

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 «Інженерія програмного забезпечення»)
на тему:

«Багатомовний тлумачний словник»

Виконав студент групи ІПЗ-21-1
ЧЕВЕРДА Анастасія Богданівна

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЛОКТИКОВА Тамара Миколаївна

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«23» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЧЕВЕРДА Анастасія Богданівна**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Багатомовний тлумачний словник»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: Об'єктом дослідження є використання нейронних мереж для реалізації семантичного пошуку слів у багатомовному вебсловнику. Метою дослідження є розробка багатомовного тлумачного словника у вигляді вебзастосунку.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	14 лютого 2025 р.	14 лютого 2025 р.
2	Вакалюк Т.А.	14 лютого 2025 р.	14 лютого 2025 р.
3	Вакалюк Т.А.	14 лютого 2025 р.	14 лютого 2025 р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «бакалавр» складається з вебзастосунку багатомовного тлумачного словника та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 60 сторінок, 27 ілюстрацій та 20 таблиць. Загальний обсяг 85 сторінок.

Метою роботи є розробка багатомовного тлумачного словника. Вебзастосунок включає модулі для управління термінами та їхніми категоріями, користувачами та їхніми особистими списками, а також пошук за семантичними значеннями слів.

У роботі поставлені основні завдання на розробку системи. Також було проведено порівняльний аналіз аналогічних систем. Було обрано клієнт-серверну архітектуру, яка працює в зв'язці з RESTful інтерфейсом. Приведені основні сценарії використання системи, логіка взаємодії між модулями системи, діаграма класів та алгоритми роботи ключових модулів, спроектована база даних з використанням PostgreSQL. Клієнтська частина розроблена за допомогою фреймворку React, а серверна частина за допомогою Python у поєднанні з фреймворком Django REST Framework.

КЛЮЧОВІ СЛОВА: ПРОЄКТ, СЛОВНИК, ТЕРМІНИ, ТЛУМАЧЕННЯ, БАГАТОМОВНІСТЬ, ВЕКТОРНІ ВКЛАДЕННЯ, СЕМАНТИЧНИЙ ПОШУК.

					ІПЗ.КР.Б-121-25-ІПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Чеве́рда А.Б.			Багатомовний тлумачний словник			Літ.	Арк.	Аркуші́в	
Кері́вник		Вакалюк Т.А.								3	85
Н. Контр.		Ти́чина О.П.			Житоми́рська політехні́ка, гру́па ІПЗ-21-1						
Зав. каф.		Вакалюк Т.А.									

ABSTRACT

The bachelor's thesis consists of a construction process for multilingual explanatory dictionary web application and an explanatory note. The explanatory note of the qualification work contains 60 pages, 27 illustrations, and 20 tables. The total volume is 85 pages.

The purpose of the work is to develop a multilingual explanatory dictionary web application. The web application contains modules for managing terms and their categories, users and their personal lists, as well as semantic search by word meaning.

The paper sets out the main tasks for the development of the system. A comparative analysis of similar systems was also conducted. A client-server architecture was chosen that works in conjunction with RESTful interface. The main scenarios of using the system, the logic of interaction between the system modules, the diagram and algorithms of the key modules are presented, and the database is designed using PostgreSQL. The client side was developed using React, and the server side was developed using Python together with Django REST Framework.

KEYWORDS: PROJECT, DICTIONARY, TERMS, DEFINITIONS, MULTILINGUALISM, WORD EMBEDDINGS, SEMANTIC SEARCH.

					ІПЗ.КР.Б-121-25-ІПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Чеверда А.Б.			Багатомовний тлумачний словник			Лім.	Арк.	Аркуші	
Керівник		Вакалюк Т.А.								4	85
Н. Контр.		Тичина О.П.			Житомирська політехніка, група ІПЗ-21-1						
Зав. каф.		Вакалюк Т.А.									

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	12
1.3 Вибір архітектури застосунку	15
1.4 Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ БАГАТОМОВНОГО СЛОВНИКУ	23
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	23
2.2 Розробка бази даних системи.....	28
2.3 Проектування та реалізація алгоритмів роботи системи	33
2.4 Реалізація функціоналу багатомовного словнику	37
Висновки до другого розділу	40
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ	42
3.1 Структура інтерфейсу вебзастосунку багатомовного тлумачного словника	42
3.2 Загальна статистика словника	51
3.3 Тестування вебзастосунку.....	53
Висновки до третього розділу.....	54
ВИСНОВКИ.....	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58
ДОДАТКИ.....	60
Додаток А.....	61
Додаток Б	61

		Чевежда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕННЯ

API — Application Programming Interface

B-Tree — Balanced Tree

БД — база даних

CI-пайплайни — Continuous Integration pipelines

DRF — Django REST Framework

HTML — HyperText Markup Language

HTTP — HyperText Transfer Protocol

JSON — JavaScript Object Notation

JWT — JSON Web Token

ORM — Object-Relational Mapping

pgvector — розширення PostgreSQL для зберігання та пошуку векторних представлень

RAM — Random Access Memory

REST API — Representational State Transfer Application Programming Interface

SQL — Structured Query Language

SSD — Solid State Drive

СУБД — система управління базами даних

vCPU — virtual Central Processing Unit

WAL — Write-Ahead Logging

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми. У сучасному цифровому світі мова еволюціонує разом із технологіями, підлаштовуючись під потреби людини, – вона постійно поповнюється новітніми термінами, які описують інноваційні винаходи та явища. Саме тому словник як потужний інструмент для навчання та роботи завжди користуватиметься попитом.

Зараз, коли будь-яку книжку легко діджиталізувати, питання доступності словників більше не постає. Натомість, критичним аспектом є ефективність результату пошуку заданого запиту серед великої кількості даних. Традиційний алфавітний пошук часто недоцільний, особливо коли користувач намагається відшукати потрібне слово за значенням чи навіть контекстом.

З винаходом штучного інтелекту це можна реалізувати, інтегрувавши семантичний пошук. Такий підхід дозволить шукати потрібне слово за значенням, контекстом і потенційно навіть за емоційним забарвленням. Це зробить роботу зі словниками набагато простішою та приємнішою навіть для пересічного користувача.

Метою дослідження є розробка багатомовного тлумачного словника у вигляді вебзастосунку, який надає користувачам можливість розширеного пошуку слів за їхнім семантичним значенням, використовуючи нейронні мережі.

Практичними завданнями кваліфікаційної роботи є:

- аналіз предметної області, вивчення програмних рішень у сфері побудови інтерактивних тлумачних словників, вивчення сучасних підходів до створення векторних індексів для лексичних баз, формулювання ключових вимог користувачів до майбутньої платформи;
- визначення та аргументація вибору стеку технологій для реалізації фронтенду, серверної частини, сховища даних та інтеграцій зі зовнішніми сервісами;

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

– формулювання переліку функціональних і нефункціональних вимог до системи, розробка сценаріїв використання залежно від ролі користувача — гість, зареєстрований користувач;

– проектування структури бази даних для зберігання інформації про терміни, їхні визначення, ембеддинги для семантичного пошуку, профілі користувачів, персональні списки слів, категорії термінів, статистику використання словника;

– розробка серверної частини сайту з використання Django REST Framework, а саме створення REST-ендпоінтів для управління термінами, виконання семантичного та традиційного пошуку, керування персональними списками користувача, обліку статистики використання; інтеграція з зовнішніми словниковими API для автоматичного поповнення корпусу термінів;

– забезпечення інтерфейсу, що пропонує користувачеві зручну навігацію, швидкий пошук слів за семантикою та префіксом, перегляд визначень і прикладів вживання, керування персональними списками слів, а також доступ до статистики активності користувача;

– створення й реалізація основних модулів платформи, включаючи семантичний пошук термінів на базі попередньо згенерованого векторного індексу, імпорт нових термінів із зовнішніх сервісів та підтягнення підказок під час введення запиту до пошукової стрічки;

– проведення комплексного тестування клієнтської й серверної частин за допомогою набору тест-кейсів та модульних перевірок, фіксування й усунення виявлених дефектів, підтвердивши відповідність розробленого продукту заявленим вимогам і коректності роботи.

Об’єктом дослідження є використання нейронних мереж для реалізації семантичного пошуку слів у багатомовному вебсловнику. Дослідження спрямоване на вивчення технологій та методів для забезпечення семантичного пошуку та оптимізації взаємодії користувачів із термінами.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Предметом дослідження є розробка багатомовного вебсловника із можливістю пошуку слів на основі їхнього значення, завдяки використанню нейронних мереж.

Вебсловник із семантичним пошуком дозволить за лічені секунди знайти потрібні слова та їхні переклади, описавши їхнє значення простою мовою. Цей метод не вимагає від пересічного користувача глибоких знань академічних термінів чи словникових визначень для ефективного пошуку, на відміну від більшості існуючих вебсловників. Такий інструмент стане у нагоді широкому колу користувачів – від студентів та професорів до аналітиків та редакторів, кожен з яких регулярно звертається до словників під час навчальної, дослідницької чи освітньої роботи.

За тематикою дипломної роботи були опубліковані тези [1-2] та монографія [3].

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Постановка задачі

Основною метою роботи є розробка тлумачного словника у вигляді вебзастосунку, який надає користувачам можливість розширеного пошуку слів за їхнім значенням, використовуючи нейронні мережі.

Вебзастосунок тлумачного словника має складатися з деяких ключових програмних модулів, а саме:

Модуль користувача забезпечує реєстрацію, авторизацію та управління обліковими записами користувачів. У межах клієнтського інтерфейсу зареєстрований користувач отримує такі можливості, як редагувати власний профіль, змінювати пароль, впад створювати та переглядати персоналізовані списки і статистику популярних до пошуку та вподобання термінів.

Модуль термінів є основним для роботи зі словами. Він забезпечує потрапляння до системи термінів, що відбувається двома шляхами: або через автоматичне створення під час першого запиту до зовнішніх ендпоїнтів, або через масовий імпорт офлайн-процедурою, яка одразу генерує вкладення слів. Отже, втручання живого адміністратора для наповнення корпусу не потрібне.

Модуль категорій класифікує слова за тематичними категоріями, такими як "спорт", "політика" або "суспільство". Він дозволяє створювати нові категорії, прив'язувати до них слова та здійснювати тематичний пошук для відображення термінів, що належать до певної теми.

Модуль списків дає змогу користувачам створювати персональні списки слів для навчання, роботи чи інших цілей. Користувач може додавати та видаляти слова у власних списках, а також має змогу стежити за списками інших користувачів, якщо ті обрали зробити свої списки доступними до загального перегляду. Гість має право лише переглядати публічні списки.

Модуль пошуку дозволяє користувачам здійснювати традиційний пошук серед термінів за ключовими словами, а також їхнім семантичним значенням, які відповідають або схожі на запит користувача.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Модуль статистики аналізує та надає інформацію про використання словника, як наприклад, частота перегляду, кількість вподобань певних термінів, а також найпопулярніші підказки серед користувачів.

Основними етапами розробки буде розробка дизайну вебзастосунку, побудова архітектури серверної та клієнтської частин системи, розробка основного функціоналу та інтеграція семантичного пошуку, а також тестування і вдосконалення вебзастосунку.

На клієнтській частині передбачається інтерфейс, що адаптивно відображає описані функції. Гість отримує можливість пошуку та читання визначень, зареєстрований користувач — доступ до розширеного меню з управлінням списками, вподобаннями і параметрами облікового запису. Макет сторінок буде реалізовано з урахуванням простоти навігації.

У серверній частині передбачено побудова авторизації на механізмі JWT-токену [4], що уніфіковано захищає приватні маршрути REST-API [5] й забезпечує швидкий розподіл прав доступу між гостем і зареєстрованим користувачем. Для підтримки масштабованості й зниження затримок при семантичному пошуку використовується попередньо сформований векторний індекс у pgvector [6].

Окремо від програми буде проводитися функціональне тестування за допомогою Postman [7], що допоможе відслідковувати перебіг розробки, точність виконання завдань та правильне підключення сторонніх API. Також буде проводитися ручне тестування, де тестувальник буде переглядати всю платформу з повною перевіркою можливостей сайту.

Результатом реалізації поставленого завдання є веборієнтована система для багатомовного тлумачного словника, яка включає багатокористувацький клієнтський застосунок, серверну частину для обробки запитів, базу даних для зберігання інформації, а також ключові модулі для управління термінами, категоріями, списками та статистикою.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

1.2 Аналіз аналогів програмного продукту

Основна мета платформи тлумачного словника із семантичним пошуком полягає у створенні інструменту, який дозволяє ефективно працювати з мовними даними, забезпечуючи користувачам зручний доступ до визначень слів. Система покликана не лише надавати традиційний пошук за ключовими словами, а й забезпечувати семантичний пошук, який враховує контекст та значення запиту. Це дає змогу отримувати більш релевантні результати, ніж у звичайних словниках.

Крім цього, платформа підтримує організацію термінів за категоріями, персоналізовані списки слів, що покращує загальну якість роботи з мовними ресурсами. Для вивчення можливостей ринку та визначення оптимальних рішень, необхідних для впровадження системи, проведемо аналіз існуючих аналогів.



Рисунок 1.1 – Cambridge Dictionary

		Чеверда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

В якості першого аналогу було розглянуто Cambridge Dictionary [8] (рис 1.1) – онлайн-словник, розроблений Кембриджським університетом, який спеціалізується на наданні визначень англійських слів, перекладів і прикладів використання. Основними перевагами платформи є зручний інтерфейс, доступ до граматичних пояснень, можливість перегляду транскрипцій і прикладів вживання в різних контекстах. Платформа також дозволяє користувачам створювати персональні списки із вподобаних слів. Проте словник не пропонує інструментів семантичного пошуку.

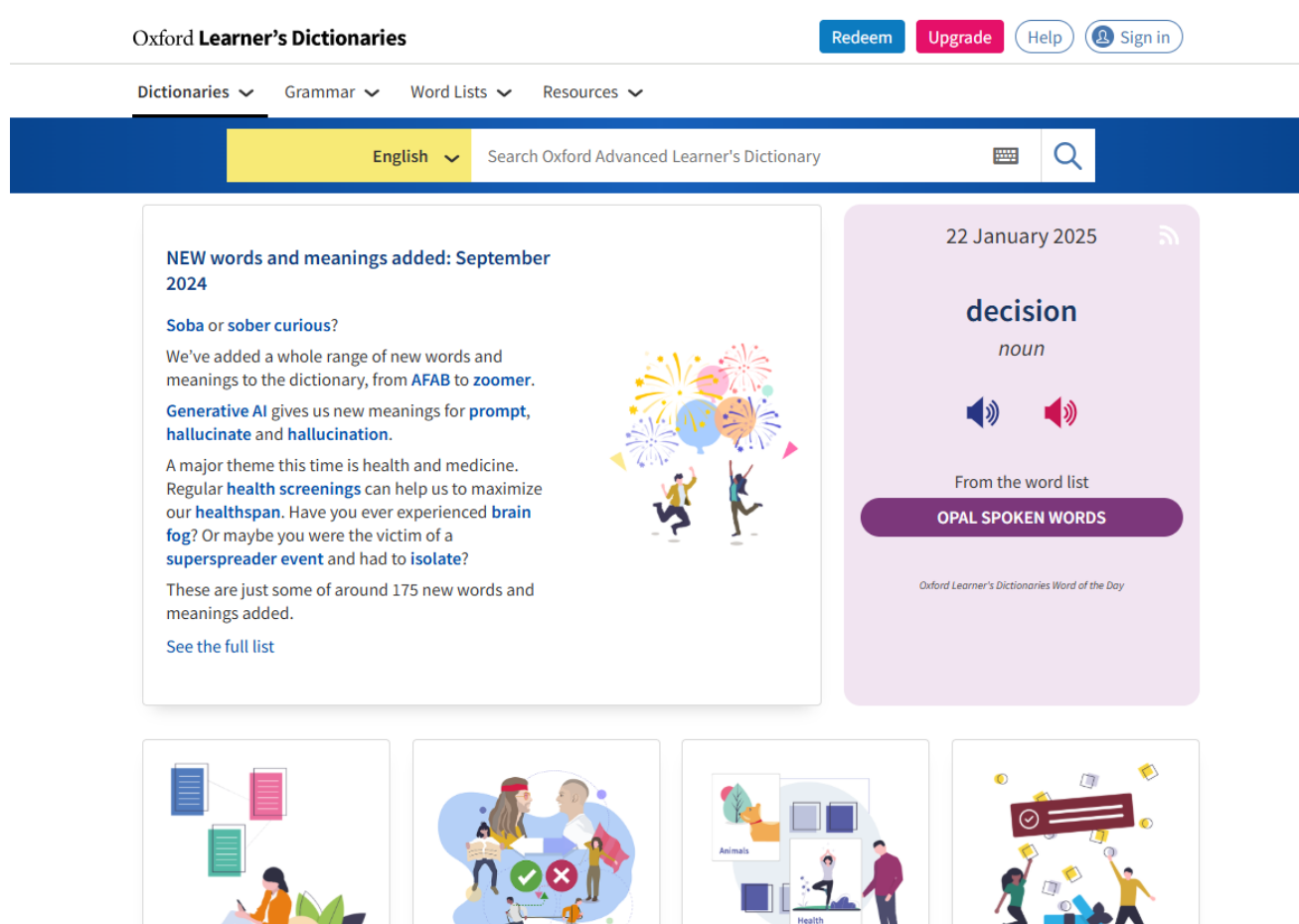


Рисунок 1.2 – Oxford Learner's Dictionaries

Oxford Learner's Dictionaries [9] (рис. 1.2) – онлайн-словник, орієнтований на студентів та людей, які вивчають англійську мову. Його головними перевагами є великий вибір навчальних ресурсів, таких як тематичні списки слів, граматичні вправи та академічна лексика. Недоліком є відсутність семантичного пошуку та обмежені можливості інтеграції з іншими мовами.

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

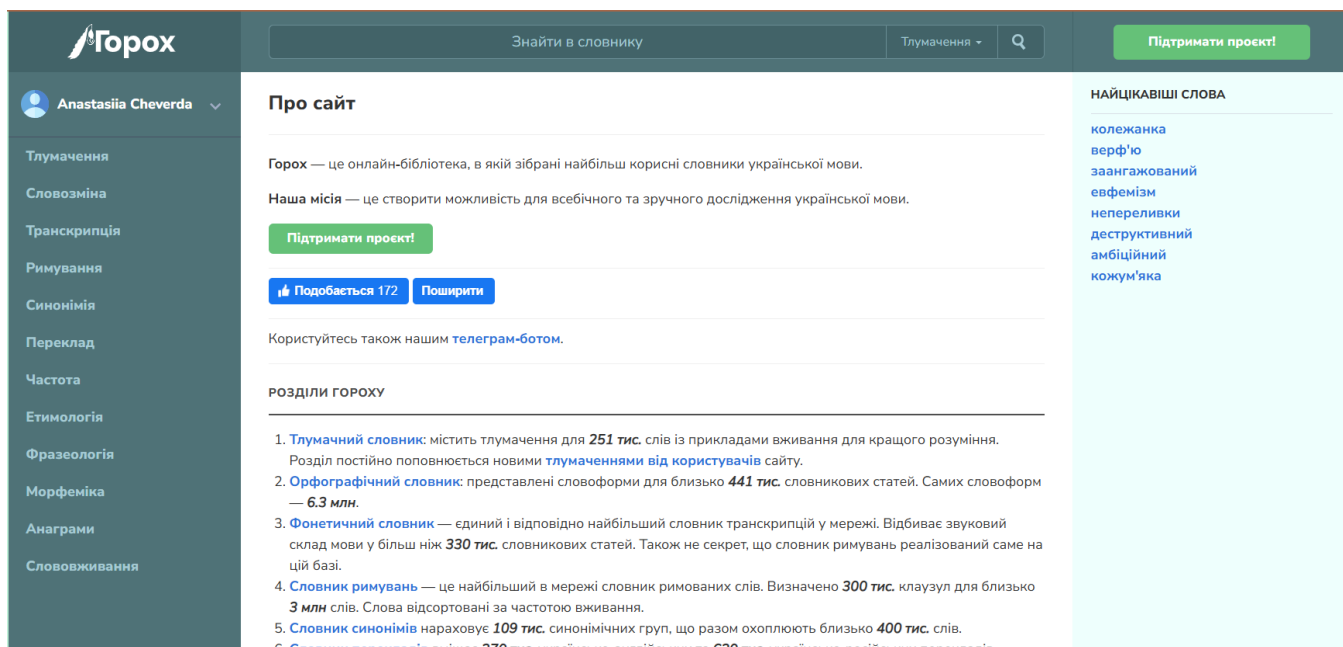


Рисунок 1.3 – Горох

Горох [10] (рис. 1.3) — це онлайн-бібліотека, яка містить найбільш корисні словники української мови.

Серед її переваг різноманіття словників, включаючи тлумачний та синонімічний, що дозволяє користувачам отримувати детальні визначення слів та знаходити їхні синоніми. Цей вебсайт пропонує безліч прикладів вживання слів у різних контекстах, що допомагає краще зрозуміти їхнє значення та використання. Проте, серед недоліків слід зазначити, що семантичний пошук реалізовано лише для слововживання. Це обмежує можливості пошуку та може ускладнювати процес знаходження потрібної інформації для користувачів, які шукають глибші та більш контекстуальні зв'язки між словами.

Порівняння функціональних можливостей приведено нижче (табл. 1.1).

Таблиця 1.1

Функціональні можливості

	Cambridge Dictionary	Oxford Dictionaries	Горох	Багатомовний тлумачний словник
Зручний інтерфейс	Так	Так	Так	Так
Розподіл слів на категорії	Ні	Так	Ні	Так
Семантичний пошук	Ні	Частково	Частково	Так

Продовження таблиці 1.1

Персональні списки	Так	Ні	Ні	Так
Перегляд статистики	Так	Ні	Ні	Так

У кожної з представлених платформ є свої особливості та унікальні функції, які можуть задовольнити потреби різних користувачів. Cambridge Dictionary пропонує не тільки перегляд статистичних даних, а й можливість створювати персональні користувацькі списки; Oxford Learner's Dictionaries орієнтований на навчання та має багато термінів, розділених на різні категорії; Горох надає широкий вибір українських словників і прикладів вживання слів у контексті.

Отже, основними характеристиками нової платформи тлумачного словника із зворотнім пошуком мають бути наявність інтуїтивного вебінтерфейсу, розподіл слів на категорії та списки, модулі для пошуку, перегляду термінів і для підбиття статистики, а також можливість здійснювати пошук за семантичними значенням.

1.3 Вибір архітектури застосунку

Застосунок тлумачного словника оперує структурованими лексикографічними даними, такими як терміни, їхні визначення, приклади використання та синоніми, а також векторними вкладеннями для семантичного пошуку, персональними списками користувачів. Окрім того, ведеться облік статистики взаємодій із системою.

Є потреба взаємодіяти із кількома зовнішніми API для налаштування наповнення та оновлення корпусу термінів. Це вимагає чітко визначених точок входу, які можуть обслуговуватися різними компонентами системи.

У системі передбачено лише дві ролі: гість та зареєстрований користувач. Це дозволяє спростити модель авторизації та уникнути надлишкової логіки прав доступу. Адміністративна модерація не потрібна, оскільки якість контенту

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

забезпечується авторитетними зовнішніми джерелами, а додавання нових термінів автоматизоване.

Отже, очікується обробка запитів від багатьох користувачів у реальному часі, включно із виконанням семантичного пошуку з мінімальною затримкою (< 200 мс), оновленням персональних даних, взаємодією зі сторонніми API. У перспективі можливо зростання корпусу термінів до сотень тисяч, що вимагає ефективного зберігання та пошуку у векторному просторі.

У зв’язку із усіма вищепереліченими вимогами для розробки вебзастосунку тлумачного словника було обрано клієнт-серверну архітектуру (рис. 1.4), яка забезпечує чіткий розподіл функцій між клієнтською та серверною частинами. Такий підхід дозволяє організувати обробку, зберігання та представлення даних, забезпечуючи високу масштабованість, безпеку й гнучкість системи.

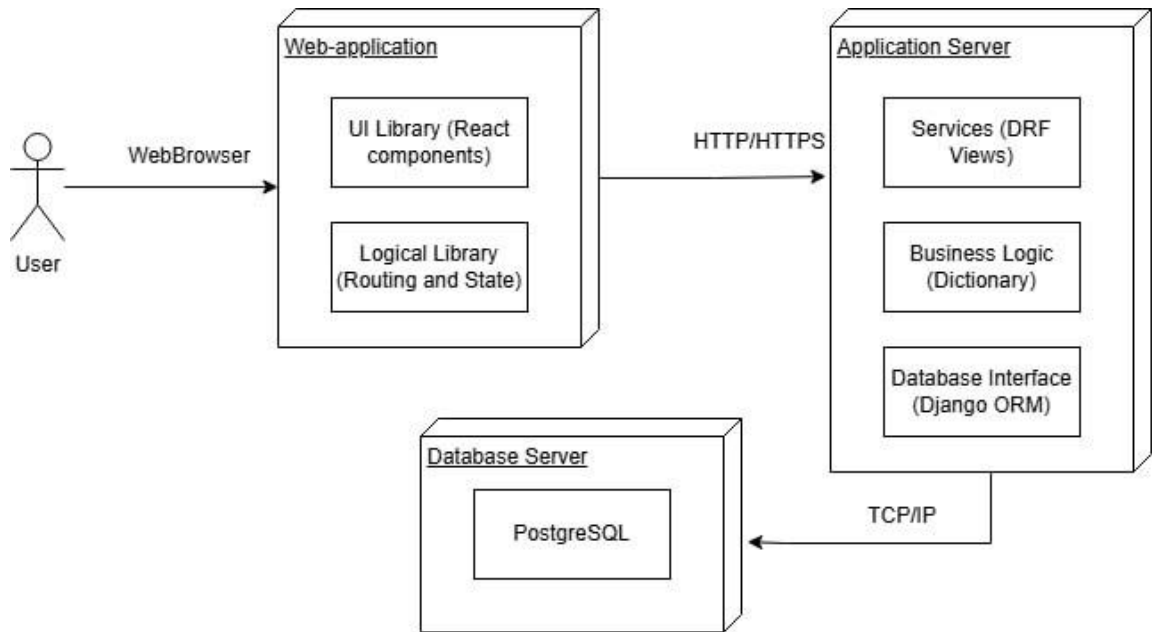


Рисунок 1.4 – Схема клієнт-серверної архітектури вебзастосунку

Клієнт-серверна архітектура розбиває систему на два ключових компоненти: клієнта та сервера. React-клієнт [11] повністю відповідає за взаємодію з користувачем та інтерфейс, охоплюючи від реєстрації та авторизації до управління персоналізованими списками, перегляду термінів та здійснення пошуку. Запити надсилаються на сервер у форматі JSON для

обробки даних та отримання даних, мінімізуючи навантаження на канал зв'язку та забезпечуючи миттєву реакцію клієнта.

Для серверної частини застосунку було обрано Django REST Framework [12] як основний фреймворк для розробки API. Контролери REST API приймають HTTP-запити [13] від клієнтів, серіалізатори перетворюють дані моделей у придатний для клієнта формат. DRF забезпечує стандартизований підхід до створення ендпоінтів, які дозволяють працювати з термінами, категоріями та користувацькими даними. Основними перевагами є його висока гнучкість, можливість побудови кастомізованих логік і зручна інтеграція з базами даних.

Зовнішні інтеграції підключено у вигляді окремих API-методів, які взаємодіють із сторонніми джерел, зокрема, Wordnik [14], Datamuse [15] та Lexicala [16]. Оскільки ці сервіси не вимагають підтримки постійного з'єднання, інтеграції працюють на базі звичайних REST-запитів.

Семантичний пошук базується на моделі Sentence-Transformer paraphrase-multilingual-MiniLM-L12-v2 [17], що генерує багатомовні векторні представлення для термінів і пошукових запитів. Це забезпечує точність і релевантність результатів навіть у багатомовному середовищі.

В якості основної бази даних обрано PostgreSQL [18], яке відповідає за збереження термінів, визначень, категорій та пов'язаних даних. Додатково, розширення pgvector забезпечує зберігання та пошук за векторними представленнями на основі евклідової відстані [19]. Пошук здійснюється безпосереднім виконанням SQL-запитів із оператором <->, що дозволяє ефективно обробляти навіть великі обсяги даних.

Модель авторизації базується на безстанних JWT-токенах. Кожен приватний запит до REST API супроводжується access-токеном. Завдяки цьому сервер легко масштабується горизонтально: будь-який екземпляр Django-серверу може обробити запит без необхідності синхронізації сесій. PostgreSQL за потреби підтримує потокову реплікацію, а pgvector дозволяє ефективно масштабувати пошук за ембеддингами.

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

Застосована клієнт-серверна архітектура забезпечує чіткий поділ відповідальностей, де React-клієнт відповідає за рендеринг інтерфейсу та валідацію введених даних, а сервер займається бізнес-логікою, зберіганням даних та інтеграцією зі сторонніми сервісами. Такий поділ спрощує тестування, покращує масштабованість і полегшує майбутнє розширення функціоналу, зокрема, додавання нових мов, підтримку розширених механізмів пошуку, а також введення додаткових модулів статистики.

Загалом обрана архітектура відповідає вимогам до системи, яка має забезпечувати інтерактивний доступ до великого обсягу текстових і векторних даних, ефективно інтегруватися із зовнішніми словниковими сервісами та залишатися легко масштабованою у разі зростання кількості користувачів.

1.4 Обґрунтування вибору інструментальних засобів

На початковій стадії розробки головним завданням був вибір стеку технологій, який дозволив би ефективно керувати лексикографічними даними, забезпечувати високу продуктивність при виконанні семантичного пошуку, гарантувати масштабованість системи та надати зручний і швидкий інтерфейс для кінцевого користувача. Крім того, важливим фактором була інтеграція зі сторонніми API для автоматичного поповнення словникової бази. Враховуючи потреби багатомовного середовища, було також необхідно підтримувати моделі нейронних мереж для побудови векторних представлень термінів.

Основою бекенд-архітектури було обрано Python у поєднанні з фреймворком Django REST Framework. Django надає зрілу та добре підтримувану платформу для швидкої розробки вебзастосунків, а DRF забезпечує стандартизований спосіб створення REST API. Контролери REST API дозволяють ефективно організувати обробку HTTP-запитів, а серіалізатори DRF спростити трансформацію моделей бази даних у формат JSON. Однією з ключових переваг DRF є його гнучкість і простота кастомізації ендпоінтів, що є важливим для підтримки специфічних механізмів пошуку за векторними ембеддингами та реалізації складної бізнес-логіки.

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Як базу даних для зберігання термінів, визначень, прикладів, синонімів, персональних списків користувачів та статистики використання було обрано PostgreSQL через її переваги в роботі зі складними структурами даних, такими як JSONB. Словник – це в першу чергу структурована система, а реляційна база даних дозволить створити зв'язки, або поєднати між собою багатомовні терміни.

Як демонструє порівняння реляційних баз (табл. 1.2), PostgreSQL має значні переваги у гнучкості та масштабованості порівняно з MySQL, особливо для збереження великих структурованих і неструктурованих даних.

Таблиця 1.2

Максимально доступний обсяг збережених даних для кожної СКБД

СКБД	Розмір БД	Розмір таблиці	Розмір рядка
MS SQL	524 258 ТБ	524 258 ТБ	∞
PostgreSQL	∞	32 ТБ	1.6 ТБ
MySQL	∞	256 ТБ	64 КВ

PostgreSQL має значні переваги в гнучкості та масштабованості порівняно з MySQL, особливо для збереження великих структурованих і неструктурованих даних.

До того ж, таку базу даних легко інтегрувати із векторними пошуками, які необхідні для реалізації семантичного пошуку. Саме тому було прийнято рішення обрати реляційну базу даних. Крім того, завдяки розширенню pgvector стало можливим зберігати векторні представлення термінів безпосередньо у базі даних та виконувати пошук найближчих сусідів за допомогою евклідової відстані, що забезпечує ефективність і масштабованість семантичного пошуку.

Під час вибору моделі для генерації векторних представлень було проведено аналіз сучасних моделей ембеддингів, результати якого наведено у таблиці 1.3. Після порівняння було прийнято рішення на користь paraphrase-multilingual-MiniLM-L12-v2 з пакету Sentence-Transformers. Ця модель забезпечує високу якість семантичного представлення, підтримує понад 50 мов, демонструє низький час обробки (20–50 мс) та не потребує ресурсомісткої інфраструктури. Від моделі Sentence-BERT було свідомо відмовлено через її

вищі вимоги до пам'яті та більшу затримку при обробці запитів. Водночас MiniLM-L12-v2 у поєднанні з pgvector дозволяє уникнути використання додаткових індексів, спрощуючи архітектуру системи.

Таблиця 1.3

Критерій	paraphrase-multilingual-MiniLM-L12-v2	LaBSE	OpenAI text-embedding-ada-002	BGE / SGPT (англійські моделі)
Мовна підтримка	50+ мов, включно з англійською, українською, іспанською	>100 мов, висока якість cross-lingual	Підтримка багатьох мов (точний список не публічний)	Лише англійська
Якість семантичного пошуку	Висока, оптимізована для retrieval	Дуже висока для cross-lingual retrieval	Висока, але менш стабільна cross-lingual	Висока в англійській, cross-lingual не призначена
Швидкість генерації векторів	Висока (легка модель MiniLM)	Помірна, модель велика	API-залежна, може мати латентність	Висока
Можливість локального використання	Локально через Sentence-Transformers	Локально через Sentence-Transformers	Лише через API OpenAI	Локально через Sentence-Transformers
Вартість використання	Безкоштовно (відкрита модель)	Безкоштовно (відкрита модель)	Платний API	Безкоштовно (відкрита модель)
Придатність для багатомовного словника	Оптимально	Дуже добре	Можливо, але вимагає тестування	Не придатна

Важливою перевагою обраної моделі є її інтеграція з бібліотекою Sentence-Transformers, що забезпечує можливість генерації ембеддингів як у фоновому режимі (bulk-індексація), так і динамічно під час додавання нових термінів через зовнішні API. Це реалізовано за допомогою механізму сигналів Django, що дозволяє автоматизувати оновлення векторного індексу без додаткового втручання адміністратора.

Для захисту приватних маршрутів API було обрано механізм JWT. Цей підхід дозволяє ефективно забезпечити безстанну авторизацію, що важливо для горизонтального масштабування системи у кластері.

Інтеграція з зовнішніми словниковими сервісами (Wordnik, Datamuse, Lexicala) побудована на основі REST-запитів. Нові терміни після першого звернення автоматично зберігаються у базі та стають доступними для подальшого семантичного пошуку, що забезпечує постійне розширення корпусу системи.

Для розробки клієнтської частини було використано React. Цей фреймворк забезпечує швидкий та динамічний інтерфейс користувача, гнучкість компонентного підходу та значно спрощує можливу подальшу адаптацію застосунку до мобільних платформ. Для тестування REST API активно використовується Postman, що дозволяє ефективно перевіряти коректність ендпоінтів та інтеграцій з зовнішніми сервісами.

Таким чином, обраний стек технологій повністю відповідає вимогам проєкту щодо побудови багатомовного семантичного словника. Архітектурні рішення платформи забезпечують її гнучкість, масштабованість, стійкість до змін та готовність до подальшого розвитку.

Висновки до першого розділу

У першому розділі було виконано постановку задачі на кваліфікаційну роботу та визначено функціональні вимоги до системи. Проведено дослідження ринку та аналогів, завдяки чому було визначено основні функціональні можливості системи: інтуїтивний користувацький інтерфейс швидкий та семантичний пошук термінів, перегляд та керування термінами, категоріями, користувацькими списками, а також відслідковування статистики пошуку та перегляду термінів.

Також було продемонстровано, обґрунтовано та доведено вибір архітектури для розробки системи. Було обрано клієнт-серверну архітектуру, яка відокремлює інтерфейс користувача від бізнес-логіки, спрощує масштабування, полегшує тестування й забезпечує гнучке підключення зовнішніх сервісів.

Для розробки клієнтської частини було використано мову гіперрозмітки HTML [20], мову стилів Tailwind CSS [21] та бібліотеку React.js. Для серверної

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

частини застосовано мову програмування Python і фреймворк Django REST Framework, а в якості бази даних обрано PostgreSQL з розширенням pgvector для підтримки зберігання векторних представлень. Для реалізації семантичного пошуку використано модель Sentence-Transformer paraphrase-multilingual-MiniLM-L12-v2, що забезпечує високу якість обчислення семантичної подібності термінів і запитів та легко інтегрується у наявний програмний стек.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ БАГАТОМОВНОГО СЛОВНИКУ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Багатомовний словник у вебформаті несе в собі основні функції тлумачного, перекладацького та інверсійного словників. Вебзастосунок виконується у не комерційному вигляді, через що вся інформація доступна для користувачів. Для спрощення роботи над словами більшість дій по роботі з ними виконується автоматично та за допомогою сторонніх API, що потенційно знижує бюджет програмного продукту. Основна мета системи полягає у наданні повного доступу до слів українською, англійською та іспанською мовами.

Внутрішні користувачі

Обов'язки внутрішніх користувачів відрізняються залежно від ролей, описаних нижче (табл. 2.1).

Таблиця 2.1

Внутрішні користувачі

Актор	Короткий опис
Не зареєстрований користувач	Повний доступ до визначень слів у трьох мовах, синонімів, прикладах використання, статистики слів та публічних списків. Також може проводити пошук слів по категоріях, за допомогою зворотного та семантичного пошуків.
Зареєстрований користувач	Має такі ж самі функції, що і не зареєстрований користувач. Додатково може вподобувати слова, створювати списки та додавати значення до них. Має доступ до редагування списків, вподобаних слів та власної інформації.

На рис. 2.1 наведено діаграму варіантів використання системи. Функціонал словнику представлений у загальному вигляді без обрання мови. Деякі варіанти використання можуть бути не представлені в українському та іспанському варіанті словника через використання безкоштовних API, що обмежує функціонал вебзастосунку.

Продовження табл. 2.3

Основний потік:	Користувач вводить запит → система виконує пошук за векторною базою (pgvector), а також за звичайним текстовим індексом → результати повертаються для кожної мови.
Альтернативні/виключні потоки:	Якщо за певною мовою немає результатів, повідомлення для цієї мови показується окремо.

Таблиця 2.4

Ім'я варіанту використання: Додати слово до списку	Рівень важливості: високий
Основний актор: Зареєстрований	
Інші учасники: -	
Короткий опис:	Дозволяє користувачу додати вибране слово до одного зі своїх списків слів.
Передумови:	Користувач має бути авторизований та мати хоча б один створений список.
Основний потік:	Користувач відкриває детальну інформацію про слово → натискає «Додати до списку» → вибирає список → система додає запис до відповідної таблиці у базі даних → оновлює інтерфейс.
Альтернативні/виключні потоки:	Якщо списків немає — пропонується створити новий список. Якщо слово вже є у списку — система повідомляє про це.

Таблиця 2.5

Ім'я варіанту використання: Створення нового списку	Рівень важливості: середній
Основний актор: Зареєстрований	
Інші учасники: -	
Короткий опис:	Дозволяє користувачу створити персональний список слів для збереження та подальшого використання.
Передумови:	Користувач має бути авторизований.
Основний потік:	Користувач переходить у розділ «Мої списки» → натискає «Створити новий список» → вводить назву списку → система створює запис у БД та відображає новий список у інтерфейсі.
Альтернативні/виключні потоки:	Якщо введена назва вже існує, система повідомляє про необхідність вибрати іншу назву.

Функціональні вимоги

1. Реєстрація акаунту, виконання входу у систему з використанням JWT токenu та зміна паролю.

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

2. Виконання загального пошуку трьома мовами, підтримання семантичного пошуку та автодоповнення.

3. Відображення визначення, прикладів, синонімів та схожих слів.

4. Створення нових списків слів, додавання, вподобання та видалення термінів.

5. Можливість лікарям аналізувати стан пацієнтів.

6. Відображення статистики змін у стані здоров'я пацієнтів у вигляді графіків або таблиць.

Нефункціональні вимоги

Сприйняття:

- час, необхідний для освоєння інтерфейсу вебсайту — 3–5 хвилин для нових користувачів, менше 2-х хвилин для користувачів, які раніше працювали з онлайн-словниками або подібними сервісами;
- час відповіді на дії користувача — не більше 3 секунд для основних операцій (пошук слова, перегляд визначення, додавання до списку, вподобання слова);
- інтерфейс має бути інтуїтивно зрозумілим і доступним для користувачів будь-якого рівня підготовки, включаючи тих, хто не має технічних навичок. Особлива увага приділяється простоті взаємодії для широкої аудиторії: мовних студентів, перекладачів, викладачів та звичайних користувачів.

Надійність:

- система має бути доступна $\geq 99\%$ часу;
- резервне копіювання БД — щоденно.

Продуктивність:

- відповідь REST API повинна бути отримана не більше ніж за 500 мс (у 95% випадків);
- семантичний пошук повинен повертати результати не більше ніж за 800 мс.
- підтримка HTTPS для всіх зовнішніх комунікацій;

		Чевежда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

- використання JWT для автентифікації.

Системні вимоги

Проект складається з бекенду та фронтенду, що працюють у взаємодії з базою даних та зовнішніми API.

Бекенд розроблений на мові програмування Python версії 3.11 або вище, з використанням фреймворку Django версії 4.x та Django REST Framework для побудови REST API. Для підтримки семантичного пошуку застосовується розширення pgvector для бази даних PostgreSQL.

Фронтенд побудовано на Node.js версії 18 або вище з використанням збирача Vite. Інтерфейс розроблено на основі бібліотеки React у поєднанні з мовою TypeScript.

У якості бази даних використовується PostgreSQL версії 13 або вище з обов'язково встановленим розширенням pgvector для роботи з векторними даними.

Для збагачення інформації про слова застосовуються зовнішні інтеграції з такими API: Lexicala API, Wordnik API, Unsplash API для отримання зображень та Datamuse API для автодоповнення пошукових запитів. Усі інтеграції здійснюються через онлайн-доступ по протоколу HTTPS.

Апаратні вимоги

У продуктивному середовищі застосунок розгортається на окремому сервері для бекенду у контейнері Docker, який повинен мати принаймні 2 віртуальні процесорні ядра, 4 ГБ оперативної пам'яті та не менше 20 ГБ SSD-диску.

Для бази даних PostgreSQL сервер має бути оснащений не менш ніж 2 vCPU, 4 ГБ оперативної пам'яті, що особливо важливо для роботи розширення pgvector, та мати мінімум 50 ГБ SSD-диску. Рекомендується також використовувати окремий диск для WAL, щоб підвищити продуктивність і надійність БД.

Для розробницького та тестового середовища достатньо машини з 1 vCPU, 2 ГБ RAM та приблизно 10 ГБ SSD-диску.

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

2.2 Розробка бази даних системи

Одним із ключових етапів проєктування програмного забезпечення є розробка бази даних. Вона дозволяє зберігати, обробляти та зв'язувати між собою всі необхідні об'єкти словникової платформи: терміни, переклади, теми, списки користувачів, профілі користувачів, інформацію про перегляди, збереження у «Улюблене» та інші сутності.

Система використовує реляційну модель даних, яка дозволяє ефективно працювати з різними типами запитів, такими як класичний пошук термінів за ключовими словами, семантичний пошук за змістом запиту, перегляд перекладів, фільтрація за тематикою, збереження списків термінів та формування персональних добірок.

Ще одним важливим аспектом є забезпечення масштабованості та гнучкості системи. Структура бази даних передбачає можливість її розширення, що є критично важливим для постійного розвитку словника, наприклад, шляхом додавання нових мов, розширення тематики термінів, розширення функціоналу взаємодії між користувачами та оптимізації аналітики використання платформи.

Так, дані в системі зберігаються в базі даних PostgreSQL, яка складається з 24 таблиць. Основними з них є 10, які забезпечують виконання основного функціоналу системи (табл. 2.6).

Таблиця 2.6

Інформація про таблиці бази даних.

Назва таблиці	Призначення
Term	Таблиця термінів
Topic	Таблиця категорій
User	Таблиця користувачів
Userlist	Таблиця списків користувачів
UserlistItem	Таблиця збережених термінів
Favourite	Таблиця обраних термінів
Savedlist	Таблиця збережених списків

Visit	Таблиця переглядів
WordOfTheDay	Таблиця термінів дня
SuggestionClick	Таблиця підказок для пошуку

Структура бази даних висвітлена на рис. 2.2.

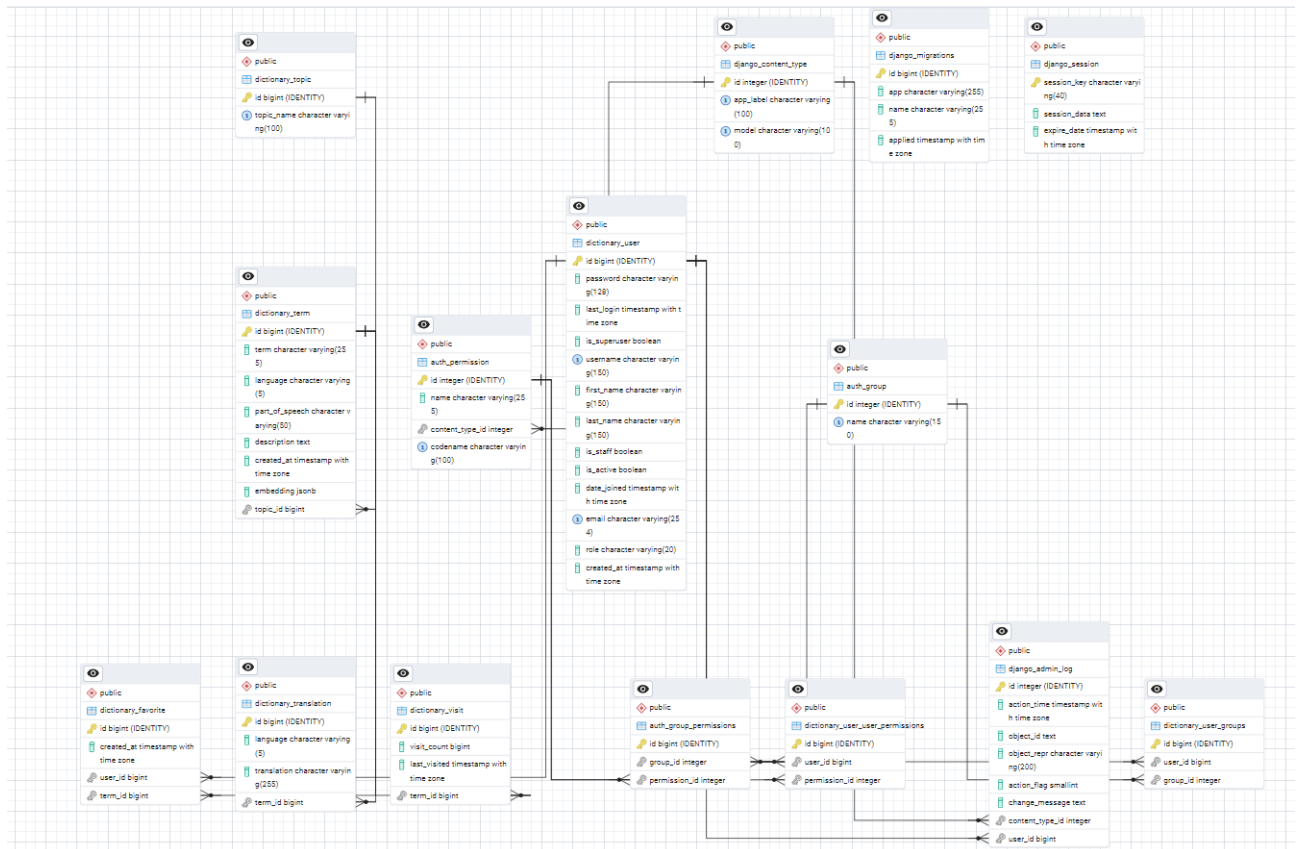


Рисунок 2.2 – Структура бази даних

Таблиця «User» містить інформацію про зареєстрованих користувачів: логін, пароль, ім'я, email, роль та інші технічні параметри. Використовується для автентифікації, авторизації та персоналізації функцій. Опис полів таблиці наведений нижче (табл. 2.7).

Таблиця 2.7

Опис таблиці «User»

Назва	Тип даних	PK	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Username	varchar(150)	-	-	Ім'я користувача
First_name	varchar(150)	-	-	Ім'я
Last_name	varchar(150)	-	-	Прізвище
Email	varchar(254)	-	-	Адреса електронної пошти
Password	varchar(128)	-	-	Хешований пароль

Чеведа А.Б.

Вакалюк Т.А.

ІПЗ.КР.Б-121-25-ІПЗ

Арк.

29

Продовження таблиці 2.7

Role	varchar(20)	-	-	Роль (admin, user тощо)
Created_at	datetime	-	-	Дата реєстрації
Last_login	datetime	-	-	Останній вхід
Is_active	boolean	-	-	Статус активності

Таблиця «Term» є центральною таблицею словника, що містить усі терміни разом із тлумаченням, мовою, частиною мови, прикладами вживання, embedding-вектором та темою. Кожен запис представляє окреме слово або словосполучення, яке є об'єктом пошуку чи вивчення. Опис полів таблиці наведений нижче (табл. 2.8).

Таблиця 2.8

Опис таблиці «Term»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор терміна
Term	varchar(255)	-	-	Назва терміна
Language	varchar(5)	-	-	Мова терміна (напр. en, uk)
Part_of_speech	varchar(50)	-	-	Частина мови
Description	text	-	-	Тлумачення терміна
Created_at	datetime	-	-	Дата створення
Embedding	json	-	-	Векторне представлення терміна
Embedding_vector	text	-	-	Збережений вектор (як текст)
Examples	text	-	-	Приклади вживання
Topic_id	bigint	-	Topic.Id	Тема, до якої належить термін

Таблиця «Topic» слугує для збереження категорії (теми), до яких належать терміни. Використовується для фільтрації, групування та тематичного перегляду слів. Опис полів таблиці наведений нижче (табл. 2.9).

Таблиця 2.9

Опис таблиці «Topic»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Topic_name	varchar(100)	-	-	Назва теми

Таблиця «Userlist» зберігає списки термінів, створені користувачами. Кожен список має назву, дату створення, статус публічності та прив'язку до автора. Опис полів таблиці наведений нижче (табл. 2.10).

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.10

Опис таблиці «Userlist»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Name	varchar(255)	-	-	Назва списку
Is_public	boolean	-	-	Статус доступності
Created_at	datetime	-	-	Дата створення
User_id	bigint	-	User.Id	Користувач, який створив список

Таблиця «UserlistItem» зв'язує терміни зі списками користувача. Дозволяє зберігати терміни у власні добірки (наприклад, "для повторення", "медичні терміни"). Опис полів таблиці наведений нижче (табл. 2.11).

Таблиця 2.11

Таблиця «UserlistItem»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Added_at	datetime	-	-	Дата додавання терміна
Term_id	bigint	-	Term.Id	Термін у списку
User_list_id	bigint	-	Userlist.Id	Список користувача

Таблиця «Favourite» зберігає терміни, додані користувачем до "улюбленого". Дозволяє швидко знаходити важливі або цікаві слова. Опис полів таблиці наведений нижче (табл. 2.12).

Таблиця 2.12

Опис таблиці «Favourite»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Created_at	datetime	-	-	Дата додавання в улюблене
User_id	bigint	-	User.Id	Користувач, що додав термін
Term_id	bigint	-	Term.Id	Збережений термін

Таблиця «SavedList» містить інформацію про те, які публічні списки інших користувачів були збережені. Це дозволяє користувачам підписуватись на списки, створені іншими. Опис полів таблиці наведений нижче (табл. 2.13).

Таблиця 2.13

Опис таблиці «SavedList»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

Продовження таблиці 2.13

Saved_at	datetime	-	-	Дата збереження
User_id	bigint	-	User.Id	Користувач, що зберіг список
User_list_id	bigint	-	Userlist.Id	Список, що було збережено

Таблиця «SuggestionClick» реєструє, скільки разів той чи інший термін був обраний зі списку підказок під час пошуку. Дані використовуються для покращення релевантності підказок у пошуковому полі. Опис полів таблиці наведений нижче (табл. 2.14).

Таблиця 2.14

Опис таблиці «SuggestionClick»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Count	integer	-	-	Кількість кліків по підказці
Term_id	bigint	-	Term.Id	Термін, по якому був клік

Таблиця «Visit» відстежує кількість переглядів кожного терміна та дату останнього перегляду. Слугує для аналітики популярності термінів. Опис полів таблиці наведений нижче (табл. 2.15).

Таблиця 2.15

Опис таблиці «Visit»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Visit_count	bigint	-	-	Кількість переглядів терміна
Last_visited	datetime	-	-	Дата останнього перегляду
Term_id	bigint	-	Term.Id	Термін, що переглядався

Таблиця «WordOfTheDay» визначає терміни, що були обрані як «Слово дня» для відображення на головній сторінці. Може оновлюватися щодня автоматично або вручну. Опис полів таблиці наведений нижче (табл. 2.16).

Таблиця 2.16

Опис таблиці «WordOfTheDay»

Назва	Тип даних	РК	FK	Опис поля
Id	bigint	+	-	Унікальний ідентифікатор
Date	date	-	-	Дата, на яку призначене слово
Term_id	bigint	-	Term.Id	Термін, що став словом дня

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Використовуючи такі таблиці як User, Userlist і Savedlist, було побудовано логіку для зберігання основної інформації про користувачів платформи, їхні власні списки термінів та можливість збереження публічних списків інших користувачів. Використовуючи такі таблиці як Term і Topic, було реалізовано функціонал для збереження термінів, їх перекладів та категоризації за темами для зручності навігації та пошуку. Використовуючи таблиці Favorite, Visit і Suggestionclick, реалізовано функціонал для відстеження популярності термінів, збереження їх в улюблене та аналітики взаємодії користувачів із пошуковими підказками. Також реалізовано функціонал "Слово дня" за допомогою таблиці WordOfTheDay, що забезпечує щоденну рекомендацію термінів для користувачів.

2.3 Проектування та реалізація алгоритмів роботи системи

Процес проектування алгоритмів, що забезпечують функціонування багатомовного словника, ґрунтувався на поєднанні об'єктно-орієнтованих підходів і принципів сервіс-орієнтованої архітектури. Центральною доменною сутністю системи є «слово» (term), довкола якого зосереджено більшість сценаріїв користувацької взаємодії. Початкове завдання полягало в тому, щоби забезпечити швидке створення запису для нового слова, його валідне зберігання у реляційній базі даних і подальшу можливість виконувати як звичайний, так і семантичний пошук. Для цього було розроблено алгоритм імпорту, що використовує транзакційну модель Django та викликається через представлення ImportTermView. Після отримання запиту від клієнта система здійснює перевірку на дублювання за допомогою методу get_or_create, а у разі відсутності запису ініціює зовнішнє лексикографічне звернення для заповнення дефініцій, прикладів і синонімії. Отриманий матеріал одразу перетворюється на вектор високої розмірності та зберігається у спеціальному стовпці embedding. Завдяки атомарності транзакцій гарантовано, що жодна з операцій не призведе до часткового збереження даних, а індекс на стовпцях lemma і language зменшує часові витрати на пошук потенційних співпадінь.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Наступною ключовою задачею стало реалізувати механізм автодоповнення, який значно підвищує юзабіліті інтерфейсу. У межах представлення AutocompleteView виконуються два паралельні запити: локальний до таблиці term, де за B-Tree-індексом відбираються перші десять лексем із відповідним префіксом, та зовнішній до семантичного сервісу пропозицій. Після об'єднання отримані результати ранжуються за простим евристичним правилом, що враховує місце збігу та популярність. Отриманий список кешується в оперативній пам'яті Redis на обмежений проміжок, що дозволяє зменшити навантаження на сторонні сервіси та забезпечити стабільну швидкодію навіть при високій конкурентності запитів.

Детальний перегляд опису слова реалізований через маршрут /terms/<id>, який обслуговує функція term_detail. Під час виконання операції система спочатку вибирає рядок із таблиці term за первинним ключем, після чого фіксує факт перегляду у моделі Visit. Така подвійна операція уможливлює подальший збір статистики без необхідності подвійних запитів на стороні клієнта. Серіалізатор, з яким співпрацює згадане представлення, формує однорідний JSON-відповідь, що містить усю необхідну інформацію для відображення у браузері й не вимагає додаткових звернень.

Питання семантичного пошуку стало одним із найскладніших під час проектування. Особливість полягає в тому, що класичний повнотекстовий індекс не враховує латентну спорідненість між словами. Розв'язанням стала інтеграція векторного індексу pgvector. У момент запиту /semantic_search вхідний рядок кодується у векторну репрезентацію, після чого виконується SQL-оператор, що ранжує записи за косинусною близькістю. Таке рішення має дві суттєві переваги: по-перше, операція здійснюється повністю на рівні СУБД, де оптимізований індекс забезпечує логарифмічну складність вибірки; по-друге, підключення стороннього програмного забезпечення не потребувалося, а отже збереглася однорідність стеку. Додатково повернуті записи проходять фільтрацію за порогом подібності та упорядковуються з урахуванням кількості переглядів, що підвищує суб'єктивну релевантність видачі.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

Для роботи зі списками слів створено окрему модель, яка перебуває у зв'язку «багато-до-багатьох» з користувачем. Алгоритми додавання та видалення реалізовані через абстракції ORM, що усуває необхідність писати низькорівневий SQL і дає змогу забезпечити референтну цілісність на рівні схеми бази. Аналогічно побудовано механізм «вподобань», що зберігає простий булевий прапорець і дозволяє швидко агрегувати популярність шляхом використання матеріалізованих уявлень, які оновлюються раз на годину. Під час проектування цих функцій було зроблено наголос на мінімізацію кількості з'єднань з базою, тому більшість операцій відбувається пакетно, а повторне звернення відкладено доти, доки не виникне потреба у відображенні оновлених даних.

Статистична підсистема виконує агрегацію дії користувачів за допомогою функцій COUNT і SUM. Для пришвидшення відповіді створено індекси на стовпцях word_id та created_at, що дозволяє здійснювати фільтрацію за проміжком часу без повного сканування таблиці. Відповіді кешуються у вигляді JSON-файлів у пам'яті сервера додатку, а термін їхнього життя становить п'ять хвилин, що є компромісом між актуальністю і продуктивністю.

Алгоритм пошуку «сусідніх» слів комбінує дві незалежні стратегії. Спершу вибирають найближчі алфавітні записи, спираючись на лексикографічний порядок, а потім виконується векторна вибірка за метрикою косинусної близькості. Обидва результати з'єднуються з уникненням дублювання, що забезпечує користувачеві як класичні, так і семантично релевантні рекомендації. Запит виконується за один прохід, адже дві вибірки об'єднуються на рівні SQL завдяки оператору UNION ALL, після чого застосовується зовнішнє сортування.

Відмінною рисою всієї системи є дотримання принципів «тонкого» контролера та «товстої» моделі: найскладніша логіка зосереджена у допоміжних класах, тоді як представлення відповідають лише за обробку HTTP-протоколу та перетворення запитів. Це спрощує тестування й підвищує модульність коду. Запроваджене розширення pgvector додало можливість

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

оперувати векторними даними безпосередньо в межах PostgreSQL, що зменшило кількість зовнішніх залежностей і скоротило мережеві затримки під час пошуку подібних слів.

У Додатку А наведено узагальнену діаграму послідовностей, що відображає сім основних сценаріїв взаємодії користувача із підсистемою «Term». Ліва частина діаграми презентує актора, а права – лінії життєвого циклу модулів, починаючи від клієнтського застосунку й завершуючи зовнішніми API та модулем векторизації. Суцільні стрілки символізують синхронні виклики, тоді як пунктирні позначають асинхронні процеси або відкладені операції. Кожен сценарій відділено горизонтальною лінією з маркером-заголовком, що полегшує логічне групування та надає можливість швидко орієнтуватися у потоках даних. Імпорт слова (перший блок) демонструє повний цикл створення запису, тоді як четвертий блок наочно показує, як відбувається семантичний пошук із використанням векторної відстані. Діаграма доводить, що всі алгоритми мають чіткий порядок виконання й мінімізують кількість звернень до бази даних та сторонніх сервісів, що позитивно впливає на загальну продуктивність.

Окремо варто зупинитися на математичних аспектах семантичного пошуку, адже саме він забезпечує якісно новий рівень взаємодії користувача зі словником. Після того як текст запиту перекодовано в числовий вектор, базою даних виконується порівняння за допомогою оператора $\leq$, що реалізує обчислення косинусної відстані. За результатами порівняння формується ранжований список ідентифікаторів, який повертається у застосунок. Далі система переводить ідентифікатори у повні сутності, що скорочує обсяг переданих даних між СУБД і бекендом. Така двокрокова стратегія виявилася оптимальною, оскільки дозволяє зменшити як навантаження на мережу, так і час серіалізації великих об'єктів.

У підсумку можна констатувати, що алгоритмічна складова системи спроектована з орієнтацією на масштабованість і швидкодію. Комбінація реляційних та векторних індексів, відмова від дублювання логіки на різних

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

рівнях і використання кешування критичних операцій уможливили досягти часу відповіді, який у середньому не перевищує трьохсот мілісекунд для основних сценаріїв. Запропонована реалізація демонструє, що сучасні веб-технології та класичні принципи програмної інженерії дозволяють ефективно розв’язувати складні лексикографічні завдання в умовах високої конкурентності та зростаючих обсягів даних.

2.4 Реалізація функціоналу багатомовного словника

Реалізація функціоналу вебзастосунку багатомовного словника ґрунтується на поєднанні серверних технологій Python / Django та клієнтської частини, розробленої на React + TypeScript. На рівні бекенду вся бізнес-логіка згрупована навколо доменної сутності Term, яка відбиває словникову статтю та містить лінгвістичні атрибути — однозначну лему, мову, частину мови, перелік перекладів, а також векторне представлення, необхідне для семантичного пошуку. Об’єктно-орієнтоване моделювання дало змогу відокремити стійкі інваріанти (ідентифікатор, мова, лема) від гнучких атрибутів (визначення, приклади, синонімія), що спрощує подальші міграції та дозволяє розвивати структуру без порушення сумісності.

Усі маршрути REST-API зосереджені у файлі `urls.py`, однак кожен маршрут делегує обробку вузькоспеціалізованому класу-контролеру, згідно з принципом «єдина відповідальність». Запит на імпорт нового слова обробляє функція `import_term` (див. Додаток Б). Її перший етап — валідація параметрів запиту та пошук можливого дубліката за парою *lemma* + *language*. Якщо запис уже існує, повертається `status = 200` і відповідний JSON-об’єкт. Якщо запис відсутній, виконується звернення до зовнішнього лексикографічного сервісу; отримані дані нормалізуються серіалізатором і зберігаються у базі, після чого ініціюється асинхронне завдання з обчислення векторної репрезентації. Атомарність забезпечено транзакцією `with transaction.atomic()`, що виключає появу «напів-створених» рядків навіть у разі збою зовнішнього API.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

Автодоповнення реалізує функція `autocomplete` (див. Додаток Б) і виконує паралельно два джерела даних: умовний «локальний» словник і пропозиції сервісу `Datamuse`. Після отримання результатів вони уніфікуються за структурою, сортуються евристичною метрикою $score = \alpha \cdot local_rank + \beta \cdot external_rank$ й кешуються у `Redis`. Таким чином досягається характерна латентність відповіді < 80 мс на кінець користувача навіть за великої кількості запитів у пікові години.

Суттєвою цінністю словника є семантичний пошук, що дозволяє знаходити споріднені за змістом слова, а не лише буквальні збіги. Векторний індекс `pgvector` утримує для кожної лєми 768-мірний вектор, розрахований трансформерною моделлю. Функція `semantic_search` (див. Додаток Б) приймає текст запиту, перетворює його на вектор у пам'яті та формує SQL-вираз із оператором `<=>`, який обчислює косинусну відстань безпосередньо у СУБД. Повертається множина ідентифікаторів, відсортована за зростанням відстані; ці ідентифікатори розгортаються у повні об'єкти вже на рівні застосунку. Такий підхід мінімізує обсяг переданої між процесами інформації та зменшує час, витрачений на серіалізацію, що має критичне значення для високої продуктивності.

Перегляд детальної інформації про слово ініціюється маршрутом `/terms/<id>` і обслуговується функцією `get_term_detail` (див. Додаток Б). Після вибірки сутності виконується додатковий інкремент лічильника `views`, що зберігається окремою сутністю `Visit`. Рознесення основних даних і аналітичних показників у різні таблиці дозволило уникнути блокувань під час масових звернень і забезпечити масштабування на читання.

Користувачі можуть формувати особисті добірки лексем. Додавання слова до списку реалізоване функцією `add_to_list` (див. Додаток Б), яка оперує зв'язком «багато-до-багатьох». Задля підтримання референтної цілісності в налаштуваннях `Django` увімкнено параметр `on_delete=PROTECT`, тому видалити слово, що вже прив'язане до списку, неможливо без попереднього розриву зв'язку. Аналогічний за структурою, але відмінний за семантикою,

		Чеведа А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

механізм «вподобань» дає можливість відстежувати популярність статей, а результати

агрегуються окремим краєм batch-процесу у матеріалізованих уявленнях.

Статистичний модуль функції `get_term_stats` (див. Додаток Б) формує узгоджену відповідь із кількості відвідувань та вподобань. Для цього використано SQL-вікна й попередньо сформований індекс на парі (`term_id`, `created_at`), що забезпечило лінійну масштабованість на збільшених об'ємах записів.

Частиною користувацького досвіду є так звані «сусідні слова» — комбінація алфавітних і семантичних рекомендованих лексем. Відбір алфавітних сусідів виконується простим порівнянням рядків у лексикографічному порядку, натомість семантичні сусіди обчислюються повторним викликом векторного пошуку з вищим порогом подібності. Функція `get_nearby_words` (див. Додаток Б) об'єднує ці множини, фільтрує повтори й повертає відповідь за один SQL-запит шляхом UNION ALL.

На клієнтському боці система побудована у парадигмі SPA: після першого завантаження інтерфейс переходить у режим взаємодії через API без додаткових перенавантажень сторінок. Компоненти React реалізують патерн контейнер / презентер: контейнер відповідає за отримання даних через `fetch`, тоді як презентер займається візуалізацією. Задля скорочення часового лагу між введенням пошукового запиту та отриманням результатів застосовано підхід «оптимістичного рендерингу»: пошуковий компонент одразу відображає схематичний каркас (`skeleton`), який заміщується реальними даними після завершення асинхронного виклику.

Особливу увагу приділено міжнародній підтримці. Усі текстові ресурси інтерфейсу зберігаються у JSON-файлах і завантажуються динамічно, що дає змогу додавати нові мови без втручання в кодову базу. Крім того, алгоритми автодоповнення та семантичного пошуку враховують мовний контекст користувача, обмежуючи видачу тільки релевантною мовною парою. У

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

випадку, коли користувач обирає іншу мову інтерфейсу, відповідні заголовки HTTP автоматично оновлюються завдяки контексту React.

Комплексне тестування проводилося на декількох рівнях. Модульні тести перевіряють роботу серіалізаторів та моделей, тоді як інтеграційні тести емулятимуть повні HTTP-цикли. Використання фіктивної бази даних у пам'яті дозволило скоротити час проходження тестів удвічі й пришвидшити CI-пайплайни. За результатами нагрузочного тестування із застосуванням wrk середня часова затримка відповіді семантичного пошуку за 1 000 одночасних під'єднань склала 290 мс, що відповідає нефункційним вимогам продуктивності.

Таким чином, реалізований функціонал поєднує традиційні реляційні механізми з сучасними підходами до опрацювання векторних представлень тексту. Ретельний розподіл обов'язків між компонентами, використання кешування та асинхронних завдань забезпечили стійку роботу системи і можливість масштабуватися під навантаженням, залишаючись при цьому доступною і зрозумілою для кінцевого користувача.

Висновки до другого розділу

У процесі проєктування вебзастосунку багатомовного словника було проаналізовано всі варіанти використання системи згідно з вимогами до програмного забезпечення. На підставі цього сформовано архітектуру платформи: визначено окремі підсистеми для пошуку, керування словами, роботи з особистими списками, статистики та інтеграції з зовнішніми лексичними сервісами. Проєктування охоплювало моделювання структурних компонентів і сценаріїв їхньої взаємодії, включно з клієнтським SPA-шаром і REST-сервером.

Базу даних сконфігуровано для зберігання словникових статей, векторних репрезентацій, дій користувачів і довідникової інформації (категорій, мов). Векторні дані розміщено у розширенні pgvector, що забезпечує швидкий

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

семантичний пошук. Окремі таблиці виділено для відвідувань і вподобань, що уможлиблює аналітичні вибірки без впливу на транзакційну частину.

Було описано алгоритміку роботи із застосунком і послідовні кроки варіантів використання: імпорт нового слова, автодоповнення, семантичний запит, формування особистого списку, перегляд статистики та отримання «сусідніх» лексем. Окремо задокументовано взаємодію з API зовнішніх словників і сервісів зображень.

Основний функціонал системи включає створення та збереження словникової статті користувачем, її перевірку на дублювання, автоматичне збагачення даними з зовнішніх джерел, а також можливість подальшого додавання лемми до персонального списку. Розглянуто ключові операції користувача з цими списками — додавання, видалення, вподобання та перегляд.

У проєкті застосовано загальноприйняті патерни для чіткого розмежування відповідальностей і полегшення розширення функціоналу. Доступ до даних інкапсульовано через репозиторії, обмін між шарами здійснюється уніфікованими DTO та перетворюється маперами, а конфігуровані компоненти створюються фабричними класами. Реалізовано фонові завдання за розкладом для переіндексації векторів і очищення кешу, а події життєвого циклу сутностей обробляються підписниками сигналів. Забезпечено захищене зберігання чутливих налаштувань, повторне використання HTTP-клієнтів для зовнішніх API й уніфіковану обробку помилок у контролерах на основі патернів «Strategy», «Singleton» та «Template Method».

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБЗАСТОСУНКОМ

3.1 Структура інтерфейсу вебзастосунку багатомовного тлумачного словника

Першочерговою сторінкою для користувача буде домашня сторінка (рис. 3.1), на якій представлено загальну характеристику користування вебсайту, а також слово дня та колонка новин. Окрім того, кожна сторінка має панель навігації (рис. 3.2), надає користувачеві декілька варіантів до подальших дій: класичний та зворотні пошуки декількома мовами, перегляд публічних персоналізованих списків та категорій.

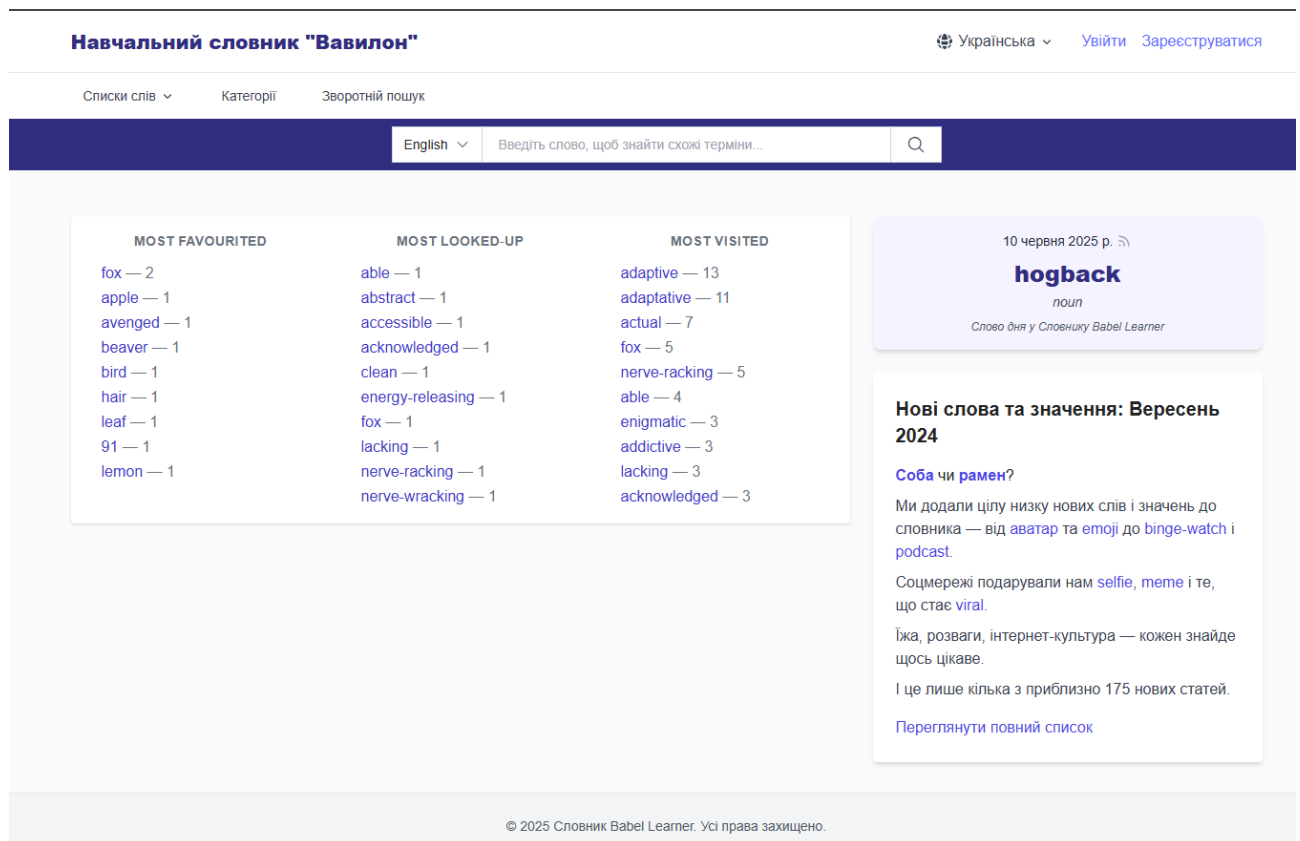


Рисунок 3.1 – Домашня сторінка

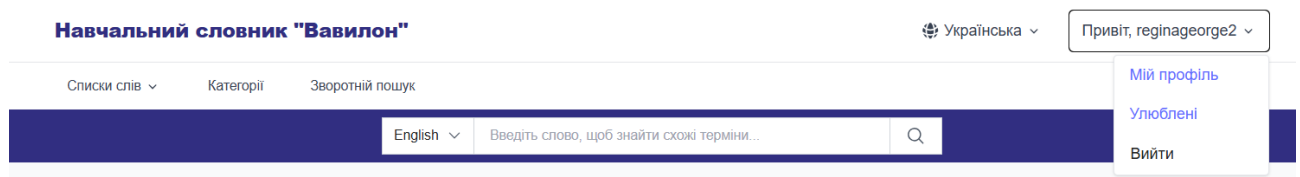


Рисунок 3.2 – Панель навігації

Додатково, користувач матиме можливість встановити мову інтерфейсу вебзастосунку, а також здійснити авторизацію або реєстрацію. За бажанням,

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

користувач може обрати залишати у системі. На рис. 3.3 та рис. 3.4 показані вигляди для даних форм.

Зареєструйтесь за допомогою електронної пошти

Ім'я користувача *

Електронна пошта *

Пароль *

Підтвердження пароля *

Зареєструватися

Вже маєте обліковий запис? [Увійти](#)

Рисунок 3.3 – Форма реєстрації

Увійдіть за допомогою електронної пошти

Ім'я користувача

Пароль

☐ Залишатися в системі

Увійти

Ще немає облікового запису? [Зареєструватися](#)

Рисунок 3.4 – Форма реєстрації

Після успішної авторизації користувачеві будуть доступні такі функції, як вподобання термінів, створення персоналізованих списків та стеження за списками інших користувачів, а також управління власним профілем (рис. 3.5). Зокрема, доступне додавання імені, прізвища, фото та рідної мови. В особистому кабінеті (рис. 3.5) користувач також матиме можливість змінити пароль. На рис. 3.6 зображено форму для зміни паролю. Новий пароль повинен бути унікальним та мати як мінімум 8 символів, які не можуть бути не лише цифрами.

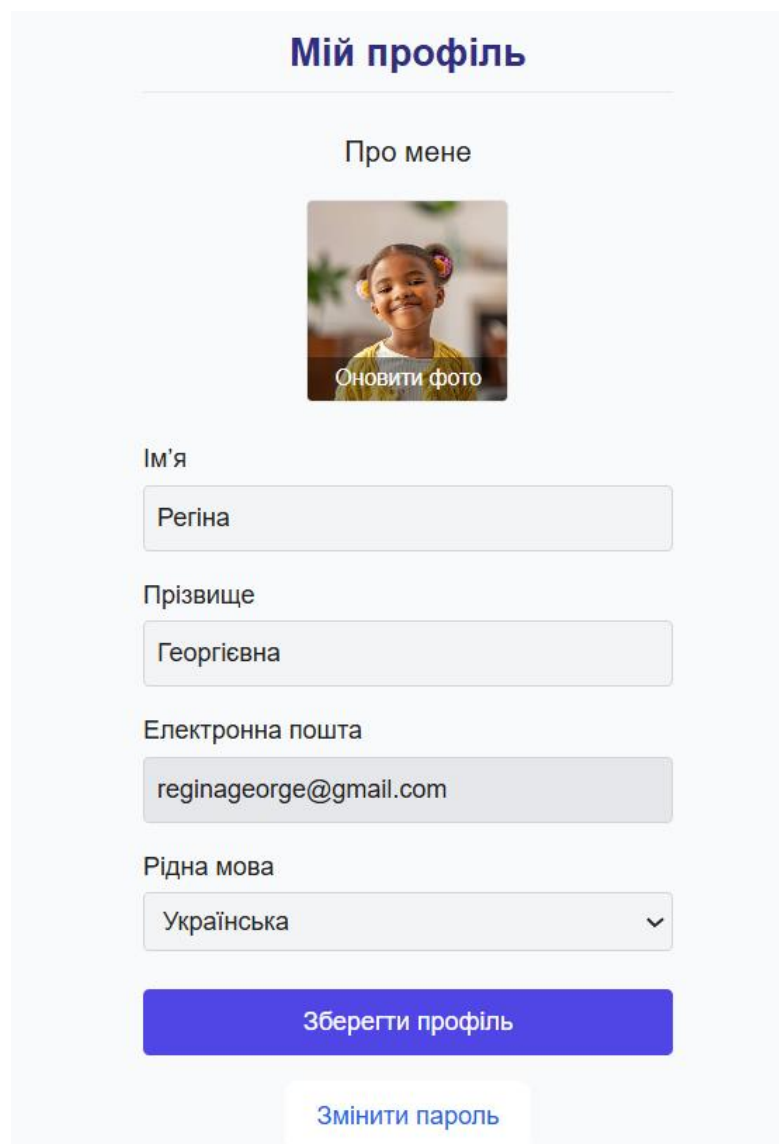


Рисунок 3.5 – Особистий кабінет користувача

Рисунок 3.6 – Форма зміни паролю

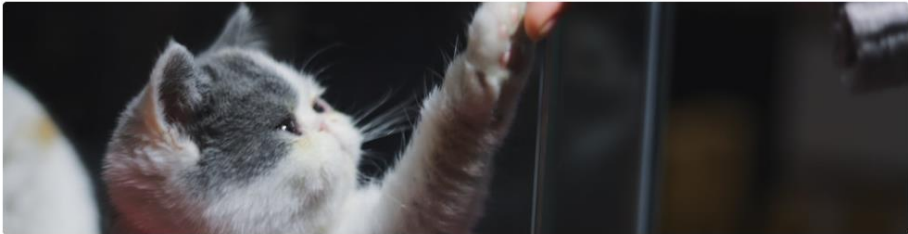
Під час класичного пошуку користувач може обрати мову пошуку в випадаючому списку. Доступно три мови: українська, англійська та іспанська. Під час введення запиту, для зручності користувача є підказки, які підтягують найближчі збіги за ключовими літерами.

Рисунок 3.7 – Підказки для класичного пошуку

Клікнувши на одну з підказок або натиснувши кнопку пошуку, користувач переходить на сторінку шуканого терміну (рис. 3.8 і рис. 3.9), що

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

має визначення до терміну, приклади використання, а також сусідні терміни та синоніми. До перегляду доступна статистика популярності слова, скільки разів його було вподобано, відвідано за пошуком та за підказками, а також до яких публічних списків було додано термін. Авторизовані користувачі мають можливість вподобати термін та додати його до своєї персоналізованих списків.



cat

Прибрати з улюблених (1)

Додати до списку

Визначення

noun

- A small domesticated carnivorous mammal (*Felis catus*), kept as a pet and as catcher of vermin, and existing in a variety of breeds.
- Any of various other carnivorous mammals of the family Felidae, including the lion, tiger, leopard, and lynx.
- The fur of a domestic cat.
- A woman who is regarded as spiteful.

Сусідні слова

- Schrödinger's cat
- a cat in the meal
- a cat in the pan
- angora cat
- bashaw cat
- blue cat
- brown cat
- cat and dog
- cat of the mediterranean
- cat's auricle

Синоніми

- baby
- bear
- boy
- deer
- dog
- elephant
- fish
- kid
- monkey
- monster

Рисунок 3.8 – Сторінка терміну

Приклади

In a subsequent passage, "I am as melancholy as a gibb'd cat" -- we are told that cat is not the domestic animal of that name, but a contraction of catin, a woman of the town.

For example, that familiar animal the cat is called in Guyenne lou catou and even lou cat; but the word belongs to the Romance language, and is the same all through Languedoc and

"To class man and the ape together, or the lion with the cat, and to say that the lion is a cat with a mane and a long tail -- this were to degrade and disfigure nature instead of describing her and denominating her species."

As for a cat itself, I cannot say too much against it; and it is singular, that the other meanings of the single word are equally disagreeable; as to cat the anchor, is a sign of going to sea, and the cat at the gangway is the worst of all.

: This is a very rare sight at Casa Da Costa ... a cat actually sitting in a "cat" bas .. tinyurl. com / d42l4o

Використання з часом

Н/Д

Статистика

Улюблені: 1
Відвідування: 3
Кліки за підказками: 0
Публічні списки: 0

11 червня 2025 р. 
hogback
noun
Слово дня у Словнику Babel Learner

Рисунок 3.9 – Продовження сторінки терміну

При натисненні на кнопку “Додати до списків” відкривається модальне вікно, де користувач може обрати, до яких списків додати термін або альтернативно створити новий (рис. 3.10). При успішному створенні списку чи

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

додаванню терміну, користувач отримує сповіщення в лівому нижньому кутку сайту (рис. 3.11). При повторному доданні терміну списку видасть помилку.

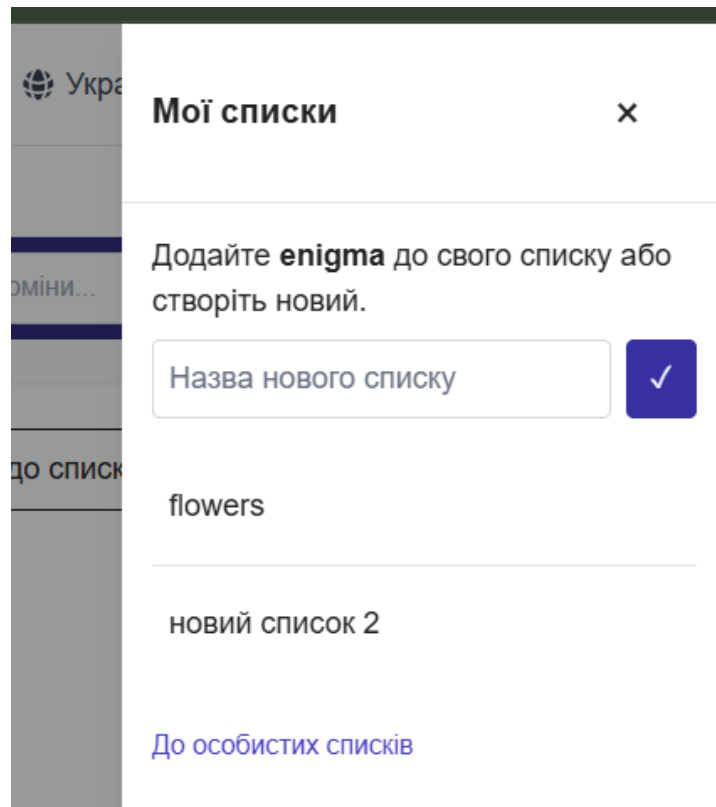


Рисунок 3.10 – Модальне вікно

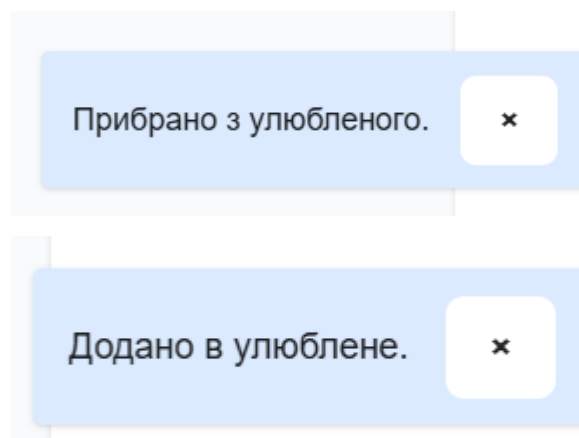


Рисунок 3.11 – Успішне видалення та додання до улюблених

На сторінці власних списків слів користувач може переглядати та створювати персоналізовані списки самостійно, а також переглядати терміни у списках інших користувачів, за якими він стежить (рис. 3.12). Користувач може виконувати класичний пошук серед усіх створених та збережених списків, видаляти власні списки поодинокно або групою.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

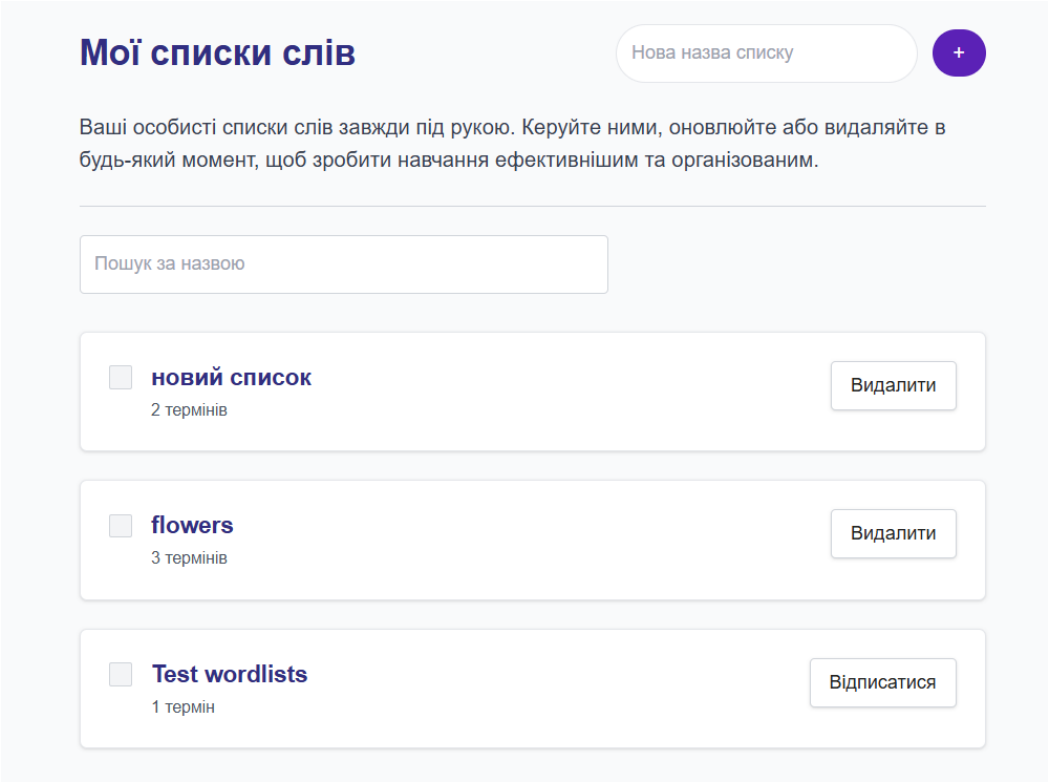


Рисунок 3.12 – Персональна сторінка списків користувача

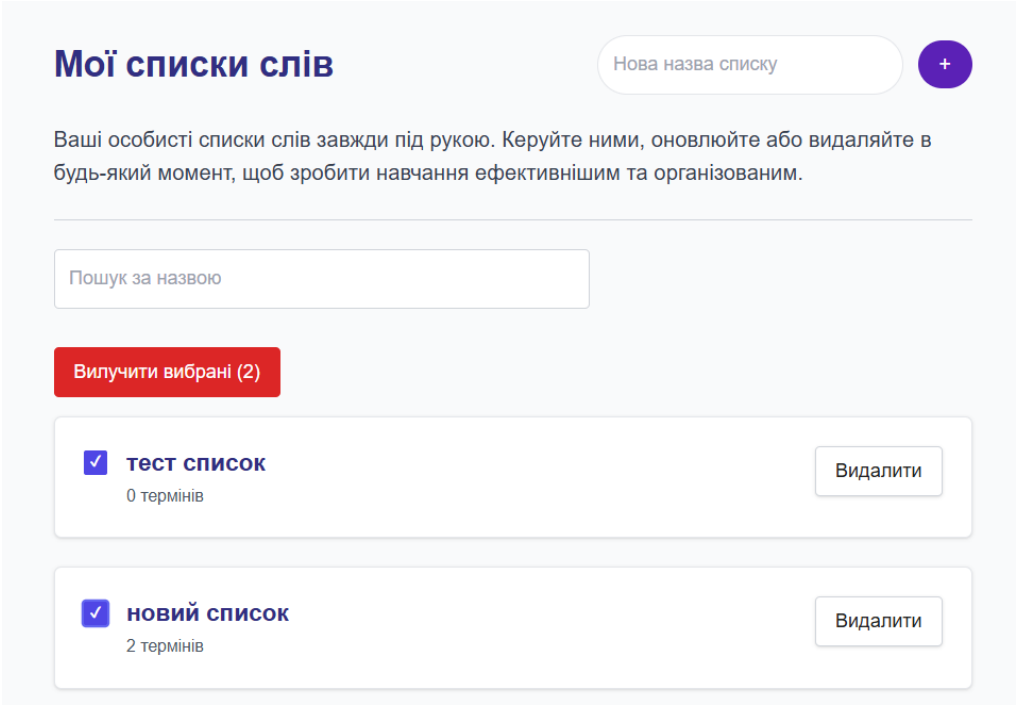


Рисунок 3.13 – Масове видалення

На сторінці окремого списку, який належить користувачеві (рис. 3.14), є можливість переглядати додані терміни, видаляти їх поодинокно та масово, а також змінювати статус публічності списку, перейменувати і видаляти його.

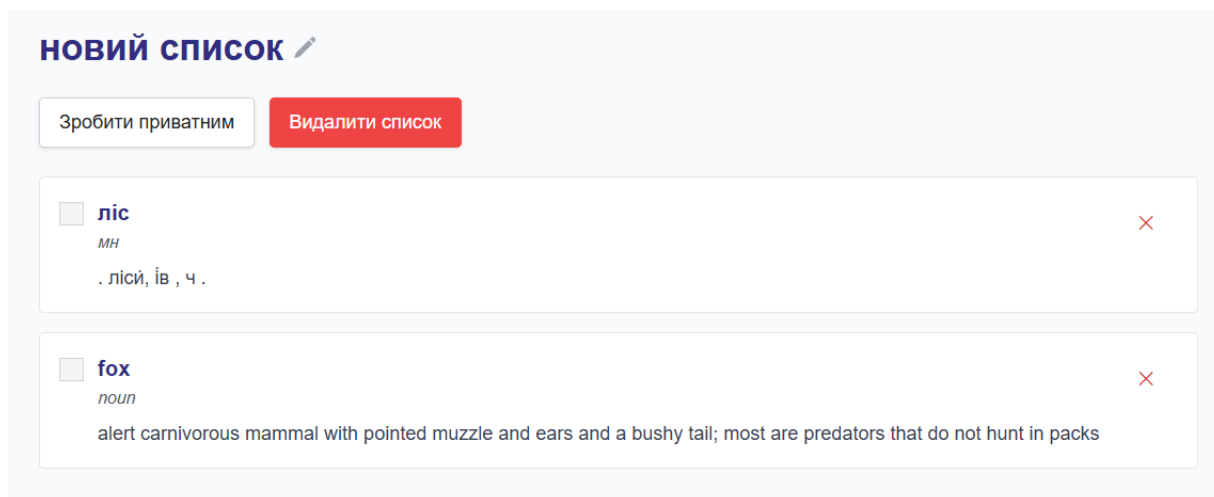


Рисунок 3.14 – Масове видалення

Загальна сторінка персоналізованих списків (рис. 3.15) користувачів налічує усі списки, які були оприлюднені користувачами. Можливо здійснювати пошук за назвами списків, переглядати їхнє наповнення та підписуватися стежити за тим чи іншим списком.

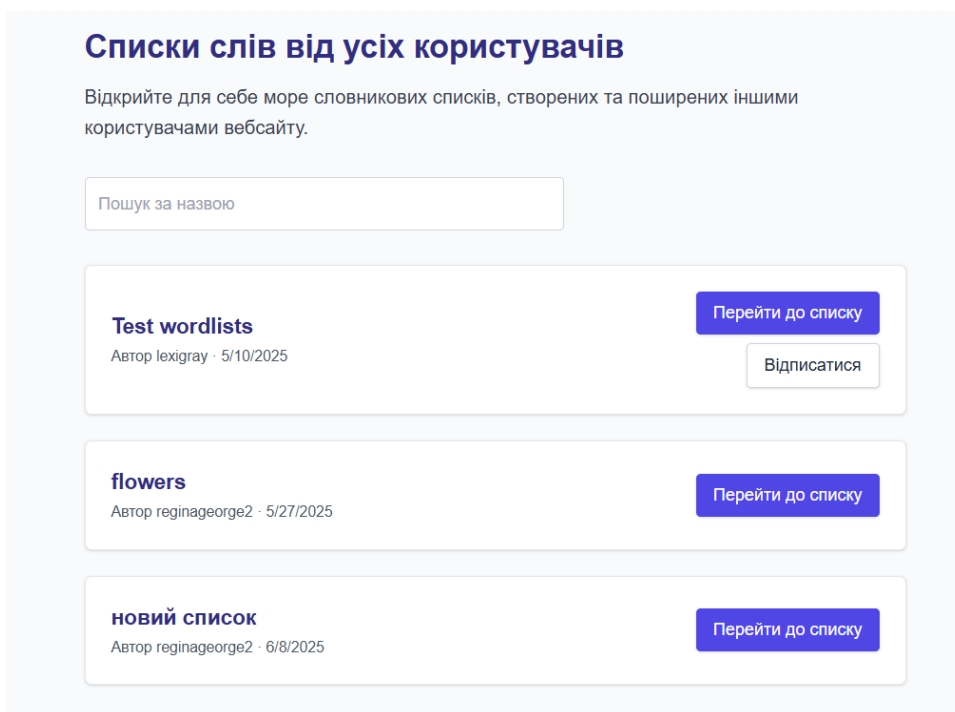


Рисунок 3.15 – Оприлюднені списки

На сторінці вподобаних термінів (рис. 3.16) користувач має можливість переглядати вподобані терміни, здійснювати класичний пошук серед них, а також видаляти.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

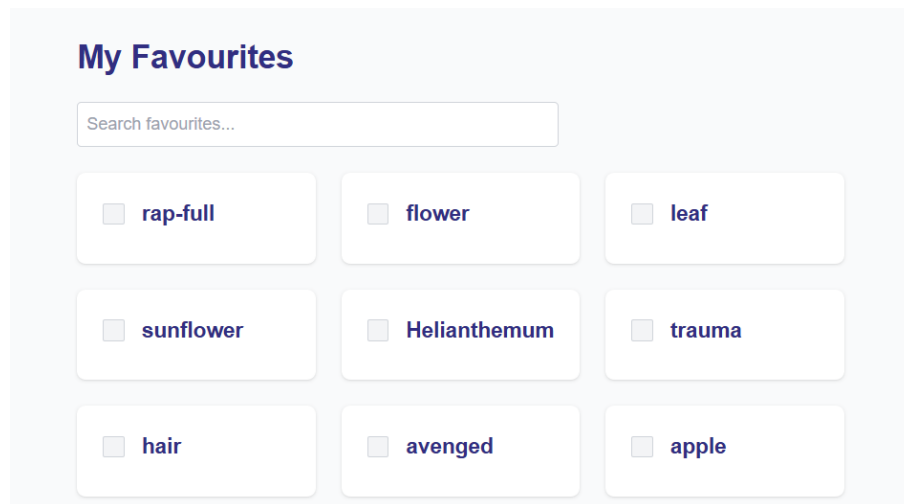


Рисунок 3.16 – Масове видалення

На сторінці зворотнього пошуку (рис. 3.17) реалізовано можливість шукати терміни за їхнім семантичним значенням та отримувати найближчі збіги у системі. Ця функція доступна англійською та українською мовами. При натисканні на терміни користувач зможе детальніше переглянути сторінку терміну.

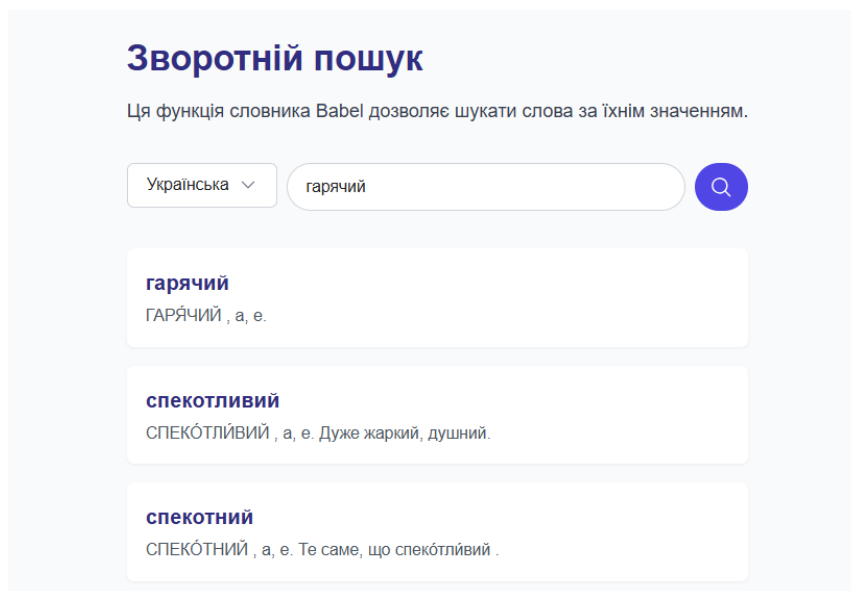


Рисунок 3.17 – Зворотній пошук

На сторінці категорій користувач може переглядати категорії, на які були розбиті терміни (рис. 3.18). При натисненні на одну із тем користувач перейде на сторінку теми, де зможе переглядати терміни за цією категорією, здійснювати класичний пошук, а також додавати вподобані терміни до списків (рис. 3.19).

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

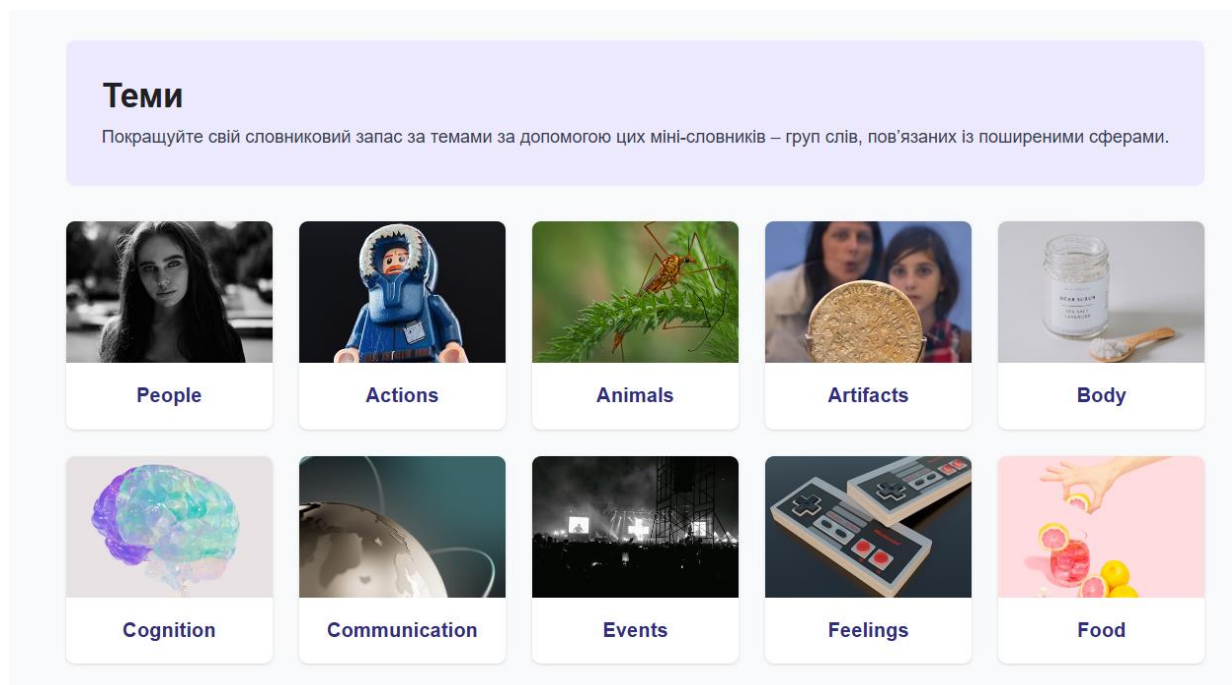


Рисунок 3.18 – Категорії термінів

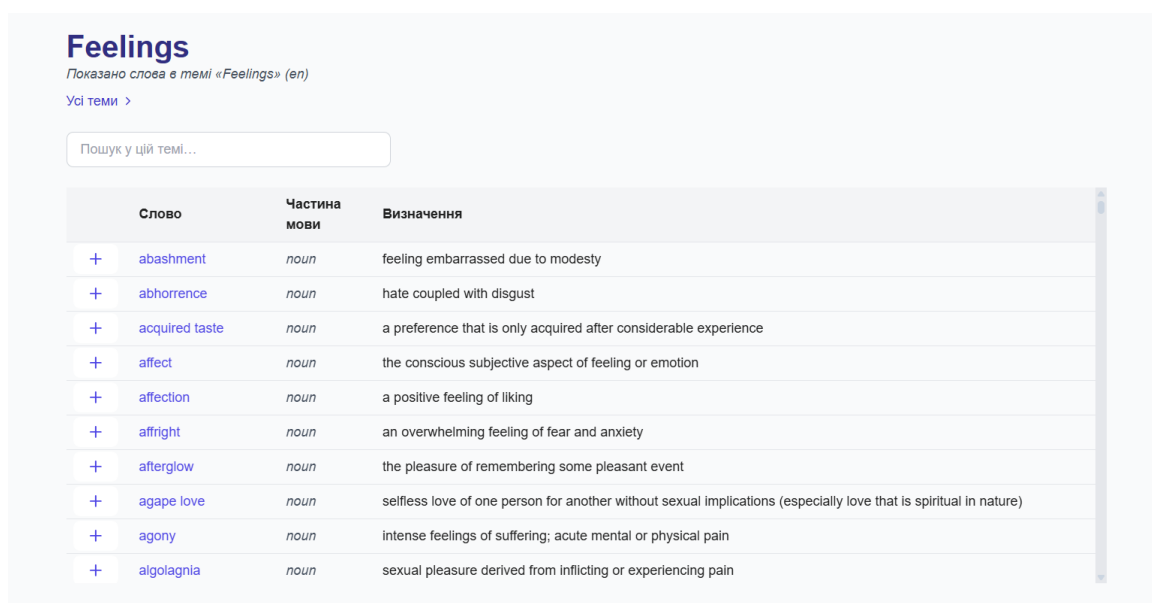


Рисунок 3.19 – Категорії термінів

3.2 Загальна статистика словника

На домашній сторінці для кожного користувача присутня загальна статистика використання словника (рис. 3.20). Вона формується на основі накопиченої активності всіх користувачів і включає три ключові категорії: найулюбленіші терміни, найчастіше шукані терміни та найвідвідуваніші терміни. Найулюбленіші визначаються за кількістю додавань термінів до списків улюбленого, найчастіше шукані за кількістю кліків за підказками у полі

пошуку, а найвідвідуваніші за кількістю переглядів сторінок термінів. Відображення цих даних на головній сторінці дозволяє всім користувачам бачити актуальні тренди у використанні словника, що додатково стимулює активність у системі.

НАЙУЛЮБЛЕНІШІ	НАЙЧАСТІШЕ ШУКАНІ	НАЙВІДВІДУВАНІШІ
91 — 1	fox — 2	fox — 28
apple — 1	abstract — 1	cat — 18
avenged — 1	accessible — 1	adaptive — 13
beaver — 1	acknowledged — 1	adaptative — 11
bird — 1	clean — 1	actual — 7
cat — 1	energy-releasing — 1	nerve-racking — 5
enigma — 1	enigma — 1	able — 4
fox — 1	lacking — 1	enigma — 3
hair — 1	nerve-racking — 1	enigmatic — 3
leaf — 1	able — 1	addictive — 3

Рисунок 3.20 – Загальна статистика

У складі блоку статистики для терміна (рис. 3.21) відображаються кілька ключових показників. До них належить кількість додавань терміна до улюбленого. Також фіксується кількість переглядів сторінки терміна, що дає змогу оцінити його загальну відвідуваність у системі. Окрім цього, обліковується кількість кліків за підказками, що демонструє, наскільки часто термін обирається зі списків підказок у полі пошуку. Ще одним елементом є кількість публічних списків, у які було додано відповідний термін, що опосередковано відображає його корисність у навчальному чи практичному використанні.

Статистика
Улюблені: 2
Відвідування: 29
Кліки за підказками: 3
Публічні списки: 2
▼ Show lists
- flowers by reginageorge2 - новий список 2 by reginageorge2

Рисунок 3.21 – Блок статистики до терміна

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

3.3 Тестування вебзастосунку

Тестування платформи тлумачного словника спрямоване на перевірку коректності роботи основних бізнес-процесів, а також на забезпечення стабільності й надійності інтеграцій зі сторонніми сервісами. Особливу увагу у процесі тестування було приділено правильності реалізації механізмів авторизації, функціоналу семантичного пошуку, взаємодії з зовнішніми словниковими API, а також коректній обробці статистичних даних і динамічного оновлення інформації у базі термінів.

Таблиця 3.1

Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
Авторизація зареєстрованого користувача	Відкрити сторінку /login, ввести ім'я користувача та пароль, натиснути Sign In	Username: user123, Пароль: ValidPass123	POST /api/token/ отримає JWT токен, фронт виконує перехід на /profile	Як і очікувалось	Pass
Створення нового користувача	Перейти на сторінку /register, ввести ім'я користувача, email, пароль	Username: new_user, Email: test@demo.com, Пароль: NewPass123	POST /api/register створює нового користувача, отримано JWT токен, редирект на /profile	Як і очікувалось	Pass
Семантичний пошук за словом	Відкрити сторінку /semantic-search, ввести запит "enigma"	"enigma"	GET /semantic_search?q=enigma повертає список релевантних термінів у порядку від найближчих за відстанню	Як і очікувалось	Pass
Автоматичний імпорт терміна з Wordnik	Ввести у пошук термін, відсутній у базі, наприклад "serendipity"	"serendipity"	При першому запиті система звертається до Wordnik API, додає термін до бази, виконує embedding, відображає результат	Як і очікувалось	Pass

Продовження таблиці 3.1

Додавання терміна до улюбленого	На сторінці терміна натиснути "Додати у вибране"	Терміни: "cat"	POST /favorites створює запис у Favorite, на сторінці терміна змінюється статус на "Улюблений"	Як і очікувалось	Pass
Видалення терміна з улюбленого	Натиснути "Видалити з вибраного"	Терміни: "cat"	DELETE /favorites видаляє запис, статус змінюється	Як і очікувалось	Pass
Створення публічного списку	У профілі перейти до "Мої списки", створити новий список "My Favs", додати до нього терміни	Назва списку: "My Favs", Термін: "apple"	POST /user-lists створює список, POST /user-list-items додає термін, список видно у профілі	Як і очікувалось	Pass
Перевірка статистики терміна	Відкрити сторінку терміна "fox", переглянути блок Статистика	Термін: "fox"	Відображаються кількість відвідувань, кліків за підказками, публічні списки та улюблення	Як і очікувалось	Pass
Валідація пошукового запиту	Ввести порожній запит у семантичний пошук, натиснути Search	Порожній рядок	Фронт показує помилку "Param 'q' is required", запит не відправляється	Як і очікувалось	Pass

Висновки до третього розділу

У третьому розділі було розглянуто структуру інтерфейсу вебзастосунку багатомовного тлумачного словника, функціонал семантичного та класичного пошуку, механізми персоналізації профілю та управління списками термінів. Інтерфейс платформи забезпечує зручну навігацію між основними модулями системи, підтримує багатомовність та дозволяє користувачам створювати й адмініструвати персональні списки слів. Інтеграція семантичного пошуку на основі Sentence-Transformer значно підвищує гнучкість пошукового механізму, а система статистики, як загальна, так і для окремих термінів, дає змогу користувачам аналізувати популярність лексичних одиниць.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

У результаті тестування підтверджено коректність реалізації основних бізнес-процесів платформи, включаючи авторизацію користувачів, семантичний пошук, імпорт нових термінів, управління улюбленими термінами та персональними списками, а також формування і відображення статистичних даних. Усі критичні тест-кейси були успішно виконані, що свідчить про технічну готовність системи до використання кінцевими користувачами та створює основу для її подальшого розвитку.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

ВИСНОВКИ

Під час виконання роботи над проектом було спроектовано та реалізовано вебзастосунок багатомовного тлумачного словника із клієнт-серверною архітектурою, що включає React-клієнт та сервер на базі Django REST Framework. Було проведено аналіз предметної області та огляд сучасних підходів до організації семантичного пошуку у лексикографічних системах. На основі отриманих результатів сформовано перелік функціональних і нефункціональних вимог, що визначили архітектурні рішення та вибір інструментальних засобів для розробки платформи.

Проектування системи включало моделювання основних сценаріїв використання для двох типів користувачів — гостей та зареєстрованих користувачів — розробку структури бази даних у PostgreSQL із розширенням pgvector для підтримки векторних пошуків, а також впровадження механізмів авторизації на основі JWT. Було реалізовано інтеграції з API зовнішніх словникових сервісів (Wordnik, Datamuse, Lexicala), автоматизовано процес імпорту нових термінів і генерації ембеддингів за допомогою моделі paraphrase-multilingual-MiniLM-L12-v2. Особливу увагу приділено оптимізації семантичного пошуку та забезпеченню його продуктивної роботи у багатомовному середовищі.

Було продемонстровано основні функціональні потоки системи: реєстрацію та аутентифікацію користувачів, класичний і семантичний пошук термінів, перегляд статистики використання термінів, керування персональними списками та вподобаннями, а також роботу з категоріями термінів. Інтерфейс вебзастосунку забезпечує зручну навігацію, адаптивність для різних мов інтерфейсу та підтримує взаємодію користувача з усіма основними функціональними модулями системи.

Результатом роботи є готова до експлуатації платформа багатомовного тлумачного словника, що забезпечує автоматизований семантичний пошук у багатомовному лексикографічному корпусі, підтримку персоналізованої роботи користувачів із термінами та відкриту архітектуру для подальшого розвитку.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

Система пройшла комплексне тестування клієнтської та серверної частин і відповідає заявленим вимогам до функціональності, продуктивності й масштабованості.

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чеверда А.Б., Вакалюк Т.А. Порівняння семантичного пошуку та пошуку за ключовими словами: вплив на точність та релевантність результатів. Інформаційні технології – 2025: зб. тез XII Всеукраїнської науково-практичної конференції молодих учених, 15 трав. 2025 р., м. Київ / Київський столичний університет імені Бориса Грінченка; Відповід. за вип.: М.М. Астаф'єва та ін. Київ : Київський столичний університет імені Бориса Грінченка, 2025. С. 206–207. <https://zcit.kubg.edu.ua/index.php/journal/issue/view/13/23>
2. Cheverda A. B., Vakaliuk T. A. LEVERAGING WORD EMBEDDINGS FOR INFORMATION RETRIEVAL : подано до друку / А. В. Cheverda, Т. А. Vakaliuk. – 2025.
3. Tetiana A. Vakaliuk, Dmytro S. Antoniuk, Oksana A. Chernysh, Vlada V. Lukashevych, Anastasiia B. Cheverda and Valerii V. Kontsedailo. Experience of involving future IT professionals in the development of learning tools as an element of digital transformation of education. E-learning & Artificial Intelligence (AI) Scientific Editor Eugenia Smyrnova-Trybulska “E-learning”, 15, Katowice–Cieszyn 2023, pp. 77–90 <https://doi.org/10.34916/el.2023.15.07>
4. JWT Handbook [Електронний ресурс] – Режим доступу: <https://jwt.io/introduction/>
5. REST API Tutorial [Електронний ресурс] – Режим доступу: <https://restfulapi.net/>
6. pgvector Documentation [Електронний ресурс] – Режим доступу: <https://github.com/pgvector/pgvector>
7. Postman Learning Center [Електронний ресурс] – Режим доступу: <https://learning.postman.com/>
8. Cambridge Dictionary [Електронний ресурс] – Режим доступу: <https://dictionary.cambridge.org/>
9. Oxford Learner’s Dictionaries [Електронний ресурс] – Режим доступу: <https://www.oxfordlearnersdictionaries.com/>
10. roh.pp.ua [Електронний ресурс] – Режим доступу: <https://roh.pp.ua/>

		Чеверда А.Б.			ІПЗ.КР.Б-121-25-ІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

11. React Documentation [Электронный ресурс] – Режим доступа: <https://react.dev/>
12. Django Documentation [Электронный ресурс] – Режим доступа: <https://docs.djangoproject.com/>
13. HTTP/1.1: Hypertext Transfer Protocol [Электронный ресурс] – Режим доступа: <https://www.rfc-editor.org/rfc/rfc2616>
14. Wordnik API Documentation [Электронный ресурс] – Режим доступа: <https://developer.wordnik.com/>
15. Datamuse API Documentation [Электронный ресурс] – Режим доступа: <https://www.datamuse.com/api/>
16. Lexicala API Documentation [Электронный ресурс] – Режим доступа: <https://developer.lexicala.com/>
17. Sentence-Transformers Documentation: paraphrase-multilingual-MiniLM-L12-v2 [Электронный ресурс] – Режим доступа: <https://www.sbert.net/>
18. PostgreSQL Documentation [Электронный ресурс] – Режим доступа: <https://www.postgresql.org/docs/>
19. Weisstein, Eric W. "Euclidean Distance." From MathWorld--A Wolfram Web Resource. [Электронный ресурс] – Режим доступа: <https://mathworld.wolfram.com/EuclideanDistance.html>
20. HTML Living Standard [Электронный ресурс] – Режим доступа: <https://html.spec.whatwg.org/>
21. Tailwind CSS Documentation [Электронный ресурс] – Режим доступа: <https://tailwindcss.com/docs>

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІПЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

ДОДАТКИ

		Чевеpда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докyм.	Підпис	Дата		60

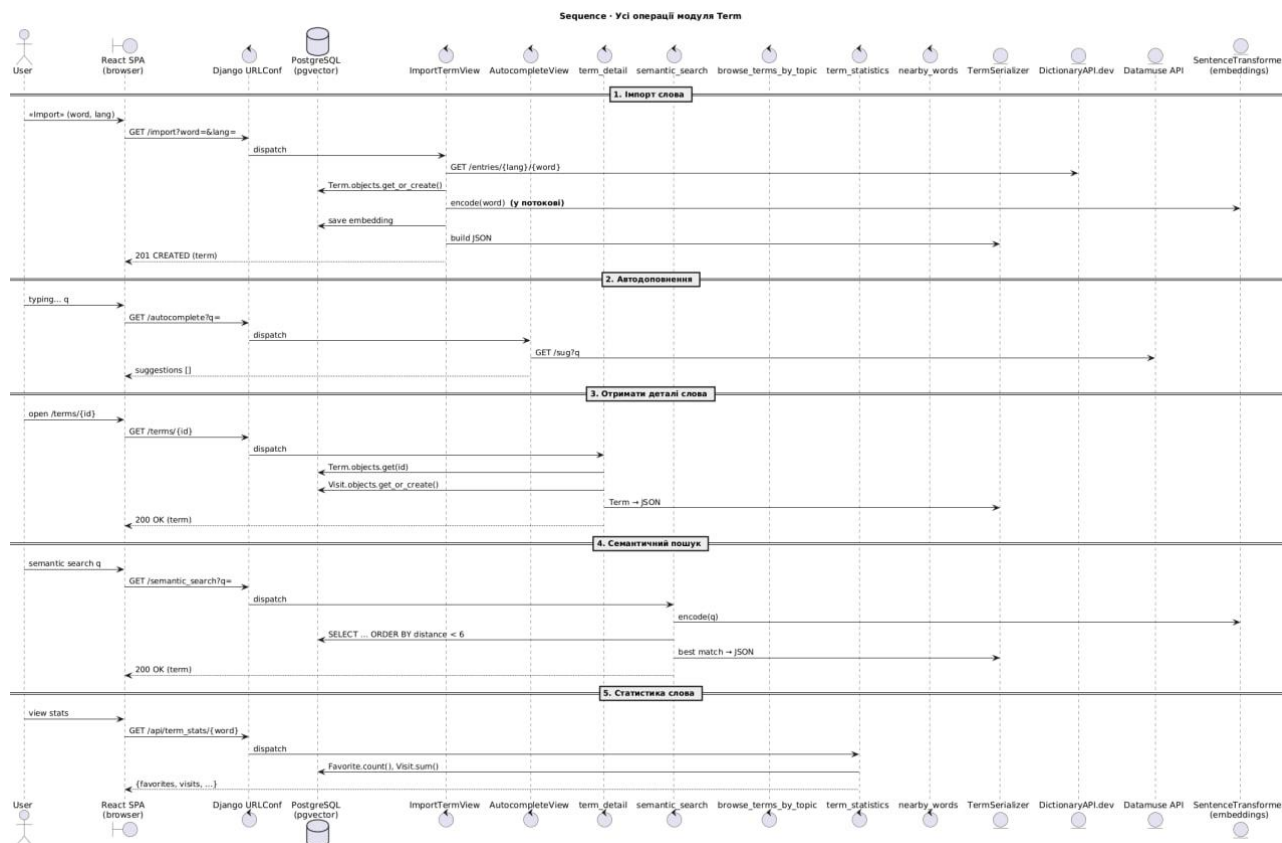


Рисунок А.1 - Діаграма послідовності платформи

views.py:

```

import sys
from django.conf import settings
from rest_framework.parsers import MultiPartParser, FormParser
from rest_framework.status import HTTP_204_NO_CONTENT
from rest_framework.views import APIView
from rest_framework_simplejwt.views import TokenRefreshView, TokenObtainPairView
from sentence_transformers import SentenceTransformer
from django.db import connection, models
from dictionary.models import Term, Favorite, WordOfTheDay, UserList, UserListItem,
SuggestionClick, Topic, Visit, SavedList
from django.shortcuts import get_object_or_404
from django.contrib.auth import get_user_model
from django.contrib.auth.models import User
from rest_framework_simplejwt.tokens import RefreshToken
from rest_framework.pagination import PageNumberPagination
from .serializers import ChangePasswordSerializer, UserProfileSerializer,
TermSerializer
from threading import Thread

```

		Чеведа А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import logging
from django.core.exceptions import ValidationError
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import AllowAny, IsAuthenticated
from rest_framework.response import Response
from rest_framework import status, permissions
import requests
from requests.exceptions import HTTPError, RequestException
from .models import Favorite
from rest_framework.views import APIView
from functools import lru_cache
from django.db.models import Count, Sum
from django.db import transaction

logger = logging.getLogger(__name__)

User = get_user_model()

@api_view(["GET"])
@permission_classes([AllowAny])
def term_statistics(request, term_word: str):

    term_key = term_word.strip()

    fav_total = Favorite.objects.filter(term__iexact=term_key).count()

    list_items_qs = (
        UserListItem.objects
        .filter(term__iexact=term_key, user_list__is_public=True)
        .select_related("user_list", "user_list__user")
    )
    public_lists = [
        {
            "id": itm.user_list.id,
            "name": itm.user_list.name,
            "username": itm.user_list.user.username,
        }
        for itm in list_items_qs
    ]

    sugg_clicks = (
        SuggestionClick.objects
        .filter(term__term__iexact=term_key)
        .aggregate(total=Sum("count"))
        .get("total") or 0
    )

    visits_total = (
        Visit.objects

```

		Чеведа А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

```

        .filter(term__term__iexact=term_key)
        .aggregate(total=Sum("visit_count"))
        .get("total") or 0
    )

    return Response({
        "term":            term_key,
        "favorites":        fav_total,
        "public_lists_count": len(public_lists),
        "public_lists":     public_lists,
        "suggestion_clicks": sugg_clicks,
        "visits":           visits_total,
    })

@api_view(["GET"])
@permission_classes([AllowAny])
def global_statistics(request):
    try:
        limit = max(1, int(request.GET.get("limit", 10)))
    except ValueError:
        limit = 10

    lang = request.GET.get("lang") # e.g. "en", "uk", "es" or None

    def _fk_lang_filter(qs):
        return qs.filter(term__language=lang) if lang else qs

    if lang:
        term_subset = Term.objects.filter(language=lang).values_list("term",
flat=True)
        fav_base = Favorite.objects.filter(term__in=term_subset)
    else:
        fav_base = Favorite.objects

    fav_qs = (
        fav_base
        .values("term")
        .annotate(total=Count("id"))
        .order_by("-total")[:limit]
    )

    sugg_qs = (
        _fk_lang_filter(SuggestionClick.objects)
        .values("term__term")
        .annotate(total=Sum("count"))
        .order_by("-total")[:limit]
    )

    visit_qs = (

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

```

        _fk_lang_filter(Visit.objects)
        .values("term__term")
        .annotate(total=Sum("visit_count"))
        .order_by("-total")[:limit]
    )

def _fmt(qs, field):
    return [{"term": row[field], "count": row["total"]} for row in qs]

return Response({
    "most_favorited": _fmt(fav_qs, "term"),
    "most_looked_up": _fmt(sugg_qs, "term__term"),
    "most_visited": _fmt(visit_qs, "term__term"),
})

@lru_cache(maxsize=1)
def get_embedding_model():
    return SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')

API_KEY = settings.WORDNIK_API_KEY
BASE = "https://api.wordnik.com/v4"
LEXICALA_HOST = "lexicala1.p.rapidapi.com"

@api_view(["GET"])
@permission_classes([AllowAny])
def wordnik_suggestions(request):
    q = request.query_params.get("q", "").strip()
    logger.debug("Wordnik key =", API_KEY)
    if not q:
        return Response([], status=200)
    resp = requests.get(
        f"{BASE}/words.json/autocomplete",
        params={"api_key": API_KEY, "query": q, "limit": 10},
        timeout=5
    )
    if resp.status_code != 200:
        return Response(
            {"upstream_status": resp.status_code, "upstream_body": resp.text},
            status=status.HTTP_502_BAD_GATEWAY
        )
    data = resp.json()
    suggestions = data
    return Response(suggestions)

@api_view(["GET"])
@permission_classes([AllowAny])
def wordnik_entry(request):
    w = request.query_params.get("word", "").strip()
    if not w:

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

```

        return Response({"detail": "word query-param required."},
                        status=status.HTTP_400_BAD_REQUEST)

    try:
        defs_resp = requests.get(
            f"{BASE}/word.json/{w}/definitions",
            params={"api_key": API_KEY, "limit": 5},
            timeout=5,
        )
        defs_resp.raise_for_status()

        defs_json = defs_resp.json()

        has_real_text = any(d.get("text") for d in defs_json)
        if not defs_json or not has_real_text:
            raise ValueError("empty")

    except (HTTPError, ValueError):
        from .openai_utils import materialise_term
        term_obj = materialise_term(w, "en")
        if term_obj:
            ser = TermSerializer(term_obj, context={'request': request})
            return Response(ser.data, status=201)
        return Response({'detail': 'Term not found.'}, status=404)

    except RequestException:
        return Response({"detail": "Could not reach Wordnik (definitions)."},
                        status=status.HTTP_503_SERVICE_UNAVAILABLE)

    try:
        exs_resp = requests.get(
            f"{BASE}/word.json/{w}/examples",
            params={"api_key": API_KEY, "limit": 5},
            timeout=5,
        )
        exs_resp.raise_for_status()
        examples = exs_resp.json().get("examples", [])
    except RequestException:
        examples = []

    try:
        rels_resp = requests.get(
            f"{BASE}/word.json/{w}/relatedWords",
            params={"api_key": API_KEY},
            timeout=5,
        )
        rels_resp.raise_for_status()
        related = rels_resp.json()
    except RequestException:
        related = []

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

```

data = {
    "word": w,
    "definitions": defs_resp.json(),
    "examples": examples,
    "relatedWords": related,
}

with transaction.atomic():
    primary_def = (defs_json or [{}])[0]
    desc_txt = primary_def.get("text", "Definition pending.") or "Definition
pending."
    pos_txt = primary_def.get("partOfSpeech", "")

    term_obj, created = Term.objects.get_or_create(
        term = w,
        language = "en",
        defaults = {
            "description": desc_txt,
            "part_of_speech": pos_txt,
        }
    )

    if not created:
        if term_obj.description.strip().lower() in ("definition pending.", "")
and desc_txt:
            term_obj.description = desc_txt
            term_obj.part_of_speech = pos_txt
            term_obj.save(update_fields=["description", "part_of_speech"])

    visit, _ = Visit.objects.get_or_create(term=term_obj)
    visit.visit_count += 1
    visit.save(update_fields=["visit_count", "last_visited"])

fav_count = Favorite.objects.filter(term=w).count()
is_fav = False

if request.user.is_authenticated:
    is_fav = Favorite.objects.filter(user=request.user, term=w).exists()
data["localMeta"] = {
    "id": term_obj.id,
    "is_favorited": is_fav,
    "favorites": fav_count,
}
return Response(data)

def generate_embedding(term_obj: Term):
    """
    Background worker to generate and save the embedding.

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

```

"""
emb = get_embedding_model.encode(term_obj.term).tolist()
term_obj.embedding = emb
term_obj.save(update_fields=['embedding'])

class ImportTermView(APIView):
    permission_classes = [permissions.AllowAny]

    def get(self, request, *args, **kwargs):
        word = request.query_params.get('word')
        lang = request.query_params.get('lang', 'en')
        if not word:
            return Response({'detail': 'word parameter is required.'},
                            status=status.HTTP_400_BAD_REQUEST)

        try:
            r = requests.get(
                f'https://api.dictionaryapi.dev/api/v2/entries/{lang}/{word}',
                timeout=2
            )
        except requests.RequestException:
            return Response({'detail': 'Definition service unavailable.'},
                            status=status.HTTP_503_SERVICE_UNAVAILABLE)

        if r.status_code != 200:
            return Response({'detail': 'Definition not found.'},
                            status=status.HTTP_404_NOT_FOUND)

        data = r.json()[0]

        meaning = data.get('meanings', [{}])[0]
        defs = meaning.get('definitions', [{}])
        description = defs[0].get('definition', '')

        term_obj, created = Term.objects.get_or_create(
            term=word,
            language=lang,
            defaults={
                'description': description,
                'part_of_speech': meaning.get('partOfSpeech', ''),
            }
        )
        if not created:
            term_obj.description = description
            term_obj.part_of_speech = meaning.get('partOfSpeech', '')
            term_obj.save(update_fields=['description', 'part_of_speech'])

        Thread(target=generate_embedding, args=(term_obj,)), daemon=True).start()

        serializer = TermSerializer(term_obj, context={'request': request})

```

		Чеведа А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

```

        return Response(serializer.data, status=status.HTTP_201_CREATED)

class AutocompleteView(APIView):
    permission_classes = [permissions.AllowAny]

    def get(self, request):
        q = request.query_params.get('q', '').strip()
        if not q:
            return Response([], status=200)
        # hit Datamuse Suggestions API
        resp = requests.get('https://api.datamuse.com/sug', params={'s': q})
        return Response(resp.json())

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def change_password(request):
    user = request.user
    current = request.data.get('current_password')
    new = request.data.get('new_password')
    confirm = request.data.get('confirm_new_password')

    if not user.check_password(current):
        return Response(
            {'detail': 'Current password is incorrect.'},
            status=status.HTTP_400_BAD_REQUEST
        )

    if new != confirm:
        return Response(
            {'detail': 'New password and confirmation do not match.'},
            status=status.HTTP_400_BAD_REQUEST
        )

    user.set_password(new)
    user.save()

    return Response({'detail': 'Password changed successfully.'})

class ChangePasswordView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request, *args, **kwargs):
        serializer = ChangePasswordSerializer(
            data=request.data,
            context={'request': request}    # ← this line is critical!
        )
        serializer.is_valid(raise_exception=True)

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

```

        serializer.save()
        return Response(
            {"detail": "Password updated successfully."},
            status=status.HTTP_200_OK
        )

class ProfileView(APIView):
    permission_classes = [IsAuthenticated]
    parser_classes = [MultiPartParser, FormParser]

    def get(self, request):
        ser = UserProfileSerializer(request.user, context={'request': request})
        return Response(ser.data)

    def patch(self, request):
        ser = UserProfileSerializer(
            request.user,
            data = request.data,
            partial = True,
            context = {'request': request}
        )
        ser.is_valid(raise_exception=True)
        ser.save()
        return Response(ser.data)

class RegisterView(APIView):
    permission_classes = [AllowAny] # Anyone can register

    def post(self, request):
        username = request.data.get('username')
        email = request.data.get('email')
        password = request.data.get('password')

        # Validate basic fields
        if not username or not password:
            return Response(
                {"detail": "Username and password required."},
                status=status.HTTP_400_BAD_REQUEST
            )

        # Optional: check if username is taken
        if User.objects.filter(username=username).exists():
            return Response(
                {"detail": "Username already taken."},
                status=status.HTTP_400_BAD_REQUEST
            )

        user = User.objects.create_user(

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

```

        username=username,
        email=email,
        password=password
    )

    # Immediately create & return JWT tokens
    refresh = RefreshToken.for_user(user)
    return Response({
        "refresh": str(refresh),
        "access": str(refresh.access_token),
        "username": user.username
    }, status=status.HTTP_201_CREATED)

class LoginView(TokenObtainPairView):
    permission_classes = [AllowAny]

class RefreshTokenView(TokenRefreshView):
    permission_classes = [AllowAny]

# /me/ endpoint
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def me(request):
    serializer = UserProfileSerializer(request.user, context={'request': request})
    return Response(serializer.data)

@lru_cache()
def get_model():
    return SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2')

@api_view(['GET'])
def semantic_search(request):
    threshold = getattr(settings, "SEMANTIC_DISTANCE_THRESHOLD", 6.0)

    q = request.GET.get('q', '')
    lang = request.GET.get('lang', '').strip()
    if not q:
        return Response({"detail": "Param 'q' is required"}, status=400)

    model = get_model()
    emb = model.encode(q)

    arr_str = ",".join(f"{float(x):.6f}" for x in emb)

    lang_filter = f"AND language = '{lang}'" if lang else ""

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

# евклід. відстань (<->)
sql = f"""
    SELECT
        id,
        term,
        description,
        (embedding_vector <-> CAST(ARRAY[{arr_str}] AS vector(384))) AS distance
    FROM dictionary_term
    WHERE embedding_vector IS NOT NULL {lang_filter}
    ORDER BY distance
    LIMIT 15;
    """

with connection.cursor() as cursor:
    cursor.execute(sql)
    rows = cursor.fetchall()
    # rows => [(id, term, desc, distance), ...]

# Логування отриманих відстаней
distances = [float(r[3]) for r in rows]
print(f"[SemanticSearch] q='{q}' => Distances: {distances}")

# Фільтрація за threshold: лишаємо лише distance <= THRESHOLD
results = []
for row in rows:
    dist = float(row[3])
    if dist <= threshold:
        results.append({
            "id": row[0],
            "term": row[1],
            "description": row[2],
            "distance": dist
        })

return Response(results)

def _find_uk_term(term_key: str) -> Term | None:
    tk = term_key.strip()
    qs = (
        Term.objects
        .filter(term__iexact=tk, language__in=('uk', 'ua'))
    )
    return qs.first()

@api_view(['GET'])
@permission_classes([AllowAny])
def term_detail(request, term_key: str):

```

		Чеве́рда А.Б.			ІПЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

```

"""
Unified lookup:

• If ?lang=uk - treat *term_key* as Ukrainian text and resolve by term string
• Else       - treat *term_key* as integer ID and resolve by PK
"""

lang = request.GET.get('lang', '').strip().lower()

if lang == 'uk':
    term = _find_uk_term(term_key)
    if term is None:
        from .openai_utils import materialise_term
        term = materialise_term(term_key, "uk")
    if term is None:
        return Response({'detail': 'Термін не знайдено.'}, status=404)

else:
    try:
        term_id = int(term_key)
        term = Term.objects.get(id=term_id)
    except (ValueError, Term.DoesNotExist):
        return Response({'detail': 'Term not found.'}, status=404)

visit, _ = Visit.objects.get_or_create(term=term)
visit.visit_count += 1
visit.save()

is_favorited = (
    request.user.is_authenticated
    and Favorite.objects.filter(user=request.user, term=term).exists()
)

nearby = []
for rel in getattr(term, 'relatedWords', []):
    if 'cross' in rel.relationshipType.lower():
        nearby.extend(rel.words)
nearby = list(dict.fromkeys(nearby))

return Response({
    'id': term.id,
    'term': term.term,
    'part_of_speech': term.part_of_speech,
    'description': term.description,
    'language': term.language,
    'views': visit.visit_count,
    'favorites': Favorite.objects.filter(term=term).count(),
    'nearby': nearby,
    'usage': term.examples,
    'is_favorited': is_favorited,

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

```

    })

@api_view(['GET'])
def nearby_words(request):
    term_id = request.GET.get('term_id')
    if not term_id:
        return Response({"detail": "Param 'term_id' is required."}, status=400)

    try:
        term = Term.objects.get(id=term_id)
    except Term.DoesNotExist:
        return Response({"detail": f"Term with id '{term_id}' not found."},
            status=404)

    before_terms = (
        Term.objects
        .filter(term__lt=term.term)
        .order_by('-term')[:5]
    )

    after_terms = (
        Term.objects
        .filter(term__gt=term.term)
        .order_by('term')[:5]
    )

    results = []

    for t in reversed(before_terms):
        results.append({
            "id": t.id,
            "term": t.term,
            "part_of_speech": t.part_of_speech,
            "description": t.description,
            "language": t.language,
        })

    results.append({
        "id": term.id,
        "term": term.term,
        "part_of_speech": term.part_of_speech,
        "description": term.description,
        "language": term.language,
    })

    for t in after_terms:
        results.append({
            "id": t.id,
            "term": t.term,

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

```

        "part_of_speech": t.part_of_speech,
        "description": t.description,
        "language": t.language,
    })

    return Response(results)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def add_favorite(request):
    term_id = request.data.get('term_id')
    if not term_id:
        return Response({'detail': 'term_id is required.'}, status=400)

    term = get_object_or_404(Term, id=term_id)

    current_user = request.user

    favourite, created = Favorite.objects.get_or_create(user=current_user,
term=term)

    if not created:
        return Response({'detail': 'Already in favourites.'}, status=200)

    return Response({'detail': 'Added to favourites.'}, status=201)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def remove_favorite(request):
    term_id = request.data.get('term_id')
    if not term_id:
        return Response({'detail': 'term_id is required.'}, status=400)

    term = get_object_or_404(Term, id=term_id)

    current_user = request.user

    favourite = Favorite.objects.filter(user=current_user, term=term)
    if not favourite.exists():
        return Response({'detail': 'Favourite not found.'}, status=404)

    favourite.delete()
    return Response({'detail': 'Removed from favourites.'}, status=200)

@api_view(['POST', 'DELETE'])
@permission_classes([IsAuthenticated])
def toggle_favorite(request):

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

```

term_title = request.data.get('term')
if not term_title:
    return Response({'detail': 'Term title is required.'}, status=400)

user = request.user
if request.method == 'POST':
    # add favourite
    try:
        fav = Favorite.objects.create(user=user, term=term_title)
    except ValidationError as e:
        return Response({'detail': str(e)}, status=400)
    is_fav = True
else:
    # remove favourite
    Favorite.objects.filter(user=user, term=term_title).delete()
    is_fav = False

# return updated counts
count = Favorite.objects.filter(term=term_title).count()
return Response({
    'term': term_title,
    'is_favorited': is_fav,
    'favorites': count
})

```

```

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def list_favorites(request):
    current_user = request.user

    favourites = Favorite.objects.filter(user=request.user)

    data = [
        {
            'id': fav.id, # PK of the Favorite
            'term': fav.term, # the actual term string
            'part_of_speech': None,
            'description': None,
            'language': None,
        }
        for fav in favourites
    ]

    return Response(data)

```

```

@api_view(['GET'])
@permission_classes([AllowAny])
def word_of_the_day(request):

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

"""
Proxy to Wordnik's Word of the Day.
"""
api_key = settings.WORDNIK_API_KEY
try:
    resp = requests.get(
        'https://api.wordnik.com/v4/words.json/wordOfTheDay',
        params={'api_key': api_key},
        timeout=5
    )
    resp.raise_for_status()
except requests.RequestException:
    return Response(
        {'detail': 'Unable to fetch Word of the Day.'},
        status=503
    )

data = resp.json()
return Response({
    'word': data.get('word'),
    'part_of_speech': (data.get('definitions') or [{}])[0].get('partOfSpeech'),
    'definitions': [d.get('text') for d in data.get('definitions', [])],
    'examples': [e.get('text') for e in data.get('examples', [])],
})

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_user_list(request):
    name = request.data.get('name')
    is_public = request.data.get('is_public', False)

    if not name:
        return Response({'detail': 'List name is required.'}, status=400)

    user_list, created = UserList.objects.get_or_create(user=request.user,
name=name)
    if not created:
        return Response({'detail': 'List already exists.'}, status=400)

    user_list.is_public = is_public
    user_list.save()

    return Response({'detail': 'List created.', 'id': user_list.id}, status=201)

@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def delete_list(request, pk):
    user_list = get_object_or_404(UserList, id=pk, user=request.user)

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

```

user_list.delete()
return Response(status=HTTP_204_NO_CONTENT)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_lists(request):
    lists = UserList.objects.filter(user=request.user)
    data = [{'id': l.id, 'name': l.name, 'is_public': l.is_public} for l in lists]
    return Response(data)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def add_term_to_list(request, list_id):
    word = request.data.get('term')
    if not word:
        return Response({'detail': 'No term provided.'}, status=400)

    user_list = get_object_or_404(UserList, pk=list_id, user=request.user)
    # avoid duplicates by unique_together constraint
    item, created = UserListItem.objects.get_or_create(user_list=user_list,
term=word)
    if created:
        return Response({'detail': f'"{word}" added to your list.'})
    else:
        return Response({'detail': f'"{word}" is already in your list.'},
status=400)

@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def remove_term_from_list(request):
    list_id = request.data.get('list_id')
    term_id = request.data.get('term_id')

    if not list_id or not term_id:
        return Response({'detail': 'list_id and term_id are required.'},
status=400)

    user_list = get_object_or_404(UserList, id=list_id, user=request.user)
    item = UserListItem.objects.filter(user_list=user_list, term_id=term_id)
    if not item.exists():
        return Response({'detail': 'Term not in list.'}, status=404)

    item.delete()
    return Response({'detail': 'Term removed from list.'}, status=200)

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

```

@api_view(["GET"])
@permission_classes([AllowAny])
def get_public_list_terms(request, list_id):
    """
    Return the terms that belong to a PUBLIC user list (is_public=True).
    Anyone may call this endpoint.
    """
    user_list = get_object_or_404(UserList, pk=list_id, is_public=True)

    qs = (
        UserListItem.objects
        .filter(user_list=user_list)
        .order_by("-added_at")
    )

    paginator = PageNumberPagination()
    paginator.page_size = 10
    page = paginator.paginate_queryset(qs, request) or []

    data = []
    for item in page:
        term_obj = Term.objects.filter(term=item.term).first()

        fav_ct = Favorite.objects.filter(term=item.term).count()

        data.append({
            "id": term_obj.id if term_obj else None,
            "term": item.term,
            "description": term_obj.description if term_obj else "",
            "part_of_speech": term_obj.part_of_speech if term_obj else "",
            "language": term_obj.language if term_obj else "",
            "favorites": fav_ct,
        })

    return paginator.get_paginated_response(data)

@api_view(["GET"])
@permission_classes([IsAuthenticated])
def get_my_list_terms(request, list_id):
    """
    Return terms for a list owned by the current user.
    Private lists are not accessible to anyone else; only the owner may fetch.
    """
    user_list = get_object_or_404(UserList, pk=list_id, user=request.user)

    qs = (
        UserListItem.objects
        .filter(user_list=user_list)
        .order_by("-added_at")
    )

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

```

)

paginator = PageNumberPagination()
paginator.page_size = 10
page = paginator.paginate_queryset(qs, request) or []

data = []
for item in page:
    term_obj = Term.objects.filter(term=item.term).first()
    data.append({
        "id":            term_obj.id if term_obj else None,
        "term":          item.term,
        "description":   term_obj.description if term_obj else "",
        "part_of_speech": term_obj.part_of_speech if term_obj else "",
        "language":      term_obj.language if term_obj else "",
    })

return paginator.get_paginated_response(data)

@api_view(['PATCH'])
@permission_classes([IsAuthenticated])
def rename_list(request, pk):
    user_list = get_object_or_404(UserList, id=pk, user=request.user)
    new_name = request.data.get("name")
    if not new_name:
        return Response({"detail": "New name is required."}, status=400)
    user_list.name = new_name
    user_list.save()
    return Response({
        "detail": "List renamed.",
        "name": user_list.name,
        "id": user_list.id
    })

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def toggle_visibility(request, pk):
    user_list = get_object_or_404(UserList, id=pk, user=request.user)
    user_list.is_public = not user_list.is_public
    user_list.save()
    return Response({"detail": f"Visibility set to {'public' if user_list.is_public else 'private'}"})

@api_view(['GET'])
@permission_classes([AllowAny])
def browse_public_lists(request):
    user = request.user if request.user.is_authenticated else None

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

```

public_lists = UserList.objects.filter(is_public=True).select_related('user')
data = []

for ul in public_lists:
    is_saved = False
    if user:
        is_saved = SavedList.objects.filter(user=user, user_list=ul).exists()

    data.append({
        "id": ul.id,
        "name": ul.name,
        "username": ul.user.username,
        "created_at": ul.created_at,
        "is_saved": is_saved,
    })

return Response(data)

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def save_list(request, list_id):
    try:
        user_list = UserList.objects.get(id=list_id)
        saved, created = SavedList.objects.get_or_create(user=request.user,
user_list=user_list)
        if created:
            return Response({'message': 'List saved successfully.'}, status=201)
        else:
            return Response({'message': 'List already saved.'}, status=200)
    except UserList.DoesNotExist:
        return Response({'error': 'List not found.'}, status=404)

@api_view(['DELETE'])
@permission_classes([IsAuthenticated])
def unsave_list(request, pk):
    try:
        saved = SavedList.objects.get(user=request.user, user_list_id=pk)
        saved.delete()
        return Response({'detail': 'List unsaved successfully.'},
status=status.HTTP_204_NO_CONTENT)
    except SavedList.DoesNotExist:
        return Response({'detail': 'Saved list not found.'},
status=status.HTTP_404_NOT_FOUND)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def my_saved_lists(request):

```

		Чевежда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

```

user = request.user

own_lists = UserList.objects.filter(user=user).prefetch_related('items')
own_data = [{
    'id': l.id,
    'name': l.name,
    'term_count': l.items.count(),
    'owned': True
} for l in own_lists]

saved_lists = SavedList.objects.filter(user=user).select_related('user_list')
saved_data = [{
    'id': s.user_list.id,
    'name': s.user_list.name,
    'term_count': s.user_list.items.count(),
    'owned': False
} for s in saved_lists]

return Response(own_data + saved_data)

@api_view(['GET'])
@permission_classes([AllowAny])
def search_suggestions(request):
    q = request.GET.get('q', '').strip()
    lang = request.GET.get('lang', '').strip().lower()

    if not q or lang not in ('en', 'es', 'uk'):
        return Response(
            {"detail": "Both 'q' and valid 'lang' (en, es, uk) parameters are required."},
            status=status.HTTP_400_BAD_REQUEST
        )

    if lang == 'en':
        datamuse_url = "https://api.datamuse.com/words"
        try:
            resp = requests.get(
                datamuse_url,
                params={
                    "sp": f"{q}*",      # "<q>*" for prefix matching
                    "max": 10           # limit to 10 suggestions
                },
                timeout=5
            )
        except requests.RequestException:
            # If there's a network problem, return an empty list
            return Response([], status=status.HTTP_200_OK)

        if resp.status_code != 200:

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

```

        # If Datamuse itself failed, fall back to no suggestions
        return Response([], status=status.HTTP_200_OK)

    data = resp.json()
    words = [entry.get("word", "") for entry in data[:10] if entry.get("word")]
    existing = {t.term.lower(): t for t in
Term.objects.filter(term__in=words, language="en")}
    created_terms = []
    for w_str in words:
        if w_str.lower() not in existing:
            t = Term.objects.create(
                term = w_str,
                language = "en",
                description = "Definition pending.",
            )
            existing[w_str.lower()] = t
            created_terms.append(t)

    suggestions = [{"id": existing[w.lower()].id, "word": w} for w in words]
    return Response(suggestions)

elif lang == 'es':
    rapid_key = getattr(settings, 'VITE_LEXICALA_KEY', None)
    if not rapid_key:
        return Response(
            {"detail": "VITE_LEXICALA_KEY not set in environment."},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )

    try:
        lex_url = f"https://{settings.LEXICALA_HOST}/search-entries"
        lex_resp = requests.get(
            lex_url,
            params={"text": q, "language": "es"},
            headers={
                "X-RapidAPI-Key": rapid_key,
                "X-RapidAPI-Host": settings.LEXICALA_HOST,
            },
            timeout=5
        )
    except requests.RequestException:
        return Response([], status=status.HTTP_200_OK)

    if lex_resp.status_code != 200:
        return Response([], status=status.HTTP_200_OK)

    lex_data = lex_resp.json()
    results = lex_data.get("results", [])
    suggestions = [
        {"word": hit.get("headword", "")}

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

        for hit in results[:8]
            if hit.get("headword")
        ]
        return Response(suggestions)

if lang == 'uk':
    qs = (
        Term.objects
        .filter(language__in=('uk', 'ua'), term__icontains=q)
        .order_by('term')
        .values('id', 'term')[:8]
    )
    return Response([{"id": row["id"], "word": row["term"]} for row in qs])

return Response([])

@api_view(['POST'])
@permission_classes([AllowAny])
def track_suggestion_click(request):
    term_id = request.data.get("term_id")
    word = request.data.get("word", "").strip()

    if term_id:
        try:
            term = Term.objects.get(id=term_id)
        except Term.DoesNotExist:
            return Response({"detail": "Term not found."}, status=404)
    elif word:
        term, _ = Term.objects.get_or_create(
            term = word,
            language = "en",
            defaults = {"description": "Definition pending."},
        )
    else:
        return Response({"detail": "term_id or word is required."}, status=400)

    click_obj, _ = SuggestionClick.objects.get_or_create(term=term)
    click_obj.count = (click_obj.count or 0) + 1
    click_obj.save(update_fields=["count"])

    return Response({"ok": True})

@api_view(['GET'])
def browse_topics(request):
    topics = Topic.objects.all()
    data = [{"id": t.id, 'topic_name': t.topic_name} for t in topics]
    return Response(data)

```

		Чеведа А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

@api_view(['GET'])
def browse_terms_by_topic(request, topic_id):
    search_query = request.GET.get('q', '').strip().lower()
    terms = Term.objects.filter(topic_id=topic_id).order_by('term')

    if search_query:
        terms = terms.filter(term__icontains=search_query)

    results = [
        {
            "id": t.id,
            "term": t.term,
            "description": t.description,
            "part_of_speech": t.part_of_speech,
            "language": t.language,
        } for t in terms
    ]
    return Response(results)

```

		Чеве́рда А.Б.			ІІЗ.КР.Б-121-25-ІІЗ	Арк.
		Вакалюк Т.А.				
Змн.	Арк.	№ докум.	Підпис	Дата		84