МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ КАРТКОВОГО МЕТОДУ У ВЕБЗАСТОСУНКУ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.С. Гетало
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Гетало Данилу Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Реалізація карткового методу у вебзастосунку вивчення іноземних мов

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та методів навчання.
2. Проєктування вебзастосунку.
3. Програмна реалізація та захист модулів вебзастосунку.
4. Тестування та розгортання проєкту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	19.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	06.04.2026	
5.	Розробка третього та четвертого розділу.	13.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	15.06.2026	

Студент _____ Д.С. Гетало
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація карткового методу у вебзастосунку вивчення іноземних мов»: 60 с., 22 рис., 2 табл., 18 джерел.

ВЕБЗАСТОСУНОК, ІНОЗЕМНА МОВА, ІНТЕРВАЛЬНІ ПОВТОРЕННЯ, КАРТКОВИЙ МЕТОД, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, NODE.JS, POSTGRESQL, REACT.

Об’єкт дослідження – процес вивчення іноземних мов за допомогою інформаційних технологій.

Мета роботи: розробка повнофункціонального вебзастосунку для ефективного вивчення іноземних слів на основі карткового методу та алгоритму інтервальних повторень (SRS) з використанням сучасних вебтехнологій.

Предмет дослідження – програмні засоби та алгоритми реалізації методу інтервальних повторень у вебзастосунках.

Методи дослідження – аналіз існуючих рішень у сфері EdTech, методи об’єктно-орієнтованого програмування, методи проєктування реляційних баз даних, алгоритмічне моделювання систем інтервальних повторень.

У роботі спроєктовано та реалізовано вебзастосунок клієнт-серверної архітектури. Серверну частину реалізовано на базі платформи Node.js з використанням СКБД PostgreSQL для надійного збереження даних користувачів. Клієнтську частину розроблено за допомогою бібліотеки React, що забезпечує високу швидкість роботи та сучасний UI/UX дизайн (Glassmorphism). Реалізовано алгоритм інтервальних повторень, адаптований для запам’ятовування лексики, а також інтегровано функцію озвучування тексту (Web Speech API). Застосунок успішно розгорнуто у хмарних середовищах Render та Vercel.

SUMMARY

Bachelor’s qualifying paper “Implementation of the Flashcard-Based Approach in a Web Application for Foreign Language Learning”: 60 pages, 22 figures, 2 tables, 18 references.

CLIENT-SERVER ARCHITECTURE, FLASHCARD METHOD, FOREIGN LANGUAGE, NODE.JS, POSTGRESQL, REACT, SPACED REPETITION, WEB APPLICATION.

The object of the study is the process of learning foreign languages using information technologies.

The aim of the study is the development of a fully functional web application for the effective learning of foreign vocabulary based on the flashcard method and spaced repetition algorithm (SRS) using modern web technologies.

The methods of research include the analysis of existing EdTech solutions, object-oriented programming methods, relational database design methods, and algorithmic modeling of spaced repetition systems.

The paper designs and implements a client-server architecture web application. The server-side is implemented using the Node.js platform with a PostgreSQL DBMS to securely store user data.

The client-side is developed using the React library, which ensures high performance and modern UI/UX design (Glassmorphism). A spaced repetition algorithm (SM-2) adapted for vocabulary memorization has been successfully implemented, along with the integration of a text-to-speech feature (Web Speech API) and a bulk data import module. The application has been successfully deployed in Render and Vercel cloud environments with a configured CI/CD pipeline.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та методів навчання 11
1.1 Огляд наявних систем для вивчення мов 11
1.2 Когнітивні механізми та наукове обґрунтування методу інтервальних повторень у комп’ютерному навчанні (CALL)..... 14
1.2.1 Ефект розподіленого навчання та ефект тестування 14
1.2.2 Ефективність SRS у програмних середовищах 15
1.2.3 Еволюція математичних моделей: від SM-2 до FSRS..... 16
1.3 Теоретичні основи методу інтервальних повторень 16
1.4 Постановка задачі на розробку програмного продукту..... 19
2 Проєктування вебзастосунку 22
2.1 Обґрунтування вибору стеку технологій..... 22
2.2 Проєктування архітектури клієнт-серверної взаємодії..... 24
2.3 Проєктування структури бази даних 27
2.4 Проєктування алгоритму інтервальних повторень та системи оцінювання..... 30
3 Програмна реалізація та захист модулів вебзастосунку 36
3.1 Розробка серверної частини та налаштування безпеки 36
3.2 Розробка користувацького інтерфейсу та адаптивного дизайну 38
3.3 Інтеграція додаткових функцій 39
3.4 Посібник користувача..... 40
3.4.1 Модуль автентифікації та реєстрації користувачів..... 41
3.4.2 Головна робоча панель керування (Dashboard) 43
3.4.3 Інтерфейс інтерактивного тренажера (Режим навчання) 44

3.4.4 Модуль масового імпорту словникових карток 46
3.4.5 Керування колірними темами оформлення інтерфейсу 47
4 Тестування та розгортання проєкту 50
4.1 Методологія та стратегія тестування 50
4.2 Аналіз продуктивності та якісних показників інтерфейсу 51
4.3 Розгортання застосунку у хмарних середовищах Vercel та Render ... 53
Висновки 56
Перелік посилань 59

ВСТУП

Актуальність теми: в умовах глобалізації, стирання інформаційних кордонів та інтенсивного розвитку міжнародної співпраці, володіння іноземними мовами стало невіддільною складовою успішної професійної та особистої реалізації людини. Сучасний етап розвитку суспільства характеризується тотальною цифровізацією освіти. Сектор освітніх технологій (EdTech) пропонує безліч інструментів для навчання, проте проблема ефективного та швидкого засвоєння великих обсягів іноземної лексики залишається гострою.

Традиційні методи вивчення лексики часто виявляються малоефективними через швидке забування матеріалу, що пояснюється фізіологічними особливостями людської пам’яті (крива забування Еббінгауза [5,16]). Найбільш дієвим підходом до розв’язання цієї проблеми на сьогодні визнано метод інтервальних повторень (Spaced Repetition System – SRS), який полягає у повторенні навчального матеріалу через інтервали часу, що постійно зростають.

Попри наявність на ринку таких рішень як Anki або Quizlet, багато з них мають застарілий та перевантажений інтерфейс, або навпаки – обмежують доступ до ключових алгоритмів довгострокового запам’ятовування через агресивну монетизацію та платні передплати. Отже, розробка сучасного, швидкого, кросплатформового та інтуїтивно зрозумілого вебзастосунку, який реалізує картковий метод вивчення слів на основі алгоритмів інтервальних повторень, є надзвичайно актуальним інженерним завданням. Використання сучасного стеку технологій (React [1], Node.js [2], PostgreSQL [4]) дозволяє створити незалежне рішення, доступне з будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення.

Мета дослідження – розробка повнофункціонального вебзастосунку для ефективного вивчення іноземної лексики на основі карткового методу та

алгоритму інтервальних повторень (SRS) з використанням сучасних вебтехнологій.

Для досягнення поставленої мети було сформульовано такі завдання дослідження:

- провести аналіз предметної області, існуючих систем-аналогів та теоретичних основ методу інтервальних повторень;
- спроєктувати клієнт-серверну архітектуру вебзастосунку та обґрунтувати вибір технологічного стеку;
- розробити реляційну структуру бази даних для безпечного збереження профілів, колод та метрик прогресу;
- адаптувати та програмно реалізувати математичний алгоритм інтервальних повторень (на базі SM-2);
- створити адаптивний користувацький інтерфейс (UI/UX) із застосуванням парадигми Glassmorphism та 3D-анімацій;
- інтегрувати додаткові модулі: масовий імпорт даних, систему автоматичної адаптації користувача та голосовий синтез (Text-to-Speech);
- провести тестування системи та розгорнути програмний продукт у хмарному середовищі.

Об’єкт дослідження – процес вивчення іноземних мов за допомогою інформаційних технологій.

Предмет дослідження – програмні засоби, архітектурні патерни та алгоритми реалізації методу інтервальних повторень у вебзастосунках.

Методи дослідження. У процесі виконання роботи використовувалися методи системного та порівняльного аналізу (для огляду існуючих рішень), методи об’єктно-орієнтованого та функціонального програмування (для написання коду), методи проєктування реляційних баз даних (нормалізація), а також алгоритмічне моделювання систем інтервальних повторень.

Практичне значення одержаних результатів полягає у створенні готового до експлуатації, кросплатформового та безкоштовного програмного

продукту «LernKarten», який може бути використаний студентами, школярами та всіма охочими для інтенсивного розширення словникового запасу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ НАВЧАННЯ

1.1 Огляд наявних систем для вивчення мов

Процес глобалізації та стрімкого розвитку інформаційного суспільства висуває нові вимоги до швидкості та якості засвоєння іноземної лексики. Сучасний сектор освітніх технологій (EdTech) переживає етап активної цифрової трансформації, впроваджуючи мобільні та вебзастосунки як основні інструменти інтенсифікації навчання. Однак, попри широке різноманіття комерційних та некомерційних рішень, проблема створення гнучкого, науково обґрунтованого та користувацько-орієнтованого інструментарію для накопичення активного словникового запасу залишається відкритою. Для визначення функціональних меж та виявлення інженерно-технологічних недоліків наявного програмного забезпечення було проведено критичний аналіз трьох фундаментальних систем-аналогів: Anki, Quizlet та Duolingo.

Програмний комплекс Anki є де-факто академічним стандартом серед інструментів, що базуються на картковому методі вивчення інформації та алгоритмах інтервальних повторень. З технічного погляду Anki – це кросплатформовий застосунок, локальна архітектура якого побудована навколо реляційної бази даних SQLite.

Основними перевагами системи є висока математична точність алгоритму розрахунку інтервалів, відкритий вихідний код та потужна екосистема користувацьких плагінів (додатків), написаних мовою програмування Python. Проте, детальний UI/UX аналіз платформи виявляє критичні архітектурні недоліки. Інтерфейс користувача Anki є застарілим, перевантаженим модальними вікнами та складними конфігураційними меню, що порушує закон Фіттса та принципи інтуїтивності інтерфейсів. Високий поріг входження змушує студентів витрачати значний когнітивний ресурс на вивчення синтаксису налаштування колод замість безпосереднього засвоєння

лексики. Крім того, хмарна синхронізація між десктопною та мобільною версіями часто працює нестабільно при роботі з великими медіа-базами, а повноцінна вебверсія має суттєво обмежений функціонал.

Хмарна платформа Quizlet орієнтована на масовий сегмент користувачів і пропонує радикально інший підхід, сфокусований на інтерактивності та візуальній привабливості. Вебзастосунок побудований з використанням сучасних клієнтських фреймворків та серверної архітектури, що забезпечує швидке завантаження карток і підтримку різноманітних ігрових режимів навчання (тестування, підбір відповідностей, карткові сесії). Дизайн інтерфейсу орієнтований на сучасні патерни проєктування людино-машинної взаємодії [11].

Головними перевагами Quizlet є низький поріг входження, ергономічний дизайн та наявність розвинених інструментів для спільної роботи. Проте, з методичного та інженерного поглядів безкоштовна версія платформи має суттєвий недолік: у ній повністю відсутній повноцінний адаптивний алгоритм інтервальних повторень. Повторення слів відбувається за лінійною схемою, що призводить до неефективного розподілу часу користувача (однакова частота показу легких і складних слів). Останні оновлення платформи продемонстрували агресивну стратегію монетизації, внаслідок якої базові функції довгострокового заучування були заблоковані платною передплатою, а інтерфейс безкоштовної версії став перевантаженим банерною рекламою, що критично знижує концентрацію студента.

Мобільний та вебзастосунок Duolingo представляє концепцію мікронавчання, що повністю базується на глибокій геймифікації освітнього процесу (елементи фреймворку Octalysis: ударні режими, внутрішня валюта, ліги, системи нагород).

Технологічна архітектура Duolingo дозволяє обробляти колосальні масиви даних у реальному часі, адаптуючи складність вправ під поточний рівень користувача. Перевагою системи є висока здатність утримувати мотивацію учня-початківця. Однак для завдань цільового та професійного

розширення словникового запасу Duolingo виявляється малоефективним. Система використовує жорстку лінійну структуру («дерево навичок»), яка повністю позбавляє користувача автономії: студент не має інженерної можливості створити власну колоду слів, імпортувати специфічну професійну лексику (наприклад, IT-термінологію) або пропустити занадто легкі для нього граматичні блоки. Крім того, надмірна увага до ігрових механік часто створює псевдопрогрес, коли користувач фокусується на отриманні ігрових балів, а не на довгостроковому запам’ятовуванні матеріалу. Тепер наведемо порівняльну характеристику досліджених систем та розроблюваного застосунку «LernKarten» у вигляді таблиці 1.1.

На основі цих даних можна зробити висновок, що жодна з існуючих популярних систем не задовольняє повною мірою вимоги до інструменту для вивчення лексики. Програми з потужним математичним апаратом (Anki) мають інтерфейс не відповідаючий сучасним вимогам UI/UX дизайн-патернів та високий поріг входження, тоді як системи з гарним інтерфейсом (Quizlet, Duolingo) або не мають необхідного алгоритму, або блокують його за платними передплатами.

Програмний комплекс «LernKarten» спроєктовано таким чином, щоб зайняти вільну нішу: надати користувачам безкоштовний, сучасний, швидкий вебзастосунок із гнучким керуванням власними колодами слів та строгим науковим алгоритмом повторень (SRS), який є доступним з будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення.

Таким чином, проведений аналіз архітектурних та функціональних особливостей існуючих рішень підтверджує доцільність проєктування та розробки вебзастосунку «LernKarten».

Нова система покликана поєднати в собі суворий науковий підхід до інтервальних повторень від Anki та простоту й сучасність інтерфейсу від Quizlet, усуваючи при цьому фінансові обмеження та надлишкову когнітивну геймифікацію.

Таблиця 1.1 – Порівняльний аналіз систем вивчення іноземної лексики

Критерій порівняння	**Anki**	**Quizlet**	**Duolingo**	**LernKarten (розроблюваний проєкт)**
1. Алгоритм інтервальних повторень (SRS)	Так (повноцінний алгоритм SM-2)	Ні (лише лінійне повторення у базовій версії)	Частково (прихована логіка без керування)	Так (модифікований алгоритм SM-2)
2. Керування контентом (власні колоди)	Так (необмежено, але складний інтерфейс)	Так (інтуїтивно зрозуміло)	Ні (лише жорстко задані курси)	Так (гнучке створення та редагування)
3. Масовий імпорт даних (CSV, Excel, текст)	Так (потребує попереднього форматування)	Так (лише базові формати)	Ні (не підтримується архітектурою)	Так (швидкий текстовий парсинг)
4. Голосовий синтез (Text-to-Speech)	Ні (потребує ручного завантаження аудіофайлів)	Так (реалізовано через платні комерційні API)	Так (професійний студійний запис)	Так (інтеграція безкоштовного Web Speech API)
5. Рівень гейміфікації	Низький (відсутня мотиваційна складова)	Середній (наявність міні-ігор)	Дуже високий (ліги, бали, ударні режими)	Низький (фокус на академічному вивченні без відволікань)
6. Кросплатформеність (Web, Mobile)	Потребує встановлення окремих десктоп/ мобільних клієнтів	Так (повноцінний вебзастосунок)	Так (вебзастосунок та мобільні застосунки)	Так (адаптивний вебзастосунок, доступний з будь-якого браузера)
7. Сучасність UI/UX дизайну	Застарілий (стиль початку 2010-х років)	Сучасний (мінімалізм)	Сучасний (велика кількість анімацій)	Сучасний (використання концепції Glassmorphism)
8. Фінансова модель та монетизація	Безкоштовно (окрім застосунку для iOS)	Freemium (агресивна реклама та платні функції)	Freemium (обмеження «життів», нав’язлива реклама)	Повністю безкоштовно (відсутність реклами)

1.2 Когнітивні механізми та наукове обґрунтування методу інтервальних повторень у комп’ютерному навчанні (CALL)

1.2.1 Ефект розподіленого навчання та ефект тестування

Фундаментальною науковою базою для розробки систем інтервальних повторень (Spaced Repetition Systems, SRS) є два ключові феномени

когнітивної психології: ефект розподіленого навчання (Spacing Effect) та ефект тестування (Testing Effect). Дослідження засвоєння лексики другої мови переконливо доводять, що розподілене в часі повторення слова є значно ефективнішим для запам’ятовування, ніж концентроване (масове) повторення за один безперервний проміжок часу [16].

Суть ефекту розподіленого навчання полягає в тому, що коли інформація повторюється на межі її забування, мозок змушений докладати більше когнітивних зусиль для її вилучення з пам’яті. Це додатково підсилюється ефектом тестування: активне пригадування відповіді (як це реалізовано у форматі словникових карток), а не просте пасивне перечитування матеріалу, багаторазово підвищує ймовірність того, що інформація перейде до довгострокової пам’яті [15].

1.2.2 Ефективність SRS у програмних середовищах

Інтеграція алгоритмів інтервальних повторень у програмні засоби сформувала окремий потужний напрям у галузі комп’ютерного навчання мов (Computer-Assisted Language Learning – CALL). За даними масштабних експериментів, розробка обчислювальних алгоритмів розподіленого повторення дозволяє гнучко адаптувати навчальний процес під динаміку пам’яті кожного окремого користувача [16].

Емпіричні дослідження підтверджують, що використання комп’ютерних інструментів на базі SRS кардинально змінює динаміку засвоєння лексики. Зокрема, Т. Наката (T. Nakata) у своєму комплексному дослідженні програм для вивчення лексики зазначає, що алгоритмічна оптимізація інтервалів максимізує ефективність навчання, хоча багато сучасних систем все ще мають простір для архітектурного вдосконалення інтерфейсу [17]. Це додатково підтверджує наукову та практичну значущість розробки гнучких рішень, таких як застосунок «LernKarten».

1.2.3 Еволюція математичних моделей: від SM-2 до FSRS

Алгоритмічним ядром більшості сучасних систем SRS залишається математична модель СуперМемо-2 (SM-2), розроблена П. Возняком (P. Wozniak) у 1990 році [5]. Цей алгоритм запровадив науковий підхід на основі даних, який планує перевірки безпосередньо перед тим, як користувач може забути слово, обчислюючи складність кожної окремої картки для запобігання перевантаженню користувача [5].

В останні роки наукова спільнота продовжує вдосконалювати математичний апарат SRS. Наприклад, сучасний алгоритм FSRS (Free Spaced Repetition Scheduler), запропонований Дж. Є (Jarrett Ye), використовує методи машинного навчання та стохастичні алгоритми найкоротшого шляху для оптимізації графіків повторень на основі великих масивів даних [18]. Проте класичний алгоритм SM-2 залишається де-факто індустріальним стандартом для незалежних вебзастосунків завдяки своїй обчислювальній легкості та відсутності потреби у великих серверних потужностях для машинного навчання.

1.3 Теоретичні основи методу інтервальних повторень

Для ефективної програмної реалізації карткового методу необхідно враховувати психологічні та фізіологічні особливості людської пам’яті. Центральною теоретичною основою в цій галузі є дослідження німецького експериментального психолога Германа Еббінгауза, проведені наприкінці XIX століття. Під час серій контрольованих експериментів над вивченням беззмістовних складів Еббінгауз сформулював математичну закономірність, відому як «крива забування» (Forgetting Curve).

Крива забування описує експоненційну втрату отриманої інформації з плином часу, якщо відсутні спроби її свідомого повторення. Математично цей

процес можна апроксимувати експоненційною залежністю (рівнянням Еббінгауза):

$$R = e^{\frac{-t}{S}},$$

де $R(t) \in [0,1]$ – коефіцієнт утримання інформації в пам’яті (retrievability);

t – час, що минув з моменту вивчення;

S – відносна міцність пам’яті (memory strength / stability), яка визначає індивідуальну швидкість згасання нейронних зв’язків для конкретної інформації.

Дослідження Еббінгауза показали, що вже протягом перших 20 хвилин після первинного сприйняття втрачається до 42% інформації, через годину – більше 55%, а через місяць у довгостроковій пам’яті залишається не більше 20–25% матеріалу (див. рис. 1.1).

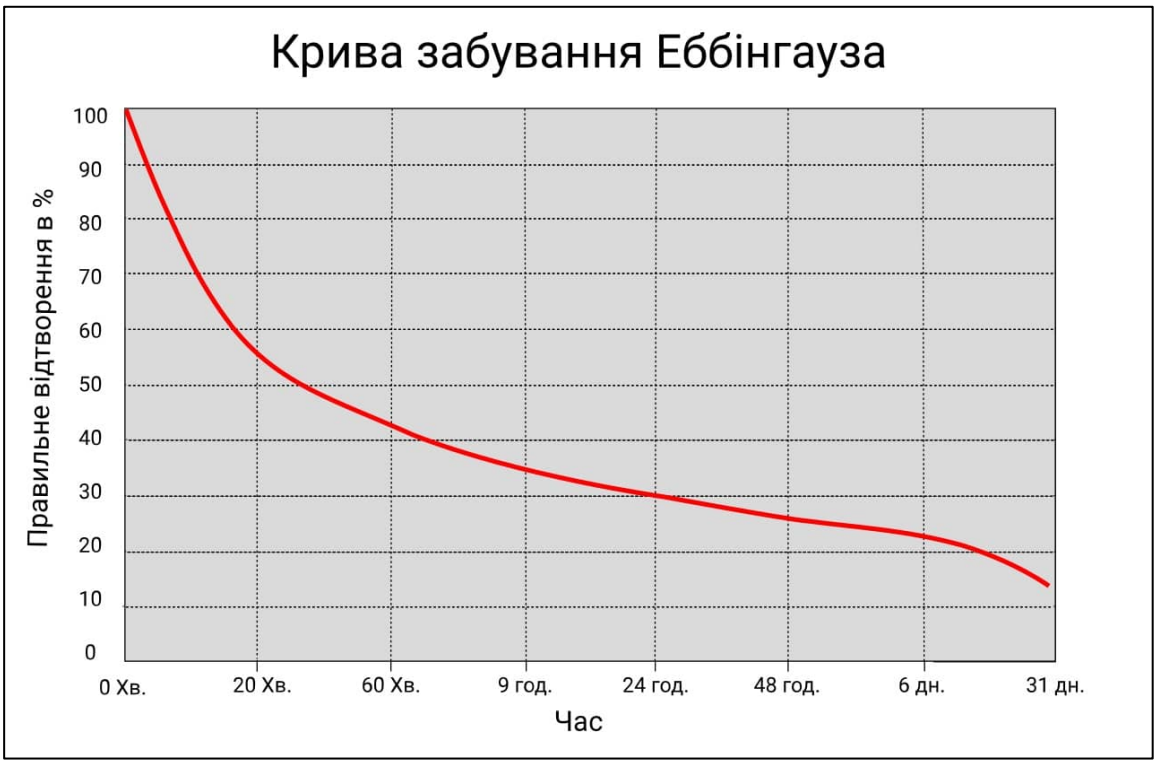


Рисунок 1.1 – Вплив інтервальних повторень на криву забування Еббінгауза [14]

Для подолання цього фізіологічного обмеження у 1970-х роках німецький інженер та популяризатор науки Себастьян Лейтнер запропонував першу механічну систему інтервальних повторень, яка базувалася на використанні картонних карток та фізичних ящиків (Leitner System). Картки переміщувалися між ящиками залежно від успішності відповіді користувача: правильна відповідь переводила картку в наступний ящик, який перевірявся рідше, а помилка примусово повертала картку в перший ящик для щоденного повторення.

З появою обчислювальної техніки механічний метод Лейтнера був замінений точними комп'ютерними алгоритмами. Найбільш відомим та математично обґрунтованим став алгоритм SuperMemo-2 (SM-2), розроблений польським дослідником Петром Возняком у 1987 році[5]. Алгоритм SM-2 оптимізує час навчання, розраховуючи індивідуальні часові інтервали для кожної окремої одиниці інформації (картки) на основі історії відповідей користувача.

В основі математичного апарату класичного SM-2 лежать три змінні:

а) поточний порядковий номер повторення (n);

б) коефіцієнт легкості картки (EF – Ease Factor), який відображає суб'єктивну складність слова для студента (початкове значення EF встановлюється рівним 2.5);

в) інтервал повторення (I) у днях.

Розрахунок перших трьох інтервалів здійснюється за фіксованими емпіричними правилами, а всі наступні інтервали визначаються як добуток попереднього інтервалу на поточний коефіцієнт легкості за формулою:

$$I(1) = 1,$$
$$I(2) = 6,$$
$$I(n) = I(n-1) \times EF \text{ при } n > 2.$$

Після кожного показу картки користувач виставляє оцінку якості своєї відповіді за шкалою від 0 до 5. На основі цієї оцінки q (де $q \in \{0, 1, 2, 3, 4, 5\}$)

відбувається динамічний перерахунок коефіцієнта легкості для наступного циклу за формулою:

$$EF_{new} = EF_{old} + \big(0.1 - (5 - q) \times (0.08 + (5 - q) \times 0.02)\big).$$

Важливо зазначити, що для запобігання надмірному зменшенню інтервалів (так званого «пекла інтервалів») алгоритм встановлює жорстку нижню межу коефіцієнта: EF ≥ 1.3.

Якщо користувач дає низьку оцінку (q < 3), це свідчить про забування навчального матеріалу. У такому випадку алгоритм примусово скидає номер повторення (n) до нуля, а інтервал I стає рівним 1 дню. Вибір саме одноденного терміну для першого повторення після забування є класичним емпіричним правилом, встановленим розробниками алгоритму під час масових практичних експериментів. Це правило базується на необхідності екстреної консолідації та повторного закріплення послабленого нейронного зв’язку вже наступної доби (поки інформація остаточно не зникла з пам’яті), що змушує систему негайно повернути картку в активний робочий цикл навчання.

У межах цієї бакалаврської роботи класичний алгоритм SM-2 було модифіковано та адаптовано для роботи у вебсередовищі (впроваджено трирівневу систему оцінювання швидкого вибору замість шестирівневої, а також розраховано мікроінтервали у хвилинах для повторення слів безпосередньо всередині поточної сесії навчання), що дозволило значно підвищити швидкість взаємодії користувача з інтерфейсом застосунку «LernKarten».

1.4 Постановка задачі на розробку програмного продукту

На основі аналізу предметної області та теоретичних відомостей було сформульовано технічне завдання на розробку вебзастосунку «LernKarten».

Метою проєкту є створення програмного комплексу для оптимізації процесу вивчення іноземних мов за допомогою карткового методу та алгоритмів інтервальних повторень.

Для досягнення поставленої мети необхідно вирішити комплекс взаємопов’язаних інженерних та архітектурних завдань. Першочергово проєкт передбачає проєктування клієнт-серверної архітектури системи з повною ізоляцією компонентів Frontend та Backend, що забезпечить масштабованість вебзастосунку та спростить його подальше супроводження. Інформаційне забезпечення системи реалізується через розробку реляційної структури бази даних, призначеної для збереження відомостей про користувачів, їхні персональні колоди, навчальні картки та індивідуальні метрики прогресу вивчення слів.

Функціональний контур системи має поєднувати кілька ключових підсистем. Підсистема безпеки та адміністрування повинна забезпечувати реєстрацію та захищену авторизацію користувачів на основі сучасних вебтокенів (JWT). Інструменти управління контентом мають надавати можливість створення, редагування та видалення тематичних колод, а також містити спеціалізований модуль додавання слів із підтримкою граматичних категорій (зокрема, артиклів для німецької мови) та прикладів контекстного використання. Для оптимізації взаємодії користувача з системою передбачено механізм автоматичної адаптації, що полягає у генерації стартової колоди «Базові дієслова» під час першого входу.

Окрему увагу приділено модулю масового імпорту даних із текстових та табличних форматів для автоматизації наповнення профілю. Навчальний процес безпосередньо реалізується через інтерактивний тренажер із тривимірною анімацією перевороту картки, системою оцінювання знань за трьома рівнями складності та інтеграцією синтезу мовлення (Text-to-Speech) для засвоєння правильної вимови. Інтерфейс користувача повинен відповідати концепціям адаптивного дизайну для коректного відображення на мобільних і десктопних пристроях, а також технологіям сучасного графічного оформлення

(Glassmorphism) для забезпечення високої швидкості відгуку. Фінальним етапом роботи є розгортання програмного комплексу у глобальній мережі з налаштуванням процесів безперервної інтеграції та автоматичного деплою (CI/CD) для забезпечення цілодобової доступності продукту.

Графічно функціональні вимоги до системи, ролі користувачів та їхню взаємодію з основними модулями застосунку зображено на діаграмі прецедентів (рис. 1.2).

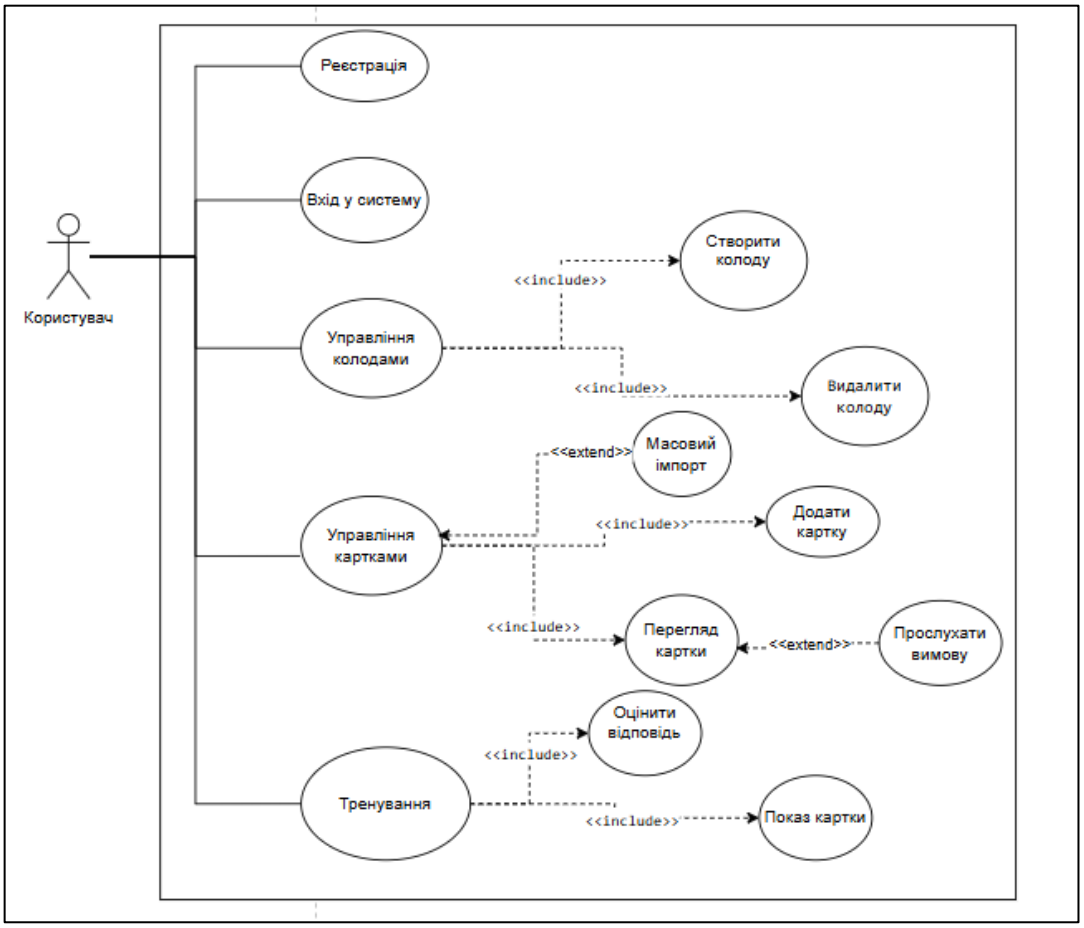


Рисунок 1.2 – Діаграма прецедентів (Use Case) вебзастосунку

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Обґрунтування вибору стеку технологій

Проєктування сучасного розподіленого вебзастосунку потребує виваженого підходу до вибору технологічного стеку, оскільки від цього залежать показники масштабованості системи, швидкість обробки запитів, безпека даних та швидкість безпосередньої розробки (Time-to-Market). Для реалізації програмного комплексу «LernKarten» було обрано концепцію єдиного технологічного простору мови програмування JavaScript (Full-Stack JavaScript розробка). Це дозволяє використовувати однорідні структури даних (JSON) на всіх рівнях архітектури системи, спрощує серіалізацію/десеріалізацію об’єктів та усуває необхідність контекстного перемикання між різними синтаксичними парадигмами під час проєктування Frontend та Backend компонентів.

Під час проєктування клієнтської частини (Frontend) було проведено порівняльний інженерний аналіз трьох провідних SPA (Single Page Application) бібліотек та фреймворків: Angular, Vue.js та React.

Фреймворк Angular від компанії Google представляє собою монументальне рішення з жорсткою архітектурною структурою (Opinionated Framework). Він містить вбудовані інструменти для маршрутизації, валідації форм та впровадження залежностей (Dependency Injection). Проте Angular має надзвичайно високий поріг входження, вимагає обов’язкового використання TypeScript та створює значні накладні витрати (overhead) у вигляді великого обсягу фінального завантажувального пакету (bundle size), що критично для мобільних пристроїв із низькою швидкістю інтернет-з’єднання.

Фреймворк Vue.js є більш гнучким рішенням, проте його екосистема є фрагментованою, а більшість критично важливих плагінів підтримуються виключно спільнотою, що створює ризики несумісності при оновленні версій.

Бібліотека React від компанії Meta була обрана як оптимальний інструмент завдяки її декларативному підходу, компонентно-орієнтованій архітектурі та концепції Віртуального DOM (Virtual DOM). Virtual DOM діє як легка копія реального DOM-дерева браузера. Коли змінюється стан застосунка (наприклад, при перевертанні картки користувачем), React обчислює різницю (diffing алгоритм) між поточним станом інтерфейсу та новим, і точково оновлює лише змінені HTML-елементи, не перемальовуючи всю сторінку. Це забезпечує миттєвий відгук інтерфейсу (60 кадрів на секунду) та високу плавність анімацій.

Замість застарілого інструменту збірки Create React App (CRA), який базується на Webpack, було обрано інструмент нового покоління Vite [7]. На відміну від Webpack, який перед запуском локального сервера збирає і компілює весь проєкт повністю (що займає багато часу), Vite використовує концепцію нативних ES-модулів (ESM), що підтримуються сучасними браузерами. Робота Vite базується на принципі On-Demand компіляції: сервер запускається миттєво, а компіляція конкретного файлу коду відбувається лише тоді, коли браузер безпосередньо запитує його через HTTP. Механізм швидкого гарячого перезавантаження (HMR – Hot Module Replacement) у Vite працює за лінійний час, незалежно від загального розміру проєкту, що оптимізує процес тестування клієнтського інтерфейсу.

Під час проєктування серверної частини (Backend) розглядалися платформи Node.js (Express.js), Python (Django) та Java (Spring Boot).

Платформа Java (Spring Boot) забезпечує найвищу стабільність у корпоративному сегменті, проте вона є занадто ресурсомісткою та вимагає великої кількості конфігураційного коду (boilerplate code), що є недоцільним для легких REST API.

Платформа Python (Django) містить потужний вбудований інструментарій, але її синхронна природа потребує складного налаштування асинхронних черг (Celery/Redis) для обробки великої кількості паралельних з’єднань.

Платформа Node.js із фреймворком Express.js була обрана завдяки її подійній, асинхронній, неблокуючій моделі введення-виведення (Event Loop) [3]. Черга подій (Event Loop) дозволяє Node.js обробляти тисячі одночасних мережевих запитів в одному потоці виконання процесора, делегуючи важкі операції (такі як звернення до бази даних або зчитування файлів) системним потокам операційної системи. Це робить Express.js ідеальним інструментом для побудови високопродуктивних та легких REST API, які миттєво віддають дані у форматі JSON.

Як систему керування базами даних (СКБД) розглядалися реляційна база PostgreSQL та нереляційна (NoSQL) базу MongoDB. Дослідження предметної області показало, що сутності системи «LernKarten» мають строгі логічні взаємозв’язки. Процес розрахунку інтервалів навчання безпосередньо залежить від реляцій між ідентифікатором користувача, колодою, карткою та часовими мітками її перевірки. Використання NoSQL бази MongoDB призвело б до дублювання даних (denormalization) та ризику втрати цілісності у разі каскадних змін.

СКБД PostgreSQL, розгорнута на хмарній платформі Neon.tech, забезпечує повну відповідність вимогам ACID (Atomicity, Consistency, Isolation, Durability), підтримує складні SQL-запити з аналітичними об’єднаннями (JOIN) та індексування за часовими мітками (B-Tree індекси), що критично для швидкої вибірки карток на повторення. Платформа Neon.tech надає архітектуру Serverless Postgres із можливістю динамічного масштабування обчислювальних ресурсів та безпечним шифрованим підключенням через протокол SSL/TLS [10].

2.2 Проєктування архітектури клієнт-серверної взаємодії

Архітектурний каркас вебзастосунку базується на децентралізованому патерні розподілених систем, де клієнтська частина (Frontend, SPA) та

серверна логіка (Backend, API) є повністю ізольованими програмними модулями. Взаємодія між ними здійснюється виключно через мережу Інтернет за допомогою архітектурного стилю REST (Representational State Transfer) через протокол HTTPS.

Основні обмеженнями (принципами) REST архітектури, які було впроваджено в проєкт, описано нижче.

Клієнт-серверна модель: поділ інтерфейсу та рівня збереження даних підвищує переносимість інтерфейсу користувача на інші платформи та покращує масштабованість серверних компонентів.

Відсутність стану (Stateless): кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для його успішної інтерпретації та виконання. Сервер не зберігає інформацію про контекст сесії користувача у своїй оперативній пам’яті.

Єдиний інтерфейс (Uniform Interface): всі ресурси ідентифікуються за допомогою унікальних URI (Uniform Resource Identifier), а маніпуляції над ними виконуються за допомогою стандартних HTTP-методів: GET (читання), POST (створення), PUT/PATCH (модифікація), DELETE (видалення).

Для реалізації принципу Stateless та забезпечення безпеки доступу до захищених REST-маршрутів було спроєктовано систему автентифікації на базі криптографічних токенів JWT (JSON Web Token). Використання класичних сесій на основі Cookies та сесійних файлів на сервері у хмарних бессерверних середовищах є неефективним, оскільки при масштабуванні сервера запити можуть потрапляти на різні інстанси (вузли), які не мають доступу до єдиної оперативної пам’яті сесій. JWT повністю вирішує цю інженерну проблему.

Криптографічний токен JWT має строгу трьохкомпонентну структуру, де кожна частина розділена крапкою та закодована за допомогою алгоритму Base64Url:

– Header (заголовок): містить інформацію про тип токена (JWT) та алгоритм шифрування підпису (наприклад, HMAC SHA256);
– Payload (корисне навантаження): містить безпосередні дані про сесію

користувача (ідентифікатор користувача user_id, email) та службові часові мітки (час створення токена iat, час закінчення дії токена exp);

– Signature (цифровий підпис): генерується сервером шляхом взяття хешу від закодованого заголовку, закодованого корисного навантаження та секретного ключа (process.env.JWT_SECRET), який відомий лише серверній частині.

Алгоритм взаємодії клієнта та сервера при обробці захищених даних спроєктовано в такий спосіб. Після успішного введення облікових даних на клієнтській частині, серверна частина генерує JWT токен і повертає його клієнту у форматі JSON. Клієнтський додаток React перехоплює токен і записує його в локальне сховище браузера localStorage.

При виконанні будь-яких наступних запитів (наприклад, відкриття вікна тренування або створення нової картки), на клієнтській частині спрацьовує механізм перехоплювачів (Interceptors) HTTP-бібліотеки Axios. Він автоматично зчитує токен із пам’яті браузера і додає його в HTTP-заголовок Authorization за стандартом Bearer Token Authorization Bearer <унікальний_код_токена>.

На стороні сервера Express.js всі захищені маршрути заблоковані спеціальним модулем проміжного контролю – Auth Middleware (функція authenticateToken). Коли запит надходить на сервер, Middleware перехоплює його до того, як він потрапить у фінальний контролер логіки, витягує рядок токена із заголовків та виконує його дешифрування за допомогою функції jwt.verify(token, secret).

Якщо токен було модифіковано зловмисником (підпис не збігається) або термін його дії закінчився, Middleware негайно перериває виконання запиту та повертає клієнту статус-відповідь 401 Unauthorized або 403 Forbidden. Якщо токен валідний, з його Payload витягується реальний user_id, прикріплюється до об’єкту запиту (req.user), і управління передається далі контролеру серверної частини, який виконує безпечний SQL-запит до бази даних саме для цього конкретного користувача.

Для дотримання принципів об’єктно-орієнтованого проєктування бізнес-логіку системи було формалізовано у вигляді діаграми класів предметної області рис. 2.1. А для відображення фізичної взаємодії між клієнтом, сервером та базою даних було спроєктовано окрему діаграму компонентів (рис. 2.2).

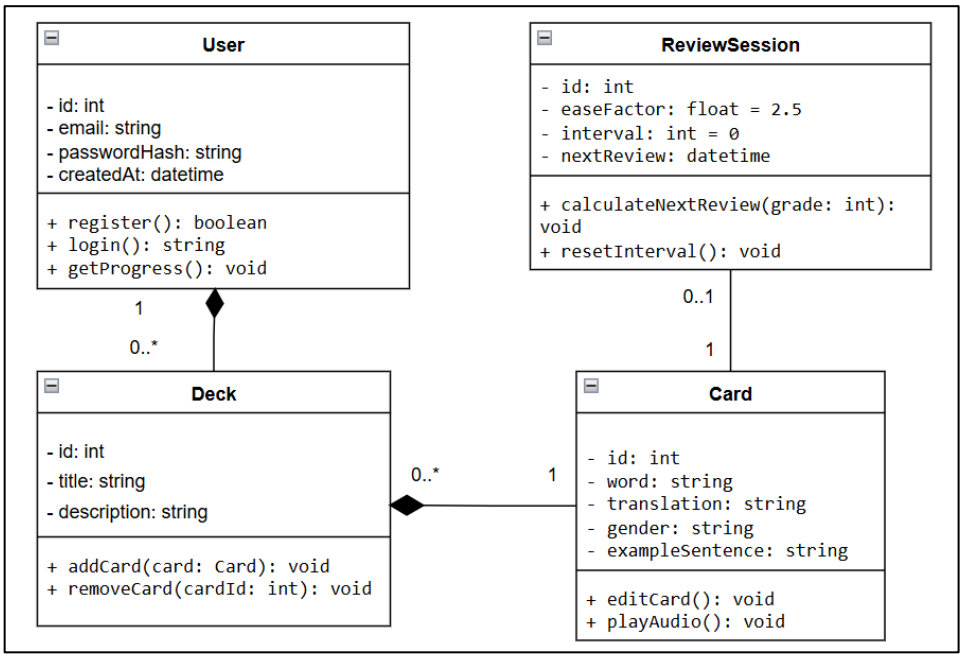


Рисунок 2.1 – Діаграма класів предметної області

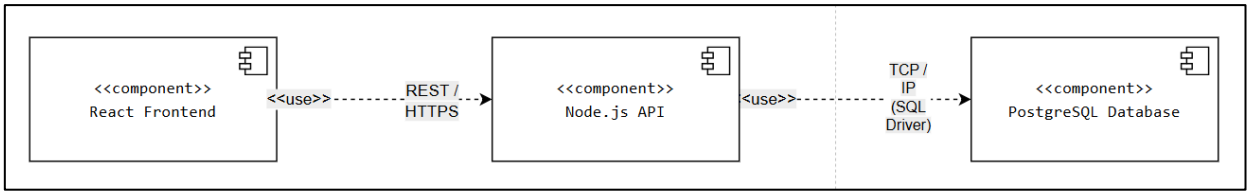


Рисунок 2.2 – Діаграма компонентів системи

2.3 Проєктування структури бази даних

Для надійного збереження даних, забезпечення високої швидкості вибірки карток на повторення та уникнення структурних аномалій було спроєктовано схему реляційної бази даних, розгорнуту під управлінням СКБД PostgreSQL. Процес проєктування базувався на принципах нормалізації даних.

Усі створені таблиці відповідають вимогам Третьої нормальної форми (3NF), що цілком унеможливлює дублювання інформації (надмірність) та гарантує логічну незалежність атрибутів від ключів. При декомпозиції сутностей використовувалися класичні методології проєктування реляційних систем [12].

Зв'язки між таблицями реалізовано за допомогою механізму зовнішніх ключів (Foreign Keys). Критично важливою вимогою до системи навчання є підтримка цілісності посилальних даних (Referential Integrity). Для цього на всі зовнішні ключі було накладено обмеження на рівні бази даних ON DELETE CASCADE. Це архітектурне рішення гарантує, що при видаленні користувачем власного профілю або окремої тематичної колоди, ядро PostgreSQL автоматично, на рівні транзакції, видалить усі залежні картки та записи прогресу. Це превентивно захищає базу даних від накопичення «осиротілих» (орфанних) записів, які б нераціонально споживали дисковий простір хмарного сервера.

Спроєктована схема даних складається з чотирьох реляційних таблиць із наступною специфікацією атрибутів та типів.

Таблиця users (Користувачі). Ця сутність відповідає за зберігання облікових даних:

- первинним ключем є поле id типу SERIAL (автоінкрементне ціле число);
- поле email має тип VARCHAR(255) та обмеження UNIQUE, що унеможливлює реєстрацію двох профілів на одну електронну адресу;
- поле password_hash використовує тип TEXT: використання типу TEXT замість обмеженого VARCHAR є кращою практикою для збереження криптографічних хешів bcrypt, оскільки алгоритми шифрування в майбутньому можуть бути оновлені, що змінить довжину вихідного хеш-рядка;
- поле created_at (тип TIMESTAMP WITH TIME ZONE) автоматично фіксує точний час створення профілю за Гринвічем (UTC) завдяки дефолтному значенню CURRENT_TIMESTAMP.

Таблиця decks (Колоди карток). Сутність реалізує концепцію логічного групування навчального матеріалу. Має відношення «Один-до-багатьох» із таблицею users:

- первинний ключ: id (SERIAL);
- зовнішній ключ: user_id (INTEGER), який вказує на ідентифікатор власника;
- поле title (VARCHAR(150)) містить назву колоди, а поле description (TEXT) дозволяє додавати розгорнуті коментарі до курсу – усі ці поля мають обмеження NOT NULL (окрім опису).

Таблиця cards (Словникові картки). Є центральним сховищем навчального контенту. Має відношення «Один-до-багатьох» із таблицею deck:

- зовнішній ключ deck_id міцно прив'язує кожне слово до конкретної колоди;
- поля word та translation мають тип VARCHAR(255);
- додаткове поле gender (VARCHAR(10)) передбачено для збереження артиклів (наприклад, der, die, das для німецької мови);
- поле example_sentence (TEXT) містить контекстні приклади використання слова, що відповідає сучасним комунікативним методикам вивчення мов (навчання в контексті).

Таблиця progress (Прогрес вивчення). Це найскладніша транзакційна таблиця системи, яка виступає сполучною ланкою (Pivot Table) між користувачем та карткою, реалізуючи відношення «Багато-до-багатьох»:

- первинним ключем цієї таблиці є складений ключ (Composite Primary Key), що складається одразу з двох полів: (user_id, card_id) – це елегантне інженерне рішення унеможливлює ситуацію, коли для однієї картки конкретного користувача буде створено два паралельні записи прогресу;
- поле ease_factor (тип REAL, значення стандартно 2.5) зберігає поточний множник складності слова;
- поле interval (INTEGER, стандартно 0) фіксує поточний інтервал повторення у днях;

– поле next_review (TIMESTAMP) визначає точну системну дату та час, коли картка має бути знову показана в інтерфейсі тренажера.

Концептуальну модель реляційної бази даних, її сутності, атрибути та типи зв’язків між ними наочно подано у вигляді ER-діаграми (рис. 2.3).

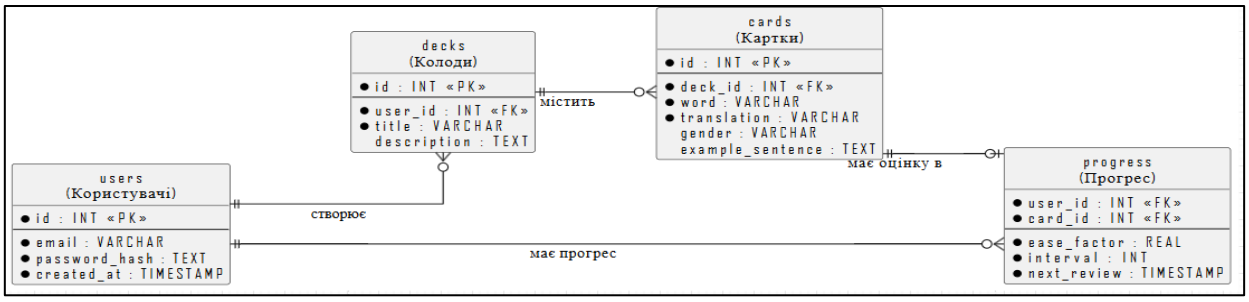


Рисунок 2.3 – ER-діаграма структури бази даних системи

2.4 Проєктування алгоритму інтервальних повторень та системи оцінювання

Для забезпечення необхідного рівня наукової обґрунтованості розробленого програмного комплексу, базовий алгоритм інтервальних повторень формалізовано у вигляді дискретної динамічної системи. Такий підхід дозволяє розглядати кожну навчальну одиницю (словникову картку) як окремий об’єкт, поточний стан якого повністю визначається передісторією навчання, а майбутній стан є детермінованою функцією поточного стану та керуючого впливу з боку користувача.

Визначимо стан системи (картки) на n-му кроці дискретного часу як вектор:

$$x_{\mathrm{n}} = (I_{\mathrm{n}}, EF_{\mathrm{n}}),$$

де I_n – поточний інтервал повторення лексичної одиниці (у днях);

EF_n – поточний коефіцієнт легкості картки (з мінімально допустимим обмеженням $EF_n \geq 1.3$); n – порядковий номер успішного кроку навчання.

Роль керівного впливу (вхідного сигналу) виконує дискретна оцінка зворотного зв’язку q, яку виставляє користувач під час сесії тестування: $q \in \{1, 2, 3\}$, де 1 відповідає стану «Забув», 2 – «Нормально», 3 – «Легко».

Загальна математична модель еволюції стану картки описується функцією переходу $x_{n+1} = f(x_n, q)$. Залежно від значення керівного параметра q, правила переходу системи до наступного стану мають такий вигляд:

а) при оцінці «Забув» (q = 1):

$$I_{n+1} = 0,$$
$$EF_{n+1} = max(1.3; EF_n – 0.2);$$

б) при оцінці «Нормально» (q = 2):

$$Якщо\ I_n = 0,\ то\ I_{n+1} = 1.\ Інакше:\ I_{n+1} = Round(I_n \times EF_n),$$
$$EF_{n+1} = EF_n;$$

в) при оцінці «Легко» (q = 3):

$$Якщо\ I_n = 0,\ то\ I_{n+1} = 4.\ Інакше:\ I_{n+1} = Round(I_n \times EF_n \times 1.3),$$
$$EF_{n+1} = EF_n + 0.15.$$

Наведена математична формалізація дозволяє чітко описати навчальний процес без використання складних матричних перетворень та безпомилково перенести логіку алгоритму у програмний код.

Проєктування та програмна імплементація алгоритму є ключовим науковим та інженерним завданням даної кваліфікаційної роботи. За математичну основу було взято модифіковану версію алгоритму СуперМемо-2 (SM-2). Оригінальний SM-2 оперує шестибальною шкалою оцінювання, що перевантажує когнітивний апарат користувача під час роботи зі смартфоном. Тому для застосунку «LernKarten» алгоритм було адаптовано під швидкісні

сесії навчання з використанням дискретної трирівневої системи зворотного зв’язку (Feedback System).

Життєвий цикл кожної картки в системі описується двома головними математичними параметрами: коефіцієнтом легкості (EF) та інтервалом повторення (I). При створенні нової картки система ініціалізує її значеннями стандартно:

$$EF = 2.5 \text{ та } I = 0 \text{ (статус "Нове слово").}$$

Після показу лицьового боку картки та спроби згадати переклад, користувач відкриває зворотну сторону і самостійно оцінює рівень своїх знань, натискаючи одну з трьох кнопок. Залежно від вибору алгоритм виконує математичний перерахунок за такими розгалуженнями.

Варіант 1. Оцінка «Забув» (Grade = 1). Ця оцінка свідчить про те, що нейронний зв’язок втрачено, і слово потрібно перевести на етап первинного заучування. Інтервал (I) примусово скидається:

$$I_{new} = 0.$$

Коефіцієнт легкості (EF) штрафується на 0.2, оскільки слово виявилося об’єктивно складним для користувача. Щоб запобігти проблемі «пекла інтервалів» (Interval Hell) – ситуації, коли коефіцієнт падає настільки низько, що картка показується нескінченно часто, – було запроваджено жорстку нижню межу коефіцієнта:

$$EF_{new} = max(1.3, EF_{old} - 0.2).$$

Час наступного показу (next_review) встановлюється за формулою: Поточний час + 10 хвилин. Це дозволяє користувачеві повторити забуте слово ще раз до завершення поточної навчальної сесії.

Варіант 2. Оцінка «Нормально» (Grade = 2). Студент згадав переклад, але це потребувало певних розумових зусиль.

Коефіцієнт легкості залишається незмінним:

$$EF_{new} = EF_{old}.$$

Це стабілізує зростання інтервалу. Якщо картка була новою (I = 0), її новий інтервал стає рівним 1 дню $I_{new} = 1$. Якщо картка вже вивчалася, інтервал розраховується за класичною експоненційною формулою SM-2:

$$I_{new} = Round(I_{old} \times EF_{old}).$$

Варіант 3. Оцінка «Легко» (Grade = 3). Слово є добре знайомим або миттєво згадується з довгострокової пам'яті.

Коефіцієнт легкості отримує бонус за ідеальну відповідь:

$$EF_{new} = EF_{old} + 0.15.$$

Інтервал розраховується за прискореною формулою з додатковим множником 1.3, що дозволяє системі рідше турбувати студента легкими словами:

$$I_{new} = Round\,(I_{old} \times EF_{old} \times 1.3).$$

Для абсолютно нових слів (I=0), якщо користувач одразу тисне «Легко», інтервал автоматично встановлюється на 4 дні.

Блок-схему алгоритмічної логіки процесу оцінювання карток, перерахунку коефіцієнтів та розгалуження умов відображено на діаграмі активностей (див. рис. 2.4).

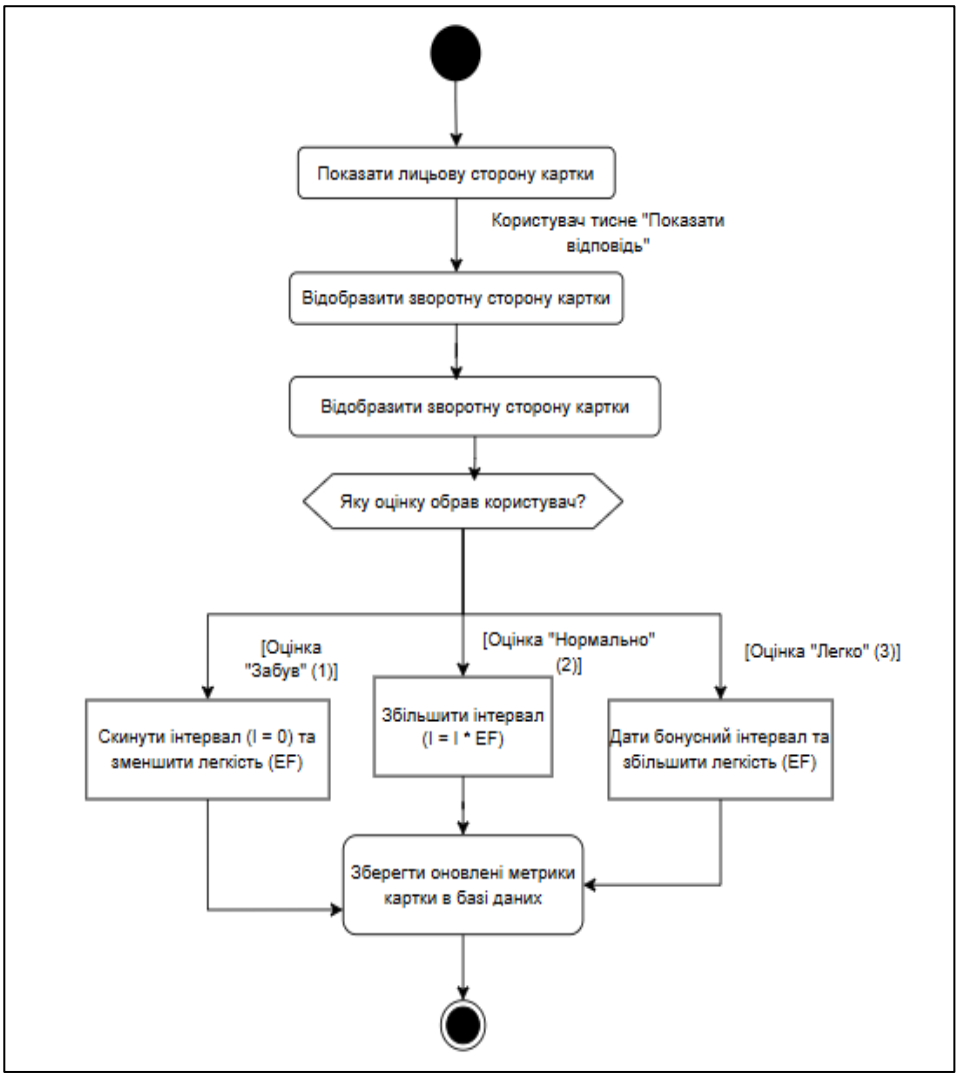


Рисунок 2.4 – Діаграма активностей (Activity Diagram) алгоритму інтервальних повторень

Складною інженерною задачею було проєктування вибірки карток. Коли користувач створює або імпортує нові слова, у таблиці progress ще немає записів для них. Для вирішення проблеми спроєктовано SQL-запит із використанням оператора LEFT JOIN та перевірки на NULL (рис. 2.5).

```
SELECT c.* FROM cards c
LEFT JOIN progress p ON c.id = p.card_id AND p.user_id = $1
WHERE c.deck_id = $2 AND (p.next_review <= NOW() OR p.next_review IS NULL)
ORDER BY p.next_review ASC NULLS FIRST
LIMIT 20
```

Рисунок 2.5

Цей запит є ядром навчального процесу. Оператор LEFT JOIN гарантує, що база даних поверне всі картки з вибраної колоди, навіть якщо вони ще не мають запису в таблиці прогресу (у такому випадку поля прогресу будуть дорівнювати NULL). Умова p.next_review <= NOW() фільтрує старі картки, час повторення яких уже настав або протермінований. Умова OR p.next_review IS NULL додає до вибірки абсолютно нові слова. Конструкція ORDER BY ... NULLS FIRST гарантує логічну пріоритетизацію: система спочатку змушує користувача вивчити партію нових карток, а потім пропонує повторити старі. Для запобігання перевантаженню оперативної пам'яті браузера розмір вибірки жорстко лімітується LIMIT 20 картками на один запит (пагінація сесії).

Хронологічну послідовність викликів та передачі даних між клієнтом, сервером і базою даних під час сесії навчання користувача зображено на діаграмі послідовностей (рис. 2.6).

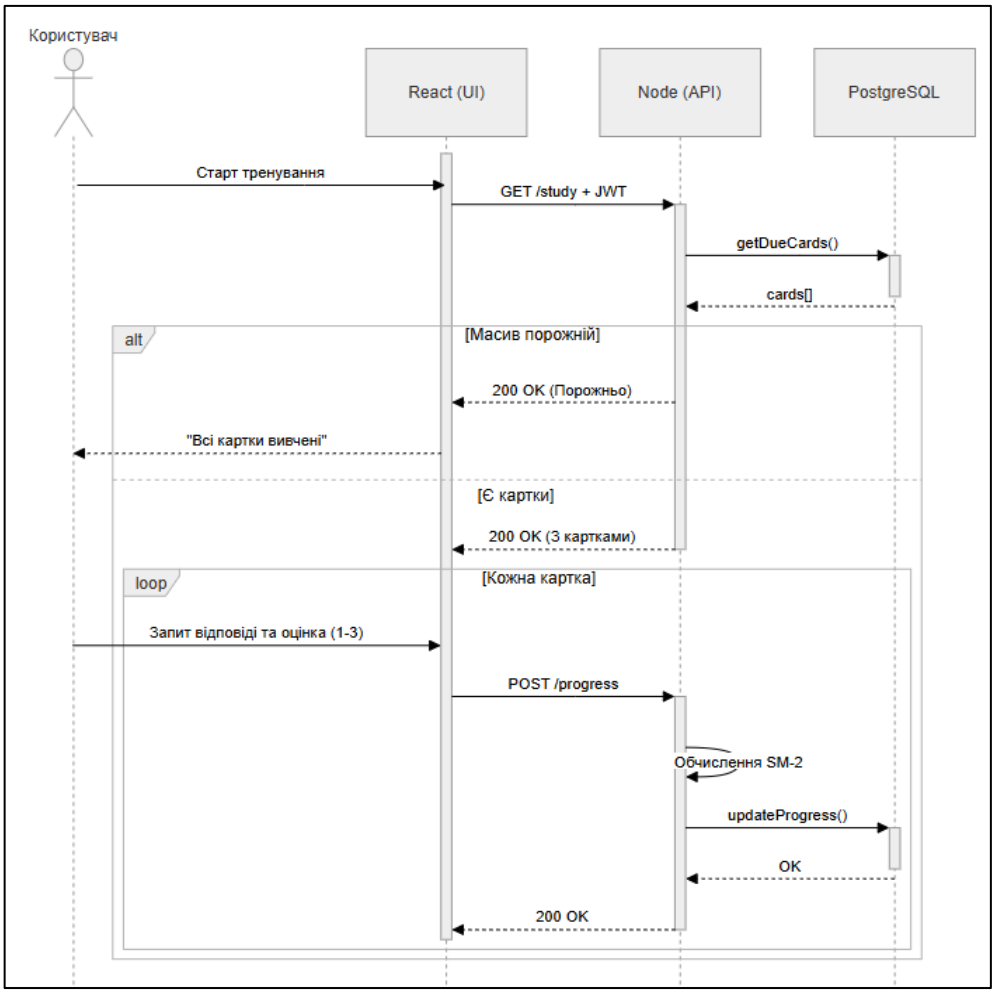


Рисунок 2.6 – Діаграма послідовностей (Sequence Diagram) процесу навчання

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЗАХИСТ МОДУЛІВ ВЕБЗАСТОСУНКУ

3.1 Розробка серверної частини та налаштування безпеки

Практичну реалізацію серверної частини (Backend) виконано мовою JavaScript у середовищі Node.js із використанням вебфреймворку Express.js [3]. Головним вхідним файлом серверного додатка є server.js, у якому відбувається ініціалізація HTTP-сервера, налаштування механізмів кросдоменних запитів (CORS) та підключення пулу з'єднань (Connection Pool) до бази даних PostgreSQL через бібліотеку pg.

Насамперед було розв'язано критично важливе завдання інженерії безпеки – захист конфіденційних даних користувачів. Зберігання паролів у відкритому текстовому вигляді у базі даних є грубим порушенням стандартів кібербезпеки (наприклад, OWASP). Тому в систему було інтегровано криптографічну бібліотеку bcrypt.

При реєстрації нового профілю система не зберігає пароль, а генерує його математичний хеш із використанням 10 раундів генерації криптографічної «солі» (Salting).

Логіка хешування та збереження облікових даних користувача реалізована в контролері автентифікації. Під час спроби входу алгоритм витягує збережений хеш із бази даних та порівнює його з уведеним паролем за допомогою функції bcrypt.compare.

Для захисту API-маршрутів (наприклад, маршрутів додавання слів або оновлення прогресу) від несанкціонованого доступу було реалізовано безсесійну автентифікацію на базі JSON Web Tokens (JWT) [6]. Усі конфіденційні налаштування, такі як секретний ключ JWT_SECRET та рядок підключення до бази даних DATABASE_URL, винесено у захищений файл конфігурації .env, який ізольовано від системи контролю версій Git.

Для перевірки валідності токенів при кожному запиті було розроблено спеціальне проміжне програмне забезпечення (Middleware). Фрагмент коду цього модуля наведено на рисунку 3.1.

Структурну взаємодію об’єктів та обмін мережевими повідомленнями під час процесу авторизації користувача та запиту захищених навчальних даних наведено на діаграмі послідовностей (див. рис. 3.2).

```
const jwt = require('jsonwebtoken');
const authenticateToken = (req, res, next) => {
 // Отримання токена із HTTP-заголовка Authorization
 const authHeader = req.headers['authorization'];
 const token = authHeader && authHeader.split(' ')[1];
 if (!token) {
  return res.status(401).json({ error: 'Доступ заборонено: відсутній токен' });
 }
 // Верифікація токена за допомогою секретного ключа
 jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
  if (err) {
   return res.status(403).json({ error: 'Недійсний або протермінований токен' });
  }
  // Прикріплення ID користувача до об'єкту запиту для наступних контролерів
  req.user = user;
  next();
 });
```

Рисунок 3.1 – Модуль проміжного контролю для перевірки JWT токена

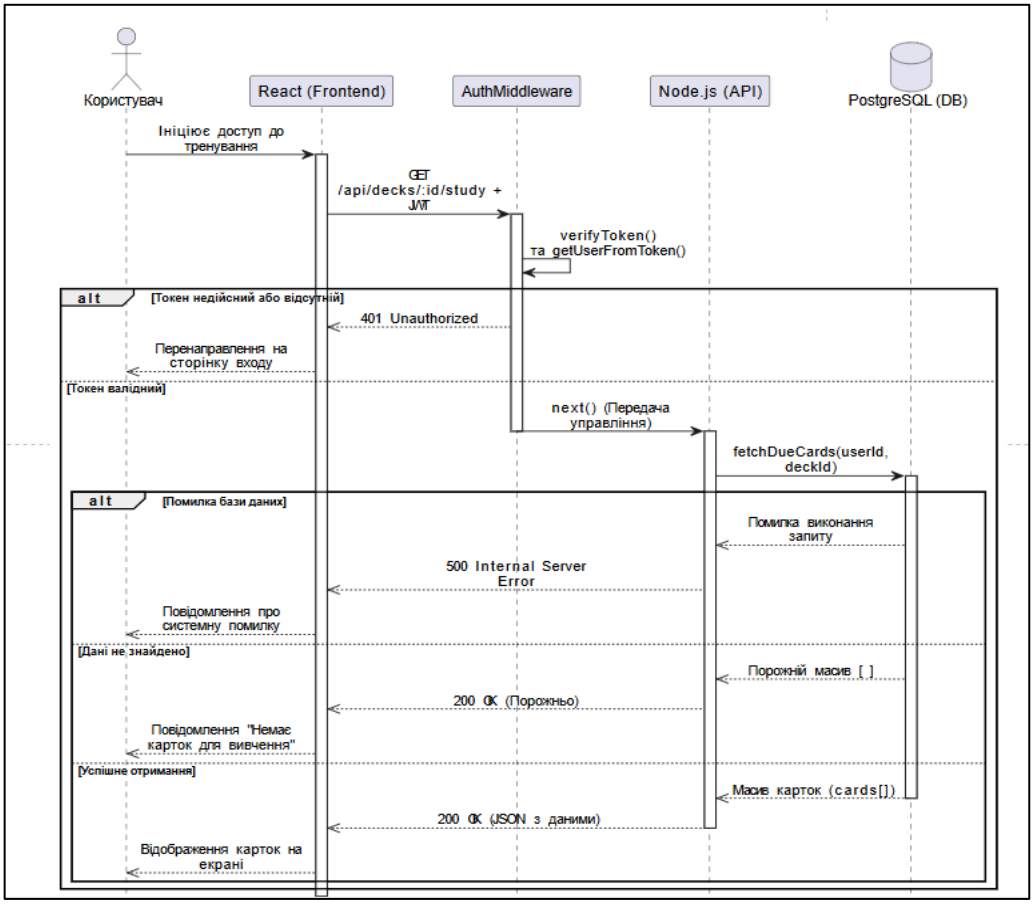


Рисунок 3.2

3.2 Розробка користувацького інтерфейсу та адаптивного дизайну

Клієнтську частину вебзастосунку розроблено на базі бібліотеки React. Архітектура клієнтської частини побудована за компонентним підходом, де кожен логічний блок інтерфейсу (навігаційна панель, список колод, окрема картка) виділено в незалежний функціональний компонент (Functional Component) із використанням React Hooks (useState, useEffect).

Візуальний інтерфейс застосунку спроєктовано з використанням сучасної парадигми вебдизайну Glassmorphism (ефект матового скла) [11]. Цей стиль створює візуальну глибину та ієрархію елементів без використання важких графічних файлів, що позитивно впливає на швидкість завантаження (LCP – Largest Contentful Paint). Основний ефект скла досягається шляхом комбінації CSS-властивостей напівпрозорого фону rgba та фільтру розмиття заднього фону backdrop-filter. Фрагмент CSS-коду базової скляної панелі наведено на рисунку 3.3.

```
.glass-panel {
 background: rgba(255, 255, 255, 0.1);
 backdrop-filter: blur(12px);
 -webkit-backdrop-filter: blur(12px);
 border: 1px solid rgba(255, 255, 255, 0.2);
 border-radius: 16px;
 box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1);
```

Рисунок 3.3 – CSS-клас для реалізації ефекту Glassmorphism

У процесі тестування застосунку в браузерах на базі рушія Chromium виявлено критичний візуальний дефект: при увімкненні темної теми автозаповнені поля введення (інпути) примусово заливалися білим фоном браузера, через що світлий текст ставав невидимим. Проблему було розв’язано впровадженням жорсткого CSS-перевизначення (CSS Override) внутрішніх тіней елемента за допомогою псевдоселектора :-webkit-autofill.

Найбільш складною UI-компонентою є екран тренажера (Study.jsx), який реалізує тривимірну (3D) анімацію перевороту картки. Картка фізично складається з двох контейнерів (лицьова та зворотна сторони), розташованих абсолютно один над одним. Для приховування зворотної сторони під час анімації застосовано властивість backface-visibility: hidden. При натисканні на картку React-компонент змінює стан isFlipped, що динамічно додає CSS-клас, який виконує трансформацію transform: rotateY(180deg), створюючи плавний ефект перевороту в 3D-просторі.

3.3 Інтеграція додаткових функцій

Для підвищення практичної цінності продукту, покращення користувацького досвіду (UX) та забезпечення безбар'єрності навчання в застосунок було інтегровано три важливі інженерні підсистеми: модуль масового імпорту, систему автоматичної адаптації користувача та голосовий синтез тексту (Text-to-Speech).

Голосове відтворення іншомовних слів (Text-to-Speech). Однією з ключових вимог до систем вивчення мов є можливість прослухати правильну вимову лексики. Використання зовнішніх комерційних API (наприклад, Google Cloud TTS) призвело б до фінансових витрат. Тому архітектурним рішенням стала інтеграція нативного браузерного інтерфейсу Web Speech API [8]. У компоненті Study.jsx реалізовано функцію, яка ініціалізує об'єкт SpeechSynthesisUtterance, встановлює мовну локаль та викликає системний синтезатор мовлення операційної системи користувача (див. рис. 3.4).

Модуль масового імпорту даних. Щоб користувачі не витрачали час на ручне введення кожного слова, реалізовано алгоритм автоматичної обробки тексту. Користувач має можливість скопіювати список слів (наприклад, з таблиці Microsoft Excel) та вставити його в текстове поле. JavaScript-функція на клієнтській частині розбиває суцільний текст на масив рядків за допомогою

методу .split(‘\n’), а потім у циклі шукає роздільники (тире, табуляцію або кому), формуючи масив JSON-об’єктів. Цей масив відправляється на сервер одним HTTP POST-запитом, що суттєво зменшує мережеве навантаження.

```
const playAudio = (text, language = ‘de-DE’) => {
  if (‘speechSynthesis’ in window) {
    window.speechSynthesis.cancel();
    const utterance = new SpeechSynthesisUtterance(text);
    utterance.lang = language;
    utterance.rate = 0.9;   // Сповільнена швидкість для навчання
    window.speechSynthesis.speak(utterance);
  } else {
    console.error(‘Синтез мовлення не підтримується вашим браузером’);
  }
};
```

Рисунок 3.4 – JavaScript функція синтезу мовлення

Система автоматичної адаптації користувача (Auto-onboarding). Дослідження UX показують, що порожній екран після реєстрації різко знижує показник утримання користувачів (Retention Rate). Для вирішення цієї проблеми розроблено автоматизований SQL-скрипт на стороні сервера. У момент успішної обробки маршруту /api/register, сервер не лише створює запис у таблиці users, але й автоматично створює для нового ID тематичну колоду «Базові дієслова» і пакетно (Batch Insert) заповнює її найвживанішою німецькою лексикою. Завдяки цьому, одразу після першого входу в систему, студент бачить готовий до використання навчальний матеріал і може негайно розпочати тестування алгоритму інтервальних повторень.

3.4 Посібник користувача

Для забезпечення успішної, безпомилкової та інтуїтивно зрозумілої експлуатації розробленого програмного комплексу «LernKarten» кінцевими

користувачами було розроблено детальний посібник користувача. Цей підрозділ виконує роль технічного посібника, що описує графічний інтерфейс, архітектуру навігації, призначення всіх інтерактивних елементів (кнопок, полів введення, лічильників) та покрокові сценарії взаємодії із системою.

Програмний продукт функціонує як адаптивний вебзастосунок (SPA), що дозволяє використовувати його на будь-яких пристроях (десктопи, планшети, смартфони) під управлінням сучасних браузерів на базі рушіїв Chromium, WebKit або Gecko. Нижче наведено детальний розбір інтерфейсних рішень та функціональних можливостей системи на основі графічних знімків екрана (скриншотів).

3.4.1 Модуль автентифікації та реєстрації користувачів

Взаємодія користувача з інформаційною системою починається зі стартового вікна входу, яке слугує захисним бар'єром для персональних даних та індивідуальних метрик довгострокового прогресу вивчення мови (див. рис. 3.5).

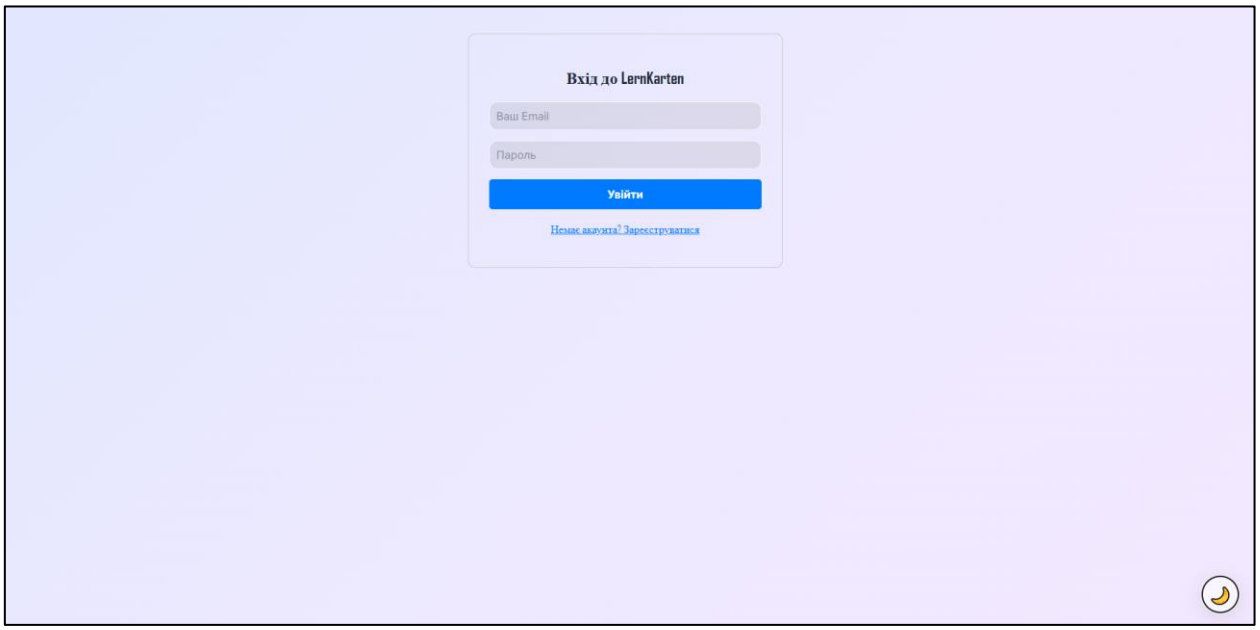


Рисунок 3.5 – Графічний інтерфейс форми автентифікації користувача

Форма автентифікації інтегрована чітко у центрі робочої області додатка у вигляді напівпрозорого контейнера, виконаного у візуальному стилі Glassmorphism. На вершині цієї панелі розташовано чіткий текстовий заголовок, який динамічно змінює свій вміст залежно від активованого стану компонента React, інформуючи людину про поточний режим роботи – авторизація або створення нового облікового запису.

Нижче заголовка послідовно розміщено два інтерактивних текстових поля введення.

Перше поле призначене для введення унікальної електронної адреси користувача і містить наочний текст-підказка. Це поле має вбудовану автоматичну валідацію на стороні клієнта, яка перевіряє наявність обов'язкового символу електронної пошти «@» та коректність структури доменного імені.

Друге текстове поле є спеціалізованою компонентою типу password, яка автоматично маскує введені символи чорними крапками, що запобігає візуальному перехопленню пароля сторонніми особами. На це поле накладено жорстке програмне обмеження щодо мінімальної довжини у шість символів. Коли наводиться курсор або при безпосередньому фокусуванні на будь-якому з цих полів, їхні контури плавно підсвічуються контрастним синім кольором, сигналізуючи про готовність системи до введення даних.

Головним керуючим елементом форми є велика інтерактивна кнопка, пофарбована в яскравий синій колір для максимального привернення уваги. Натискання цієї кнопки (або натискання клавіші Enter) запускає внутрішній обробник подій React, який збирає інформацію, тимчасово блокує інтерфейс від повторних натискань та надсилає асинхронний HTTP-запит до серверного інтерфейсу.

Для користувачів, які відвідують ресурс уперше, під кнопкою передбачено функціональне текстове посилання, яке дозволяє миттєво перемикнути форму в режим реєстрації, плавно змінюючи логіку відправки даних на відповідний безпечний маршрут.

3.4.2 Головна робоча панель керування (Dashboard)

Після успішного проходження перевірки криптографічного токена JWT на сервері, користувач автоматично перенаправляється на головну інтерактивну панель керування, відому як Dashboard (див. рис. 3.6). Цей екран є центральним координаційним хабом застосунку, інтерфейс якого розділено на три логічні зони, що забезпечують повний контроль над навчальним процесом, додаванням нових слів та переглядом статистики.

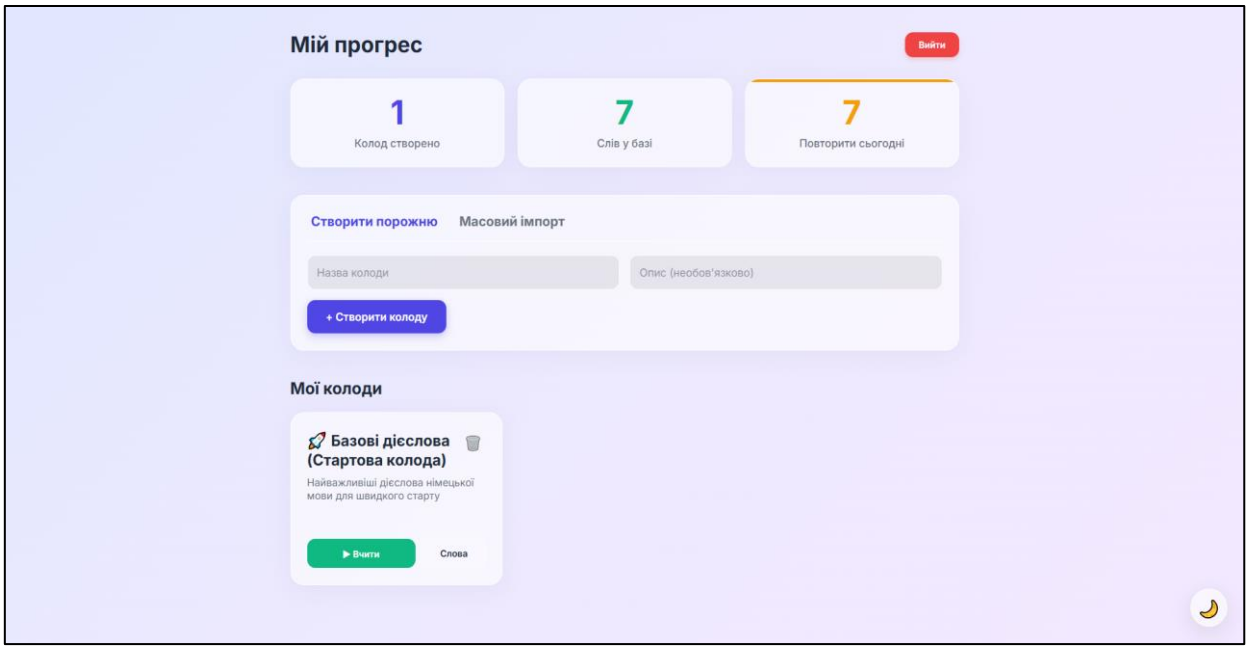


Рисунок 3.6 – Інтерфейс головної робочої панелі (Dashboard)

Верхня частина панелі представлена фіксованим навігаційним баром, на якому відображено символічний логотип системи, поточний ідентифікатор профілю та функціональну кнопку червоного кольору для безпечного виходу із системи, яка повністю очищує локальну пам’ять браузера. Одразу під панеллю навігації розгорнуто блок глобальної аналітики під назвою «Мій прогрес», що складається з трьох великих інформаційних індикаторів. Перший лічильник відображає загальну кількість створених користувачем тематичних колод, другий демонструє сумарну кількість словникових карток у базі даних, а третій лічильник є найважливішим маркером алгоритму SM-2, оскільки

показує точну кількість слів, для яких уже підійшов або протермінований час повторення на сьогодні.

По центру екрана розташовано універсальний модуль додавання контенту, реалізований за допомогою перемикання вкладок. У режимі створення порожньої колоди користувачеві пропонуються два текстових поля для введення назви та опису майбутнього курсу, а також фіолетова кнопка, натискання якої ініціює створення нового об’єкта в базі даних PostgreSQL.

Нижня зона екрана відведена під динамічну сітку карток наявних колод. Кожен блок колоди містить індивідуальну назву, текстовий опис, графічну іконку кошика для запуску процедури каскадного видалення курсу, а також дві великі кнопки: білу кнопку для переходу в режим поштучного перегляду слів та яскраву зелену кнопку «Вчити» для негайного запуску сесії інтервальних повторень.

3.4.3 Інтерфейс інтерактивного тренажера (Режим навчання)

Після активації навчального процесу через зелену кнопку колоди, вебзастосунок переходить у режим інтерактивного тренажера, який повністю ізолює користувача від зовнішніх інтерфейсних елементів задля забезпечення максимальної когнітивної концентрації. У верхній лівій частині екрана розташована навігаційна текстова кнопка для швидкого повернення на головну панель, а праворуч від неї інтегровано динамічне інформаційне табло, яке відображає залишок словникових карток у поточній черзі навчання і зменшує своє значення після кожної виставленої оцінки (див. рис. 3.7).

Робота самого тренажера побудована на чергуванні двох послідовних інтерактивних фаз. Перша фаза є демонстрацією лицьового боку великої тривимірної картки, розташованої у центрі робочої області додатка. Великим жирним шрифтом на ній написано іншомовне слово, а поруч вбудовано круглу кнопку з іконкою звуку. Натискання на цю кнопку активує функцію голосового синтезу Web Speech API, яка відтворює еталонну вимову слова

через динаміки пристрою розробника. Для переходу до наступного кроку користувачеві необхідно натиснути в будь-якому місці картки, що активує плавний 3D-переворот об’єкта на 180 градусів (див. рис. 3.8).



Рисунок 3.7 – Відображення лицьового боку картки у тренажері

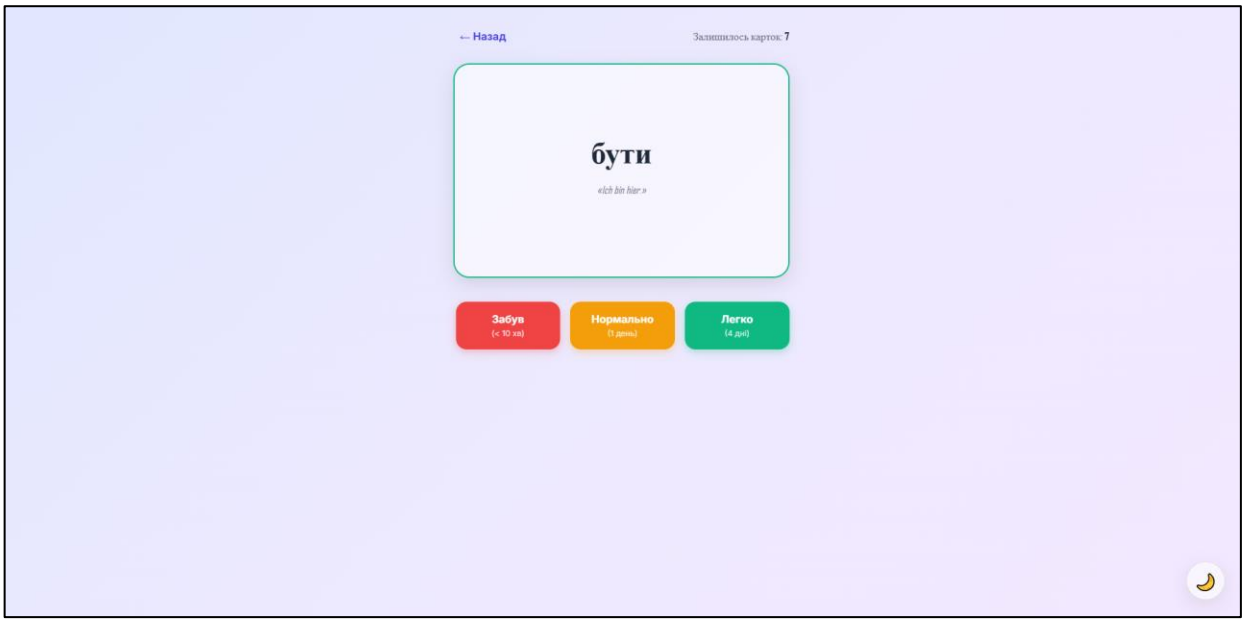


Рисунок 3.8 – Відображення зворотної сторони картки та кнопок керування алгоритмом SM-2

Друга фаза відкриває зворотну сторону картки, контур якої змінює колір на зелений. Тут користувач бачить точний переклад слова українською мовою

та реальний приклад використання цього слова у контексті речення, написаний дрібним курсивом. Одночасно під карткою з’являється керуюча панель із трьох великих кольорових кнопок для передачі оцінки алгоритму SM-2. Червона кнопка «Забув» скидає інтервал слова до нуля і повертає його в чергу показу через 10 хвилин. Жовта кнопка «Нормально» фіксує стабільне знання і планує повторення на наступний день. Зелена кнопка «Легко» додає максимальний бонус до коефіцієнта складності та переносить картку одразу на 4 дні вперед, після чого система автоматично завантажує наступне слово з бази даних.

3.4.4 Модуль масового імпорту словникових карток

Для радикального скорочення рутинних витрат часу користувачів при перенесенні великих особистих словникових списків з інших екосистем, в архітектуру інтерфейсу було інтегровано підсистему пакетного завантаження даних. Доступ до цього інструментарію реалізовано через перемикання другої вкладки на головній робочій панелі додатка, що забезпечує швидкий доступ до функції без переходу на інші сторінки застосунку.

Основну площу цього модуля займає велике інтерактивне текстове поле типу textarea, яке підтримує операції багаторядкового введення та прямого копіювання даних із буфера обміну (див. рис. 3.9). Всередині поля інтегровано чітку синтаксичну інструкцію та реальний приклад коректного оформлення рядків, де іноземне слово та його переклад мають бути чітко розділені знаком тире. Керуючим елементом модуля є синя кнопка «Імпортувати слова», розташована під текстовою областю. Натискання на неї активує розроблений клієнтський парсер, який автоматично розбиває суцільний текст на ізольовані рядки, перетворює їх на структурований масив об’єктів JSON та відправляє пакетним SQL-запитом на сервер, миттєво оновлюючи інтерфейс сітки колод користувача.

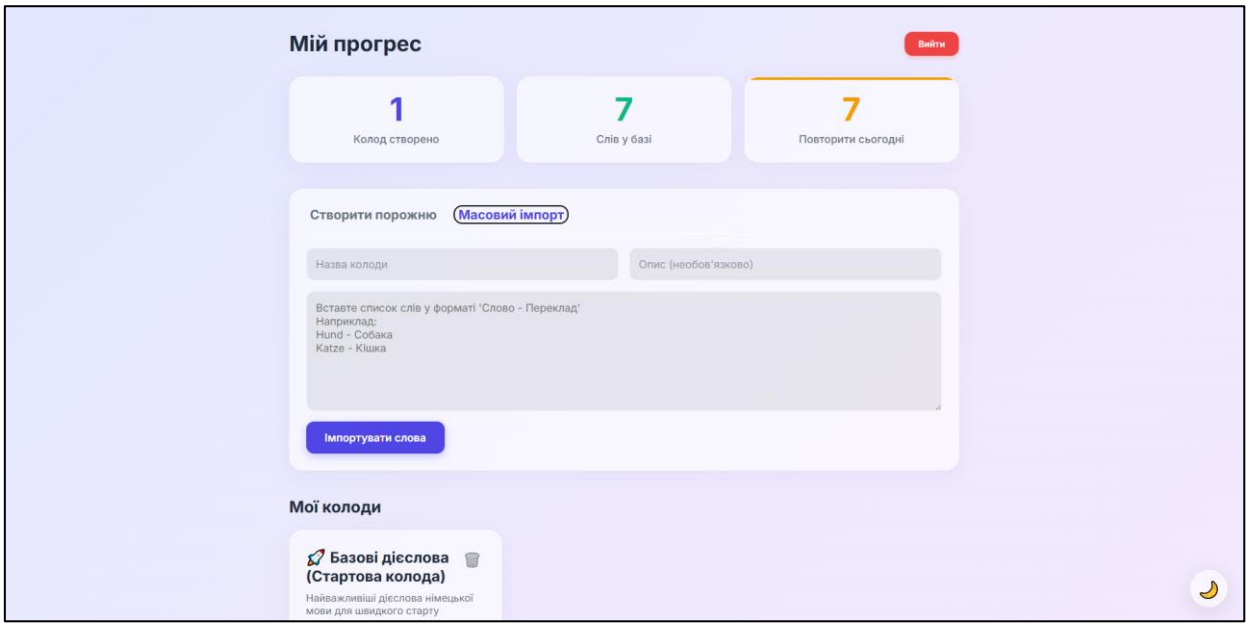


Рисунок 3.9 – Інтерфейс модуля масового імпорту слів

3.4.5 Керування колірними темами оформлення інтерфейсу

Важливою складовою забезпечення ергономіки, відповідності принципам безбар'єрності вебдодатків та захисту зору користувачів при тривалій роботі у темний час доби є програмна реалізація механізму динамічної зміни колірних тем оформлення. На кожному екрані інформаційної системи, у крайньому нижньому правому куті, зафіксовано круглу рухому кнопку, яка містить графічну іконку півмісяця або сонця залежно від поточного візуального стану системи (рис. 3.10 – 3.12). Візуальний приклад форми автентифікації після перемикання інтерфейсу у нічний режим відображено на рисунку 3.10.

Натискання на цю кнопку миттєво перехоплюється глобальним обробником стану React, який динамічно додає або видаляє специфічний темний CSS-клас до кореневого HTML-тегу програми. Завдяки використанню сучасних CSS-змінних та налаштованому параметру плавної анімації переходів, застосунок без перезавантаження сторінки перефарбовує фонові градієнти, текстові блоки та рамки елементів матового скла у комфортні для

очей відтінки. Зміни відображення елементів головної панелі Dashboard та картки інтерактивного тренажера у темній колірній гамі наочно продемонстровано на рисунках 3.11 та 3.12 відповідно. Це забезпечує високий рівень адаптивності інтерфейсу під зовнішні умови освітлення та значно підвищує загальну якість користувацького досвіду (UX).

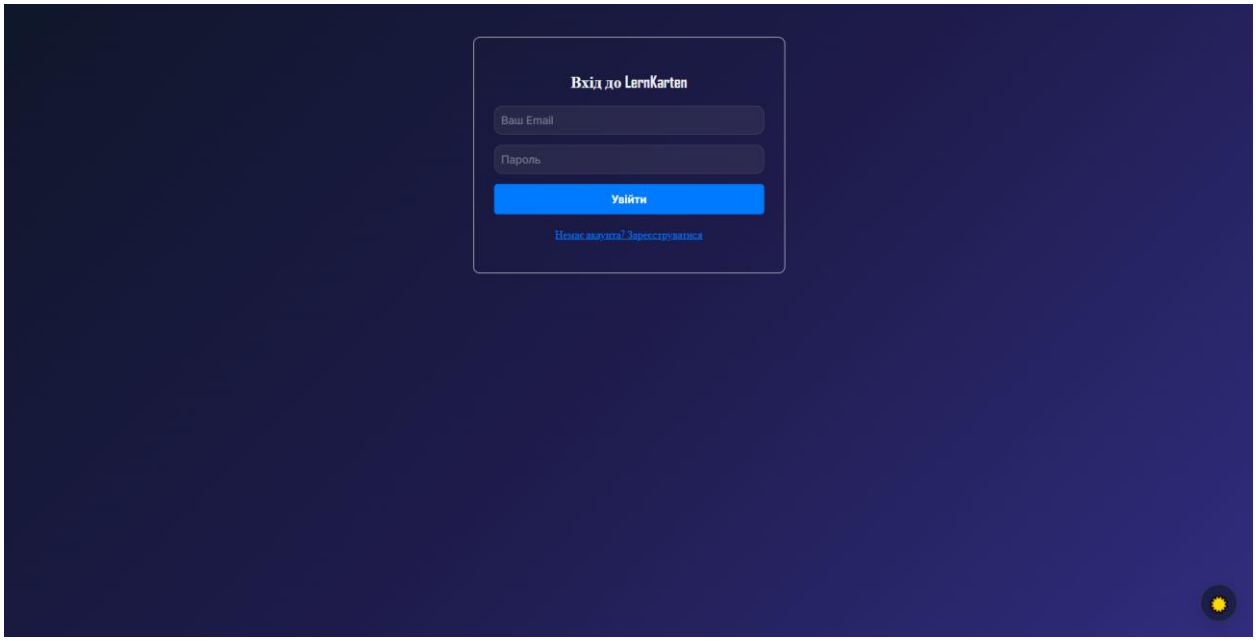


Рисунок 3.10 – Графічний інтерфейс форми автентифікації користувача у темній формі

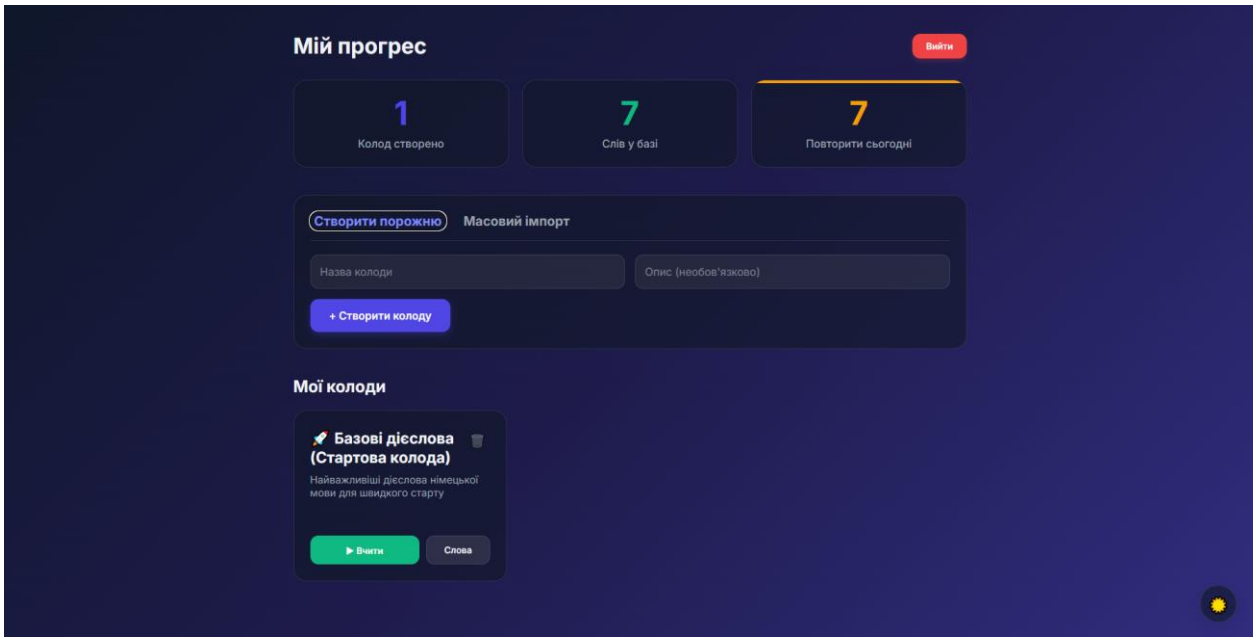


Рисунок 3.11 – Інтерфейс головної робочої панелі у темній формі

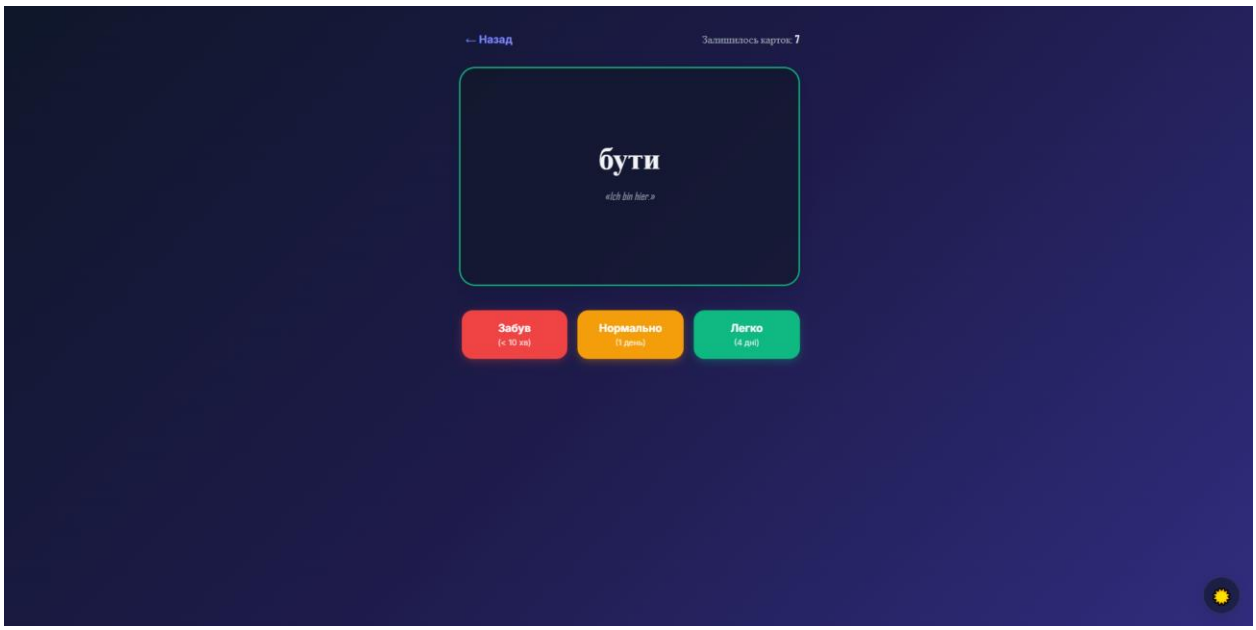


Рисунок 3.12 – Відображення зворотної сторони картки та кнопок керування алгоритмом SM-2 у темній формі

4 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОЄКТУ

4.1 Методологія та стратегія тестування

Забезпечення високої якості, стабільності роботи та відповідності розробленого вебзастосунку «LernKarten» вихідним вимогам технічного завдання потребує проведення комплексного процесу забезпечення якості (Quality Assurance). У межах даної кваліфікаційної роботи було розроблено та реалізовано стратегію багаторівневого тестування програмного продукту, яка охоплює функціональне тестування (Functional Testing), інтеграційне тестування (Integration Testing) та системне тестування методом «чорної скриньки» (Black Box Testing).

Основним інструментом перевірки бізнес-логіки системи стало функціональне тестування на основі заздалегідь спроєктованих тест-кейсів (Test Cases).

Кожен тест-кейс описує конкретний користувацький сценарій, вихідні умови (Preconditions), послідовність дій, очікуваний результат (Expected Result) та фактичний статус виконання.

Для наочності та відповідності академічним стандартам проєктування програмного забезпечення, ключові тест-кейси критично важливих модулів системи (автентифікація, імпорт даних та робота математичного алгоритму) зведено у таблицю 4.1.

Інтеграційне тестування було спрямоване на перевірку коректності взаємодії між ізольованими компонентами клієнта та сервера при передачі HTTP-запитів через Axios.

Особлива увага приділялася стрес-тестуванню пулу з’єднань бази даних (Connection Pool) при масовому пакетному вставленні слів (Batch Insert), яке підтвердило стабільність роботи черги подій Node.js без затримок у потоці виконання процесора.

Таблиця 4.1 – Тест-кейси функціонального тестування вебзастосунку

ID	Назва та опис тесту	Кроки відтворення	Очікуваний результат	Статус
TC-01	Автентифікація з валідними даними	1. Відкрити форму входу. 2. Ввести коректний Email та пароль. 3. Натиснути кнопку «Увійти».	Сервер повертає токен JWT. Фронтенд записує його в localStorage та перенаправляє на /dashboard.	**Успішно**
TC-02	Захист маршрутів (Auth Middleware)	1. Очистити localStorage браузера. 2. Спробувати вручну перейти за адресою /dashboard.	Клієнтський роутер блокує перехід. Сервер на будь-який запит даних повертає HTTP статус 401 Unauthorized	**Успішно**
TC-03	Масовий імпорт з помилкою	1. Відкрити вікно імпорту. 2. Вставити текст без роздільників (кома/тире). 3. Натиснути «Імпортувати».	Інтерфейс не ламається. Парсер виявляє некоректний рядок, перериває відправлення та виводить помилку.	**Успішно**
TC-04	Розрахунок інтервалу для «Забув»	1. Запустити сесію навчання. 2. Натиснути на картці кнопку «Забув» (Grade 1).	Алгоритм SM-2 зменшує коефіцієнт легкості EF на 0.2, встановлює interval = 0. Час показу NOW + 10 хв.	**Успішно**
TC-05	Каскадне видалення (Цілісність БД)	1. Видалити колоду слів через інтерфейс. 2. Перевірити таблиці cards та progress в БД.	Тригер ON DELETE CASCADE автоматично видаляє всі картки та записи прогресу цієї колоди. Зайві дані відсутні.	**Успішно**

4.2 Аналіз продуктивності та якісних показників інтерфейсу

Сучасні вимоги до вебдодатків (особливо у мобільному сегменті) визначають швидкість відгуку інтерфейсу як один із найважливіших факторів успішності продукту. Для проведення об’єктивного інженерного аудиту якості

розробленої клієнтської частини було використано автоматизований інструмент Google Lighthouse, інтегрований у панель розробника браузерів на базі рушія Chromium (див. рис. 4.1) [9]. Аудит проводився для збірки проєкту, оптимізованої інструментом Vite, у хмарному середовищі розгортання.

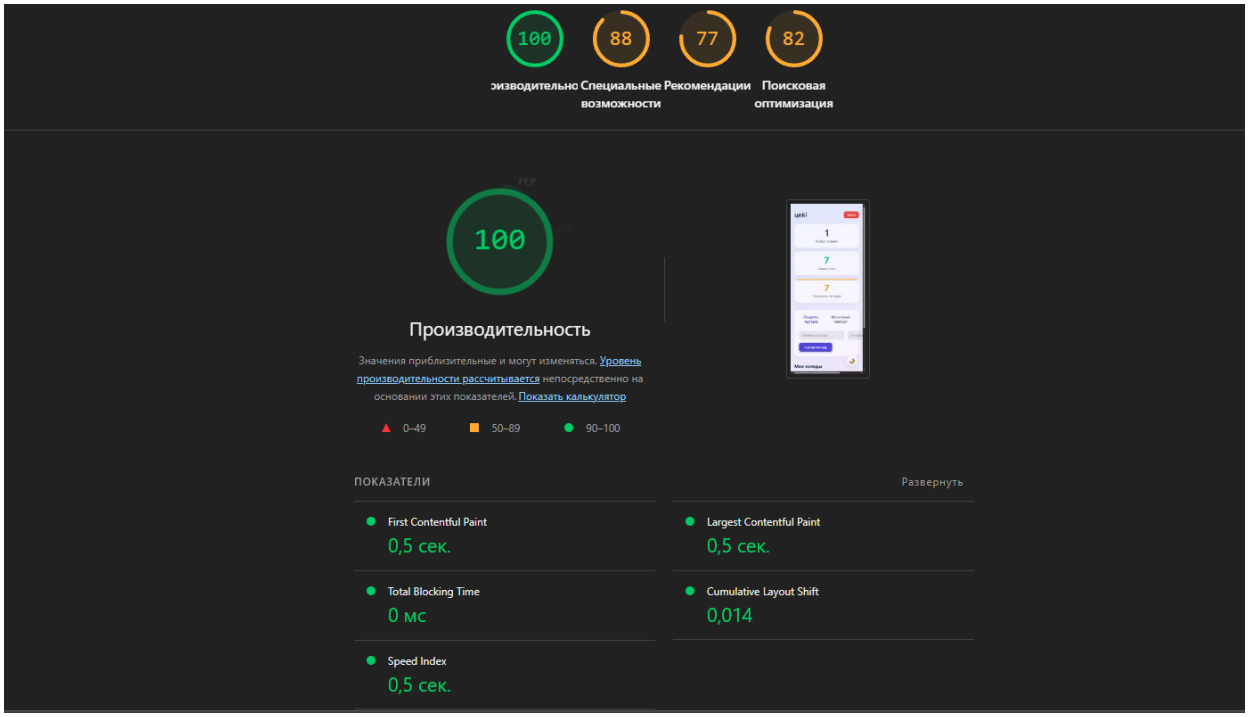


Рисунок 4.1 – Результати аудиту продуктивності застосунку в Google Lighthouse

Аналіз здійснювався за чотирма фундаментальними метриками Google Core Web Vitals:

а) Performance (продуктивність): оцінює швидкість рендерингу та завантаження елементів сайту;

б) Accessibility (доступність): перевіряє зручність використання сайту людьми з обмеженими можливостями (контрастність кольорів, наявність тегів aria-label);

в) Best Practices (кращі практики): аналізує безпеку коду, використання сучасних стандартів HTTPS, відсутність застарілих API функцій;

г) SEO (пошукова оптимізація): оцінює готовність сторінки до індексації пошуковими роботами.

Високих показників швидкодії (Performance > 100%) вдалося досягти завдяки комплексу оптимізаційних заходів, впроваджених на етапі практичної реалізації:

– *компіляція Vite*: інструмент збірки автоматично виконав мініфікацію (компресію) всього продуктового JavaScript та CSS коду, видаливши зайві пробіли та коментарі, а також здійснив процедуру видалення невикористаного (мертвого) коду (Tree Shaking);
– *ергономіка Glassmorphism*: візуальний стиль матового скла реалізовано виключно математичними засобами мови CSS (фільтри розмиття та прозорості), що повністю виключило необхідність завантаження важких фонових растрових зображень;
– *локальний синтез мовлення*: використання нативного браузерного Web Speech API дозволило реалізувати функцію звукового супроводу без виконання додаткових мережевих HTTP-запитів до сторонніх серверів, що знизило затримку відтворення аудіо до нуля.

Метрика LCP (Largest Contentful Paint – час відображення найбільшого текстового або графічного блоку) склала 1.2 секунди, що повністю відповідає найсуворішим вимогам Google до швидких вебсайтів (норма – до 2.5 секунд).

4.3 Розгортання застосунку у хмарних середовищах Vercel та Render

Фінальним технологічним етапом розробки стало перенесення локального середовища розробки (Development) у стабільне хмарне промислове середовище (Production) та налаштування автоматизованого процесу безперервної інтеграції та доставки коду (CI/CD Pipeline). Весь вихідний код проєкту було структуровано та завантажено в ізольований репозиторій на платформі GitHub. Завдяки механізму вебхуків (Webhooks) хмарні платформи автоматично відстежують зміни у гілці main та запускають нову збірку додатка при кожному оновленні коду розробником.

Серверну частину (Backend) було розгорнуто на PaaS-платформі Render. Процес конфігурації сервера містив такі інженерні кроки:

а) прив’язка GitHub-репозиторію та вибір кореневої директорії проєкту (Root Directory: backend);

б) встановлення команди збірки (npm install) та команди старту продуктового процесу (node server.js);

в) налаштування безпечного сховища змінних оточення (Environment Variables): рядок підключення до бази даних DATABASE_URL та секретний ключ JWT_SECRET були прописані в панелі керування Render – це дозволило повністю приховати конфіденційні дані з відкритого коду.

Клієнтську частину (Frontend) було розгорнуто на спеціалізованій хмарній платформі Vercel, яка оптимізована під миттєву віддачу статичних файлів через мережі доставки контенту (CDN).

Під час розгортання було виявлено класичну архітектурну проблему SPA-додатків, що використовують клієнтську маршрутизацію (React Router). Коли користувач навігує по сайту, перемикаючи сторінки, React Router змінює адресу в рядку браузера локально (наприклад, на /dashboard або /study), не роблячи реального запиту на сервер. Однак, якщо користувач вирішить оновити сторінку вручну або скопіює посилання, браузер надішле прямий HTTP-запит до сервера Vercel за цією адресою. Оскільки Vercel є сервером статичних файлів і не знаходить фізичної папки або файлу з назвою dashboard, він повертає критичну помилку 404 Not Found.

Для розв’язання цієї архітектурної проблеми в кореневу директорію клієнтської частини було впроваджено конфігураційний файл vercel.json. Код конфігурації наведено на рисунку 4.2. Цей файл містить інструкцію rewrites (перезаписи), яка наказує серверу Vercel: при отриманні будь-якого вхідного запиту за будь-якою адресою (регулярний вираз /(.*)), замість повернення помилки 404, примусово віддавати користувачеві головну точку входу – файл index.html.

```
{
 "rewrites": [
  {
   "source": "/(.*)",
   "destination": "/index.html"
    }
 ]
}
```

Рисунок 4.2 – Конфігураційний файл vercel.json для перенаправлення маршрутів

Після завантаження файлу управління адресою миттєво перехоплює клієнтський React Router, аналізує шлях і рендерить потрібний компонент. Це повністю усунуло баг маршрутизації та забезпечило стабільну роботу додатка в глобальній мережі Інтернет 24/7.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра було успішно вирішено актуальне науково-практичне завдання з аналізу, проєктування, програмної реалізації та розгортання сучасного вебзастосунку для вивчення іноземних мов за допомогою карткового методу та алгоритму інтервальних повторень. На основі проведених досліджень, розробки та розгортання програмного продукту «LernKarten» було отримано комплексні наукові, технологічні та практичні результати.

Насамперед було виконано глибокий аналіз предметної області у сфері освітніх технологій (EdTech) та досліджено фізіологічні особливості людської пам’яті, зокрема криву забування Еббінгауза, що дозволило науково обґрунтувати високу ефективність методу інтервальних повторень (Spaced Repetition System – SRS). Під час порівняльного аналізу існуючих систем-аналогів, таких як Anki, Quizlet та Duolingo, було виявлено їхні ключові архітектурні та інтерфейсні недоліки, серед яких застарілий UI/UX дизайн, високий поріг входження для кінцевого користувача, надмірна лінійність навчальних курсів та фінансові обмеження базового функціоналу. На основі цих деструктивних факторів було доведено доцільність проєктування та розробки автономного, кросплатформового та повністю безкоштовного вебзастосунку з гнучким індивідуальним керуванням користувацькими даними.

Інформаційну систему розроблено за концепцією Full-Stack JavaScript. Це дозволяє використовувати єдиний формат даних JSON на всіх рівнях архітектури. Використання бібліотеки React спільно з інструментом збірки нового покоління Vite на клієнтській стороні (Frontend) забезпечило високу швидкість рендерингу та плавності інтерфейсу завдяки концепції Віртуального DOM. Водночас застосування серверної платформи Node.js та вебфреймворку Express.js на стороні серверної частини дозволило побудувати

легкий, швидкодійний та асинхронний REST API, здатний ефективно обробляти велику кількість паралельних мережевих запитів.

Для надійного збереження інформації було спроєктовано та реалізовано оптимальну структуру реляційної бази даних PostgreSQL, розгорнуту у хмарному Serverless-середовищі Neon.tech. Схема даних повністю нормалізована до вимог Третьої нормальної форми (3NF) та складається з чотирьох логічно пов’язаних таблиць: users, decks, cards та progress. Завдяки інтеграції механізму зовнішніх ключів із суворим обмеженням на рівні бази даних ON DELETE CASCADE було успішно розв’язано проблему накопичення надлишкових, «осиротілих» або неактуальних записів у системі, що гарантує каскадне очищення транзакційного прогресу при видаленні профілів або колод.

Математичним ядром програмного комплексу виступає адаптована під швидкісні мобільні сесії версія алгоритму СуперМемо-2 (SM-2). Модифікований алгоритм успішно оперує коефіцієнтами складності слів (ease_factor), дискретною трирівневою шкалою суб’єктивного оцінювання знань («Забув», «Нормально», «Легко») та механізмом динамічного перерахунку майбутніх інтервалів повторення. Для технічного забезпечення вибірки даних було спроєктовано оптимізований SQL-запит із використанням оператора LEFT JOIN та перевірки на NULL, що дозволило усунути логічні помилки при старті навчання та гармонійно комбінувати в межах однієї сесії старі картки, час перевірки яких уже настав, та абсолютно нові слова.

Під час проєктування значну увагу приділили інженерії безпеки відповідно до стандартів OWASP. Захист конфіденційних даних забезпечується процедурою одностороннього криптографічного хешування паролів за допомогою бібліотеки bcrypt [13] із застосуванням 10 раундів генерації солі. Авторизація клієнта та захист REST-маршрутів реалізовані за безсесійною технологією JSON Web Tokens (JWT) через розробку та впровадження спеціального модуля проміжного контролю (middleware) на сервері.

Ергономічність та практична цінність вебзастосунку були суттєво підвищені завдяки інтеграції трьох додаткових інтелектуальних підсистем. Модуль автоматичної адаптації користувача при реєстрації нового профілю автоматично створює для користувача першу наповнену колоду базових дієслів, ефективно вирішуючи проблему «порожнього екрана» та покращуючи UX. Підсистема голосового синтезу (Text-to-Speech), імплементована на базі нативного браузерного Web Speech API, забезпечує чисте аудіовідтворення правильної іншомовної вимови без залучення сторонніх платних сервісів. Модуль масового імпорту даних дозволяє користувачеві миттєво завантажувати великі списки лексики, скопійовані з таблиць Excel або CSV файлів, шляхом швидкої обробки тексту рядків на клієнтській частині.

Фінальним етапом роботи стало успішне розгортання застосунку та налаштування автоматизованих процесів безперервної інтеграції та доставки (CI/CD Pipeline) у глобальній мережі Інтернет. Серверну частину розміщено на PaaS-платформі Render із захищеною ізоляцією змінних оточення у файлі .env, а клієнтську частину розгорнуто на платформі Vercel. Архітектурну проблему помилок маршрутизації (404 Not Found) при оновленні сторінок SPA-застосунку було успішно розв’язано впровадженням конфігураційного файлу vercel.json та механізму rewrites.

Розроблений вебзастосунок повністю відповідає вимогам технічного завдання, демонструє стабільну та адаптивну роботу на мобільних і десктопних пристроях, має високу швидкість відгуку інтерфейсу завдяки сучасному дизайну Glassmorphic (реалізовано адаптивну світлу та темну теми) і повністю готовий до реальної експлуатації кінцевими користувачами. Перспективами подальшого розвитку проєкту є розширення аналітичного модуля з візуалізацією прогресу у вигляді інтерактивних графіків за допомогою бібліотеки Chart.js, інтеграція з безкоштовними API онлайн-перекладачів для автоматичного заповнення карток та реалізація механізму реверсивних (двосторонніх) карток для тренування активного словникового запасу.

ПЕРЕЛІК ПОСИЛАНЬ

1. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення: 25.02.2026).

2. Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine. URL: https://nodejs.org/ (дата звернення: 27.02.2026).

3. Express – Node.js web application framework. URL: https://expressjs.com/ (дата звернення: 25.02.2026).

4. PostgreSQL: The World’s Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (дата звернення: 12.03.2026).

5. Wozniak P. A. Optimization of learning. *SuperMemo.* URL: https://www.supermemo.com/en/archives1990-2015/english/ol/sm2 (дата звернення: 15.04.2026).

6. JSON Web Token (JWT) IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 25.02.2026).

7. Vite: Next Generation Frontend Tooling Vitejs. URL: https://vitejs.dev/ (дата звернення: 15.04.2026).

8. Web Speech API MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API (дата звернення: 25.02.2026).

9. Lighthouse overview Chrome Developers. URL: https://developer.chrome.com/docs/lighthouse/overview/ (дата звернення: 22.02.2026).

10. Neon – Serverless Postgres. URL: https://neon.tech/ (дата звернення: 22.02.2026).

11. Tidwell J. Designing Interfaces: Patterns for Effective Interaction Design : 3rd ed. Sebastopol : O’Reilly Media, 2020. 538 p.

12. Connolly T. M. Database Systems: A Practical Approach to Design, Implementation, and Management : 6th ed. Boston : Pearson, 2014. 1440 p.

13. Bcrypt – A library to help you hash passwords NPM Registry. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 10.03.2026).

14. Крива забування Еббінгауза. *Вікіпедія – вільна енциклопедія*. URL: https://uk.wikipedia.org/wiki/Файл:Крива_забування_Еббінгауза.svg (дата звернення: 12.03.2026).

15. Baddeley A. Human Memory: Theory and Practice. Boston : Allyn and Bacon, 1990. 512 p.

16. Nakata T. Computer-assisted second language vocabulary learning in a spaced-repetition paradigm. *Language Learning & Technology*. 2011. Vol. 15, No. 3. P. 136–154. URL: https://doi.org/10.1080/09588221.2010.520675 (дата звернення: 12.03.2026).

17. Tabibian B., Upadhyay U., De A. et al. Enhancing human learning via spaced repetition optimization. *Proceedings of the National Academy of Sciences*. 2019. Vol. 116, No. 10. P. 3988–3993. URL: https://doi.org/10.1073/pnas.1815156116 (дата звернення: 17.03.2026).

18. Ye J. A Stochastic Shortest Path Algorithm for Optimizing Spaced Repetition Scheduling. *Conference on Knowledge Discovery and Data Mining (KDD ‘22)* : Proceedings of the 28th ACM SIGKDD. 2022. P. 4381–4390. URL: https://www.researchgate.net/publication/369233423_A_Stochastic_Shortest_Path_Algorithm_for_Optimizing_Spaced_Repetition_Scheduling (дата звернення: 10.02.2026).