потребує повторного розгортання інших.

Для корекції тексту на рівні ML-сервісу розглянуто три підходи, першим підходом є виключно аналіз на основі правил LanguageTool, другим підходом є застосування однієї нейронної моделі та третім підходом є гібридний метод з кількома джерелами і голосуванням. Підхід аналізу на основі правил забезпечує передбачувані результати для формалізованих типів помилок, але не здатний виявляти контекстно-залежні помилки, а підхід з однією нейронною моделлю дає кращу якість для складних помилок, але схильний до галюцинацій та хибних виправлень коректного тексту. Гібридний підхід поєднує переваги обох методів.

Гібридний алгоритм голосування між трьома джерелами обрано як основний підхід до корекції тексту. Нейронна модель Pravopysnyk/best-unlp спеціалізується на граматичній корекції україномовних текстів і навчена на корпусі UA-GEC. Нейронна модель grammarly/spivavtor-large є інструктивно-налаштованою великою моделлю для редагування тексту українською мовою. LanguageTool забезпечує формалізовану перевірку за базою правил, що налічує понад 500 правил для української мови. Алгоритм MergeStrategy об'єднує їхні результати на рівні токенів, приймаючи лише ті виправлення, які підтримують щонайменше два джерела.

Вибір саме трьох джерел корекції є компромісом між якістю та швидкістю. Використання лише однієї нейронної моделі робить результат

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

залежним від її окремих помилок, а додавання великої кількості моделей суттєво збільшує час обробки без помітного приросту якості. Дві спеціалізовані моделі разом із аналізатором на основі правил утворюють непарну кількість джерел, що дозволяє ухвалювати рішення простою більшістю голосів без додаткових правил для розв'язання нічиїх ситуацій. Голосування на рівні окремих токенів робить корекцію передбачуваною, оскільки кожне виправлення має підтримку щонайменше двох незалежних джерел.

Обрана архітектура з модульним монолітом та виокремленим ML-сервісом забезпечує незалежне масштабування ML-компонента, технологічну свободу при виборі стеку для кожного рівня (додаток А, рис. 1). Такий вибір відповідає функціональним і нефункціональним вимогам до системи та усуває виявлені обмеження наявних аналогів.

Дотримання цих принципів спрощує не лише розробку, а й подальше тестування системи. Оскільки кожен компонент має чітко окреслену зону відповідальності, його можна перевіряти ізольовано, не запускаючи всю систему. Так, логіку шлюзу API можна тестувати з підставним ML-сервісом, а вебклієнт перевіряти на заздалегідь підготовлених відповідях. Низька зв'язаність між компонентами також зменшує ризик того, що зміна в одній частині системи призведе до помилок в іншій. Це особливо важливо для проєкту, який передбачає подальший розвиток і додавання нових можливостей.

Обрана архітектура відповідає принципам поділу відповідальностей та єдиної відповідальності. Кожен компонент системи вирішує конкретну задачу. ML-сервіс відповідає за якісну корекцію тексту. Шлюз API забезпечує безпеку, маршрутизацію та збереження даних. Вебклієнт відповідає за інтерфейс користувача. Подібне розмежування уможливлює незалежне вдосконалення та заміну кожного компонента без впливу на решту системи.

Безпеку системи забезпечено на кількох незалежних рівнях, на рівні автентифікації FlexibleAuthGuard перевіряє підпис JWT або хеш API-ключа при кожному захищеному запиті. На рівні авторизації RolesGuard перевіряє роль користувача перед виконанням адміністративних операцій. На рівні зберігання

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

паролі хешуються bcrypt, а API-ключі зберігаються виключно у вигляді SHA-256 хешу.

Вебклієнт реалізовано як односторінковий застосунок, що виконується безпосередньо у браузері користувача. Вебклієнт не зберігає конфіденційних даних і взаємодіє із системою виключно через шлюз API.

Рівень шлюзу API реалізовано за шаблоном модульного моноліту. Усі модулі бізнес-логіки розгортаються в одному Node.js-процесі та взаємодіють через внутрішній механізм ін'єкції залежностей NestJS, а не через мережеві виклики. Для шлюзу API саме це рішення є оптимальним. Модулі Auth, Users, Corrector, ApiKeys, Admin та Pdf чітко розмежовані логічно, але їхня низька обчислювальна інтенсивність не потребує окремого розгортання кожного.

Логування запитів реалізовано асинхронно, після повернення відповіді клієнту модуль CheckRequestsModule зберігає деталі запиту та виявлені помилки в базі даних у окремій задачі. Клієнт не відчуває жодної додаткової затримки навіть за умови ведення повного журналу перевірок.

Конфігурація системи здійснюється виключно через змінні середовища без жорсткого кодування будь-яких параметрів у вихідному коді. Такий підхід відповідає методології «12-factor app» і дозволяє розгортати одну й ту саму кодову базу в різних середовищах (розробка, тестування, робоча збірка) лише зміною значень змінних без перезбірки застосунку. Стійкість системи до збоїв забезпечується незалежністю її компонентів. Якщо ML-сервіс стає тимчасово недоступним, шлюз API продовжує обслуговувати запити, що не потребують корекції тексту, наприклад вхід, перегляд історії або керування ключами, і повертає зрозумілий код стану для запитів на перевірку. Так само тимчасова недоступність вебклієнта не впливає на роботу REST API для зовнішніх розробників. Відсутність єдиної точки відмови на рівні всієї системи підвищує її загальну надійність.

Взаємодія між компонентами системи відбувається виключно через HTTP. Шлюз API є єдиною точкою входу до систем. ML-сервіс не має прямого зовнішнього доступу і приймає запити лише від шлюзу API через внутрішню

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

мережу. Зовнішній клієнт позбавлений прямого доступу до ML-сервісу і не може обійти механізми автентифікації та ліміти шлюзу API, що суттєво підвищує захищеність системи.

1.4 Обґрунтування вибору інструментальних засобів

Формування технологічного стеку сервісу здійснювалося з урахуванням функціональних вимог системи, обраної архітектури та необхідності забезпечення підтримуваності й розширюваності програмного продукту.

Сервіс машинного навчання реалізовано мовою Python – стандартом галузі машинного навчання та обробки природної мови. Python надає доступ до найширшої екосистеми бібліотек і готових моделей для вирішення NLP-задач [12].

Java та C++ з відповідними бібліотеками розглядалися як можливі альтернативи. Однак Python є фактичним стандартом у сфері машинного навчання. Більшість фреймворків (PyTorch, TensorFlow, Transformers) розробляються першочергово для Python, а бібліотека Hugging Face Transformers, що використовується в проєкті, не має повноцінних альтернатив на інших мовах з аналогічним рівнем підтримки.

Нейронну корекцію тексту виконують дві моделі з відкритим доступом. Pravopysnyk/best-unlp [13] є спеціалізованою моделлю для корекції українського тексту, навченою на великих україномовних корпусах. Grammarly/spivavtor-large є потужною моделлю від команди Grammarly, адаптованою для слов'янських мов [14]. Поєднання двох нейронних моделей з наступним голосуванням на рівні токенів підвищує точність корекції і зменшує кількість хибних виправлень [13].

Як фреймворк для побудови REST API сервісу машинного навчання обрано FastAPI. Цей фреймворк забезпечує асинхронну обробку запитів, автоматичну генерацію документації OpenAPI та високу продуктивність завдяки використанню ASGI-сервера [15, 16].

Серед альтернатив розглядався Flask – легковажний мікрофреймворк для Python. Проте Flask не має вбудованої підтримки асинхронного виконання та автоматичної валідації запитів. FastAPI базується на Starlette та Pydantic, що

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує нативну підтримку `async/await`, автоматичну генерацію документації OpenAPI та суворий контроль типів на рівні схем запитів і відповідей.

Аналіз на основі правил реалізовано за допомогою бібліотеки `language-tool-python`, що є обгорткою для `LanguageTool` [17]. Включення `LanguageTool` у загальний процес обробки дозволяє виявляти пунктуаційні помилки та стилістичні неточності, які нейронні моделі можуть пропустити, а також застосовувати додаткові проходи корекції після нейронної обробки [6].

Шлюз API реалізовано на платформі `Node.js` із фреймворком `NestJS`. `NestJS` є модульним `TypeScript`-фреймворком, що реалізує принципи SOLID і надає вбудовані засоби для управління залежностями, декоративну систему маршрутизації та підтримку `Passport.js` для автентифікації [18]. Типізація `TypeScript` знижує ймовірність помилок і спрощує довгострокову підтримку коду [19].

Як альтернативи розглядалися `Express.js` та `Fastify`. `Express.js` є більш гнучким, проте не нав'язує жодної структури проєкту, що ускладнює підтримку при зростанні кодової бази. `Fastify` забезпечує вищу продуктивність, але має меншу екосистему готових модулів. `NestJS` обрано через вбудовану підтримку `TypeScript`, модульну архітектуру, декораторний стиль, що наближений до `Angular`, та наявність рішень для `DI`, `Passport`, `TypeORM` з мінімальним налаштуванням [20].

Як систему управління базами даних обрано `PostgreSQL`, об'єктно-реляційну СУБД з відкритим вихідним кодом, широко застосовувану у вебзастосунках різного масштабу. `PostgreSQL` підтримує транзакції ACID, розширені типи даних, зокрема `JSONB` для зберігання напівструктурованих відомостей про помилки, та надійні механізми індексування [21]. Для типобезпечної взаємодії з базою даних використано ORM-бібліотеку `TypeORM`.

`MySQL` та `MongoDB` оцінювалися як альтернативні СУБД. `MySQL` є зрілою реляційною СУБД, але поступається `PostgreSQL` у підтримці складних типів даних та розширень. `MongoDB` як документоорієнтована СУБД могла б спростити зберігання результатів перевірок, проте реляційна модель даних із

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

зовнішніми ключами є більш природною для структурованих сутностей системи (користувачі, ключі, запити, помилки) та забезпечує цілісність даних на рівні СУБД.

JWT використовується для автентифікації вебклієнта. Токен доступу має термін дії 15 хвилин, що відповідає сучасним практикам безпеки. Для програмного доступу до API реалізовано систему API-ключів, яка зберігає лише хеш ключа в базі даних. Це унеможливлює відновлення оригінального значення у разі компрометації бази даних [22].

Клієнтська частина реалізована з використанням бібліотеки React. Для збірки проекту застосовано Vite, сучасний інструмент збірки, що забезпечує миттєвий старт у режимі розробки та оптимізовану продуктивну збірку [23]. На стороні вебклієнта використовується TypeScript [24]. React Router DOM забезпечує клієнтську маршрутизацію між сторінками застосунку [25].

Вимоги до апаратного забезпечення для розгортання ML-сервісу визначаються обчислювальною складністю нейронних моделей. Для повноцінної роботи в точному режимі рекомендується використання GPU з обсягом відеопам'яті щонайменше 8 ГБ, що дозволяє скоротити час обробки одного запиту з приблизно 9 секунд на CPU до 1–2 секунд. Шлюз API та вебклієнт не мають підвищених вимог до апаратного забезпечення і можуть розгортатися на стандартних хмарних інстансах.

AWS Lambda та AWS S3 обрані для реалізації підсистеми генерації PDF-звітів. Lambda дозволяє виконувати ресурсоємну генерацію документів поза основним процесом шлюзу API, що не збільшує час відповіді для інших запитів. Тарифікація Lambda відбувається за фактичним часом виконання, що є економічно вигідним при рідкісних запитах на генерацію PDF. S3 забезпечує надійне довготривале зберігання згенерованих файлів з публічним доступом за URL.

Платформа Hugging Face Spaces обрана для розгортання ML-сервісу завдяки підтримці Docker-контейнерів, автоматичному кешуванню завантажених моделей між перезапусками та наявності безкоштовного рівня з

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

CPU-ресурсами, що є достатнім для навчального і демонстраційного середовища. При необхідності GPU-прискорення платформа надає платні тарифи з картами NVIDIA T4 без зміни коду сервісу.

Взаємодію між компонентами системи організовано за архітектурним стилем REST, що забезпечує уніфікований інтерфейс через стандартні HTTP-методи та JSON як формат обміну даними. Підхід є стандартним для вебAPI і підтримується будь-якою мовою програмування, що робить інтеграцію із сервісом максимально доступною для зовнішніх розробників.

React Router DOM використовується для маршрутизації між сторінками вебклієнта. Бібліотека axios застосовується для HTTP-запитів до шлюзу API з централізованим інтерсептором, що автоматично додає JWT-токен до заголовків вихідних запитів. Axios також обробляє відповіді зі статусом 401 та перенаправляє користувача на сторінку входу при закінченні строку дії токена.

Висновки до першого розділу

У першому розділі було розглянуто наявні рішення для перевірки та корекції граматики тексту й визначено, яке місце серед них може зайняти розроблюваний сервіс. Порівняння Grammarly, LanguageTool та UkrainianCorrection показало, що жоден із них не пропонує повноцінної нейронної корекції української мови в межах безкоштовного публічного API. Десь українська підтримується лише частково, десь немає програмного доступу, а просунуті функції доступні тільки платно. Саме ці прогалини й роблять розробку власного сервісу доцільною.

На основі аналізу було сформовано вимоги до системи та обрано її архітектуру. Оскільки нейронні моделі працюють на Python, а основна логіка на Node.js, від єдиного моноліту довелося відмовитися. Ресурсомісткий ML-сервіс винесено в окремий процес, який можна масштабувати незалежно, а решту функцій зібрано у шлюзі API, що став єдиною точкою входу до системи. Такий поділ дозволив підібрати найзручніший стек для кожної частини й спростив подальшу підтримку.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Відповідно до вимог визначено технології. Для ML-сервісу взято Python із FastAPI, для самої корекції гібридний підхід із двох українських нейромоделей та LanguageTool, для шлюзу NestJS, в якості БД PostgreSQL, а для вебклієнта React. Обрані рішення відповідають поставленим задачам та усувають недоліки наявних аналогів.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСЕРВІСУ ПЕРЕВІРКИ ТА КОРЕКЦІЇ ГРАМАТИКИ УКРАЇНСЬКОЇ МОВИ

2.1 Аналіз вимог та варіанти використання

Вебсервіс орієнтований на чотири категорії акторів: гість, що може переглядати головну сторінку та виконувати перевірку тексту без реєстрації; зареєстрований вебкористувач, що взаємодіє через браузерний інтерфейс; зовнішній розробник, що інтегрує сервіс через REST API за допомогою API-ключа; адміністратор, що має повний доступ до статистики системи та управління обліковими записами. Для наочного відображення взаємодії між акторами та функціями системи побудовано діаграму варіантів використання (рис. 2.1).

Функціональні вимоги визначають конкретні можливості, які система повинна надавати користувачам. Реєстрація нового облікового запису відбувається за адресою електронної пошти та паролем. Зареєстровані користувачі отримують JWT-токен після автентифікації та можуть перевіряти текст у трьох режимах – швидкому, середньому й точному. Відповідь на запит перевірки повинна містити виправлений текст, перелік виявлених помилок із зазначенням типу, позиції та запропонованого виправлення, а також час обробки. Авторизованим користувачам доступний журнал усіх виконаних перевірок із можливістю перегляду.

Передбачено генерацію API-ключів та управління ними, завдяки чому зовнішні розробники здійснюють програмну інтеграцію без браузерного інтерфейсу. Адміністративна панель повинна надавати статистику всіх запитів у системі, перелік зареєстрованих користувачів та можливість видалення облікових записів. Система також повинна підтримувати генерацію PDF-звіту для будь-якого збереженого запиту перевірки.

Нефункціональні вимоги визначають якісні характеристики системи. Паролі користувачів зберігаються виключно у вигляді bcrypt-хешу, API-ключі – у вигляді SHA-256 хешу, а передача даних між компонентами здійснюється через

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

HTTPS. Архітектура системи допускає горизонтальне масштабування ML-сервісу без внесення змін до коду шлюзу API.

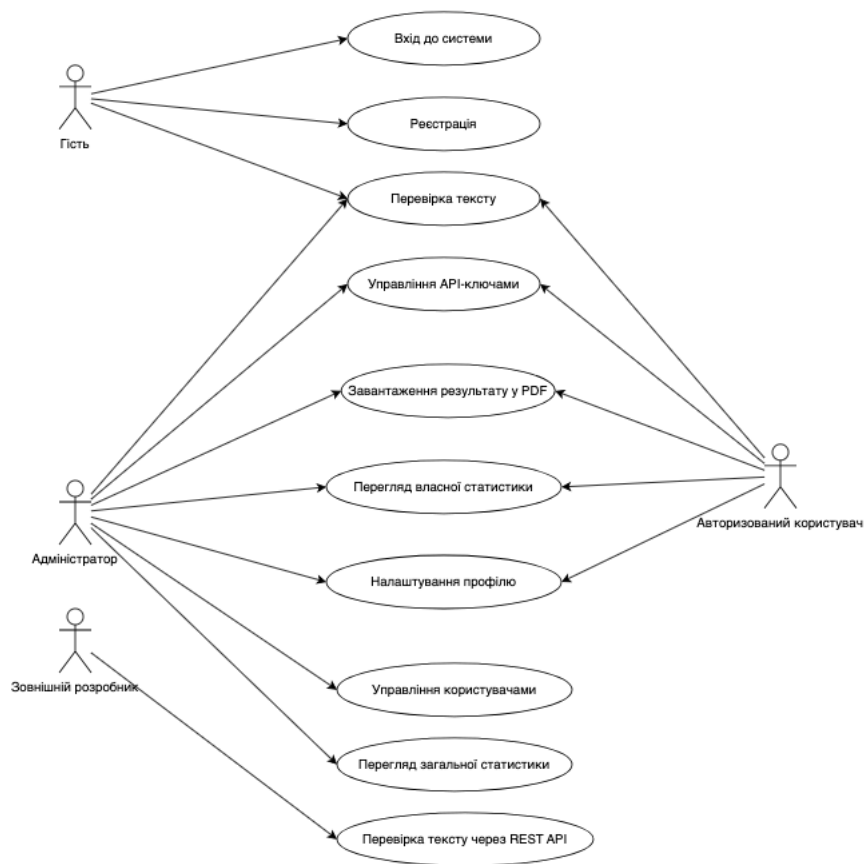


Рис. 2.1. Діаграма варіантів використання

Опис кожного варіанту використання наведено в таблицях 2.1-2.9.

Таблиця 2.1

Опис прецеденту «Реєстрація та вхід до системи»

Назва варіанту використання	Реєстрація та вхід до системи / Вхід через Google
Основні актори	Гість
Короткий опис	Незареєстрований користувач може створити обліковий запис, вказавши електронну адресу та пароль, або увійти через обліковий запис Google (OAuth 2.0). Зареєстрований користувач виконує вхід і отримує JWT-токен для автентифікації подальших запитів.

Таблиця 2.2

Опис прецеденту «Перевірка тексту»

Назва варіанту використання	Перевірка тексту
Основні актори	Авторизований користувач, Зовнішній розробник, Гість, Адміністратор
Короткий опис	Користувач або зовнішній клієнт надсилає текст і обирає режим перевірки (швидкий, середній або точний). Система повертає виправлений текст, перелік знайдених помилок із зазначенням позиції, типу і запропонованого виправлення, а також час обробки.

Таблиця 2.3

Опис прецеденту «Управління API-ключами»

Назва варіанту використання	Управління API-ключами
Основні актори	Авторизований користувач, Адміністратор
Короткий опис	Користувач генерує нові API-ключі для програмного доступу до сервісу, переглядає їх перелік (лише префікс ключа), перейменовує та видаляє наявні ключі. Повне значення ключа відображається одноразово під час генерації.

Таблиця 2.4

Опис прецеденту «Завантаження результату у PDF»

Назва варіанту використання	Завантаження результату у PDF
Основні актори	Авторизований користувач, Адміністратор
Короткий опис	Авторизований користувач формує PDF-звіт для будь-якого збереженого запиту перевірки. Звіт містить оригінальний та виправлений текст, перелік помилок із типами та позиціями, а також час обробки.

Таблиця 2.5

Опис прецеденту «Перегляд власної статистики»

Назва варіанту використання	Перегляд власної статистики
Основні актори	Адміністратор, Авторизований користувач

Короткий опис	Авторизований користувач переглядає власний журнал перевірок із детальною інформацією про кожен запит (оригінальний і виправлений текст, кількість та типи помилок, час обробки).
---------------	---

Таблиця 2.6

Опис прецеденту «Налаштування профілю»

Назва варіанту використання	Налаштування профілю
Основні актори	Авторизований користувач, Адміністратор
Короткий опис	Авторизований користувач переглядає та редагує дані свого облікового запису (електронну адресу, пароль).

Таблиця 2.7

Опис прецеденту «Управління користувачами»

Назва варіанту використання	Управління користувачами
Основні актори	Адміністратор
Короткий опис	Адміністратор переглядає перелік зареєстрованих користувачів та має можливість видалення облікових записів.

Таблиця 2.8

Опис прецеденту «Перегляд загальної статистики»

Назва варіанту використання	Перегляд загальної статистики
Основні актори	Адміністратор
Короткий опис	Адміністратор переглядає зведену статистику всіх запитів у систем (загальну кількість перевірок, розподіл за режимами, найпоширеніші типи помилок та активність користувачів).

Таблиця 2.9

Опис прецеденту «Перевірка тексту через REST API»

Назва варіанту використання	Перевірка тексту через REST API
Основні актори	Зовнішній розробник

Короткий опис	Зовнішній розробник надсилає POST-запит до REST API з текстом, обраним режимом та API-ключем у заголовку. Система повертає структурований JSON-результат із виправленим текстом, переліком помилок та часом обробки.
---------------	--

Діаграма послідовності відображає повний цикл обробки запиту на перевірку тексту – від дії користувача у браузері до збереження результату в базі даних.

Користувач вводить текст у редакторі вебклієнта, обирає режим перевірки та натискає кнопку «Перевірити». Вебклієнт формує POST-запит до ендпоінту `/api/check` з параметрами `text` і `mode`, додаючи до заголовка токен авторизації.

Перед тим як запит потрапить до контролера, він проходить через `FlexibleAuthGuard`. Охоронець перевіряє тип авторизації за трьома сценаріями, якщо передано JWT-токен – викликається `jwtService.verify()` і в об'єкт запиту записуються ідентифікатор, пошта та роль користувача; якщо передано API-ключ з префіксом `sk-...` – викликається `apiKeysService.validate()`, який повертає `userId` та `apiKeyId`; якщо токен відсутній – `request.user` встановлюється в `null` і запит продовжується як анонімний.

Після успішної автентифікації API Gateway проксує запит до ML-сервісу. ML-сервіс обробляє текст залежно від обраного режиму. У швидкому режимі виконується лише перевірка через `LanguageTool`. У середньому режимі текст спочатку обробляється `Pravopysnyk`, після чого результат додатково перевіряється `LanguageTool`. У точному послідовно запускаються `Pravopysnyk`, `Spivavtor` і `LanguageTool`, результати об'єднуються через `MergeStrategy.merge()`, після чого `LanguageTool` виконує два додаткових проходи для уточнення, а `ErrorClassifier` класифікує всі знайдені помилки за типами.

ML-сервіс повертає структуровану відповідь із полями `corrected`, `errors`, `error_count` та `processing_time_ms`. API Gateway асинхронно зберігає запис про перевірку в PostgreSQL через `loggingService.log()` із зазначенням `userId`, `clientType`, оригінального та виправленого тексту, кількості та переліку помилок.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебклієнт отримує відповідь 200 ОК, підсвічує знайдені помилки в редакторі та відображає виправлений текст разом зі списком помилок користувачу.

Для детальнішого опису взаємодії між компонентами системи в рамках основного сценарію побудовано діаграму послідовності перевірки тексту (додаток А, рис. 2)

Вебклієнт надсилає POST-запит до шлюзу API з текстом і обраним режимом. Шлюз виконує автентифікацію запиту через JWT або API-ключ, після чого проксує запит до ML-сервісу. ML-сервіс виконує корекцію тексту відповідно до обраного режиму й повертає структурований результат.

2.2 Проектування архітектури та бази даних

Проектування архітектури системи базується на принципі розділення відповідальності між трьома незалежними частинами, кожна з яких виконує чітко визначену роль.

ML-сервіс реалізований на Python з використанням FastAPI і розгорнутий у хмарному середовищі Hugging Face Spaces. Він відповідає виключно за обчислювальну частину – отримує текст і режим перевірки, виконує корекцію та повертає структурований результат. Шлюз API побудований на NestJS як модульний моноліт і є єдиною точкою входу для всіх клієнтів – він відповідає за автентифікацію, маршрутизацію, журналювання та взаємодію з базою даних. Вебклієнт реалізований на React і виконується безпосередньо у браузері користувача. Загальну архітектуру системи зображено на додатку А, рис. 1.

Взаємодія між усіма частинами системи відбувається виключно через HTTP. Вебклієнт і зовнішні розробники звертаються до шлюзу API через REST із JWT-автентифікацією або автентифікацією через API-ключ. Шлюз API звертається до ML-сервісу через внутрішній HTTP-запит, не відкриваючи ML-сервіс назовні – це суттєво підвищує безпеку системи, оскільки весь вхідний трафік проходить через єдину контрольовану точку.

Шлюз API побудовано за модульним принципом, де кожен модуль відповідає за окрему предметну область: Auth, Users, ApiKeys, Admin, Corrector,

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

CheckRequests та Pdf. Модулі взаємодіють через механізм ін'єкції залежностей NestJS, що забезпечує слабку зв'язаність і спрощує тестування кожного модуля окремо. На рис. 2.2 наведено діаграму компонентів системи.

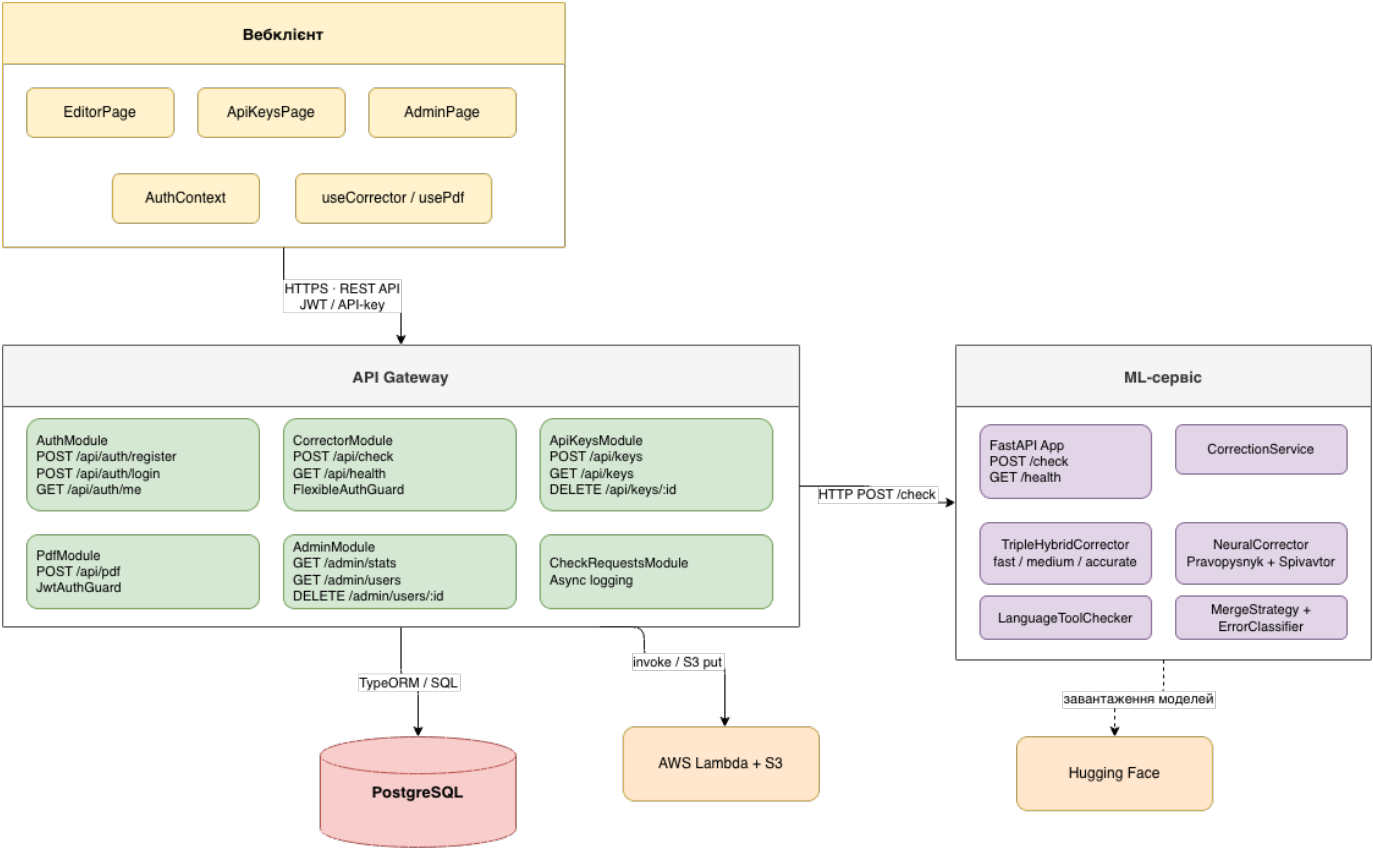


Рис. 2.2. Діаграма компонентів системи

При проєктуванні системи застосовано ряд класичних патернів проєктування [26], що сприяють слабкій зв'язаності компонентів, зручності розширення та тестування.

Патерн Strategy реалізовано в класі TripleHybridCorrector. Залежно від значення параметра mode метод correct() делегує обробку одній із трьох стратегій: FastStrategy передає текст лише до LanguageTool, MediumStrategy послідовно застосовує Pravopysnyk і LanguageTool, AccurateStrategy запускає обидві нейронні моделі паралельно і об'єднує результати через MergeStrategy (див. додаток Б). Це дозволяє додавати нові режими без зміни коду клієнта.

Патерн Проху застосовано в CorrectorModule шлюзу API. Модуль приймає запит клієнта, виконує валідацію та логування, після чого прозоро пересилає запит до ML-сервісу, а відповідь повертає без змін. Клієнт взаємодіє виключно з

проксі і не знає про деталі реалізації ML-сервісу, що спрощує заміну або горизонтальне масштабування останнього.

Патерн Facade реалізовано на рівні шлюзу API в цілому. Шлюз надає єдиний уніфікований інтерфейс для вебклієнта і зовнішніх розробників, приховуючи внутрішню складність, розподіл між ML-сервісом і базою даних, деталі автентифікації та асинхронне логування. Будь-які зміни у внутрішній архітектурі не вимагають оновлення клієнтського коду.

Патерн Repository застосовано через TypeORM. Кожна сутність (User, ApiKey, CheckRequest, RequestError, GeneratedPdf) має власний репозиторій, що абстрагує CRUD-операції від бізнес-логіки. Сервіси отримують репозиторії через ін'єкцію залежностей і не залежать від конкретної СУБД, що спрощує тестування через підміну репозиторію.

Патерн Chain of Responsibility реалізовано в системі guard-ів NestJS. Кожен вхідний запит послідовно проходить через ланцюжок перевірок. FlexibleAuthGuard визначає метод автентифікації (JWT або API-ключ), JwtAuthGuard валідує підпис токена, RolesGuard перевіряє роль користувача для адміністративних маршрутів. Якщо будь-яка ланка ланцюжка відхиляє запит, подальша обробка не виконується.

Патерн Observer реалізовано в асинхронному логуванні запитів. CheckRequestsModule підписується на події завершення перевірки. Після повернення відповіді клієнту модуль асинхронно зберігає деталі запиту і виявлені помилки в базі даних. Це забезпечує відсутність додаткової затримки для клієнта при збереженні повного журналу перевірок.

Патерн Adapter застосовано в компоненті LanguageToolChecker. LanguageTool надає Java-інтерфейс через JVM-процес, а Python-бібліотека language-tool-python є адаптером цього інтерфейсу для Python-коду. LanguageToolChecker додатково адаптує формат відповіді до внутрішньої структури помилок системи, ізолюючи решту ланцюжка від деталей LanguageTool API.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Патерн Singleton забезпечується механізмом провайдерів NestJS. За замовчуванням усі сервіси (AuthService, CorrectorService, ApiKeysService тощо) реєструються як синглтони в межах кореневого модуля. Це гарантує, що з'єднання з базою даних, HTTP-клієнт до ML-сервісу і кеш конфігурації створюються лише один раз і повторно використовуються для всіх вхідних запитів.

База даних. Зберігання даних забезпечує PostgreSQL. БД складається з п'яти таблиць: users, api_keys, check_requests, request_errors та generated_pdfs. Між таблицями встановлено зв'язки через зовнішні ключі, що забезпечує цілісність даних. На рис. 2.3 наведено ER-діаграму бази даних [21].

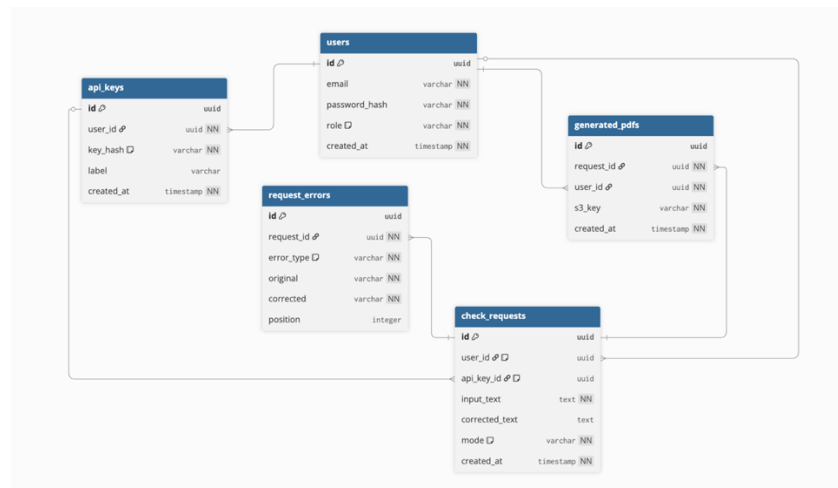


Рис. 2.3. ER-діаграма бази даних

Схема бази даних відповідає третій нормальній формі. Кожен атрибут функціонально залежить лише від первинного ключа і не залежить від інших неключових атрибутів. Первинні ключі всіх таблиць реалізовані як UUID, що генеруються функцією gen_random_uuid() PostgreSQL. Витік інформації про кількість записів через передбачуваність послідовних ідентифікаторів виключено.

Зовнішні ключі між таблицями забезпечують декларативну цілісність даних. Зовнішній ключ api_keys.user_id → users(id) налаштований із каскадним видаленням. При видаленні користувача всі його API-ключі видаляються автоматично. Зовнішній ключ check_requests.user_id → users(id) налаштований із SET NULL при видаленні, що дозволяє зберегти анонімні запити гостей і статистику навіть після видалення облікового запису.

Продуктивність запитів підвищено через індекси на полях із найвищою частотою вибірки. Таблиця `check_requests` індексується за `user_id` для швидкого отримання журналу конкретного користувача і за `created_at` для пагінованого виведення за датою. Таблиця `api_keys` має унікальний індекс на `key_hash` для ефективного пошуку при кожному API-запиті. Таблиця `users` має унікальний індекс на електронну пошту, що запобігає дублюванню облікових записів на рівні СУБД.

Поля з обмеженим набором значень реалізовано як PostgreSQL `enum`. Поле `role` таблиці `users` приймає значення `user` або `admin`, поле `client_type` таблиці `check_requests` - `web` або `api`. Використання `enum` замість рядкового типу `varchar` забезпечує цілісність на рівні бази даних. Некоректне значення відхиляється СУБД до виконання будь-якої бізнес-логіки застосунку.

Опис кожної з таблиць бази даних наведено в таблицях 2.10–2.14.

Таблиця 2.10

Опис таблиці `users`

Поле	Тип	Призначення
<code>id</code>	<code>uuid</code>	Унікальний ідентифікатор запису
<code>email</code>	<code>varchar</code>	Електронна адреса користувача
<code>password</code>	<code>varchar</code>	Хеш паролю (bcrypt)
<code>googleId</code>	<code>varchar</code>	Ідентифікатор облікового запису Google (nullable)
<code>role</code>	<code>enum</code> (user, admin)	Роль користувача у системі
<code>createdAt</code>	<code>timestamp</code>	Дата та час реєстрації

Таблиця 2.11

Опис таблиці `api_keys`

Поле	Тип	Призначення
<code>id</code>	<code>uuid</code>	Унікальний ідентифікатор ключа
<code>user_id</code>	<code>uuid</code>	Ідентифікатор власника (FK → users)
<code>key_hash</code>	<code>varchar</code>	SHA-256 хеш API-ключа
<code>key_prefix</code>	<code>varchar(8)</code>	Перші 8 символів ключа для відображення

продовж. табл. 2.11

name	varchar	Довільна назва ключа
is_active	boolean	Ознака активності ключа
created_at	timestamp	Дата та час створення
last_used_at	timestamp	Дата та час останнього використання

Таблиця 2.12

Опис таблиці check_requests

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор запиту
user_id	uuid	Ідентифікатор користувача (FK → users)
api_key_id	uuid	Ідентифікатор API-ключа (FK → api_keys)
client_type	enum (web, api)	Тип клієнта, що здійснив запит
original_text	text	Оригінальний текст запиту
corrected_text	text	Виправлений текст після обробки
error_count	integer	Загальна кількість виявлених помилок
processing_time_ms	float	Час обробки в мілісекундах
created_at	timestamp	Дата та час запиту

Таблиця 2.13

Опис таблиці request_errors

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор помилки
request_id	uuid	Ідентифікатор запиту (FK → check_requests)
error_type	varchar	Тип помилки (spelling, grammar, punctuation, style)
original	varchar	Оригінальний фрагмент тексту з помилкою
suggestion	varchar	Запропоноване виправлення
position	integer	Позиція помилки у вихідному тексті (символів)

Таблиця 2.14

Опис таблиці generated_pdfs

Поле	Тип	Призначення
id	uuid	Унікальний ідентифікатор запису
userId	uuid	Ідентифікатор власника (FK → users)
s3Url	varchar	Публічна URL-адреса PDF-файлу

s3Key	varchar	Ключ об'єкта у сховищі S3
createdAt	timestamp	Дата та час генерації файлу

2.3 Алгоритм гібридної корекції тексту

Алгоритм корекції побудовано на принципі поєднання трьох незалежних джерел виправлень з подальшим узгодженням їхніх результатів. Першим джерелом виступає нейронна модель Pravopysnyk/best-unlp, спеціалізована на корекції україномовних текстів. Другим джерелом є велика мовна модель grammarly/spivavtor-large з підтримкою інструктивної взаємодії. Третім джерелом слугує аналізатор LanguageTool, що працює за заздалегідь визначеними граматичними правилами. Поєднання нейронних моделей з аналізатором заснованим на правилах дозволяє компенсувати слабкі сторони окремих методів. Нейронні моделі краще опрацьовують контекстно залежні помилки, тоді як аналізатор на основі правил надійно виявляє формальні порушення на зразок пунктуації та узгодження за відмінками.

Загальний процес корекції складається з кількох послідовних етапів. Спочатку вхідний текст надходить до всіх трьох джерел, і кожне з них формує власний виправлений варіант. Перед подальшою обробкою система підраховує, скільки слів змінило кожне джерело. Якщо жодне джерело не внесло змін, текст повертається без обробки. У протилежному випадку отримані варіанти зіставляються з оригіналом на рівні окремих слів, і для кожного слова ухвалюється рішення, який варіант залишити. Після цього текст проходить додаткову нормалізацію, а на завершення система визначає тип кожної внесеної зміни, щоб показати користувачеві не лише виправлений текст, а й перелік помилок із поясненнями.

Модель Pravopysnyk/best-unlp базується на архітектурі mBART, що є багатомовною трансформерною моделлю з кодувальником та декодувальником. Її навчено на корпусі UA-GEC, який містить автентичні україномовні тексти з анотованими виправленнями. Модель отримує вхідний текст безпосередньо, без додаткових інструкцій, і генерує виправлений варіант методом beam search із шириною пучка 5. Для довгих текстів, що перевищують ліміт у 400 токенів,

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				37
Змн.	Арк.	№ докум.	Підпис	Дата		

реалізовано посегментну обробку. У такому режимі вхід розбивається на речення за пунктуаційними межами, а кожне речення обробляється окремо із подальшим склеюванням результатів.

Модель *grammarly/spivavtor-large* є інструктивно-налаштованою великою мовною моделлю для редагування тексту з підтримкою слов'янських мов. На відміну від *Pravopysnyk*, яка працює виключно у форматі *seq2seq*, *Spivavtor* приймає текст разом із текстовою інструкцією «Виправте помилки в цьому тексті», що забезпечує гнучкішу поведінку моделі. Large-варіант обрано через його здатність моделювати складніші граматичні залежності порівняно з базовими варіантами. Модель була донавчена на корпусах текстів із виправленнями, що дозволяє їй ефективно виявляти та коригувати орфографічні, граматичні й стилістичні помилки. Технічні параметри генерації, зокрема ширина *beam search* та обмеження довжини послідовності, аналогічні до *Pravopysnyk*.

Аналізатор *LanguageTool* виконується як окремий JVM-процес, до якого ML-сервіс звертається через бібліотеку *language-tool-python*. Для кожного запиту *LanguageTool* повертає перелік об'єктів типу *Match*, кожен з яких містить позицію помилки, категорію правила та впорядкований за рейтингом перелік запропонованих замін. Алгоритм застосовує лише першу пропозицію з переліку, оскільки вона має найвищий внутрішній рейтинг релевантності. Захист від помилкових виправлень забезпечує додатковий фільтр на основі коефіцієнта схожості, обчисленого через *SequenceMatcher*. Якщо схожість між оригінальним та запропонованим словами становить менше 0,7, заміна не застосовується, що захищає від випадків спотворення малознайомих моделі лексем.

Поріг схожості 0,7 підібрано експериментально. За нижчого значення система пропускала надто сміливі заміни, коли аналізатор пропонував зовсім інше слово замість незнайомого йому терміна. За вищого значення відхилялися навіть коректні виправлення з невеликою різницею в написанні. Значення 0,7 дозволяє приймати виправлення окремих літер і закінчень, але відхиляє повну

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

заміну слова, яку правилний аналізатор не може впевнено підтвердити. Завдяки цьому рідковживані слова, власні назви та терміни залишаються недоторканими.

Сервіс підтримує три режими обробки тексту з різним балансом між швидкістю та точністю. Швидкий режим використовує лише LanguageTool без залучення нейронних моделей. Середній режим виконує послідовно один прохід Pravopysnyk та один прохід LanguageTool, що дає прийнятну якість корекції за помірний час обробки. Точний режим активує всі три джерела з подальшим голосуванням і додатковими проходами LanguageTool, забезпечуючи максимальну точність ціною найбільшого часу очікування. Загальну схему ланцюжка гібридної корекції у точному режимі зображено в додатку А, рис. 3.

Ключовим компонентом гібридної корекції є модуль MergeStrategy (додаток А, рис. 4), що відповідає за узгодження результатів трьох джерел на рівні окремих слів. Для кожного з трьох виправлених варіантів спочатку виконується вирівнювання з оригінальним текстом за допомогою алгоритму SequenceMatcher [27], який визначає відповідність між позиціями токенів через операції збереження, заміни, видалення та вставки. Результатом вирівнювання є три відображення «позиція оригінального слова – відповідне виправлене слово», що надалі використовуються для прийняття консенсусного рішення.

Окремої уваги потребують випадки, коли довжина оригінального та виправленого фрагментів не збігається. Якщо кількість слів однакова, кожному оригінальному слову ставиться у відповідність його виправлений варіант. Якщо нейронна модель намагається замінити кілька слів одним, таке злиття відхиляється, оскільки воно часто свідчить про втрату частини тексту. Якщо джерело пропонує видалити слово, рішення залежить від його типу. Для нейронних моделей видалення відхиляється через високий ризик втрати змісту, а для LanguageTool воно приймається, бо правилний аналізатор видаляє слова лише у формально визначених випадках. Вставлені слова приєднуються до сусіднього токена, щоб зберегти правильний порядок слів у реченні.

Прийняття рішення для кожної позиції токена ґрунтується на правилах більшості. Якщо всі три джерела пропонують той самий варіант, він приймається безумовно. Якщо два джерела пропонують однакову заміну, перевага надається консенсусу двох незалежно від того, що пропонує третє. У випадку, коли всі три джерела дають різні виправлення, спрацьовує пріоритетна послідовність Pravopysnyk, Spivavtor, LanguageTool, оскільки спеціалізована модель найкраще розуміє україномовний контекст.

Якщо змінило слово лише одне джерело, рішення залежить від типу цього джерела. Для LanguageTool така одиночна заміна приймається, тоді як для нейронних моделей виконується додаткова перевірка на наявність артефакту розбиття слова, яка убезпечує від помилок на зразок «неможливо» → «нем ожливо». Випадки, коли нейронна модель намагається видалити слова або об'єднати їх у блок, відхиляються через високий ризик втрати інформації.

Перевірка на артефакт розбиття слова працює таким чином. Якщо нейронна модель змінила слово і водночас змінила сусіднє з ним, система склеює відповідну пару слів в оригіналі та у виправленні і порівнює їх без урахування регістру та розділових знаків. Якщо склеєні рядки збігаються, це означає, що модель не виправила помилку, а лише неправильно поділила слово на частини. У такому разі зміна відхиляється, і оригінальне слово залишається без змін. Подібна перевірка усуває типову ваду нейронних моделей, коли коректне слово помилково розривається пробілом, як у прикладі «неможливо» та «нем ожливо».

У точному режимі після завершення основного голосування виконуються додаткові оптимізаційні проходи. LanguageTool повторно обробляє вже скоригований текст щонайбільше двічі поспіль для усунення помилок, які могли виникнути на стику виправлень від різних нейронних моделей. Цикл припиняється достроково, якщо черговий прохід не змінює тексту. Завершальним етапом є фінальний прохід Pravopysnyk виключно для нормалізації регістру літер. Якщо модель пропонує інше написання слова, але оригінал та пропозиція збігаються при порівнянні в нижньому регістрі, застосовується лише виправлення регістру без зміни самого слова.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

Перед поверненням результату текст проходить етап нормалізації. На ньому усуваються повторені поспіль розділові знаки, прибирається випадкове дублювання останнього слова та зайві пробіли, що могли виникнути під час склеювання виправлень від різних джерел. Цей етап не змінює зміст тексту, а лише приводить його до охайного вигляду перед відображенням користувачеві.

Після формування остаточного варіанта тексту модуль ErrorClassifier здійснює класифікацію виявлених розбіжностей між оригінальним і виправленим текстами (див. додаток Г). Категорії правил LanguageTool відображаються на уніфіковану таксономію з п'яти типів помилок. Категорії GRAMMAR, TYPOS, PUNCTUATION та CASING конвертуються відповідно у граматичні, орфографічні, пунктуаційні помилки та помилки регістру, а решта категорій належать до стилістичних зауважень. По кожній помилці фіксуються позиція в оригінальному тексті, її довжина, оригінальний фрагмент, запропонована заміна та текстове повідомлення для користувача. Уніфікований формат структури помилки забезпечує сумісність між усіма трьома режимами роботи та спрощує подальшу візуалізацію у вебклієнті.

До переліку виявлених помилок окремо додаються стилістичні зауваження від LanguageTool. Після основної класифікації система переглядає стилістичні зауваження аналізатора й додає лише ті з них, що не перетинаються за позицією з уже знайденими помилками. Це потрібно, щоб одна й та сама ділянка тексту не позначалася двічі. Стилiстичні зауваження мають рекомендаційний характер і не завжди змінюють текст, тому вони показуються користувачеві як поради щодо покращення, а не як обов'язкові виправлення. Після додавання всі помилки впорядковуються за позицією в тексті, що забезпечує їх послідовне відображення в редакторі від початку до кінця документа..

Класифікація помилок виконується у два кроки. Спочатку система намагається визначити тип помилки за категорією відповідного правила LanguageTool, зіставляючи позицію зміни з позиціями знайдених правил. Якщо для певної зміни правило не знайдено, тип визначається за набором простих ознак. Зміна, що відрізняється лише регістром літер, вважається помилкою

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савiцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

реєстру. Зміна, що зачіпає тільки розділові знаки, належить до пунктуаційних. Заміна одного слова схожим за написанням класифікується як орфографічна, а одночасна зміна кількох слів вважається граматичною. Перед класифікацією великі блоки змін розбиваються за межами речень, щоб кожна помилка стосувалася окремого фрагмента й мала точну позицію в тексті.

2.4 Реалізація функціоналу

Шлюз API надає REST-інтерфейс для взаємодії з системою [28]. Усі маршрути, що потребують автентифікації, захищені компонентом FlexibleAuthGuard, який підтримує два способи автентифікації Bearer JWT-токен у заголовку Authorization або API-ключ. Основні маршрути системи описано в таблиці 2.15.

Таблиця 2.15

Основні маршрути REST API

Метод	Маршрут	Опис	Автентифікація
POST	/api/auth/register	Реєстрація нового користувача	Ні
POST	/api/auth/login	Вхід та отримання JWT-токена	Ні
GET	/api/auth/me	Отримання даних поточного користувача	JWT
POST	/api/check	Перевірка та корекція тексту	JWT або API-ключ
GET	/api/health	Перевірка стану ML-сервісу	Ні
POST	/api/keys	Генерація нового API-ключа	JWT
GET	/api/keys	Отримання переліку власних ключів	JWT
DELETE	/api/keys/:id	Видалення API-ключа	JWT
GET	/api/admin/stats	Статистика всіх запитів	JWT (admin)
GET	/api/auth/google	Перенаправлення до Google для OAuth 2.0	Ні

GET	/api/auth/google	Перенаправлення до Google для OAuth 2.0	Ні
GET	/api/auth/google/callback	Обробка OAuth-callback, видача JWT	Ні
POST	/api/pdf	Генерація PDF-звіту для перевірки	JWT
GET	/user/history	Журнал запитів поточного користувача	JWT
GET	/user/history/:id	Деталі конкретного запиту	JWT
GET	/user/stats	Статистика запитів користувача	JWT
GET	/admin/errors/top	Розподіл помилок за типами (admin)	JWT (admin)
GET	/admin/users	Перелік всіх користувачів (admin)	JWT (admin)
DELETE	/admin/users/:id	Видалення облікового запису (admin)	JWT (admin)

Маршрут POST /api/check приймає JSON-тіло з полями text (рядок, від 1 до 10 000 символів) і режим (одне зі значень швидкий, середній або точний). Успішна відповідь містить corrected_text, масив errors із полями type, start, end, original і suggestion, а також processing_time_ms. Некоректні вхідні дані повертають статус 400 із переліком порушених правил валідації.

Усі відповіді API повертаються у форматі JSON зі стандартними HTTP-кодами стану. Код 200 - успішний запит, 201 - успішне створення ресурсу, 400 - некоректні вхідні дані, 401 - відсутня або недійсна автентифікація, 403 - недостатньо прав доступу, 409 - конфлікт (наприклад, електронна пошта вже зайнята), 503 - тимчасова недоступність ML-сервісу. У разі помилки тіло відповіді містить поле message з описом причини та поле statusCode.

API-ключі генеруються як криптографічно стійкі 32-байтні рядки через Node.js crypto.randomBytes(). У базі даних зберігається лише SHA-256 хеш і перші 8 символів як префікс для відображення в інтерфейсі. При автентифікаційному запиті сервер обчислює SHA-256 хеш ключа з заголовка і

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

порівнює з хешем у базі даних, що унеможлиблює відновлення оригінального значення навіть у разі витоку бази даних.

Система автентифікації та авторизації. Безпека вебсервісу забезпечується дворівневою системою автентифікації, на основі JWT-токенів для вебклієнта та на основі API-ключів для зовнішніх інтеграцій [22]. Обидва механізми підтримуються єдиним компонентом FlexibleAuthGuard на рівні шлюзу API, що усуває необхідність дублювання логіки перевірки у різних маршрутах.

Система підтримує вхід через обліковий запис Google за протоколом OAuth 2.0 [29]. Маршрут GET /api/auth/google перенаправляє браузер на сторінку авторизації Google, після чого Google повертає код авторизації на зворотній маршрут GET /api/auth/google/callback. Шлюз API обмінює код на профіль користувача, знаходить або створює запис у таблиці users і видає JWT-токен для подальшої автентифікації.

Щойно автентифікацію підтверджено, видається JWT-токен доступу з терміном дії 15 хвилин. Термін дії токена мінімізує вікно можливих зловживань у разі його перехоплення [22].

Токен містить стандартні поля JWT: ідентифікатор користувача, пошта та роль. Підпис виконується алгоритмом HS256 з секретним ключем, що зберігається у змінних середовища. При кожному захищеному запиті FlexibleAuthGuard перевіряє підпис та термін дії токена.

Паролі користувачів зберігаються виключно у вигляді хешів, обчислених алгоритмом bcrypt із коефіцієнтом навантаження 10. Алгоритм bcrypt за рахунок адаптивної складності обчислення забезпечує стійкість до атак із використанням передобчислених таблиць (rainbow tables) та до брутфорс-атак на сучасному обладнанні. При вході система обчислює хеш введеного пароля і порівнює результат з хешем у базі даних за допомогою функції bcrypt.compare, що виключає порівняння у вигляді відкритого тексту [30].

Система API-ключів забезпечує програмний доступ до сервісу без використання JWT. Під час генерації нового ключа система створює криптографічно безпечний рядок довжиною 32 байти з використанням

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

crypto.randomBytes. Повне значення ключа повертається користувачу одноразово під час створення і не зберігається у відкритому вигляді. У базі даних зберігається лише SHA-256 хеш ключа та перші 8 символів (префікс) для ідентифікації ключа у списку. Під час кожного запиту система обчислює SHA-256 хеш отриманого ключа і порівнює результат з хешованими значеннями у таблиці `api_keys`.

Розмежування доступу реалізовано за допомогою рольової моделі (RBAC). Система підтримує дві ролі `user` та `admin`. Звичайний користувач має доступ до перевірки тексту, управління власними API-ключами та перегляду особистої статистики. Адміністратор додатково має доступ до агрегованої статистики всіх запитів та управління обліковими записами. Перевірка ролі здійснюється декоратором `Roles` у комбінації з `RolesGuard`, що виконується після `FlexibleAuthGuard` у ланцюжку стражів `NestJS` [18].

Запит від клієнта проходить через `FlexibleAuthGuard`, який визначає тип автентифікації за наявністю заголовка `Authorization: Bearer` (JWT або API-ключ з префіксом `sk-`). У разі JWT виконується верифікація підпису та перевірка терміну дії. У разі API-ключа обчислюється SHA-256 хеш і виконується пошук відповідного запису у базі даних. У разі успіху запит передається до контролера з об'єктом користувача у контексті запиту.

Інтерфейс користувача. Вебклієнт реалізовано з використанням бібліотеки `React` та мови `TypeScript` [23, 24]. Такий підхід забезпечує плавну навігацію між розділами без перезавантаження сторінки та дозволяє реалізувати інтерактивний редактор тексту з підсвічуванням помилок у режимі реального часу. Збірка проєкту виконується інструментом `Vite`, який забезпечує миттєвий старт сервера розробки та оптимізовану виробничу збірку [25].

Маршрутизація на стороні клієнта реалізована за допомогою `React Router` [31]. Застосунок містить такі основні маршрути: `/` (головна сторінка з редактором тексту), `/keys` (управління API-ключами), `/profile` (профіль, журнал перевірок та налаштування), `/admin` (панель адміністратора, доступна лише для ролі `admin`). Захищені сторінки перевіряють наявність автентифікації самостійно

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

через хук `useAuth`, якщо токен відсутній, виконується перенаправлення на сторінку входу.

Головна сторінка містить основний функціональний елемент - редактор тексту з підтримкою підсвічування помилок. Користувач вводить текст у текстове поле, обирає режим перевірки (швидкий, середній або точний) і натискає кнопку «Перевірити». Отримавши відповідь від API, застосунок відображає виправлений текст із підсвіченими фрагментами, що відповідають виявленим помилкам. Натискання на підсвічений фрагмент відкриває спливаючий блок з детальною інформацією: тип помилки, оригінальний фрагмент та запропоновані виправлення.

Підсвічування помилок реалізовано компонентом `HighlightedText`, який розбиває виправлений текст на фрагменти відповідно до масиву об'єктів помилок, отриманого від API. Кожен об'єкт помилки містить поля `position`, `length`, `error_type` та `suggestion`. Фрагменти тексту, що відповідають помилкам, відображаються як `span`-елементи з кольоровим підкресленням, колір якого визначається типом помилки: червоний для орфографічних (`spelling`), фіолетовий для граматичних (`grammar`), помаранчевий для пунктуаційних (`punctuation`), синій для стилістичних (`style`).

Компоненти застосунку організовані у дві основні групи. Група `layout` (`MainLayout`, `Header`) відповідає за загальну розмітку та навігацію між сторінками. Група `editor` містить компоненти `TextEditorPanel`, `HighlightedText`, `ErrorSidebar`, `EditorToolbar`, `SuggestionPopup`, `StatusBar`, `ErrorLegend` та `SidebarErrorItem`, які використовуються для відображення редактора тексту та підсвічування виявлених помилок.

Управління станом застосунку реалізовано за допомогою `React Context API` та хука `useState`. Глобальний контекст `AuthContext` зберігає дані поточного користувача та стан автентифікації. Для сторінки редактора використовується хук `useCorrector`, що інкапсулює логіку взаємодії з API, управління станом тексту, помилок та обраного режиму перевірки (див. додаток Д). Такий підхід

усуває необхідність підключення сторонньої бібліотеки управління станом (Redux або MobX) для проєктів середнього масштабу.

Сторінка профілю (/profile) реалізована у вигляді трьох вкладок: статистика, журнал перевірок та налаштування. Вкладка статистики відображає загальну кількість перевірок користувача та розподіл виявлених помилок за типами. Вкладка журналу перевірок відображає пагінований список минулих запитів із зазначенням дати, кількості помилок і типу клієнта (web або api). При натисканні на запис відкривається детальний перегляд із оригінальним та виправленим текстом, а також переліком знайдених помилок. Вкладка налаштувань надає можливість змінити пароль та пошта облікового запису.

Сторінка управління API-ключами (/keys) відображає таблицю наявних ключів із полями: назва, префікс ключа (перші 8 символів), дата створення та дата останнього використання. Повне значення ключа відображається одноразово безпосередньо після генерації у захищеному діалоговому вікні, яке рекомендує користувачу скопіювати значення перед закриттям.

Панель адміністратора (/admin) надає агреговану статистику системи: загальна кількість запитів, кількість унікальних користувачів, розподіл запитів за типами клієнтів, середній час обробки. Дані відображаються у вигляді карток зі значеннями та графіків, побудованих бібліотекою recharts.

Генерація PDF-звітів. PDF-звіт генерується за запитом користувача через вебінтерфейс або REST API. Клієнт надсилає POST-запит до /api/pdf з оригінальним текстом, виправленим текстом і списком помилок; шлюз перевіряє JWT і передає дані до AWS Lambda [32].

Lambda генерує документ із такими розділами: заголовок (дата, режим перевірки), оригінальний і виправлений текст, таблиця помилок (тип / оригінал / пропозиція / позиція), статистика за типами та час обробки. Кирилиця підтримується через шрифт Unicode (DejaVu Sans або Noto Sans). Готовий файл завантажується до Amazon S3, Lambda повертає публічну URL і S3-ключ, які шлюз зберігає в таблиці generated_pdfs і повертає клієнту. Для закритих бакетів можна використовувати pre-signed URL з обмеженим терміном дії [33 - 35].

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

Логування та моніторинг. Шлюз API логує всі HTTP-запити, зовнішні виклики до ML-сервісу, операції з БД та помилки - із часовою міткою, рівнем (log / warn / error / debug) і контекстом. Доступність ML-сервісу перевіряється через /api/health. Шлюз надсилає GET до ML-сервісу і повертає агрегований статус. Якщо сервіс недоступний або відповідає довше 5 секунд - повертається 503.

Статистика запитів агрегується SQL через TypeORM безпосередньо з таблиці check_requests - без окремих агрегаційних таблиць і фонових задач. Для продуктивності на полях created_at, user_id і client_type створено індекси.

Висновки до другого розділу

У другому розділі було спроектовано систему та описано, як вона працює. Визначено основні групи користувачів: гість, зареєстрований користувач, зовнішній розробник, адміністратор. На основі цього сформовано функціональні вимоги: перевірка тексту в кількох режимах, ведення журналу запитів та керування API-ключами, а також вимоги до безпеки й масштабованості. Основні сценарії роботи показано на діаграмах варіантів використання та послідовності.

Окрему увагу приділено архітектурі й базі даних. Систему поділено на три незалежні частини, що спілкуються лише через HTTP, де шлюз API приховує ML-сервіс від зовнішнього доступу й бере на себе автентифікацію, перевірку прав і журналювання. Щоб знизити зв'язаність компонентів, застосовано низку класичних патернів проектування. Для зберігання даних взято PostgreSQL із п'ятьма таблицями, зв'язаними зовнішніми ключами, що забезпечує цілісність даних навіть після видалення окремих записів.

Центральною частиною розділу став алгоритм гібридної корекції. Він поєднує дві українські нейромоделі з редактором на основі правил LanguageTool і працює у трьох режимах, що відрізняються балансом між швидкістю та точністю. Результати джерел узгоджує голосування MergeStrategy, яке приймає лише ті виправлення, що підтверджені принаймні двома джерелами, а наприкінці кожна знайдена помилка отримує свій тип (див. додаток В). Такий підхід зменшує кількість хибних виправлень, властивих окремій нейромоделі.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБСЕРВІСОМ ПЕРЕВІРКИ ТА КОРЕКЦІЇ ГРАМАТИКИ УКРАЇНСЬКОЇ МОВИ

3.1 Порядок встановлення та налаштування параметрів системи

Система складається з чотирьох розгорнутих компонентів, а саме ML-сервіс на Hugging Face Spaces, шлюз API і база даних на Railway, вебклієнт на Vercel. Кожен компонент розгортається незалежно, що дозволяє оновлювати окремі частини системи без зупинки інших.

Шлюз API і база даних PostgreSQL розміщені на Railway в одній приватній мережі, що забезпечує низьку затримку між ними та унеможливорює прямий доступ до бази даних ззовні. ML-сервіс розгорнуто на Hugging Face Spaces у Docker-контейнері і доступний виключно через шлюз API – прямі запити від вебклієнта до нього не передбачені. Вебклієнт розгорнуто на Vercel, файли якого розповсюджуються через глобальну CDN-мережу. Генерація PDF виконується через AWS Lambda із збереженням готового файлу у Amazon S3, що усуває необхідність виділеного сервера для цієї операції [33, 35]. Діаграму розгортання системи наведено на рис. 3.1.

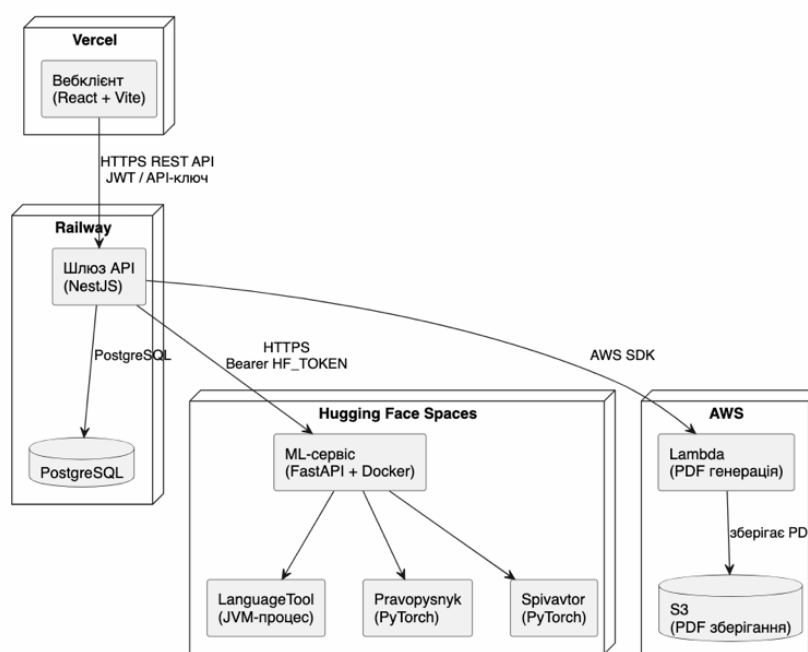


Рис. 3.1. Діаграма розгортання системи

ML-сервіс розгортається автоматично з репозиторію через Docker-контейнер [36]. Hugging Face Spaces зчитує Dockerfile з кореня репозиторію і

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

будує образ при кожному push до гілки main. Контейнер базується на образі python:3.10-slim, встановлює Java 21 для LanguageTool та Python-залежності з requirements.txt. Сервіс запускається через uvicorn на порту 7860, який є стандартним для Hugging Face Spaces.

Після розгортання сервіс доступний за публічною адресою вигляду `https://<username>-<space-name>.hf.space`. Цю адресу необхідно вказати у змінній середовища. Захист ML-сервісу від прямих звернень забезпечується токеном, який вмикається в налаштуваннях Space – тоді всі запити мають містити заголовок `Authorization: Bearer <HF_TOKEN>`.

Розгортання шлюзу API і бази даних на Railway. Railway автоматично визначає NestJS-проект і будує його через `npm run build`. База даних PostgreSQL розгортається як окремий сервіс Railway і підключається до шлюзу API через змінну середовища `DATABASE_URL`, яку Railway підставляє автоматично.

Для коректної роботи шлюзу API необхідно налаштувати такі змінні середовища в панелі Railway таблиця 3.1.

Таблиця 3.1

Налаштування середовища Railway

DATABASE_URL	URL підключення до PostgreSQL (підставляється Railway автоматично)
JWT_SECRET	Секретний ключ для підпису JWT-токенів
ML_SERVICE_URL	URL ML-сервісу на Hugging Face Spaces
HF_TOKEN	Токен доступу до Hugging Face
GOOGLE_CLIENT_ID	Ідентифікатор OAuth-застосунку Google
GOOGLE_CLIENT_SECRET	Секрет OAuth-застосунку Google
GOOGLE_CALLBACK_URL	URL зворотного маршруту після авторизації Google
AWS_ACCESS_KEY_ID	Ключ доступу AWS для Lambda та S3
AWS_SECRET_ACCESS_KEY	Секретний ключ AWS
AWS_REGION	Регіон AWS
AWS_LAMBDA_FUNCTION_NAME	Назва Lambda-функції для генерації PDF

Схема бази даних створюється автоматично під час першого запуску шлюзу API завдяки параметру `synchronize: true` у конфігурації TypeORM [20].

При старті застосунку TypeORM зчитує визначення всіх сутностей системи – User, ApiKey, CheckRequest, RequestError та GeneratedPdf – і порівнює їх із поточним станом бази даних. Якщо потрібних таблиць ще не існує, вони створюються автоматично разом із зовнішніми ключами, індексами та обмеженнями, описаними в коді сутностей. Завдяки цьому розгортання чистого екземпляра системи не потребує жодних ручних SQL-скриптів, адже достатньо підключити порожню базу даних, і структура сформується самостійно..

Вебклієнт розгортається на платформі Vercel, яка спеціалізується на хостингу статичних застосунків та інтегрується безпосередньо з репозиторієм проєкту. Vercel автоматично визначає, що проєкт побудовано на Vite, і виконує збірку командою `npm run build` без додаткового налаштування. Під час збірки Vite компілює вихідний код React та TypeScript у набір оптимізованих статичних файлів – HTML, CSS та JavaScript, – які розміщуються в каталозі `dist`. Ці файли розповсюджуються через глобальну мережу доставки контенту (CDN) Vercel, завдяки чому застосунок завантажується швидко незалежно від географічного розташування користувача, а з'єднання захищене HTTPS за замовчуванням [25, 37].

Збірка запускається автоматично при кожному надсиланні змін до основної гілки репозиторію, тож оновлення вебклієнта не потребує ручних дій. Для кожного коміту чи окремої гілки Vercel також може створювати попередні (preview) розгортання, що дозволяє перевірити зміни на окремій адресі ще до їх публікації в основній версії. Оскільки застосунок реалізовано як односторінковий, платформа налаштована повертати головний файл `index.html` для всіх маршрутів, завдяки чому клієнтська маршрутизація React Router працює коректно навіть при прямому переході за внутрішнім посиланням або оновленні сторінки.

Для підключення вебклієнта до шлюзу API в налаштуваннях проєкту Vercel задається одна змінна середовища `VITE_API_URL`, яка містить публічну адресу шлюзу API на Railway. Особливістю Vite є те, що змінні з префіксом `VITE_` підставляються у код під час збірки, а не зчитуються під час виконання,

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

тому після зміни цієї адреси проєкт потрібно зібрати заново. Окремо на стороні шлюзу API адресу вебклієнта додають до переліку дозволених джерел (CORS), інакше браузер блокуватиме запити з домену Vercel до іншого домену Railway. Таке розмежування доменів є типовим для сучасних вебзастосунків і не впливає на зручність роботи користувача.

Порядок першого запуску. Оскільки компоненти системи залежать один від одного, їх розгортання виконується у визначеній послідовності, де кожен наступний крок використовує результати попереднього. Недотримання цього порядку призвело б до того, що окремі компоненти не змогли б знайти своїх залежностей під час старту.

Першим розгортається ML-сервіс на Hugging Face Spaces. Після успішної збірки Docker-образу та запуску контейнера фіксується його публічна адреса, а за потреби активується токен захисту від прямих звернень. Ця адреса є вхідним параметром для наступного кроку, тому без працездатного ML-сервісу немає сенсу піднімати решту системи.

Другим кроком розгортається база даних і шлюз API на Railway. У панелі Railway створюється екземпляр PostgreSQL, а у змінні середовища шлюзу вносяться всі необхідні значення, зокрема адреса ML-сервісу, отримана на попередньому кроці, секретний ключ для підпису JWT, дані OAuth-застосунку Google та параметри доступу до AWS. Під час першого запуску шлюз підключається до бази даних і автоматично створює потрібну схему, після чого стає доступним за власною публічною адресою. На цьому етапі доцільно перевірити стан системи через службовий маршрут перевірки доступності ML-сервісу, що підтверджує правильність налаштувань.

Останнім розгортається вебклієнт на Vercel із зазначеною адресою шлюзу API у змінній VITE_API_URL. Оскільки до цього моменту шлюз уже працює й має відому адресу, зібраний клієнт одразу спрямовує свої запити за правильним призначенням. Така послідовність забезпечує наявність усіх залежностей до моменту першого запиту користувача, адже ML-сервіс готовий приймати текст на обробку, шлюз API має доступ і до нього, і до бази даних, а вебклієнт знає,

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

куди надсилати запити. Після завершення всіх трьох кроків система вважається повністю розгорнутою й готовою до роботи.

3.2 Інтерфейс вебкористувача

Вебклієнт реалізовано як односторінковий застосунок. Це означає, що інтерфейс завантажується один раз, а подальша взаємодія з користувачем відбувається без повного перезавантаження сторінки, що забезпечує плавну роботу редактора й миттєве відображення результатів перевірки. Застосунок виконується безпосередньо у браузері та взаємодіє із сервером виключно через шлюз API.

Навігація організована через фіксований заголовок. У ньому розміщено логотип сервісу, що веде на головну сторінку з редактором. Для неавторизованих відвідувачів у заголовку відображаються посилання «Увійти» та «Реєстрація», а для авторизованих – електронна адреса користувача (перехід до профілю), посилання на керування API-ключами та кнопка виходу. Адміністраторам додатково показується посилання на адміністративну панель.

Неzareєстрований користувач одразу отримує доступ до головної сторінки з редактором і може скористатися перевіркою тексту в гостьовому режимі – окремого екрана входу при першому відвідуванні не передбачено. Вхід і реєстрація винесені на окремі сторінки, перехід до яких здійснюється за відповідними посиланнями. Натомість сторінки профілю, керування ключами та адміністрування є захищеними, спроба відкрити їх без авторизації перенаправляє на сторінку входу. Сторінку входу наведено на рис. 3.2.

Форма входу містить поля для адреси електронної пошти та пароля, а форма реєстрації – ті самі два поля без додаткового підтвердження пароля. Поле пошти браузер перевіряє на відповідність формату, а пароль під час реєстрації має містити щонайменше шість символів. Обидві форми також дозволяють увійти або зареєструватися через обліковий запис Google за протоколом OAuth 2.0 [29]. Після успішного входу JWT-токен зберігається в локальному сховищі браузера (localStorage) і додається до заголовка Authorization наступних запитів [22]. Термін дії токена становить 15 хвилин; після його завершення сесія

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

вважається завершеною, і застосунок повертається в неавторизований (гостьовий) стан.

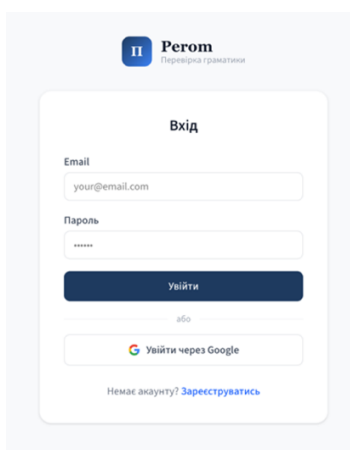


Рис. 3.2. Сторінка входу та реєстрації у системі

Головна сторінка є текстовим редактором із панеллю вибору режиму перевірки. Центральну частину займає область введення тексту. У нижній частині редактора відображається рядок стану з лічильником символів і слів, а також кількістю знайдених помилок у розбивці за типами. Кнопка «Перевірити» залишається недоступною, поки поле введення порожнє. Зовнішній вигляд редактора наведено на рис. 3.3.

Режим перевірки обирається повзунком із трьох позицій. Режим «Швидка» виконує перевірку лише через LanguageTool. Режим «Середня» поєднує нейронну модель Pravorusnyk із LanguageTool. Режим «Точна» активує повний гібридний ланцюжок із трьома джерелами корекції і рекомендується для текстів, де критично важлива максимальна точність.

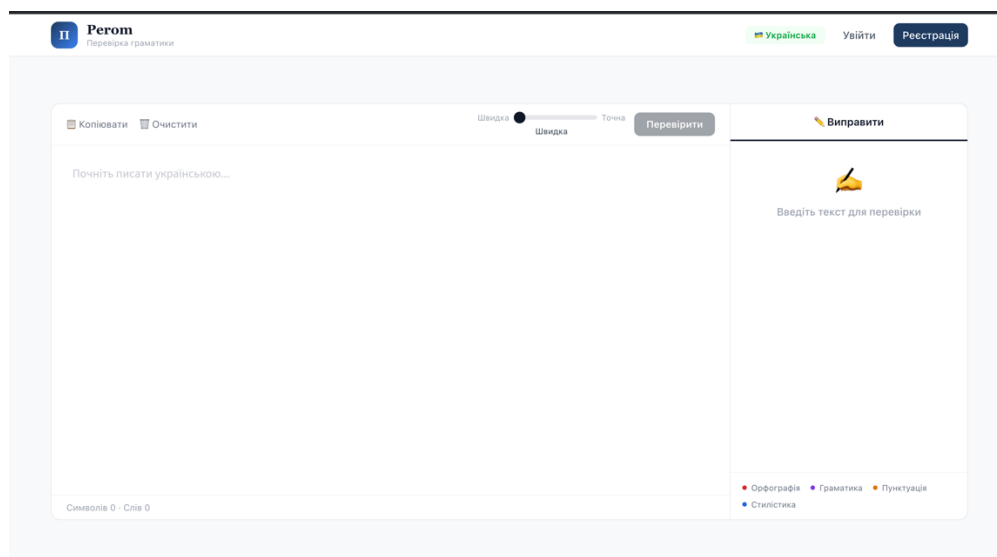


Рис. 3.3. Головна сторінка редактора тексту

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

Результати перевірки з'являються на тій самій сторінці одразу після завершення обробки, без перезавантаження. Виправлений текст відображається з підсвіченими помилками, кожен фрагмент отримує пунктирне підкреслення та легку заливку фоном, колір яких визначається типом помилки – червоний для орфографічних, фіолетовий для граматичних, помаранчевий для пунктуаційних, синій для стилістичних. Натискання на підсвічений фрагмент відкриває спливаючий блок із типом помилки, оригінальним фрагментом, запропонованим виправленням і кнопками «Виправити» та «Ігнорувати». Праворуч відображається панель із переліком усіх знайдених помилок. Приклад результату перевірки наведено на рис. 3.4.

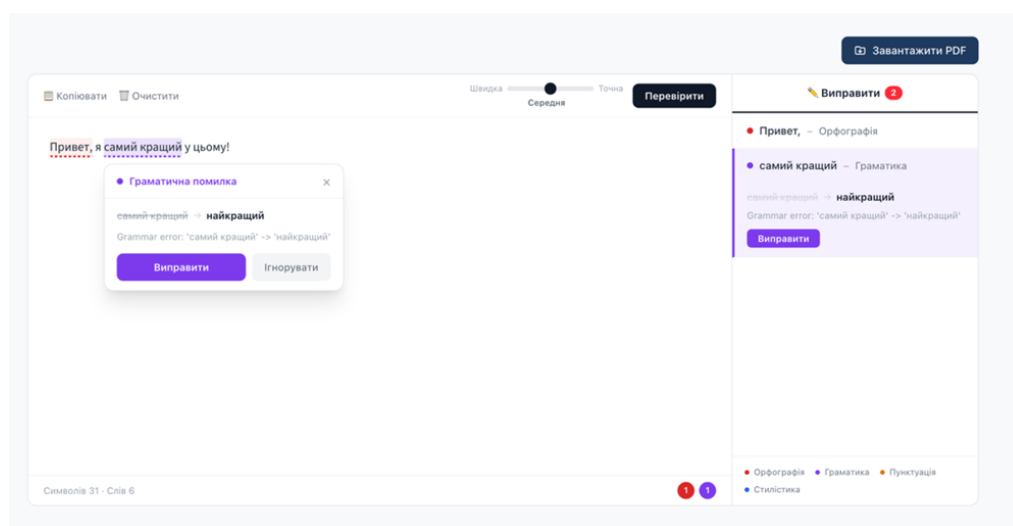


Рис. 3.4. Результат перевірки тексту з підсвіченими виправленнями та переліком помилок

Для збереження результату у форматі PDF авторизований користувач натискає кнопку «Завантажити PDF», яка з'являється після перевірки. Система формує звіт через AWS Lambda і після готовності ініціює завантаження файлу в браузері. Звіт містить вихідний і виправлений текст та таблицю помилок із класифікацією (тип, оригінальний фрагмент і запропоноване виправлення).

Сторінка профілю містить три вкладки статистика, історія перевірок та налаштування. Вкладка статистики відображає загальну кількість виконаних перевірок і розподіл виявлених помилок за типами у вигляді прогрес-барів із відсотками. Вкладка історії відображає пагінований список минулих запитів; для кожного запису показано початок тексту, дату, тип клієнта (вебінтерфейс або

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савицький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

API) та кількість помилок. При натисканні на запис відкривається детальний перегляд із оригінальним і виправленим текстом та переліком помилок. Вкладка налаштувань дозволяє змінити пароль і адресу електронної пошти облікового запису, а також вийти із системи. Історію перевірок наведено на рис. 3.5.

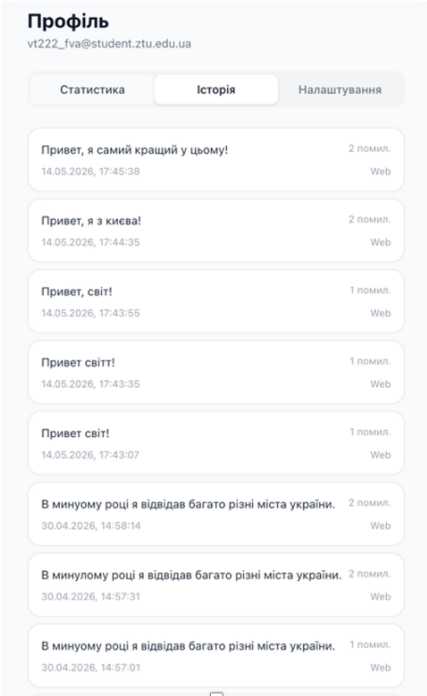


Рис. 3.5. Журнал перевірок на сторінці профілю користувача

Сторінка керування API-ключами відображає список наявних ключів із полями: префікс (перші символи ключа у замаскованому вигляді), назва, дата створення та дата останнього використання. Для створення нового ключа користувач за бажанням вводить назву і підтверджує. Система генерує ключ і одноразово показує його повне значення у виділеній панелі з кнопкою копіювання та попередженням, що повторно ключ показано не буде. Для відкриття ключа достатньо натиснути відповідну кнопку в його рядку. Вигляд сторінки керування ключами наведено на рис. 3.6.

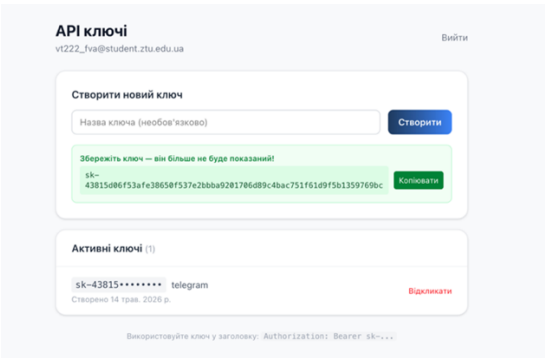


Рис. 3.6. Сторінка управління API-ключами

Адміністративна панель доступна виключно для облікових записів із роллю адміністратора. Спроба переходу до адміністративного розділу зі звичайного облікового запису призводить до автоматичного перенаправлення на головну сторінку.

Панель є єдиною сторінкою, у верхній частині якої наведено зведену статистику у вигляді карток: загальна кількість запитів, кількість запитів з вебінтерфейсу та через API, кількість авторизованих і анонімних запитів, кількість зареєстрованих користувачів, середня довжина запиту та середній час обробки. Нижче розміщено стовпчасту діаграму запитів за останні 30 днів, кругову діаграму співвідношення запитів з вебінтерфейсу та через API, а також розподіл помилок за типами у вигляді прогрес-барів. Верхню частину адміністративної панелі зі статистикою наведено на рис. 3.7.

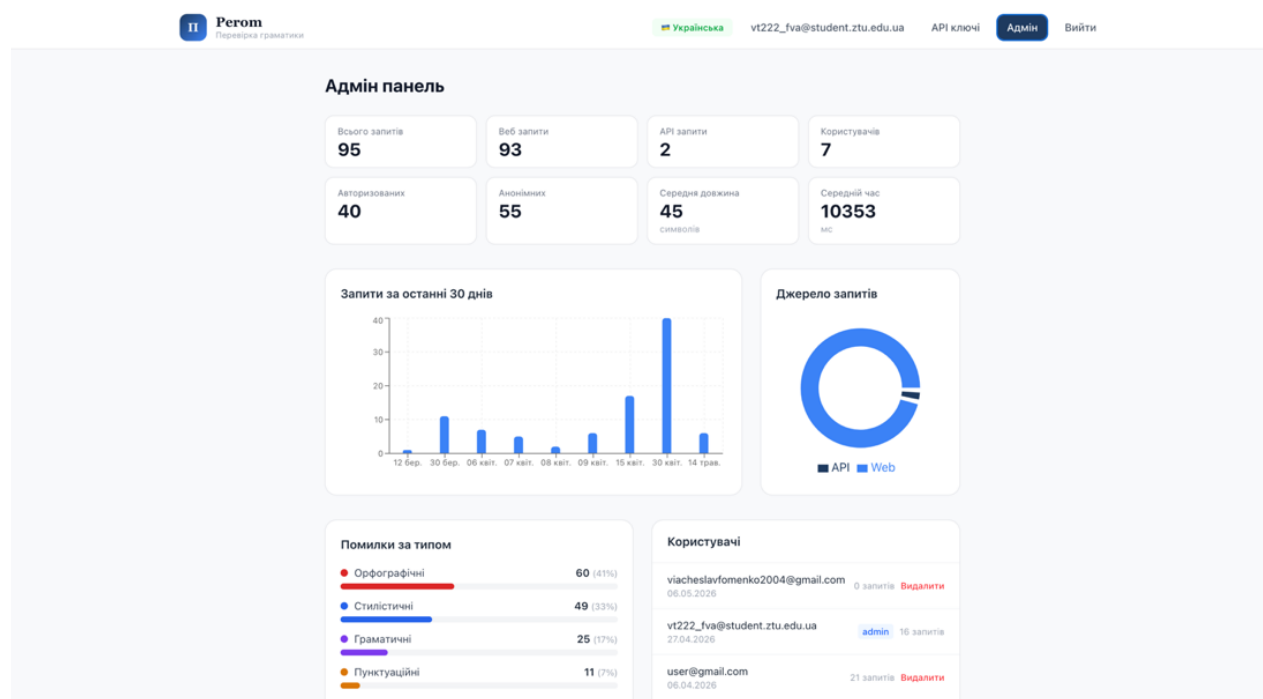


Рис. 3.7. Головна сторінка адміністративної панелі

У нижній частині панелі розміщено список зареєстрованих користувачів із полями: адреса пошти, дата реєстрації, роль та кількість виконаних запитів. Для звичайних користувачів доступна операція видалення після підтвердження у діалоговому вікні; для облікових записів із роллю адміністратора кнопку видалення приховано, що унеможлиблює видалення адміністративних акаунтів через панель. Список користувачів наведено на рис. 3.8.

Користувачі		
viacheslavfomenko2004@gmail.com	0 запитів	Видалити
06.05.2026		
vt222_fva@student.ztu.edu.ua	admin	16 запитів
27.04.2026		
user@gmail.com	21 запитів	Видалити
06.04.2026		
test123@gmail.com	admin	1 запитів
30.03.2026		
admin@gmail.com	admin	2 запитів
12.03.2026		

Рис. 3.8. Розділ управління користувачами в адміністративній панелі

Сервіс підтримує програмний доступ через REST API з використанням API-ключа. Зовнішній розробник реєструється в системі, генерує API-ключ та передає його в кожному запиті у заголовок Authorization у форматі Bearer.

Запит на перевірку надсилається методом POST на маршрут /api/check. Тіло запиту у форматі JSON містить поле text із рядком тексту (до 5000 символів) та поле mode з обраним режимом. Відповідь містить виправлений текст, масив знайдених помилок із класифікацією та час обробки в мілісекундах. Кожен об'єкт помилки містить поля error_type, position, length, original, suggestion та message. Приклад запиту через Postman наведено в (додатку А, рис. 5).

Гостьовий доступ до функції перевірки тексту можливий без реєстрації, заголовок Authorization можна не передавати, і результат перевірки повертається повністю. Такий запит все одно фіксується в журналі як анонімний (без прив'язки до облікового запису), тому він не з'являється в особистій історії користувача. Генерація PDF-звіту та перегляд особистої історії й статистики потребують авторизованого облікового запису.

3.3 Тестування системи

Тестування системи проводилося вручну методом функціонального тестування, коли перевіряється поведінка системи з боку користувача за заданими сценаріями без аналізу внутрішнього коду. Метою тестування було підтвердити, що сервіс виконує свою основну задачу – перевірку та корекцію тексту, – а також правильно реагує на некоректні дії та хибні вхідні дані.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Вебінтерфейс перевірявся у браузері, а програмний доступ через REST API – за допомогою застосунку Postman [38].

Перевірку побудовано на наборі тест-сценаріїв, що охоплюють як штатні (позитивні) випадки, коли користувач діє правильно, так і негативні випадки, коли на вхід подаються помилкові дані або виконуються дії без потрібних прав. Для кожного сценарію визначено вхідні дані або послідовність дій, очікуваний результат та фактичний результат перевірки. Перелік основних тест-сценаріїв наведено в таблиці 3.2.

Таблиця 3.2

Тест сценарії

№	Тест-сценарій	Вхідні дані / дії	Очікуваний результат	Статус
1	Реєстрація користувача	Коректні email і пароль (≥6 символів)	Обліковий запис створено, видано токен, перехід до інтерфейсу	Пройдено
2	Вхід у систему	Коректні email і пароль	Успішний вхід, JWT збережено	Пройдено
3	Вхід через Google	Обліковий запис Google (OAuth 2.0)	Успішний вхід, видача JWT	Пройдено
4	Перевірка тексту, режим «Швидка»	Текст із навмисними помилками	Помилки знайдено й підсвічено, повернуто виправлений текст	Пройдено
5	Перевірка тексту, режим «Середня»	Текст із навмисними помилками	Поглиблена корекція, виправлений текст і перелік помилок	Пройдено
6	Взаємодія з помилкою	Натискання на фрагмент, «Виправити» / «Ігнорувати»	Виправлення застосовується або ігнорується	Пройдено
7	Експорт у PDF	Натискання «Завантажити PDF»	Сформовано звіт з оригіналом, виправленим текстом і таблицею помилок	Пройдено
8	Журнал перевірок	Відкриття сторінки профілю	Виконана перевірка відображається в історії	Пройдено

9	Перевірка через REST API	POST /api/check із дійсним API-ключем	Код 200, JSON із виправленим текстом, помилками й часом обробки	Пройдено
10	Гостьова перевірка	POST /api/check без токена	Код 200, результат повертається, запит фіксується як анонімний	Пройдено
11	Вхід із невірними даними	Неправильний пароль	Код 401, повідомлення «Невірний email або пароль»	Пройдено
12	Реєстрація на зайнятий email	Наявна адреса	Код 409, «Користувач з таким email вже існує»	Пройдено
13	Зміна пароля з помилками	Неспівпадіння / короткий / невірний поточний	Відповідні повідомлення про помилку	Пройдено
14	Некоректний запит до API	Порожній текст або невірний режим	Код 400 з описом помилки валідації	Пройдено
15	Перевищення розміру запиту	Текст понад допустимий обсяг	Код 413	Пройдено
16	Доступ без прав	Маршрут без токена / admin-маршрут з роллю user	Код 401 / код 403 «Доступ заборонено»	Пройдено

Спочатку перевірено основний сценарій роботи сервісу. У редактор уведено речення з навмисними орфографічними, граматичними та пунктуаційними помилками. Перевірку запущено в режимах «Швидка» та «Середня». Система знайшла внесені помилки та підсвітила їх відповідним кольором за типом. Виправлений текст повернувся коректно. Натискання на підсвічений фрагмент відкривало блок із типом помилки та запропонованим виправленням. Кнопки «Виправити» та «Ігнорувати» працювали правильно. Результат успішної перевірки наведено на рис. 3.9.

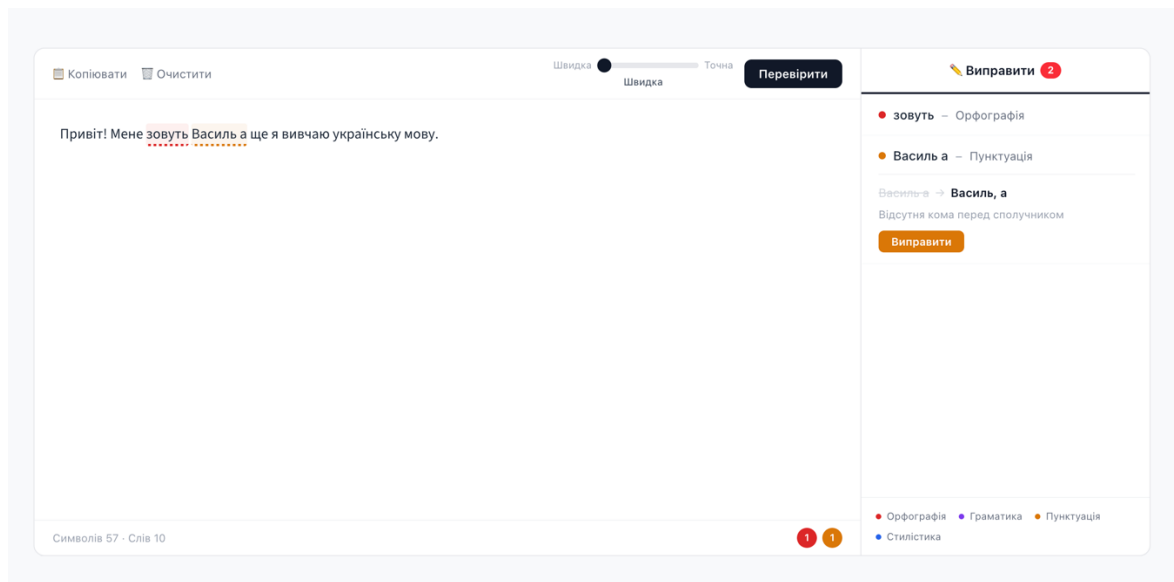


Рис. 3.9. Результат успішної перевірки тексту з підсвіченими виправленнями та переліком помилок

Далі перевірено супутні функції авторизованого користувача. Реєстрація нового облікового запису, вхід за електронною поштою та вхід через обліковий запис Google завершувалися успішно з видачею токена й переходом до робочого інтерфейсу. Збереження результату у форматі PDF формувало звіт із оригінальним і виправленим текстом та таблицею помилок, причому кирилиця відображалася без спотворень. Виконані перевірки одразу з'являлися в журналі на сторінці профілю (рис. 3.5), а зведені показники оновлювалися у вкладці статистики.

Перевірено програмний доступ через REST API. Запит на перевірку тексту надсилався методом POST на маршрут `/api/check`. У заголовку передавався дійсний API-ключ. Сервер повертав код 200 і коректний JSON із виправленим текстом, переліком помилок та часом обробки. Також перевірено гостьовий доступ. Без заголовка авторизації запит теж повертав результат із кодом 200 і фіксувався в журналі як анонімний. Успішну відповідь API наведено в додатку А, рис. 5.

Після перевірки штатних сценаріїв окрему увагу приділено реакції системи на некоректні дії. Першим перевірявся вхід із невірними обліковими даними. При введенні неіснуючої адреси або неправильного пароля сервер відхиляв запит, а застосунок виводив повідомлення «Невірний email або пароль» без

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

розкриття того, що саме було введено неправильно. Повідомлення про помилку входу наведено на рис. 3.10.

Рис. 3.10. Повідомлення про помилку при спробі входу з невірними даними

Далі перевірялася валідація під час реєстрації. Поле електронної пошти браузер перевіряв на відповідність формату, а пароль мав містити щонайменше шість символів – за недотримання цих умов форма не відправлялася, а проблемне поле підсвічувалося. Якщо адреса вже зайнята, сервер повертав помилку, і застосунок показував повідомлення «Користувач з таким email вже існує». Приклад помилок реєстрації наведено на рис. 3.11.

Рис. 3.11. Помилки валідації та повідомлення про зайняту електронну пошту при реєстрації

Окремо перевірено обробку помилок під час зміни пароля у профілі. Якщо новий пароль і його підтвердження не збігалися, виводилося повідомлення «Паролі не співпадають», за надто короткого пароля – «Мінімум 6 символів», а при невірному поточному паролі сервер відхиляв зміну з повідомленням «Невірний поточний пароль». Приклади цих повідомлень наведено на рис. 3.12.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

Рис. 3.12. Повідомлення про помилки під час зміни пароля

На рівні REST API перевірялася реакція на некоректні запити у Postman. Запит з порожнім полем тексту або з неприпустимим значенням режиму відхилявся з кодом 400. У відповіді наводився опис помилки валідації. Запит із надмірним обсягом даних відхилявся з кодом 413. Звернення до захищеного маршруту без токена поверталось з кодом 401. Звернення з обліковим записом без потрібних прав поверталось з кодом 403 і повідомленням «Доступ заборонено». Приклад помилкової відповіді API наведено на рис. 3.13.

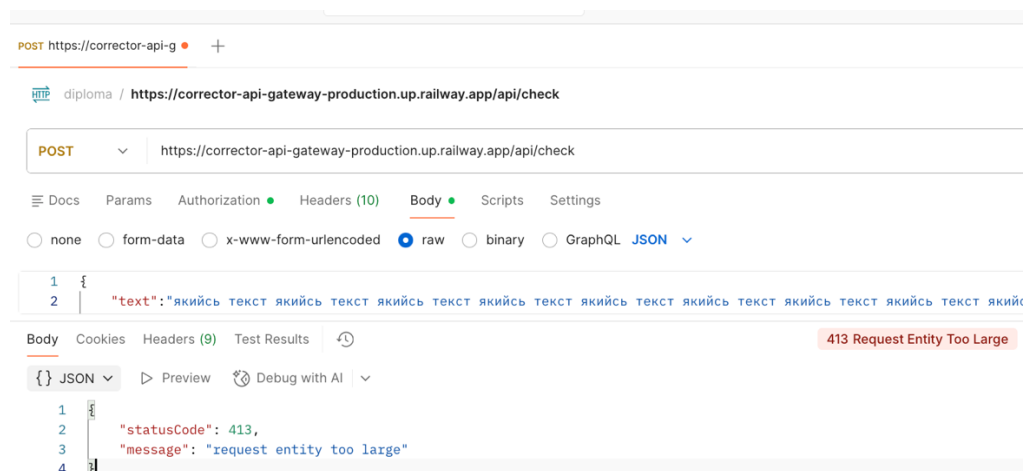


Рис. 3.13. Відповідь REST API при перевищенні розміру запиту (код 413) у Postman

За результатами ручного тестування підтверджено, що система коректно опрацьовує як штатні сценарії, так і помилкові ситуації. Основний функціонал перевірки та корекції тексту працює відповідно до вимог, некоректні дані відхиляються, користувач отримує зрозумілі повідомлення, а API повертає відповідні коди стану. Виявлені під час перевірки недоліки було усунено.

Висновки до третього розділу

У третьому розділі було показано, як система розгортається й працює з боку користувача, а також перевірено її роботу. Усі компоненти розгорнуто незалежно, ML-сервіс на Hugging Face Spaces, шлюз API з базою даних на Railway, вебклієнт на Vercel, що дозволяє оновлювати окремі частини без зупинки решти. Описано й порядок розгортання та змінні середовища, потрібні для запуску. Компоненти піднімаються в такій послідовності, щоб усі залежності були готові до моменту першого запиту.

Описано інтерфейс сервісу, редактор тексту з вибором режиму перевірки й підсвічуванням помилок різними кольорами залежно від типу, сторінки профілю та керування API-ключами, а також адміністративну панель зі статистикою й керуванням користувачами. Показано також збереження результату у форматі PDF та програмний доступ через REST API, зокрема в гостьовому режимі без реєстрації.

Роботу системи перевірено вручну за методом мануального тестування. Основну увагу приділено тому, як сервіс реагує на помилкові дії, а саме невдалий вхід, помилки під час реєстрації, спроби доступу без прав і некоректні запити до API. У кожному випадку перевірялося, чи показує застосунок зрозуміле повідомлення та чи повертає правильний код стану. Перевірка підтвердила, що система правильно опрацьовує і звичайні, і помилкові ситуації, а виявлені під час тестування недоліки було усунено.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно розроблено та реалізовано вебсервіс автоматизованої перевірки й корекції граматики української мови, що інтегрує нейромережеві моделі, аналіз на основі правил і сучасні вебтехнології задля підвищення якості україномовних текстів. Проведений аналіз існуючих рішень виявив суттєві обмеження наявних аналогів. Grammarly підтримує українську мову лише в бета-режимі й не надає публічного API, LanguageTool обмежує безкоштовний доступ та не має нейронної корекції, а UkrainianCorrection.com позбавлений і програмного інтерфейсу, і нейромоделей. Розроблена система усуває ці недоліки, поєднуючи гібридну нейронну корекцію з трьома режимами перевірки, веденням персонального журналу запитів, експортом результатів у PDF, керуванням API-ключами та повноцінним REST API для інтеграції зі сторонніми застосунками.

Обрана клієнт-серверна архітектура складається з трьох незалежних компонентів – сервісу машинного навчання, шлюзу API та вебклієнта, що взаємодіють виключно через HTTP. Через несумісність технологічних стеків обчислювально навантажений ML-сервіс на Python виокремлено в окремий процес із можливістю незалежного масштабування, тоді як шлюз реалізовано на NestJS як модульний моноліт і єдину точку входу, що приховує ML-сервіс від зовнішньої мережі та зосереджує автентифікацію, авторизацію й журналювання. Такий поділ забезпечив технологічну свободу при виборі стеку для кожного рівня та багаторівневий захист системи – JWT і API-ключі, рольову модель доступу, хешування паролів bcrypt і ключів SHA-256.

Центральним технічним аспектом роботи є алгоритм гібридної корекції, що поєднує дві спеціалізовані нейронні моделі – Pravopysnyk/best-unlp і grammarly/spivavtor-large – із аналізатором LanguageTool. Узгодження їхніх результатів виконує алгоритм MergeStrategy, який ухвалює пословне рішення за правилом більшості, приймаючи лише ті виправлення, що їх незалежно підтверджують щонайменше два джерела, чим суттєво зменшується кількість хибних змін, властивих окремо взятій нейромоделі. Додатково реалізовано шлюз

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

API з дворівневою автентифікацією та входом через Google OAuth 2.0, базу даних PostgreSQL, вебклієнт на React і TypeScript із підсвічуванням помилок та підсистему генерації PDF-звітів на AWS Lambda зі зберіганням у S3, а розгортання компонентів розподілено між Hugging Face Spaces, Railway і Vercel. Проведене ручне функціональне тестування підтвердило коректну роботу як штатних сценаріїв, так і обробку помилкових ситуацій із поверненням зрозумілих повідомлень та відповідних кодів стану.

Практична значущість роботи полягає у створенні доступного інструменту для перевірки та корекції україномовних текстів, придатного для використання як кінцевими користувачами через вебінтерфейс, так і програмними системами через REST API. Перспективи подальшого розвитку охоплюють розширення набору нейронних моделей і правил для підвищення якості корекції, оптимізацію часу обробки в режимі максимальної точності, розвиток адміністративної аналітики та розширення інтеграційних можливостей сервісу.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Центр Разумкова. Ідентичність громадян України: тенденції змін. 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://razumkov.org.ua/napriamky/sotsiologichni-doslidzhennia/identychnist-gromadian-ukrainy-tendentsii-zmin-cherven-2024r>.
2. Syvokon O. et al. A Low-Resource Approach to the Grammatical Error Correction of Ukrainian. Proceedings of the Second Ukrainian NLP Workshop (UNLP 2023) [Електронний ресурс] – Режим доступу до ресурсу: <https://aclanthology.org/2023.unlp-1.14/>.
3. Fomenko V. A. ВИКОРИСТАННЯ HUGGING FACE HUB ЯК СЕРВІСУ БЕКЕНДУ ДЛЯ РОЗГОРТАННЯ МОВНИХ МОДЕЛЕЙ : матеріали International Scientific and Practical Internet Conference «Development of Education, Science and Business: Results 2025», 18–19 грудня 2025 р. / V. A. Fomenko. – Дніпро : FOP Marenichenko V. V., 2025. – 622 с. [Електронний ресурс] – Режим доступу до ресурсу: <http://www.wayscience.com/wp-content/uploads/2025/12/Conference-Proceedings-December-18-19-2025-1.pdf>.
4. Автоматизований пошук помилок в українському тексті з використанням моделей штучного інтелекту : матеріали VIII Всеукраїнської наук.-техн. конф., м. Житомир, 02–03 грудня 2025 р. – Житомир : ЖТУ, 2025. [Електронний ресурс] – Режим доступу до ресурсу: <https://conf.ztu.edu.ua/wp-content/uploads/2026/01/197.pdf>.
5. Grammarly [Електронний ресурс] – Режим доступу до ресурсу: <https://www.grammarly.com/>.
6. LanguageTool [Електронний ресурс] – Режим доступу до ресурсу: <https://languagetool.org/uk>.
7. UkrainianCorrection.com [Електронний ресурс] – Режим доступу до ресурсу: <https://www.ukrainiancorrection.com/>.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

8. Microsoft Word: [Електронний ресурс] – Режим доступу до ресурсу: <https://word.cloud.microsoft/uk-ua/>.
9. Google docs: [Електронний ресурс] – Режим доступу до ресурсу: <https://www.googleapps.com/docs/about/?hl=en>.
10. Hugging Face Spaces. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://huggingface.co/docs/hub/spaces>.
11. Railway. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://railway.com/>.
12. Hugging Face Transformers. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://huggingface.co/docs/transformers>.
13. Pravopysnyk/best-unlp: Мовна модель [Електронний ресурс] – Режим доступу до ресурсу: <https://huggingface.co/pravopysnyk/best-unlp>.
14. Grammarly/spivavtor-large: Мовна модель [Електронний ресурс] – Режим доступу до ресурсу: <https://huggingface.co/grammarly/spivavtor-large>.
15. FastAPI. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>.
16. Uvicorn. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.uvicorn.org/>.
17. language-tool-python. Бібліотека [Електронний ресурс] – Режим доступу до ресурсу: <https://pypi.org/project/language-tool-python/>.
18. NestJS. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/>.
19. Passport.js. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.passportjs.org/docs/>.
20. TypeORM. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://typeorm.io/>.
21. PostgreSQL. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>.

- 22.JWT.io. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>.
- 23.React. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/>.
- 24.TypeScript. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>.
- 25.Vite. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://vitejs.dev/guide/>.
- 26.Refactoring Guru. Патерни проєктування [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/design-patterns>.
- 27.Python difflib. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python.org/3/library/difflib.html>.
- 28.REST API Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>.
- 29.Google Identity. OAuth 2.0 [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/identity/protocols/oauth2>.
- 30.Bcryptjs. Бібліотека [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/bcryptjs>.
- 31.React Router. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/>.
- 32.AWS SDK for JavaScript. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/sdk-for-javascript/>.
- 33.AWS Lambda. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/lambda/>.
- 34.Pdf-lib. Бібліотека [Електронний ресурс] – Режим доступу до ресурсу: <https://www.npmjs.com/package/pdf-lib>.
- 35.AWS S3. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.aws.amazon.com/s3/>.
- 36.Docker. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/>.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

37.Vercel. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://vercel.com/docs>.

38.Postman. Документація [Електронний ресурс] – Режим доступу до ресурсу: <https://learning.2man.com/docs/>.

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіцький Р.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Савіуький Р.С.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

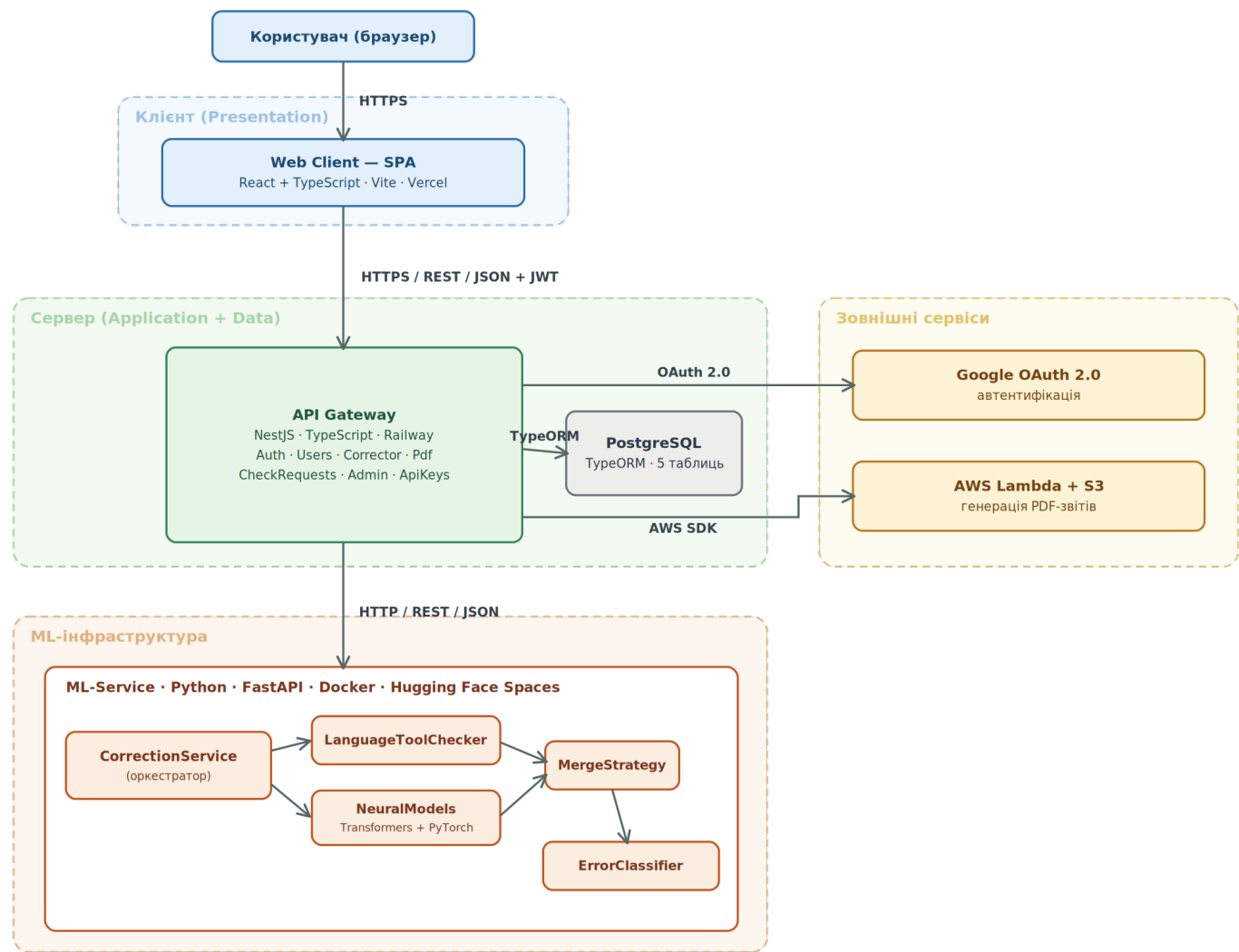


Рис. 1. Схема архітектури

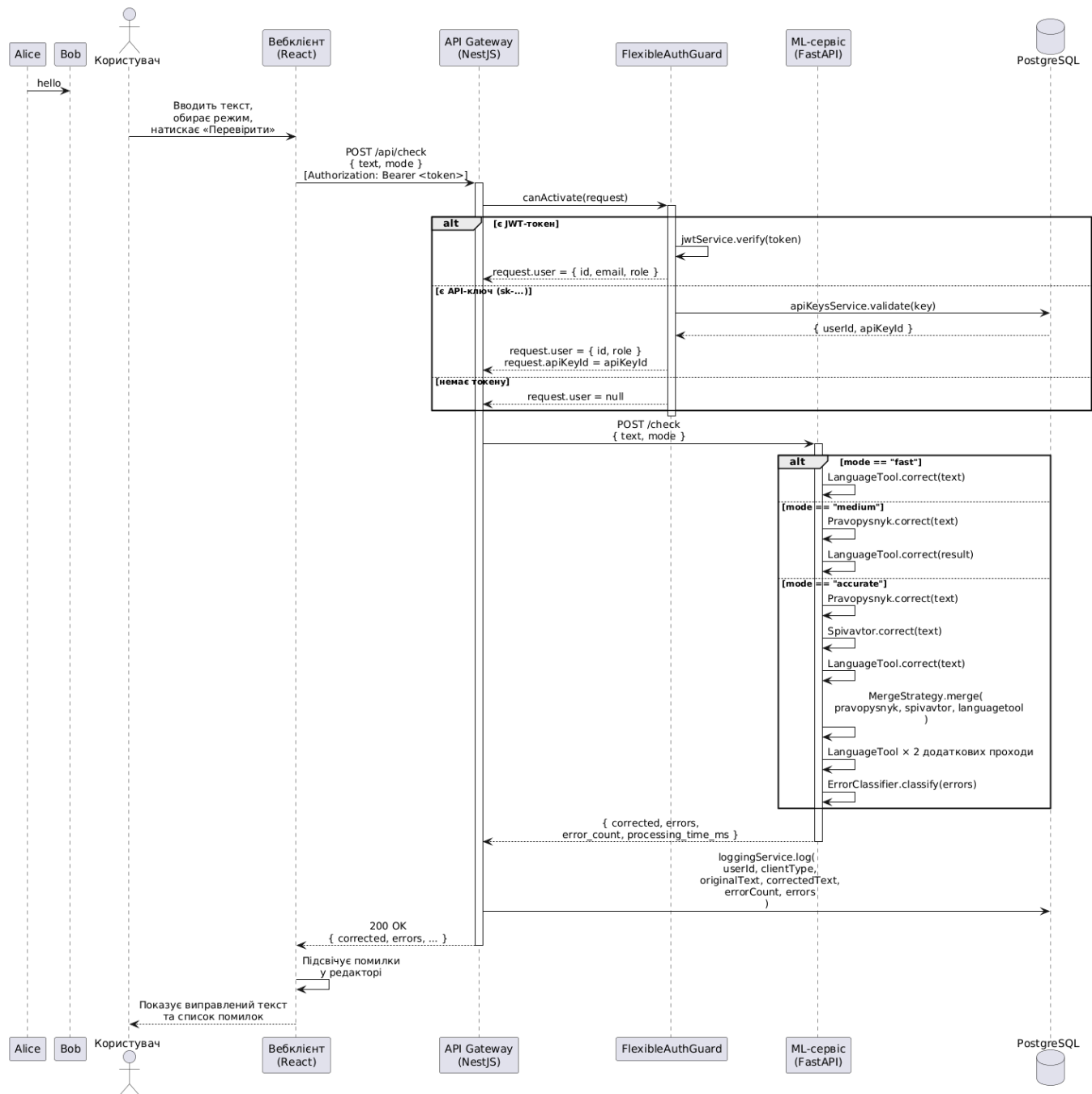


Рис. 2. Діаграма послідовності виконання перевірки тексту

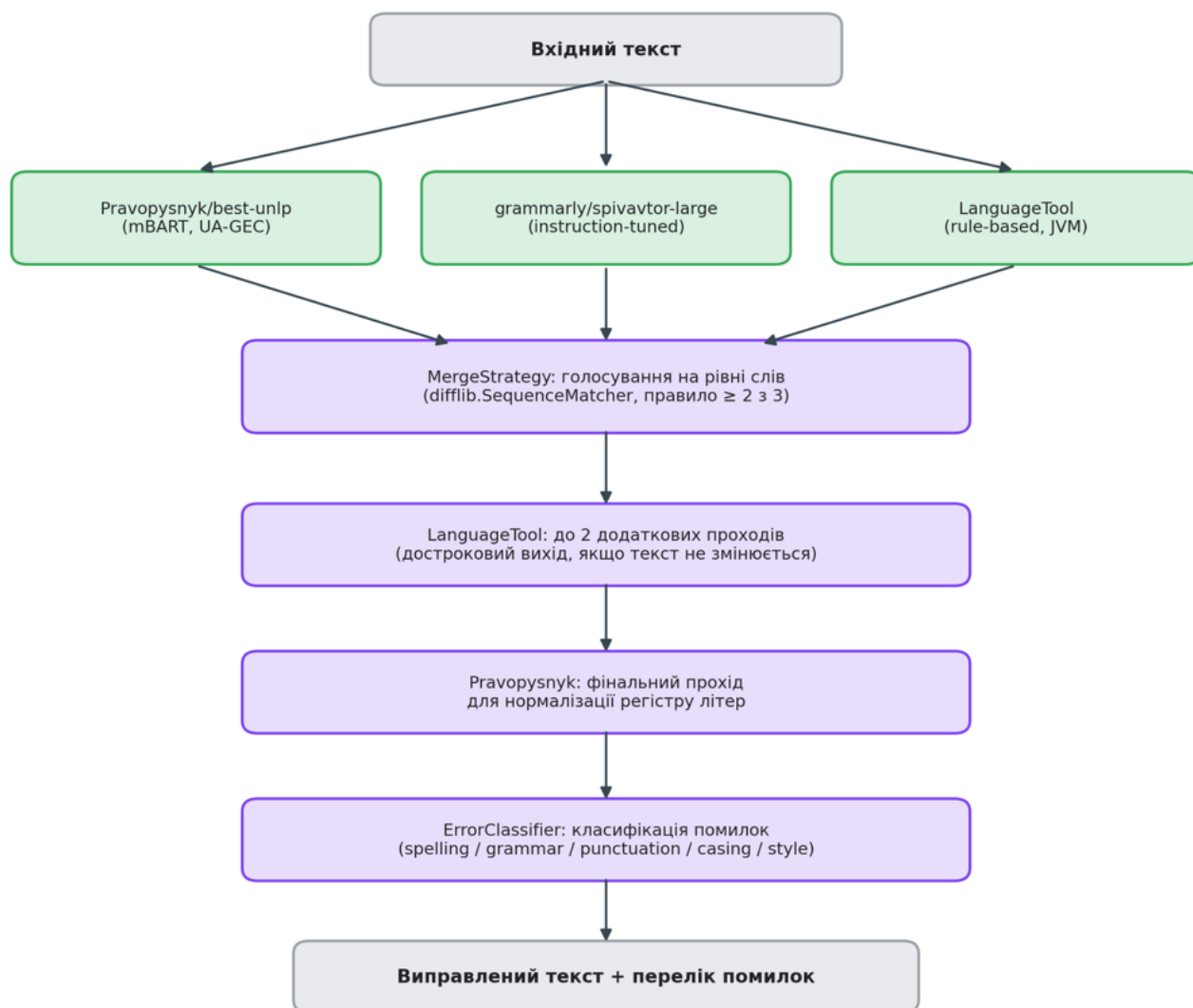


Рис. 3. Схема гібридної корекції у точному режимі

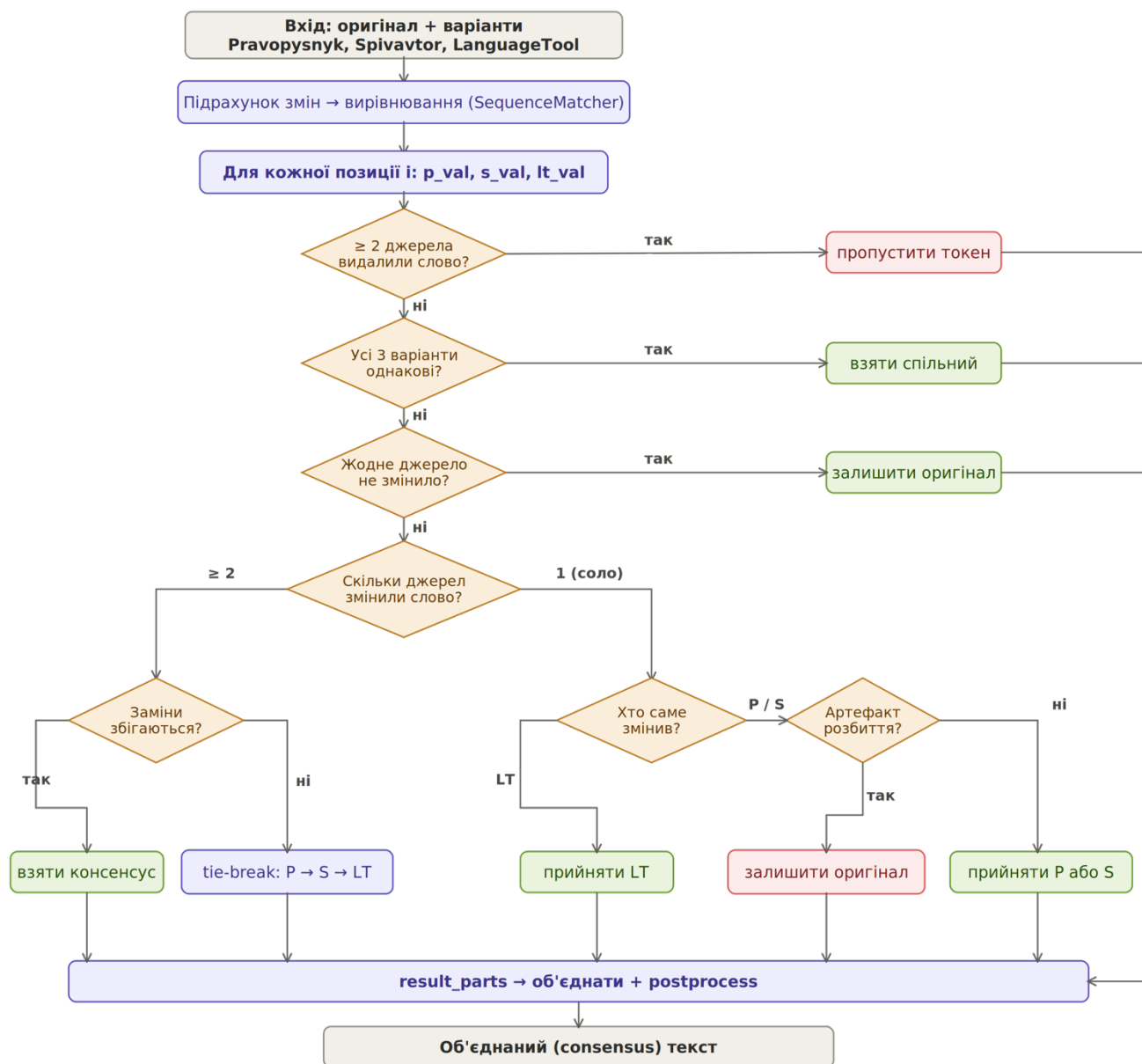


Рис. 4. Схема алгоритму MergeStrategy

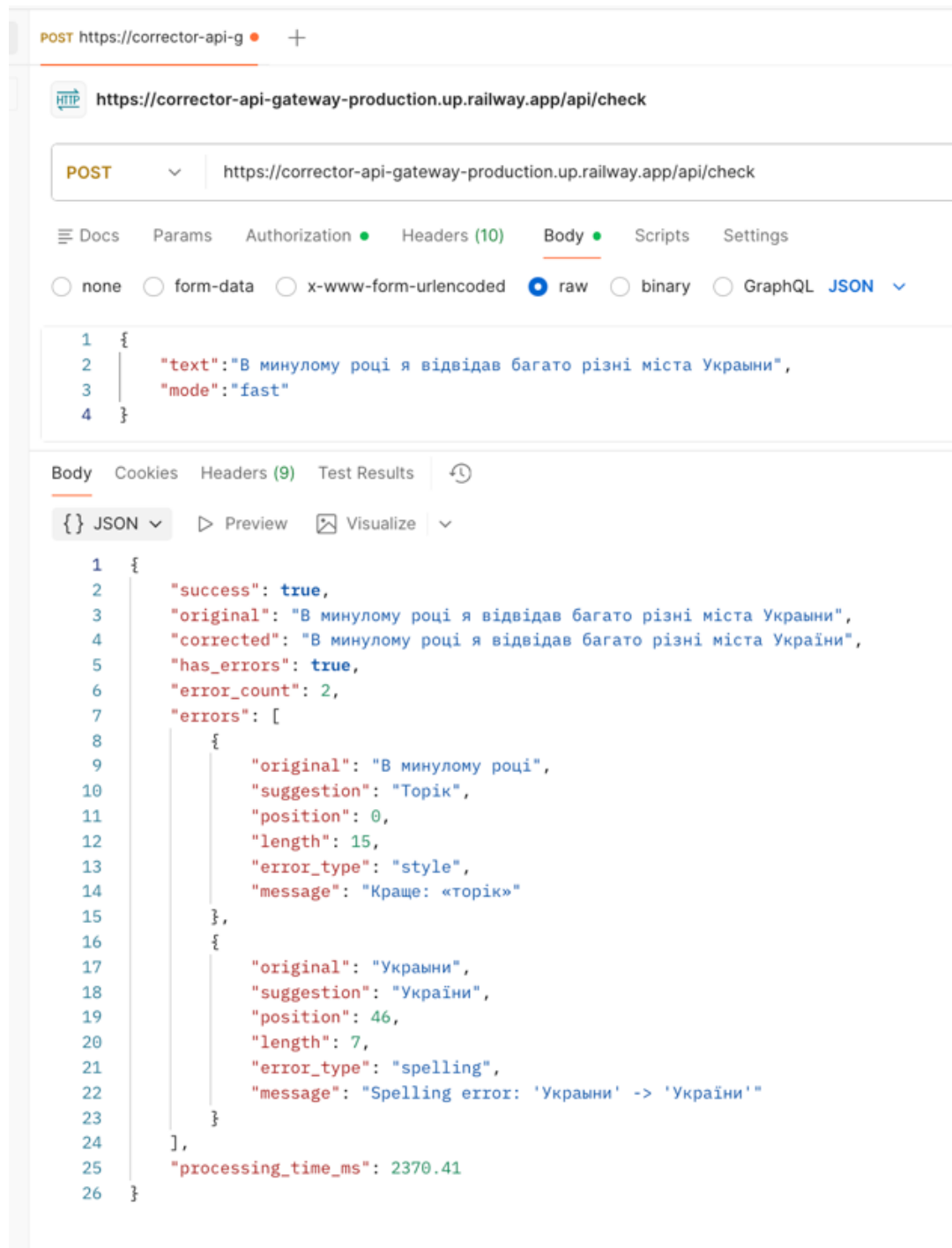


Рис. 5. Запит з використанням API токєну (Postman)

Лістинг коду файлу corrector.py

```

import logging
import time
from typing import Dict

from .neural_models import NeuralCorrector
from .languagetool_checker import LanguageToolChecker
from .merge_strategy import MergeStrategy
from .error_classifier import ErrorClassifier
from .languagetool_setup import LanguageToolSetup
from schemas.request_response import ErrorDetail, ErrorType

logger = logging.getLogger(__name__)

class TripleHybridCorrector:

    def __init__(
        self,
        model_name_1: str = "Pravopysnyk/best-unlp",
        model_name_2: str = "grammarly/spivavtor-large",
    ):
        logger.info("Initializing...")

        self.neural = NeuralCorrector(model_name_1, model_name_2)

        grammar_tool = LanguageToolSetup.setup()
        self.lt_checker = LanguageToolChecker(grammar_tool)

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савіцький Р.С.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

self.merger = MergeStrategy()
self.classifier = ErrorClassifier(grammar_tool)

```

```

logger.info("Ready")

```

```

def check_text(self, text: str, return_details: bool = True, mode: str = "accurate") ->
Dict:

```

```

    start_time = time.time()

```

```

    try:

```

```

        logger.info(f'Checking [{mode}]: '{text[:50]}...')

```

```

        if mode == "fast":

```

```

            return self._check_fast(text, return_details, start_time)

```

```

        elif mode == "medium":

```

```

            return self._check_medium(text, return_details, start_time)

```

```

        else:

```

```

            return self._check_accurate(text, return_details, start_time)

```

```

    except Exception as e:

```

```

        logger.error(f'Error: {e}', exc_info=True)

```

```

        raise

```

```

def _merge_style_errors(self, all_errors: list, lt_errors: list) -> list:

```

```

    lt_style_errors = [e for e in lt_errors if e.error_type == ErrorType.STYLE]

```

```

    for style_err in lt_style_errors:

```

```

        overlaps = any(

```

```

            e.position < style_err.position + style_err.length

```

```

            and style_err.position < e.position + e.length

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуцький Р.С.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        for e in all_errors
    )
    if not overlaps:
        all_errors.append(style_err)
all_errors.sort(key=lambda e: e.position)
return all_errors

```

```

def _check_fast(self, text: str, return_details: bool, start_time: float) -> Dict:

```

```

    lt_corrected, lt_errors = self.lt_checker.get_corrections(text)

```

```

    processing_time = (time.time() - start_time) * 1000

```

```

    return {

```

```

        "success": True,

```

```

        "original": text,

```

```

        "corrected": lt_corrected,

```

```

        "has_errors": text != lt_corrected,

```

```

        "error_count": len(lt_errors),

```

```

        "errors": lt_errors if return_details else None,

```

```

        "processing_time_ms": round(processing_time, 2),

```

```

    }

```

```

def _check_medium(self, text: str, return_details: bool, start_time: float) -> Dict:

```

```

    pravopysnyk_corrected = self.neural.get_pravopysnyk_correction(text)

```

```

    lt_corrected, lt_errors = self.lt_checker.get_corrections(pravopysnyk_corrected)

```

```

    current = lt_corrected

```

```

    all_errors = []

```

```

    if text != current and return_details:

```

```

        all_errors = self.classifier.classify_errors(text, current)

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

```

if return_details:
    all_errors = self._merge_style_errors(all_errors, lt_errors)

processing_time = (time.time() - start_time) * 1000
return {
    "success": True,
    "original": text,
    "corrected": current,
    "has_errors": text != current,
    "error_count": len(all_errors),
    "errors": all_errors if return_details else None,
    "processing_time_ms": round(processing_time, 2),
}

```

```

def _check_accurate(self, text: str, return_details: bool, start_time: float) -> Dict:

```

```

    LT_EXTRA_PASSES = 2

```

```

    pravopysnyk_corrected = self.neural.get_pravopysnyk_correction(text)

```

```

    spivavtor_corrected = self.neural.get_spivavtor_correction(text)

```

```

    lt_corrected, lt_errors = self.lt_checker.get_corrections(text)

```

```

    current = self.merger.merge_corrections(

```

```

        original=text,

```

```

        pravopysnyk=pravopysnyk_corrected,

```

```

        spivavtor=spivavtor_corrected,

```

```

        languagetool=lt_corrected,

```

```

    )

```

```

    logger.info("Pass 1 (neural + LT) done")

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

for pass_num in range(LT_EXTRA_PASSES):
    lt_corrected, _ = self.lt_checker.get_corrections(current)
    if lt_corrected == current:
        logger.info(f"LT stable after extra pass {pass_num + 1}")
        break
    logger.info(f"LT extra pass {pass_num + 1}: text changed")
    current = lt_corrected

pravopysnyk_final = self.neural.get_pravopysnyk_correction(current)
if pravopysnyk_final != current:
    current = self._apply_casing_only(current, pravopysnyk_final)

final_corrected = current
all_errors = []

if text != final_corrected and return_details:
    all_errors = self.classifier.classify_errors(text, final_corrected)

if return_details:
    all_errors = self._merge_style_errors(all_errors, lt_errors)
processing_time = (time.time() - start_time) * 1000

return {
    "success": True,
    "original": text,
    "corrected": final_corrected,
    "has_errors": text != final_corrected,
    "error_count": len(all_errors),
    "errors": all_errors if return_details else None,
    "processing_time_ms": round(processing_time, 2),

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

```
}
```

```
@staticmethod
```

```
def _apply_casing_only(original: str, corrected: str) -> str:
```

```
    orig_words = original.split()
```

```
    corr_words = corrected.split()
```

```
    if len(orig_words) != len(corr_words):
```

```
        return original
```

```
    result = []
```

```
    for orig_w, corr_w in zip(orig_words, corr_words):
```

```
        if orig_w.lower() == corr_w.lower() and orig_w != corr_w:
```

```
            logger.info(f'Casing fix: '{orig_w}' -> '{corr_w}')
```

```
            result.append(corr_w)
```

```
        else:
```

```
            result.append(orig_w)
```

```
    return ''.join(result)
```

```
def get_statistics(self) -> Dict:
```

```
    return {
```

```
        "neural_device": self.neural.device,
```

```
        "models_loaded": True,
```

```
        "languagetool_available": self.lt_checker.grammar_tool is not None,
```

```
    }
```

```
def quick_check(self, text: str) -> str:
```

```
    result = self.check_text(text, return_details=False)
```

```
    return result["corrected"]
```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

Лістинг коду файлу merge_strategy.py

```

import re
import logging
from typing import Dict, List
from difflib import SequenceMatcher
from collections import Counter

from .text_utils import count_real_changes, postprocess

logger = logging.getLogger(__name__)

class MergeStrategy:
    @staticmethod
    def merge_corrections(
        original: str,
        pravopysnyk: str,
        spivavtor: str,
        languagetool: str
    ) -> str:
        logger.info("Merging corrections...")

        p_changes = count_real_changes(original, pravopysnyk)
        s_changes = count_real_changes(original, spivavtor)
        lt_changes = count_real_changes(original, languagetool)

        logger.info(f'Changes: P={p_changes}, S={s_changes}, LT={lt_changes}')

        if p_changes == 0 and s_changes == 0 and lt_changes == 0:

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return original

orig_words = original.split()
corrections = {
    'pravopysnyk': pravopysnyk,
    'spivavtor': spivavtor,
    'lt': languagetool,
}

model_aligned = MergeStrategy._align_corrections(orig_words, corrections)
result_parts = MergeStrategy._vote_for_words(orig_words, model_aligned)

final = ''.join(part for part in result_parts if part)
final = postprocess(final)

logger.info(f'Result: '{final}''')
return final

```

```

@staticmethod
def _align_corrections(orig_words: List[str], corrections: Dict[str, str]) -> Dict[str,
Dict[int, str]]:
    model_aligned = {}
    for model_name, corr_text in corrections.items():
        corr_words = corr_text.split()
        matcher = SequenceMatcher(None, orig_words, corr_words)
        word_map = {}

        for tag, i1, i2, j1, j2 in matcher.get_opcodes():
            if tag == 'equal':
                for idx in range(i1, i2):

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

word_map[idx] = orig_words[idx]

elif tag == 'replace':
    orig_chunk = orig_words[i1:i2]
    corr_chunk = corr_words[j1:j2]

    if len(orig_chunk) == len(corr_chunk):
        for k, idx in enumerate(range(i1, i2)):
            word_map[idx] = corr_chunk[k]
    elif model_name in ('pravopysnyk', 'spivavtor') and len(orig_chunk) >
len(corr_chunk):
        logger.info(
            f"[{model_name}] block-collapse rejected: "
            f"{orig_chunk} → {corr_chunk}"
        )
        for idx in range(i1, i2):
            word_map[idx] = orig_words[idx]
    else:
        word_map[i1] = ''.join(corr_chunk)
        for idx in range(i1 + 1, i2):
            word_map[idx] = "

elif tag == 'delete':
    if model_name in ('pravopysnyk', 'spivavtor'):
        logger.info(
            f"[{model_name}] deletion rejected: "
            f"{orig_words[i1:i2]}"
        )
        for idx in range(i1, i2):
            word_map[idx] = orig_words[idx]

```

```

        else:
            for idx in range(i1, i2):
                word_map[idx] = "

elif tag == 'insert':
    insert_text = ''.join(corr_words[j1:j2])
    if i1 > 0 and (i1 - 1) in word_map:
        word_map[i1 - 1] += ' ' + insert_text
    else:
        existing = word_map.get(i1, orig_words[i1] if i1 < len(orig_words)
else ")

        word_map[i1] = insert_text + ' ' + existing

model_aligned[model_name] = word_map

return model_aligned

@staticmethod
def _vote_for_words(orig_words: List[str], model_aligned: Dict[str, Dict[int, str]])
-> List[str]:
    result_parts = []

    for i, orig_word in enumerate(orig_words):
        p_val = model_aligned['pravopysnyk'].get(i, orig_word)
        s_val = model_aligned['spivavtor'].get(i, orig_word)
        lt_val = model_aligned['lt'].get(i, orig_word)

        next_is_empty = (i + 1) < len(orig_words)
        if next_is_empty and p_val != orig_word and
model_aligned['pravopysnyk'].get(i + 1) == ":
```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				86
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        logger.info(f"Word [{i}]: pravopysnyk block-collapse
'{orig_word}'→'{p_val}' neutralized")
        p_val = orig_word
        if next_is_empty and s_val != orig_word and model_aligned['spivavtor'].get(i
+ 1) == "":
            logger.info(f"Word [{i}]: spivavtor block-collapse '{orig_word}'→'{s_val}'
neutralized")
            s_val = orig_word

    variants = {
        'pravopysnyk': p_val,
        'spivavtor': s_val,
        'lt': lt_val,
    }

    # Check deletion
    empty_count = sum(1 for v in variants.values() if v == "")
    if empty_count >= 2:
        continue

    if empty_count == 1:
        result_parts.append(
            MergeStrategy._handle_single_deletion(i, orig_word, variants,
model_aligned, orig_words, result_parts)
        )
        continue

    # всі моделі згодні з оригіналом (нічого не мінялось)
    values = list(variants.values())
    if len(set(values)) == 1:

```

```

result_parts.append(values[0])
continue

# Check who changed
changed = [(name, val) for name, val in variants.items() if val != orig_word]

if len(changed) == 0:
    result_parts.append(orig_word)
    continue

if len(changed) >= 2:
    changed_vals = [v for _, v in changed]

    if len(set(changed_vals)) == 1: #["пішов", "пішов","пішов"]
        result_parts.append(changed_vals[0])
        continue

    counter = Counter(values)
    best_val, best_count = counter.most_common(1)[0]

    if best_count >= 2: #["пішов", "пішов","ПАшов"]
        result_parts.append(best_val)
        continue

# всі 3 дали різну відповідь
# пріоритет
# 1) pravopysnyk
# 2) spivavtor
# 3) languagetool

result_parts.append(MergeStrategy._tiebreak(i, orig_word, variants))

```

continue

changer_name, changer_val = changed[0]

#ТІЛЬКИ ОДНА МОДЕЛЬ ЗАМІНИЛА

result_parts.append(

MergeStrategy._handle_single_change(i, orig_word, changer_name,
changer_val, variants, model_aligned, orig_words)
)

return result_parts

@staticmethod

def _handle_single_deletion(

i: int,

orig_word: str,

variants: Dict[str, str],

model_aligned: Dict[str, Dict[int, str]],

orig_words: List[str],

result_parts: List[str]

) -> str:

absorber = [n for n in variants if variants[n] == ""][0]

absorber_map = model_aligned[absorber]

has_block_replacement = False

for prev_idx in range(i - 1, -1, -1):

prev_val = absorber_map.get(prev_idx, "")

if prev_val and prev_val != orig_words[prev_idx]:

has_block_replacement = True #["не", "можу"] => ["неможу", ""]

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савінький Р.С.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        break
    elif prev_val != "":
        break

    if has_block_replacement and result_parts and i > 0 and result_parts[-1] !=
orig_words[i - 1]:
        return "
    return orig_word

@staticmethod
def _handle_single_change(
    i: int,
    orig_word: str,
    changer_name: str,
    changer_val: str,
    variants: Dict[str, str],
    model_aligned: Dict[str, Dict[int, str]],
    orig_words: List[str]
) -> str:
    if changer_name in ('pravopysnyk', 'spivavtor'):
        is_artifact = MergeStrategy._is_split_artifact(i, changer_name, changer_val,
model_aligned, orig_words)

        if is_artifact:
            logger.info(f'Word [{i}]: '{orig_word}' kept ({changer_name} solo,
artifact)")
            return orig_word

    changer_map = model_aligned[changer_name]
    if (i + 1) < len(orig_words) and changer_map.get(i + 1) == "":

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        logger.info(f"Word [{i}]: '{orig_word}' kept ( {changer_name} solo, block-
collapse rejected)")
        return orig_word

```

```

        logger.info(f"Word [{i}]: '{orig_word}' -> '{changer_val}' ( {changer_name}
solo)")

```

```

        return changer_val

```

```

    else:

```

```

        #LT змінив в соло(тоді приймаємо)

```

```

        logger.info(f"Word [{i}]: '{orig_word}' -> '{changer_val}' (LTsolo)")

```

```

        return changer_val

```

```

@staticmethod

```

```

def _is_split_artifact(

```

```

    i: int,

```

```

    model_name: str,

```

```

    word_val: str,

```

```

    model_aligned: Dict[str, Dict[int, str]],

```

```

    orig_words: List[str]

```

```

) -> bool:

```

```

    model_map = model_aligned[model_name]

```

```

    #оригінал:["не", "можливо"]

```

```

    #модель: ["нем", "ожливо"]

```

```

    for neighbor in (i - 1, i + 1):

```

```

        if 0 <= neighbor < len(orig_words):

```

```

            neighbor_val = model_map.get(neighbor, orig_words[neighbor])

```

```

            if neighbor_val != orig_words[neighbor] and neighbor_val != "":

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савіцький Р.С.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if neighbor < i:
            orig_pair = orig_words[neighbor] + orig_words[i]
            corr_pair = neighbor_val + word_val
        else:
            orig_pair = orig_words[i] + orig_words[neighbor]
            corr_pair = word_val + neighbor_val

    orig_clean = re.sub(r'[.,!?:;]', '', orig_pair).lower()
    corr_clean = re.sub(r'[.,!?:;]', '', corr_pair).lower()

    if orig_clean == corr_clean:
        return True

    return False

@staticmethod
def _tiebreak(i: int, orig_word: str, variants: Dict[str, str]) -> str:
    if variants['pravopysnyk'] != orig_word:
        logger.info(f'Word [{i}]: '{orig_word}' -> '{variants["pravopysnyk"]}'
(tiebreak: P)')
        return variants['pravopysnyk']
    elif variants['spivavtor'] != orig_word:
        logger.info(f'Word [{i}]: '{orig_word}' -> '{variants["spivavtor"]}' (tiebreak:
S)')
        return variants['spivavtor']
    else:
        return orig_word

```

Лістинг коду файлу error_classifier.py

```

import re
import logging
from typing import List
from difflib import SequenceMatcher

from schemas.request_response import ErrorDetail, ErrorType

logger = logging.getLogger(__name__)

class ErrorClassifier:

    def __init__(self, grammar_tool=None):
        self.grammar_tool = grammar_tool
        logger.info("ErrorClassifier initialized")

    def classify_errors(self, original: str, corrected: str) -> List[ErrorDetail]:
        errors = []

        # "я іду додому"
        # [("я",0,1), ("іду",2,5), ("додому",6,12)]
        orig_words = [(m.group(), m.start(), m.end()) for m in re.finditer(r'\S+',
original)]
        corr_words = [(m.group(), m.start(), m.end()) for m in re.finditer(r'\S+',
corrected)]

        orig_texts = [w[0] for w in orig_words]
        corr_texts = [w[0] for w in corr_words]

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіцький Р.С.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

matcher = SequenceMatcher(None, orig_texts, corr_texts)

for tag, i1, i2, j1, j2 in matcher.get_opcodes():
    if tag in ('replace', 'delete', 'insert'):

        #ділимо на речення
        sub_blocks = self._split_at_sentence_boundaries(
            tag, i1, i2, j1, j2, orig_texts, corr_texts
        )
        for si1, si2, sj1, sj2 in sub_blocks:
            # "спить добре"
            orig_phrase = ''.join(orig_texts[si1:si2]) if si1 < si2 else "
            # "відпочиває"
            corr_phrase = ''.join(corr_texts[sj1:sj2]) if sj1 < sj2 else "

            if not orig_phrase and not corr_phrase:
                continue

            position = orig_words[si1][1] if si1 < len(orig_words) else 0
            length = (orig_words[si2 - 1][2] - position) if si2 > si1 else
len(orig_phrase)

            error_type = self._determine_error_type(orig_phrase, corr_phrase,
original, position)

            errors.append(ErrorDetail(
                original=orig_phrase,
                suggestion=corr_phrase,
                position=position,

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        length=length,
        error_type=error_type,
        message=self._generate_message(orig_phrase, corr_phrase,
error_type),
    ))

```

```

logger.info(f'Classified {len(errors)} errors')
return errors

```

```

def _split_at_sentence_boundaries(

```

```

    self,
    tag: str,
    i1: int, i2: int,
    j1: int, j2: int,
    orig_texts: List[str],
    corr_texts: List[str],

```

```

) -> List[tuple]:

```

```

    orig_slice = orig_texts[i1:i2]
    corr_slice = corr_texts[j1:j2]

```

```

    if len(orig_slice) <= 1:
        return [(i1, i2, j1, j2)]

```

```

    split_points_o = [
        k + 1
        for k in range(len(orig_slice) - 1)
        if re.search(r'[.!?]$', orig_slice[k])
    ]

```

```

    if not split_points_o:

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return [(i1, i2, j1, j2)]

    results = []
    prev_o, prev_c = 0, 0
    for o_split in split_points_o:
        if len(orig_slice) == len(corr_slice):
            c_split = o_split
        else:
            ratio = o_split / len(orig_slice)
            c_split = max(1, min(len(corr_slice) - 1, round(ratio * len(corr_slice))))

        results.append((i1 + prev_o, i1 + o_split, j1 + prev_c, j1 + c_split))
        prev_o, prev_c = o_split, c_split

    results.append((i1 + prev_o, i2, j1 + prev_c, j2))
    return results

def _determine_error_type(self, error_text: str, suggestion: str, full_text: str,
position: int) -> ErrorType:

    if error_text and suggestion and error_text.lower() == suggestion.lower():
        return ErrorType.CASING

    lt_type = self._get_error_type_from_languagetool(error_text, suggestion,
full_text, position)
    return lt_type if lt_type != ErrorType.UNKNOWN else
self._auto_classify_error_type(error_text, suggestion)

def _get_error_type_from_languagetool(self, error_text: str, suggestion: str,
full_text: str, position: int) -> ErrorType:

```

```

if not self.grammar_tool:
    return ErrorType.UNKNOWN

try:
    matches = self.grammar_tool.check(full_text)
    for match in matches:
        if abs(match.offset - position) <= 3:
            return self._map_category_to_type(match.category)
    return ErrorType.UNKNOWN
except Exception:
    return ErrorType.UNKNOWN

```

```

def _map_category_to_type(self, category: str) -> ErrorType:
    mapping = {
        "GRAMMAR": ErrorType.GRAMMAR,
        "TYPOS": ErrorType.SPELLING,
        "PUNCTUATION": ErrorType.PUNCTUATION,
        "CASING": ErrorType.CASING,
        "STYLE": ErrorType.STYLE,
    }
    return mapping.get(category.upper(), ErrorType.UNKNOWN)

```

```

def _auto_classify_error_type(self, error_text: str, suggestion: str) -> ErrorType:
    if not error_text or not suggestion:
        return ErrorType.UNKNOWN

    if error_text.lower() == suggestion.lower():
        return ErrorType.CASING

    error_no_punct = re.sub(r'[^\w\s]', "", error_text)

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіуький Р.С.				97
Змн.	Арк.	№ докум.	Підпис	Дата		

```

suggestion_no_punct = re.sub(r'[\^w\s]', "", suggestion)

if error_no_punct == suggestion_no_punct:
    return ErrorType.PUNCTUATION

#дуже короткі рядки тяжко класифікувати
if len(error_text) < 2 or len(suggestion) < 2:
    return ErrorType.SPELLING

#виправлення більше 1 слова
if len(error_text.split()) > 1 or len(suggestion.split()) > 1:
    return ErrorType.GRAMMAR

# схожість слів
similarity = SequenceMatcher(None, error_text.lower(),
suggestion.lower()).ratio()
if similarity > 0.7:
    return ErrorType.SPELLING

return ErrorType.GRAMMAR

@staticmethod
def _generate_message(original: str, suggestion: str, error_type: ErrorType) -> str:
    type_names = {
        ErrorType.SPELLING: "Spelling error",
        ErrorType.GRAMMAR: "Grammar error",
        ErrorType.PUNCTUATION: "Punctuation error",
        ErrorType.CASING: "Casing error",
        ErrorType.STYLE: "Style suggestion",

```

		Фоменко В.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савіцький Р.С.				98
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        ErrorType.UNKNOWN: "Correction",
    }

    type_name = type_names.get(error_type, "Correction")

    if not suggestion or suggestion.strip() == "":
        return f"{type_name}: '{original}' (delete)"

    if not original or original.strip() == "":
        return f"{type_name}: (insert) '{suggestion}'"

    return f"{type_name}: '{original}' -> '{suggestion}'"

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савіиький Р.С.				99
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг коду файлу useCorrector.ts

```

import { useState } from "react";
import { checkText } from "../api/corrector";
import type { CheckMode } from "../api/corrector";
import type { TextError } from "../types";

interface UseCorrector {
  text: string;
  setText: (val: string) => void;
  errors: TextError[];
  loading: boolean;
  isChecked: boolean;
  activeId: string | null;
  setActiveId: (id: string | null) => void;
  expandedId: string | null;
  setExpandedId: (id: string | null) => void;
  mode: CheckMode;
  setMode: (mode: CheckMode) => void;
  checkNow: () => void;
  applyFix: (currentError: TextError) => void;
  ignoreError: (error: TextError) => void;
  clearAll: () => void;
}

export function useCorrector(): UseCorrector {
  const [text, setTextRaw] = useState("");
  const [errors, setErrors] = useState<TextError[]>([]);
  const [loading, setLoading] = useState(false);
  const [isChecked, setIsChecked] = useState(false);

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савіуький Р.С.				100
Змн.	Арк.	№ докум.	Підпис	Дата		

```
const [activeId, setActiveId] = useState<string | null>(null);
const [expandedId, setExpandedId] = useState<string | null>(null);
const [mode, setMode] = useState<CheckMode>("fast");
```

```
function setText(val: string) {
  setIsChecked(false);
  setTextRaw(val);
}
```

```
async function checkNow() {
  if (!text.trim()) {
    setErrors([]);
    return;
  }
  setErrors([]);
  setActiveId(null);
  setExpandedId(null);
  setLoading(true);
  try {
    const found = await checkText(text, mode);
    setErrors(found);
    setIsChecked(true);
  } catch (err) {
    console.error("Check failed:", err);
  } finally {
    setLoading(false);
  }
}
```

```
function applyFix(currentError: TextError) {
```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Савінький Р.С.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

```

console.log(currentError)

const before = text.slice(0, currentError.offset); // текст до помилки
const after = text.slice(currentError.offset + currentError.length); // текст після
помилки

const delta = currentError.suggestion.length - currentError.length; // різниця
довжини між виправленням та помилкою

const updated = errors
    .filter((e) => e.id !== currentError.id) // залишаємо всі помилки окрім тої яку
виправляємо
    .map((e) => e.offset > currentError.offset ? { ...e, offset: e.offset + delta } : e); //
після зміни, побібно додати або відняти оффсет у всіх помилок які йдуть після
виправленої помилки (згідно по делті)

setErrors(updated);

setTextRaw(before + currentError.suggestion + after);
setActiveId(null);
setExpandedId(null);
}

function ignoreError(error: TextError) {
    setErrors((prev) => prev.filter((e) => e.id !== error.id));
    setActiveId(null);
    setExpandedId(null);
}

function clearAll() {
    setTextRaw("");

```

```

setErrors([]);
setIsChecked(false);
setActiveId(null);
setExpandedId(null);
}

```

```

return {
  text,
  setText,
  errors,
  loading,
  isChecked,
  activeId,
  setActiveId,
  expandedId,
  setExpandedId,
  mode,
  setMode,
  checkNow,
  applyFix,
  ignoreError,
  clearAll,
};
}

```

		Фоменко В.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Савіуький Р.С.				103
Змн.	Арк.	№ докум.	Підпис	Дата		