**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

# Дипломний проект

**на здобуття ступеня бакалавра**

**з напряму підготовки 6.050103 «Програмна інженерія»**

**на тему: «Програмні засоби для підтримки процесу вивчення англійської мови»**

Виконав:
студент IV курсу, групи КП-52
Уруков Дмитро Олексійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Т.М. _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н.,
Онай М.В. _____

Рецензент:
Доцент кафедри ММСА, к.т.н., доцент,
Дідковська М.В. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 6.050103 «Програмна інженерія»

«ЗАТВЕРДЖУЮ»

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

**ЗАВДАННЯ**
**на дипломний проект студенту**
Урукову Дмитру Олексійовичу

1. **Тема** проекту «Програмні засоби для підтримки процесу вивчення англійської мови» затверджена наказом по університету № 1331-С від «22» травня 2019 р.
2. **Термін подання** студентом завершеного проекту: «7» червня 2019 р.
3. **Вихідні дані для дипломного проектування:** див. Технічне завдання.
4. **Перелік задач, які мають бути вирішені:**
   – розробити загальну структуру програмного забезпечення;
   – розробити алгоритми роботи програми;
   – розробити дизайн сторінок та графічних елементів;
   – виконати програмну реалізацію системи відповідно до вимог технічного завдання;
   – виконати тестування розробленої системи.
5. **Перелік обов'язкового ілюстративного матеріалу:**
   – структура бази даних системи підтримки процесу вивчення англійської мови (креслення);
   – схема потоку даних системи підтримки процесу вивчення англійської мови (креслення);
   – структурна схема системи підтримки процесу вивчення англійської мови (плакат);
   – діаграма прецедентів системи (плакат).
6. **Консультанти розділів проекту**

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Нормоконтроль | Онай М.В., доцент | | |

7. **Дата видачі завдання:** «31» жовтня 2018 р.

**КАЛЕНДАРНИЙ ПЛАН-ГРАФІК**

| № з/п | Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Вивчення літератури за тематикою проекту | 15.10.2018 | |
| 2. | Розроблення та узгодження технічного завдання | 19.11.2018 | |
| 3. | Розроблення структури програмного забезпечення | 03.12.2018 | |
| 4. | Підготовка матеріалів першого розділу дипломного проекту | 17.12.2018 | |
| 5. | Розроблення дизайну системи та графічних елементів | 01.02.2019 | |
| 6. | Підготовка матеріалів другого розділу дипломного проекту | 25.02.2019 | |
| 7. | Програмна реалізація дипломного проекту | 15.03.2019 | |
| 8. | Тестування програмного забезпечення | 19.04.2019 | |
| 9. | Підготовка матеріалів третього розділу дипломного проекту | 03.05.2019 | |
| 10. | Підготовка матеріалів четвертого розділу дипломного проекту | 15.05.2019 | |
| 11. | Оформлення документації дипломного проекту | 03.06.2019 | |

**Керівник дипломного проекту** _____ Т.М. Заболотня

**Студент** _____ Д.О. Уруков

# АНОТАЦІЯ

Даний дипломний проект присвячений розробленню програмних засобів для підтримки процесу вивчення англійської мови.

Розроблені програмні засоби являють собою розширення для веб-браузеру Google Chrome, призначене для підтримки процесу збагачення словникового запасу англійської мови користувача за популярним нині методом інтервальних повторень. Функціональність клієнтської частини розширення забезпечує інтеграцію системи у повсякденний процес перегляду сторінок у веб-браузері шляхом надання можливості отримати переклад та значення будь-якого слова на веб-сторінці, зберегти його до персонального списку слів для подальшого заучування вищеозначеним методом за допомогою інтерфейсу, що відображається на новій вкладці веб-браузера. Доступ до додатку забезпечується після реєстрації, гостьовий доступ не передбачено. Під час реєстрації користувач проходить швидкий тест для оцінки обсягу словникового запасу англійської мови, результати якого використовуються додатком для виділення на веб-сторінках, що переглядаються, слів іноземної мови, що є потенційно незнайомими для користувача.

У даному дипломному проекті розроблено: архітектуру браузерного розширення та пов’язаної з ним серверної частини, алгоритм визначення потенційно незнайомих слів, алгоритм вибору слів для повторення за методом інтервальних повторень, а також графічні елементи та дизайн клієнтської частини.

## ABSTRACT

This diploma project deals with the development of software to support the process of learning English.

Developed software is a Google Chrome web browser extension, intended to support the process of increasing the English vocabulary of the user by using a popular method of spaced repetition. The functionality of the client-side part of the extension provides integration into the everyday process of web browsing by allowing the user to view a translation and a meaning of any word on the web page, save it to his/her personal list for further memorization using a spaced repetition method via the interface on a blank tab of the web browser. Access to the application is granted after the registration, guest access is not allowed. The user passes a quick test during the registration in order to estimate the volume of his/her vocabulary. This estimation is used by the application to highlight the words which are potentially unknown to the user during his web browsing.

While working on this project, the following artefacts have been developed: the architecture of the browser extension and the connected server-side part, the algorithm to detect potentially unknown words, the algorithm to select words for repetition according to the spaced repetition method, graphic elements and the design of the client-side part.

ДП.045440-01-90 Програмні засоби для підтримки процесу вивчення англійської мови. Відомість проекту

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | Документація проекту | | |
| | | | |
| ДП.045440-02-91 | Програмні засоби для | 4 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Технічне завдання | | |
| | | | |
| ДП.045440-03-81 | Програмні засоби для | 57 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Пояснювальна записка | | |
| | | | |
| ДП.045440-04-51 | Програмні засоби для | 4 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Програма та методика | | |
| | тестування | | |
| | | | |
| ДП.045440-05-34 | Програмні засоби для | 5 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Керівництво | | |
| | користувача | | |
| | | | |
| | | | |
| | | | |
| | | | |

| Позначення | Найменування | Кіл-ть | Примітка |
|---|---|---|---|
| | | | |
| ДП.045440-06-99 | Програмні засоби для | 1 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Структура бази даних. | | |
| | ERD-діаграма | | |
| | | | |
| ДП.045440-07-99 | Програмні засоби для | 1 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Потік даних сервісу | | |
| | аналізу потенційно | | |
| | незнайомих слів. | | |
| | Схема потоку | | |
| | | | |
| ДП.045440-08-98 | Програмні засоби для | 1 | |
| | підтримки процесу | | |
| | вивчення англійської мови. | | |
| | Компакт-диск | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ І.А. Дичка
(підпис)
"___" _____ 2018 р.

# ПРОГРАМНІ ЗАСОБИ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ

**Технічне завдання**

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Д.О. Уруков

2018

**ЗМІСТ**

## 1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

**Назва розробки**: Програмні засоби для підтримки процесу вивчення англійської мови.

**Галузь застосування**: інформаційні технології.

## 2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

## 3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання у веб-браузерах в якості сервісу для підтримки процесу збагачення словникового запасу англійської мови користувача.

## 4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмний продукт повинен забезпечувати такі основні функції:

1) можливість використання у веб-браузері;
2) можливість отримання значення довільного слова на веб-сторінці;
3) можливість зберегти довільне слово з веб-сторінки для подальшого заучування засобами продукту;
4) можливість переглядати та редагувати список збережених слів;
5) можливість повторювати та заучувати нові слова за методом інтервальних повторень;

## 5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) керівництво користувача;

4) креслення:

- «Структура бази даних. ERD-діаграма»;
- «Потік даних сервісу аналізу потенційно незнайомих слів. Схема потоку».

## 6. ЕТАПИ ПРОЕКТУВАННЯ

Вивчення наявної літератури за тематикою проекту .....12.10.2018

Розроблення та узгодження технічного завдання.....16.11.2018

Розроблення структури програмного забезпечення .....07.12.2018

Розроблення дизайну інтерфейсів .....08.02.2019

Програмна реалізація дипломного проекту .....15.03.2019

Тестування програмного забезпечення.....19.04.2019

Підготовка матеріалів текстової частини проекту.....03.05.2019

Підготовка матеріалів графічної частини проекту .....24.05.2019

Оформлення технічної документації проекту.....03.06.2019

## 7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

# ПРОГРАМНІ ЗАСОБИ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ

**Пояснювальна записка**

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Д.О. Уруков

2019

**ЗМІСТ**

## СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ОС – операційна система.

ПЗ – програмне забезпечення.

Веб-браузер – програмне забезпечення для перегляду інформації в мережі Інтернет.

Розширення браузера – відносно невелике програмне забезпечення, що додає додаткові функціональні можливості до веб-браузера.

Інтерпретатор – програмне забезпечення, що застосовується для прямого виконання програмних інструкцій, написаних одною з мов програмування високого рівня.

DOM – об’єктна модель документа, програмний інтерфейс для доступу до елементів HTML-документів.

HTML – мова розмітки гіпертекстових документів, мова розмітки для створення веб-сторінок.

CSS – каскадні таблиці стилів, описують яким чином елементи HTML розташовуються та виглядають на екрані.

Фреймворк – набір абстракцій, які надають базову функціональність програмного забезпечення для подальшої композиції та модифікації.

SPA – односторінковий інтерфейс, веб-застосунок, який функціонує в межах однієї веб-сторінки без необхідності її перезавантаження при переходах між екранами додатку.

JSX – HTML-подібний синтаксис, що використовується бібліотекою. React та дозволяє поєднувати разом в одній деревовидній структурі код мовою JavaScript та шаблони розмітки мовою HTML.

Webpack – пакувальник модулів у проектах, написаних мовою JavaScript. Дозволяє трансформувати та збирати воєдино

власне код, розмітку HTML, стилі CSS та статичні ресурси, наприклад, шрифти або картинки.

БД – база даних.

СУБД – система управління базами даних, програмне забезпечення для керування даними.

ACID – спеціальні вимоги до СУБД: Atomicity (укр. «атомарність»), Consistency (укр. «узгодженість»), Isolation (укр. «ізольованість»), Durability (укр. «довговічність»).

Транзакція – єдина логічна одиниця роботи, яка отримує доступ та, можливо, модифікує вміст бази даних.

Шардування – тип розбиття бази даних, при якому велика база даних розділюється на менші, швидші та більш керовані частини, які називаються шардами.

Реплікація – неперервне копіювання змін в одній базі даних до іншої бази даних, які, як правило, знаходяться на різних фізичних серверах, з метою резервного копіювання або зменшення навантаження на одну з баз данних.

JSON – текстовий формат обміну даними, що базується на мові програмування JavaScript.

HTTP – Hypertext Transfer Protocol (укр. «протокол передачі гіпер-текстових документів»), протокол прикладного рівня для передачі даних у комп’ютерних мережах.

API – Application Programming Interface (укр. «інтерфейс прикладного програмування») – набір стандартів та методів взаємодії окремих компонентів програмної системи.

AJAX – технологія звернення до серверу за допомогою HTTP-запитів без перезавантаження сторінки.

REST – Representational State Transfer (укр. «передача репрезентативного стану»), архітектурний стиль побудови взаємодії між комп’ютерними системами та додатками в мережі Інтернет, наприклад, між клієнтським веб-додатком та зовнішнім сервером.

# ВСТУП

Вміння розмовляти англійською мовою є одною з ключових навичок, яка має бути наявна в людини у сучасному світі, де глобалізаційні процеси стають все більш помітними, поступово стираючи кордони між державами, культури та релігіями. Англійська мова є рідною для більш ніж 400 мільйонів людей, але набагато більше – приблизно 1.5 мільярда жителів нашої планети використовують її в якості другої або третьої мови, тому вона по праву вважається мовою міжнародного спілкування. Її широко використовують у бізнесі, туризмі, сфері освіти, виробництві, політиці тощо. Саме тому знання англійської мови в наш час є показником успішності та освіченості людини, адже воно не тільки відкриває доступ до скарбниці світового мистецтва та культури, а й значно покращує позиції на ринку праці та в бізнесі, надаючи змогу працювати на міжнародних ринках та взаємодіяти з іноземними партнерами.

Вивчення англійської мови – це надзвичайно складна задача, яка потребує неабиякої працьовитості, мотивації та цілеспрямованості. Разом із розвитком інформаційних технологій почали з’являтися та стрімко набирати популярність цифрові додатки та сервіси, що додавали інтерактивний елемент до процесу вивчення та значно розширювали обсяг доступної для людини навчальної інформації та завдань для самовдосконалення. Створені професійними педагогами, такі додатки дозволили автоматизувати процес вивчення англійської без втрати якості засвоєння матеріалу порівняно із традиційними методами навчання із вчителем, а головне – дозволили людям навчатися де завгодно та скільки завгодно.

Ринок ПЗ для підтримки процесу вивчення англійської мови почав стрімко зростати протягом останніх кількох років. Постійно з’являються нові сервіси, орієнтовані на різні аспекти вивчення мови: від таких, що пропонують повноцінні інтерактивні курси із вправами для тренування усіх

навичок, до таких, що фокусуються на якомусь конкретному аспекті вивчення англійської, як-от збагачення словникового запасу або тренування навичок слухання.

У той же час, через те, що популярність вищеописаних сервісів ще не досягла свого піку, їх автори постійно експериментують із різними підходами до процесу навчання користувача, кожен з яких містить свої недоліки. Одним із таких вважається невміння більшості додатків забезпечити неперервність процесу навчання без втрати мотивації з боку користувача. Саме тому створення системи для підтримки процесу навчання, яка б була позбавлена цього недоліку та якомога сильніше інтегрувала користувача в процес навчання, є актуальною на наш час задачею.

Метою даного дипломного проекту є розробка системи для підтримки процесу вивчення англійської мови, що дозволить уникнути недоліків існуючих програмних рішень.

# 1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

## 1.1. Навички, необхідні в процесі вивчення англійської мови

Мовні навички – це набір здібностей, що дозволяють людині розуміти і відтворювати розмовну та письмову мову для належного і ефективного міжособистісного спілкування [1].

Традиційно виділяють такі навички, найбільше задіяні у процесі вивчення англійської мови:

1. Listening (укр. «слухання»)
2. Speaking (укр. «говоріння»)
3. Reading (укр. «читання»)
4. Writing (укр. «письмо»)

Для кожної з цих навичок існують певні усталені методи їх покращення. Для слухання – це прослуховування аудіокниг та перегляд фільмів мовою оригіналу, для читання – очевидно, читання книжок, періодичних видань, газет та інших письмових джерел, для письма – написання коротких есе на задані тем, для говоріння – комунікативна взаємодія із носіями мови або ж із більш досвідченими знавцями англійської як іноземної.

Окрім вищезгаданих основних мовних навичок виділяють і дві додаткових: Pronunciation (укр. «вимова») та Vocabulary (укр. «словниковий запас»). Однак у той час, як вимова слів зазвичай покращується паралельно із розвитком інших усних навичок (слухання та говоріння), розширення словникового запасу є окремим і незалежним процесом. Прогресу у вивченні нових слів майже неможливо досягти, якщо читати нові тексти англійською, шукати незнайомі слова в словнику та ніяким чином не взаємодіяти із цими словами, тобто не запам’ятовувати їх. Саме тому в даній роботі було вирішено зосередитися саме на створенні системи, спрямованої на збагачення словникового запасу при вивченні англійської мови. На відміну від програмних засобів для розвитку інших мовних навичок, для

створення такої системи немає необхідності генерувати велику початкову базу знань, для якої знадобилися б спеціальні філологічні знання.

### 1.2. Огляд популярних методик запам’ятовування нових слів

Однією з необхідних умов опанування англійської мови є постійне поповнення словникового запасу новими словами. Однак, не для всіх студентів цей елемент навчання є простим та інтуїтивним. Саме через це фахівці радять при вивченні та запам’ятовуванні нових слів використовувати наступні техніки [2], що значно пришвидшують цей процес та підвищують його ефективність:

1. Техніка створення асоціативних зв’язків між словами, що вже знайомі студенту, та тими, що вивчаються. Ці слова не обов’язково повинні бути пов’язані за змістом: навпаки, чим абсурдніше придуманий зв’язок, тим краще запам’ятовується нове [3].
2. Техніка запам’ятовування у складі словосполучення (у контексті). Якщо нове слово було помічено в тексті, слід виписати його разом із оточуючими словами, наприклад, у парі дієслово + іменник, прикметник + іменник тощо, якщо ж ні – на допомогу приходять онлайн-словники, які, зазвичай, містять прості приклади використання слова.
3. Техніка запам’ятовування синонімів/антонімів: слід знайти/підібрати вже знайомі синоніми або антоніми до нового слова, адже мозок в такому разі автоматично створює між ними мнемонічний зв’язок.
4. Метод інтервальних повторень. Дослідження довели [4], що наявність довгих інтервалів між короткими періодами заучування нової інформації (у тому числі іноземних слів) значно покращує якість запам’ятовування у порівнянні з довгим та інтенсивним заучуванням, що або містить короткі інтервали для відпочинку, або не містить їх взагалі, адже дозволяє задіяти довгострокову пам’ять.

Наприклад, студент хоче запам’ятати слово «wanderlust» (укр. «жага до пригод»): повторивши його за годину, потім – за чотири, пізніше – за день, і, врешті-решт, за тиждень, він вже, найбільш ймовірно, не забуде його значення. При цьому слід пам’ятати, що бажані інтервали між повтореннями є повністю індивідуальними для кожного студента та потребують коригування власноруч.

В результаті проведення аналізу популярних методів та технік запам’ятовування нових слів було прийняти рішення в рамках даного дипломного проекту сфокусуватися саме на дослідженні способів прикладного використання методу інтервальних повторень, адже, на відміну від інших, до цього методу можуть бути застосовані підходи з автоматизації та алгоритмізації його застосування.

## 1.3. Аналіз існуючих програмних рішень

### *1.3.1. Anki*

Anki – відкрите програмне забезпечення, що використовується в якості навчального інструменту, що допомагає запам’ятовувати нову інформацію за методикою інтервальних повторень [5]. Anki дозволяє самостійно створювати так звані колоди навчальних карток для використання під час навчання. Зазвичай, картка на лицевій стороні містить потенційно новий для користувача текст / зображення / аудіо / формулу, а на тильній стороні – відповідь (також у вигляді тексту, зображення, аудіо чи формули), яку користувач має намір запам’ятати (див. рис. 1.1).

Під час роботи з кожною із карток користувач має позначити, наскільки легкою, або ж навпаки – важкою, є для нього ця картка. Ця інформація використовується алгоритмами Anki для адаптації внутрішньої системи ранжування карток у кожній колоді. Відповідно до правил методу інтервальних повторень, картки, що були позначені користувачем як легкі для нього, будуть показуватися рідше, ніж картки, позначені складними [5].

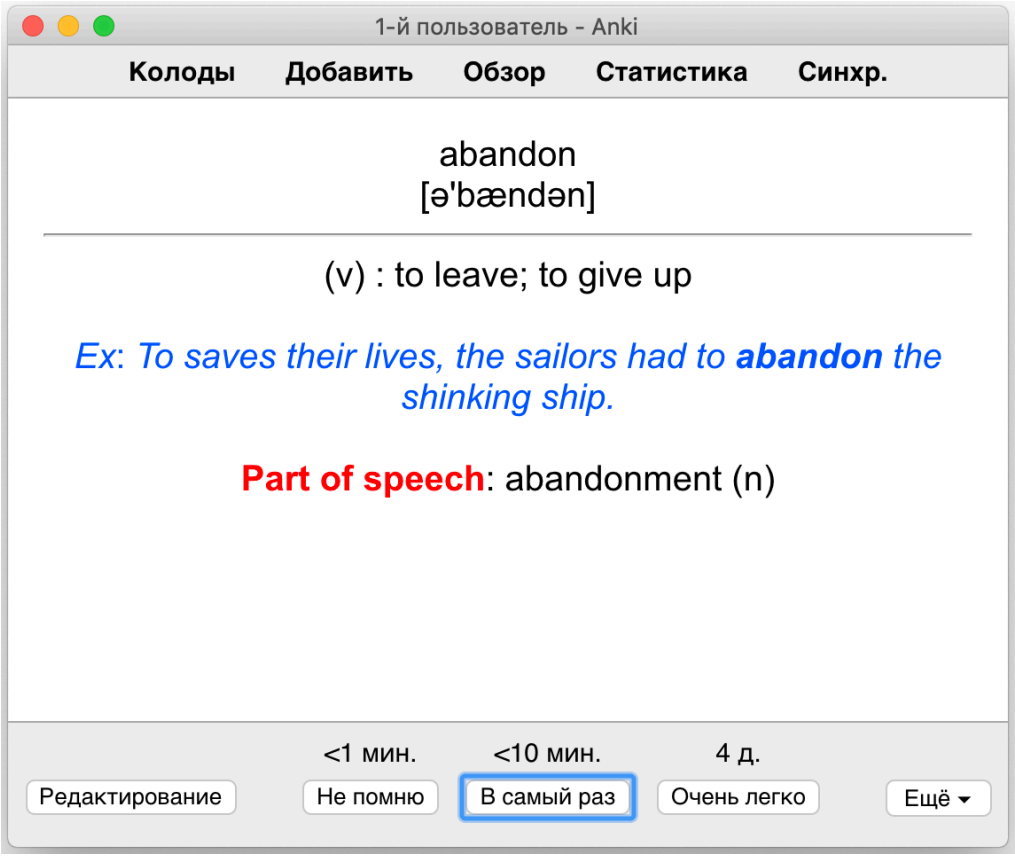



Рис. 1.1. Інтерфейс додатку Anki для ОС macOS

Якщо користувач не бажає створювати свої власні колоди карток, він може скористатися вже готовими колодами, створеними спільнотою, які доступні у вбудованому в додаток онлайн-каталозі із десятками тисяч готових колод на будь-яку тематику.

Функціональними особливостями даного ПЗ є:

- широкий вибір інструментів для створення карток;
- велика кількість безкоштовних плагінів;
- можливість налаштовувати алгоритм навчання під індивідуальні потреби;
- доступність у додатку детальної статистики використання;
- підтримка більшості сучасних десктопних платформ;
- синхронізація даних через профіль користувача, що створюється на офіційному порталі Anki.

До недоліків реалізації даного програмного застосунку можна віднести:

- застарілий інтерфейс користувача із неочевидними дизайн-рішеннями;
- відсутність будь-якої категоризації колод в онлайн-каталозі, що значно погіршує пошук бажаної колоди;
- неоднорідність якості колод в онлайн-каталозі та відсутність їх модерації;
- відсутність будь-яких інтеграцій з операційної системою, як-от системних нотифікацій із нагадуваннями про необхідність повернутися до користування додатком.

### *1.3.2. Memrise*

Memrise – сервіс, доступний як у вигляді веб-сервісу, так і мобільного додатку, що допомагає вивчати та запам’ятовувати нові слова іноземної мови [6]. Це досягається за допомогою гейміфікованого інтерфейсу в космічному стилі, де прекрасно поєднуються дизайн та функціональність.

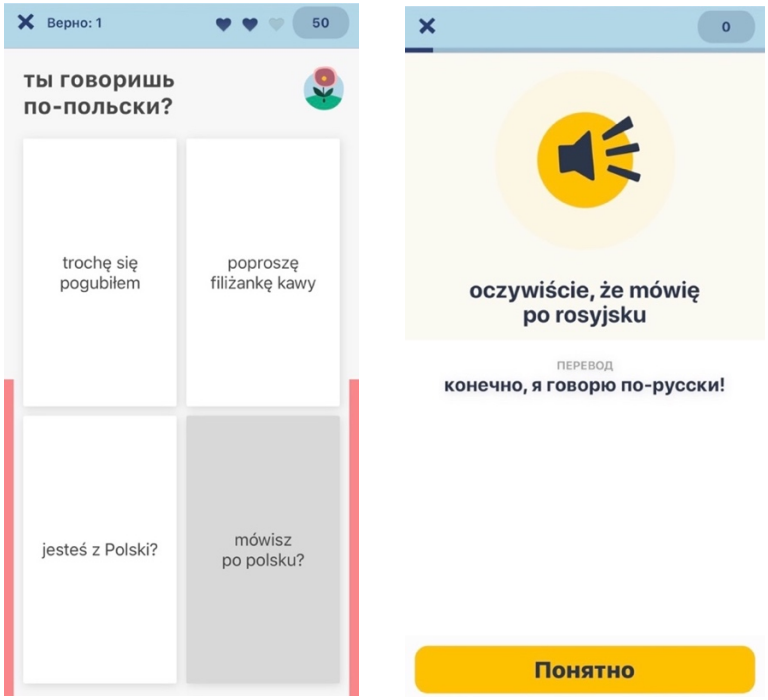



Рис. 1.2. Інтерфейс мобільного додатку Memrise для iOS

Процес навчання у Memrise є доволі схожим на той, що пропонують інші додатки, створені за методом інтервальних повторень, але із великою кількістю позитивних відмінностей від, наприклад, Anki. По-перше, для більшості мовних курсів Memrise має прекрасно озвучені справжні носіями мови слова та фрази: саме можливість не тільки бачити, а й слухати нові слова є найбільш помітним прискорювачем їх запам'ятовування. По-друге, режим навчання в Memrise побудований таким чином, щоб максимально часто чергувати різні види навчання, доступні в додатку, а саме:

- вибір правильного перекладу слова;
- вибір правильного слова за наявним озвученим (аудіювання);
- відтворення слова за наявними перемішаними літерами;
- відтворення порядку слів у реченні за перемішаними словами;
- запис власної вимови слова чи фрази для порівняння із вимовою носія мови.

Режим повторення в Memrise влаштований так, що додаток автоматично додає до поточної навчальної сесії (уроку) слова з попередніх уроків, що, за розрахунками внутрішніх алгоритмів, вже могли підзабутися.

Помітними недоліками додатку для вивчення іноземних мов Memrise є:

- дуже обмежений набір функцій у безкоштовній версії;
- висока ціна преміум-версії: $40 на рік;
- недостатньо велика кількість вправ на граматику;
- надзвичайно мала кількість мовних курсів, доступних українською мовою;
- веб-версія значно поступається можливостями порівняно із мобільною версією.

### *1.3.3. Google Dictionary (Chrome Extension)*

Google Dictionary – розширення для веб-браузеру Google Chrome, що надає доступ до даних потужного онлайн-словнику компанії Google [7].

Головною його особливістю є можливість використання словнику прямо під час перегляду будь-якої веб-сторінки за допомогою браузеру Google Chrome шляхом подвійного кліку на незнайоме слово. Це призводить до появи невеликої «бульбашки» із тлумаченням та/або перекладом цього слова. Крім цього, в користувача є можливість прослухати вимову слова, записану носієм мови.

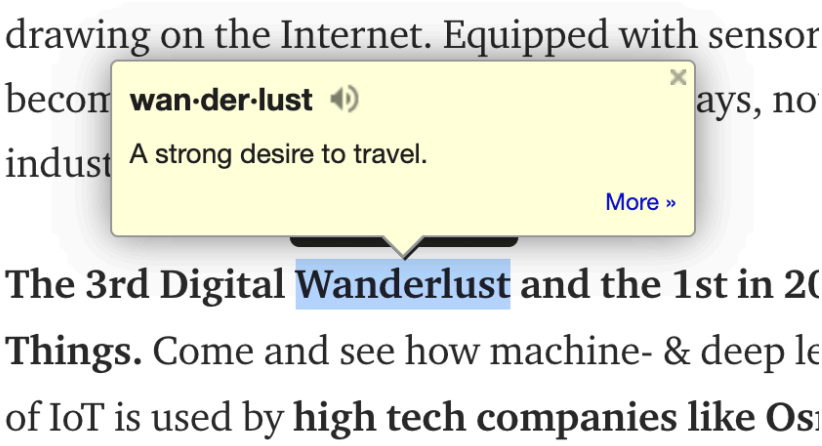



Рис. 1.3. Приклад використання розширення Google Dictionary

Розширення підтримує переклад між близько 20 найбільш популярними у світі мовами. При натисканні на посилання «Більше» розширення переадресовує користувача до результатів пошуку Google із списком популярних онлайн-словників, що містять більше інформації про це слово.

Недоліком розширення є відсутність можливості зберігати слова, з якими взаємодіяв користувач, до будь-якого списку, з яким можна було б у якийсь корисний спосіб взаємодіяти пізніше (доступний лише експорт у CSV).

### *1.3.4. 30 Seconds of Knowledge*

30 Seconds of Knowledge – розширення для браузерів Google Chrome та Mozilla Firefox, орієнтоване, у першу чергу, на розробників та тих, хто бажає ними стати, що дозволяє отримувати нові навички та покращувати

свої галузеві знання просто відкриваючи нову вкладку у своєму браузері [8]. Розширення містить у собі велику бібліотеку фрагментів програмного коду із поясненнями у великій кількості галузей розроблення, як-от: JavaScript, CSS, React, PHP, Ruby, C++ тощо.


Рис. 1.4. Приклад фрагменту коду, який показує розширення 30 Seconds of Knowledge на новій вкладці

Основна ідея розширення описана у його назві: показувані фрагменти коду є доволі короткими, тож їх можна переглянути дуже швидко, протягом 30 секунд. Тим не менш, кожен розробник десятки разів на день відкриває вкладку браузеру, тож у випадку, якщо його погляд зачепиться на цікавому для нього фрагменту коду, у сукупності за день кількість переглянутих фрагментів буде доволі значною.

Користувач має можливість як зберегти показаний фрагмент коду до вибраних, так і видалити його з ротації, натиснувши на відповідну кнопку над фрагментом. Налаштування розширення дозволяють обрати цікаві користувачеві галузі розроблення для показу, а також змінити візуальний стиль розширення (кольорова тема, розмір шрифту тощо).

Недоліком розширення є неможливість додавати в ротацію свої фрагменти коду, тобто воно є жорстко обмеженим джерелами даних розробника розширення, які, тим не менш, доволі часто оновлюються.

### 1.4. Результати проведення аналізу

Дані, отримані при детальному аналізі схожих за функціональними особливостями рішень, їх переваг та недоліків, свідчать про те, що жодне з розглянутих рішень в повній мірі не реалізує усіх концепцій методу інтервальних повторень одночасно із високим рівним залучення користувача до неперервного процесу вивчення нових слів.

При цьому, кожне з існуючих рішень дуже добре реалізує якусь частину бажаних функціональних особливостей. Наприклад, додаток Anki чудово реалізує метод інтервальних повторень, але через відсутність зовнішніх інтеграцій не дає можливості легко наповнювати словник без необхідності відкривати додаток. З іншого боку, розширення Google Dictionary є чудово інтегрованим у процес веб-серфінгу, дозволяючи перекласти будь-яке слово, що зустрілося, але не надає можливостей для його подальшого запам’ятовування користувачем. В той же час, розширення “30 Seconds of Knowledge” хоча є й далеким за своєю темою від теми розроблюваного рішення, але є ідеальним прикладом реалізації додатку не на окремому веб-сервісі, який потребував би від користувача окремих дій для переходу на нього, а на новій вкладці, таким чином повністю інтегруючись у звичайний процес роботи користувача із браузером.

В результаті аналізу наявних аналогів, їх особливостей, переваг та недоліків було сформульовано список бажаних функціональних особливостей розроблюваної системи:

1. Система має надавати можливість реєстрації нового користувача та автентифікації існуючого користувача.
2. Система має визначати приблизний рівень знання користувачем лексики англійської мови шляхом проведення тесту.
3. Система має надавати користувачеві можливість отримати переклад слова на веб-сторінці, яку той переглядає.
4. Система має дозволяти користувачеві зберегти слово з веб-сторінки для подальшого заучування засобами системи.
5. Система має дозволяти користувачеві переглядати список раніше збережених слів та дані про них, видаляти збережені та додавати нові слова до нього.
6. Система має обирати слова із списку збережених слів за методом інтервальних повторень та відображати їх користувачеві по одному під час кожного початку взаємодії із системою.
7. Система має надавати можливість вказати приблизний рівень володіння слова користувачем для подальшого використання у роботі системи.

# 2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

## 2.1. Вибір мови програмування для розроблення серверної частини

Після аналізу сформульованих функціональних особливостей було зроблено висновок у необхідності розбиття системи на два окремі модулі, а саме: модуль серверної частини, що відповідатиме за взаємодію із базою даних та зовнішніми ресурсами, та модуль клієнтської частини для реалізації користувацької взаємодії за методом інтервальних повторень.

До мови програмування для розроблення серверної частини було висунуто наступні вимоги:

- інтерпретована модель виконання;
- наявність інструментів функціональної парадигми;
- наявність великої кількості доступних бібліотек;
- наявність фреймворку для побудови веб-серверу.

### *2.1.1. Python*

Python – інтерпретована мова програмування високого рівня із динамічною типізацією та підтримкою багатьох популярних парадигм програмування, а саме об’єктно-орієнтованої, функціональної, процедурної (імперативної) тощо [9].

Основними та найбільш значущими є такі характеристики мови Python:

1. Інтерпретованість: інтерпретатор мови виконує код послідовно рядок за рядком, що значно полегшує процес відлагодження.
2. Кросплатформеність: інтерпретатор мови доступний для більшості популярних платформ, таких як Windows, Linux, macOS тощо.
3. Відкритість: сирцевий код інтерпретатору мови Python є відкритим для перегляду та вивчення, а сам інтерпретатор доступний для вільного завантаження та використання.

4. Об’єктно-орієнтованість: мова підтримує та реалізує більшість концепцій об’єктно-орієнтованої парадигми програмування, тобто надає можливість розробнику маніпулювати об’єктами створених класів, хоча й у доволі відмінний від інших популярних мов спосіб.
5. Розширюваність та інтегрованість: модуль, написаний іншими мовами, наприклад C, C++, Java, може бути легко викликаний із коду мовою Python, і навпаки.
6. Стандартна бібліотека мови містить величезну кількість модулів та функцій для пришвидшення розроблення необхідного ПЗ.

Вважається [9], що мова Python є доволі простою для вивчення, адже її унікальний синтаксис направлений на збільшення читабельності, а отже розробники (особливо початківці) можуть сприймати код, написаний мовою Python, швидше за код, написаний іншими мовами.

У екосистемі навколо мови Python існує безліч рішень для реалізації серверної частини веб-додатків, адже мова підтримує як багатопоточне, так і асинхронне виконання коду – дві найбільш популярні концепції роботи серверних додатків. Найчастіше для побудови серверних рішень використовуються фреймворки Flask (більш мінімалістичний) та Django (містить більше вбудованих модулів та компонентів) .

### *2.1.2. PHP*

PHP – скриптова (інтерпретована) мова програмування, специфічним призначенням якої є розроблення веб-додатків [10]. За даними сучасних досліджень [11], саме PHP є мовою, якою написано більшість (73%) існуючих працюючих в мережі Інтернет веб-серверів.

Мова PHP має динамічну типізацію, тобто не потребує як оголошення типу змінних, так і самих змінних. Інтерпретатор мови виконує рядки сирцевого коду один за одним без попередньої компіляції усього модулю.

Інтерпретатор мови PHP доступний на усіх популярних платформах, що у свій час призвело до значного зростання популярності мови, що, вкупі

з її надзвичайною простотою для новачків, сформувало надзвичайно низький поріг входження у програмування мовою PHP. Це призвело до того, що у вільному доступі наявна безліч бібліотек та модулів мовою PHP, але їх велика частка була написана розробниками або із початковим рівнем компетенції, або без використання спільних угод до проектування бібліотек та стилю написання коду. Деякі з таких бібліотек були включені до списку вбудованих бібліотек мови, а це в свою чергу призвело до помітної неузгодженості синтаксису різних бібліотек.

Відомим недоліком мови PHP є порівняно велика складність відловлювання помилок, а отже й відлагодження помилок у написаних скриптах.

У той же час, мова приваблює веб-мастерів порівняно невеликою вартістю розгортання веб-серверу із працюючим додатком. Для створення веб-сайтів найчастіше використовуються такі фреймворки як Zend Framework, Yii, Symphony тощо.

### *2.1.3. C#*

C# – сучасна об’єктно-орієнтована мова програмування загального призначення, що функціонує в рамках розробленої компанією Microsoft платформи .NET [12]. Фреймворк .NET призначений, у першу чергу, для створення та запуску мультиплатформних додатків, написаних, окрім власне мови C#, мовами C++, F#, Visual Basic, COBOL тощо.

Особливостями мови C#, що відрізняють її від інших мов, є такі:

- наявність інструментів для реалізації усіх традиційних парадигм та шаблонів ООП;
- типобезпечність, яка дозволяє виявити більшість можливих помилок ще на етапі компіляції програми;
- простий для розуміння новачками синтаксис;
- величезна бібліотека вбудованих модулів та компонентів;
- автоматичне керування пам’яттю (наявність збирача сміття)

- гігантська спільнота розробників;
- постійне еволюціонування мови.

Безумовним лідером серед фреймворків для розроблення серверної частини веб-додатків є фреймворк ASP.NET та його остання версія ASP.NET Core, що працює на новій платформі .NET Core, головним нововведенням якої є не тільки значне покращення швидкодії, а й можливість запуску на платформах macOS та Linux, що значно знижує витрати на розгортання серверу із запущеним веб-додатком.

### *2.1.4. JavaScript*

JavaScript – динамічна прототипова скриптова мова програмування, призначена у першу чергу для реалізації складної взаємодії користувача із веб-сторінкою; є одним з основних інструментів побудови клієнтської частини веб-сторінок разом із HTML (Hypertext Markup Language) – мовою розмітки елементів веб-сторінки та CSS (Cascading Style Sheets) – мовою опису стилів веб-сторінки [13]. Окрім динамічного керування контентом веб-сторінки, мова дозволяє керувати мультимедіа-ресурсами, анімувати зображення, створювати інтерактивні користувацькі форми і т.д.

Хоча сучасний стандарт мови JavaScript – ES7 – підтримує велику кількість інструментів для реалізації концепцій ООП (класи, об’єкти, методи об’єктів та класів і т.д.), найбільш популярною є функціональна парадигма програмування – стиль побудови програм, при якому хід роботи програми уявляється як результат обчислення математичних функцій, які при цьому уникають наскільки це можливо зміни стану даних (мутації).

Визначною особливістю JavaScript відносно інших мов програмування є її асинхронна модель виконання коду, що реалізується відповідними двигунами браузерів (V8 у Google Chrome, SpiderMonkey в Mozilla Firefox [14]), які замість синхронної та послідовної реакції на події на веб-сторінці будують з них асинхронну чергу подій, що у свою чергу обробляється так званим «циклом подій». Така модель виконання дозволяє

виконувати одну задачу під час очікування результату виконання іншої (наприклад, HTTP-запиту до зовнішнього ресурсу).

Саме асинхронна модель подій двигуна V8 лягла в основу фреймворку для виконання JavaScript на сервері під назвою Node.js [15], який окрім повноцінного інтерпретатору JavaScript (V8) містить у собі допоміжні бібліотеки та модулі для взаємодії із операційною системою (файловую системою, сокетами тощо).

Таблиця 2.1

Порівняння мов програмування для розроблення серверної частини

| | Інтерпретована модель виконання | Функціональна парадигма | Велика кількість бібліотек | Фреймворк для веб-серверу |
|---|---|---|---|---|
| Python | + | +/– | + | + |
| PHP | + | – | +/– | + |
| C# | – | – | + | + |
| JavaScript | + | + | + | + |

Після проведення аналізу популярних мов програмування для реалізації серверної частини було обрано мову JavaScript, яка не тільки відповідає всім встановленим вимогам, а й дозволяє повторно використати деяку частину написаного коду у клієнтській частині системи.

## 2.2. Вибір технології для розроблення клієнтської частини

Основними вимогами до технології (фреймворку) для реалізації клієнтської частини є:

- швидкість оновлення стану сторінки;
- компонентний підхід до архітектури;
- наявність великої кількості модулів або бібліотек.

### *2.2.1. Angular*

Angular – клієнтський фреймворк для побудови односторінкових веб-додатків [16]. В якості основного архітектурного шаблону Angular використовує підхід Model-View-Controller (MVC):

- Model (модель): структура даних, що керує інформацією та отримує дані від контролера;
- View (вид): представлення інформації;
- Controller (контролер): реагує на користувацьке введення даних та відповідним чином взаємодіє із моделлю.

Архітектура додатків у Angular основана на компонентному підході, де компоненти представляються у вигляді невеликих частин інтерфейсу, незалежних одна від одної. Завдяки цьому досягається не тільки висока читабельність та підтримуваність таких компонентів для інших розробників, а й можливість перевикористовувати окремі компоненти в абсолютно різних частинах додатку.

Angular написаний із використанням мови TypeScript, яка в своїй основі є надбудовою над JavaScript. Вона повністю компілюється у JavaScript, але дозволяє помітити та виправити найчастіші помилки ще під час написання коду: маленькі проекти можливо і не потребують такого покращення, але для великих проектів enterprise-рівня це призводить до значного покращення якості та чистоти коду [17]. З іншого боку, використання TypeScript у проекті вимагає в розробників великої кількості часу для вивчення та адаптування до особливостей роботи Angular.

За декілька років свого існування навколо фреймворку Angular зібралася велика спільнота розробників, було створено велику кількість сторонніх модулів, а корпорація Google підтвердила свої наміри про довгострокову підтримку розроблення фреймворку [18].

### *2.2.2. React*

React – бібліотека, написана мовою JavaScript, призначена для побудови користувацьких інтерфейсів [19].

В основі усіх React-додатків лежать компоненти: модулі, які показують певне виведення даних. У вигляді компонентів можна представити такі елементи як, наприклад, кнопка або поле введення. Компоненти можна об'єднувати: один компонент може містити у своєму виведенні один або більше інших компонентів. Таким чином, компоненти вибудовуються у багатокомпонентну деревовидну структуру від компонентів низького рівня до компонентів високого рівня, формуючи інтерфейс веб-додатку.

Важливою особливістю внутрішньої будови бібліотеки React є використання технології Virtual DOM: замість маніпулювання об'єктною моделлю документу при кожній зміні стану додатку напряму, React застосовує зміни до DOM, який повністю побудований та запущений у пам'яті додатку. Після оновлення Virtual DOM, React у ефективний спосіб обчислює, які саме зміни треба внести до справжнього браузерного DOM. Час маніпуляцій із віртуальним DOM є мізерно малим відносно маніпуляцій зі справжнім DOM, отже такий підхід дозволяє отримати величезний приріст у швидкості оновлень стану веб-додатку.

Унікальною особливістю React стало використання шаблонів JSX, що є дуже подібними до HTML і використовуються для того, аби код мовою JavaScript та розмітка могли існувати разом у XML-подібній структурі, фактично вбудовуючи логіку (значення змінних, умовні вирази тощо) у розмітку. На етапі попереднього оброблення коду пакувальником webpack такий синтаксис трансформується у звичайний JavaScript. Такий підхід дозволяє значно покращити читабельність компонентів та підвищує розуміння їх коду нетехнічними спеціалістами (наприклад, дизайнерами).

Бібліотека React активно підтримується компанією Facebook, а кількість доступних для використання бібліотек та модулів для. React вимірюється тисячами.

Проведений аналіз свідчить про те, що обидві технології повністю або частково відповідають встановленим вимогам. Це підтверджується також і статистикою пошукових запитів Google [20], у якій React є лише трохи більш популярним, ніж Angular.

В результаті, для розроблення клієнтської частини було обрано бібліотеку React, адже її підхід до створення компонентів є ближчим до функціональної парадигми, яка використовуватиметься і при створенні серверної частини веб-додатку. Крім того, їх код є більш виразним та легшим для подальшого підтримування, а використання бібліотеки React не потребує додаткового налаштування середовища до роботи із іншою мовою, як цього потребує Angular із TypeScript.

## 2.3. Вибір СУБД

В результаті аналізу початкових вимог було висунуто такі вимоги до СУБД, що використовуватиметься при побудові серверної частини системи:

- вільне розповсюдження (безкоштовність);
- можливість розгортання на віддаленому сервері;
- реляційна модель даних;
- відповідність вимогам ACID;
- підтримка індексів;
- підтримка повнотекстового пошуку.

### *2.3.1. MySQL*

MySQL – система управління реляційними базами даних, що розроблюється та вільно розповсюджується компанією Oracle [21]. Написана мовами C та C++, СУБД MySQL є сумісною з усіма основними операційними системами.

Основними особливостями СУБД MySQL є:

1. Повна відповідність вимогам ACID.
2. Простота вивчення та використання: відмінність синтаксису MySQL від стандарту SQL є незначною, підтримуються всі наявні в SQL типи даних.
3. Клієнт-серверна архітектура: MySQL виступає в ролі сервера, до якого під'єднуються незалежні клієнти – додатки для взаємодії із базою даних.
4. Ємність: СУБД може вмістити практично будь-яку кількість інформації.
5. Наявність транзакцій, рівнів ізоляції транзакцій та функції їх відката.
6. Наявність тригерів, збережуваних процедур та представлень.
7. Розповсюдженість: існують бібліотеки для використання СУБД для абсолютної більшості мов програмування.
8. Невисока ціна розгортання та утримання відносно конкурентів.

До відомих вад MySQL відносять [22]:

1. Важкість масштабування: налаштування ефективного горизонтального масштабування є нетривіальною задачею.
2. Важкість самостійного налаштування операцій, наявних «з коробки» в інших СУБД, наприклад налаштування інкрементного резервного копіювання
3. Падіння швидкодії на надскладних запитах.

MySQL здобула значну популярність серед розробників веб-орієнтованих додатків через велику кількість спеціальних функцій, таких як, наприклад, типи даних HTML. Вона є частиною популярної архітектури для створення повноцінних веб-додатків LAMP: Linux, Apache, MySQL, PHP.

### *2.3.2. PostgreSQL*

PostgreSQL – популярна відкрита реляційна СУБД, яка часто використовується при створенні веб-додатків [23]. Вона була одною з перших СУБД, що з'явилися на ринку. Дозволяє оперувати як структурованими, так і неструктурованими даними. Може бути запущена на більшості сучасних операційних систем. Містить зручні вбудовані інструменти для імпорту даних із інших СУБД.

Визначними перевагами СУБД PostgreSQL є:

1. Повна відповідність вимогам ACID.
2. Клієнт-серверна архітектура: сервер підтримує значну кількість одночасних окремих підключень від додатків-клієнтів.
3. Дуже потужний повнотекстовий пошук, наявна підтримка регулярних виразів.
4. Великі можливості масштабування бази даних.
5. Можливість виконувати код, написаний деякими популярними мовами програмування, прямо на сервері СУБД.
6. Великі можливості управління транзакціями.
7. Можливість зберігати XML, JSON та багато інших унікальних типів даних.

Відомими недоліками PostgreSQL є:

1. Неповна підтримка стандарту ANSI SQL.
2. Помітне падіння швидкодії при групових операціях.
3. Порівняна складність встановлення та конфігурації для новачків.

В результаті проведеного аналізу було встановлено, що обидві розглянуті СУБД частково або повністю відповідають початковим вимогам. Тим не менш, для використання при розробленні серверної частини системи було обрано СУБД PostgreSQL, у тому числі через її порівняно більші можливості повнотекстового пошуку, а також здатність зручно зберігати та взаємодіяти із JSON.

# 3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

## 3.1. Опис вимог до розроблюваної системи

Фаза збору та опису вимог найчастіше є найпершою у довгому процесі розроблення нового програмного забезпечення. Вона дозволяє перетворити людські ідеї та погляди у чітку та конкретну специфікацію вимог.

Фаза збору вимог включає в себе набір задач, які допомагають визначити вплив програмного забезпечення на організацію, потреби клієнтів, а також як саме користувачі взаємодіятимуть із розроблюваним ПЗ. Очевидно, що наявність помилки у специфікації вимог призведе до помилок у кінцевому продукті. При цьому доведено [24], що вартість усунення помилки у вимогах ще на етапі їх збору є в десятки разів меншою відносно вартості виправлення цієї ж помилки коли кінцевий розроблюваний продукт вже готовий.

Згідно зі стандартом розроблення вимог ISO 29148, вимога – це певний стан або можливість, яку повинні містити система або компонент системи для відповідності контракту, стандарту, специфікації або іншому формально визначеному документу.

Виділяють такі типи вимог:

- функціональні вимоги;
- нефункціональні вимоги;
- бізнес-вимоги.

Функціональні вимоги описують бажані функціональні особливості системи та її компонентів – правила, за якими система реагує (або не реагує) на певні вхідні дані. Ця поведінка може включати в себе обчислення, управління даними, деякий бізнес-процес, користувацьку взаємодію тощо.

Нефункціональні вимоги пов’язані із такими системними властивостями як надійність та час відгуку. Вони з’являються через наявність певних користувацьких вимог, бюджетних обмежень, корпоративних політик тощо і не пов’язані прямо із якоюсь конкретною

функцією системи. Розроблюване ПЗ має відповідати встановленим нефункціональним вимогам для того, аби зробити його роботу більш ефективною.

До типів нефункціональних вимог належать:

- вимоги до ефективності: опис границь використання ресурсів, швидкості виконання та споживаної пам’яті;
- вимоги до надійності: опис допустимої частоти відмов;
- вимоги до зручності використання: мінімальна кількість переходів для певної дії, вимоги до інтерфейсу тощо;
- корпоративні вимоги: походять від корпоративних політик та процедур;
- вимоги до розповсюдження: яким чином кінцевий продукт має доставлятися користувачу;
- вимоги до реалізації: опис бажаних архітектурних підходів, мов програмування, цільових платформ.

В результаті проведення аналізу предметної галузі було сформульовано такий набір функціональних вимог до розроблюваного ПЗ:

1. Система має надавати можливість реєстрації нового користувача та автентифікації існуючого користувача.
2. Система має визначати приблизний рівень знання користувачем лексики англійської мови шляхом проведення тесту.
3. Система має надавати користувачеві можливість отримати переклад слова на веб-сторінці, яку той переглядає.
4. Система має дозволяти користувачеві зберегти слово з веб-сторінки для подальшого заучування засобами системи.
5. Система має дозволяти користувачеві переглядати список раніше збережених слів та дані про них, видаляти збережені та додавати нові слова до нього.

6. Система має обирати слова із списку збережених слів за методом інтервальних повторень та відображати їх користувачеві по одному під час кожного початку взаємодії із системою.
7. Система має надавати можливість вказати приблизний рівень володіння слова користувачем для подальшого використання у роботі системи.

Проілюструємо сформульовані функціональні вимоги відповідними діаграмами прецедентів (англ. “use case diagram”).

Прецедент 1: «Зареєструватися та/або авторизуватися у системі» – описує процес початку взаємодії із системою (рис. 3.1).

1. Користувач відкриває додаток.
2. Якщо користувач не має створеного раніше облікового запису, він може пройти процедуру реєстрації.
3. Якщо користувач вже був зареєстрований, він вводить свої дані для авторизації (логін та пароль).
4. Якщо користувач не може згадати пароль від свого облікового запису, він може зробити запит на відновлення паролю та згенерувати новий пароль, який після цього він зможе використати для входу на сайт.

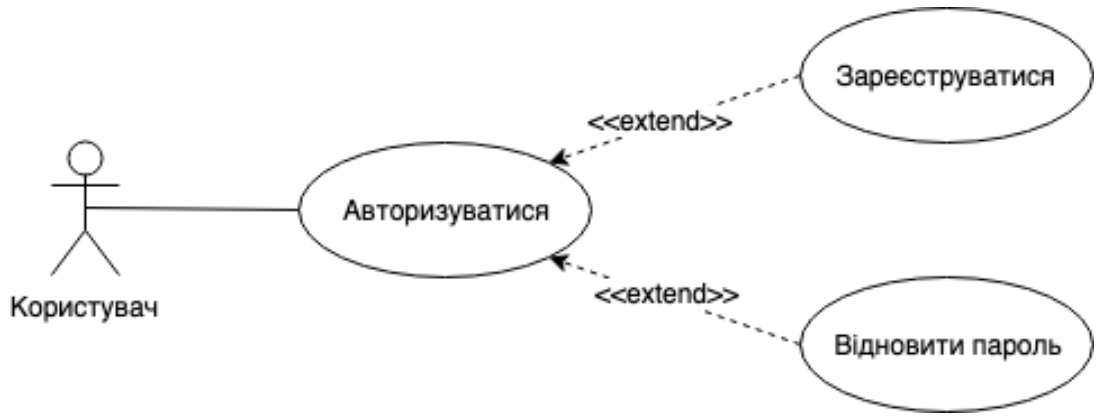



Рис. 3.1. Прецедент «Зареєструватися та/або авторизуватися у системі»

Прецедент 2: «Пройти початковий тест на визначення рівня знання англійської лексики» (рис. 3.2).

1. Користувач починає тест.
2. Користувачеві показується слово англійською та 4 варіанти його перекладу.
3. Користувач обирає правильний, на його думку, варіант перекладу слова.
4. Користувач виконує кроки 2-3 доки не встановлено приблизний рівень його знання англійської лексики.
5. Користувач закінчує тест.


Рис. 3.2. Прецедент «Пройти початковий тест на визначення рівня знання англійської лексики»

Прецедент 3: «Отримати переклад слова у браузері» (рис. 3.3).

1. Користувач натискає на будь-яке слово на веб-сторінці свого браузера.
2. Користувач переглядає переклад слова, його визначення, приклади використання тощо.
3. Користувач може перейти за посиланням до онлайн-словнику з більш повною інформацією та більшою кількістю прикладів використання слова.
4. Користувач може зберегти слово до свого користувацького словника для подальшої взаємодії із ним.

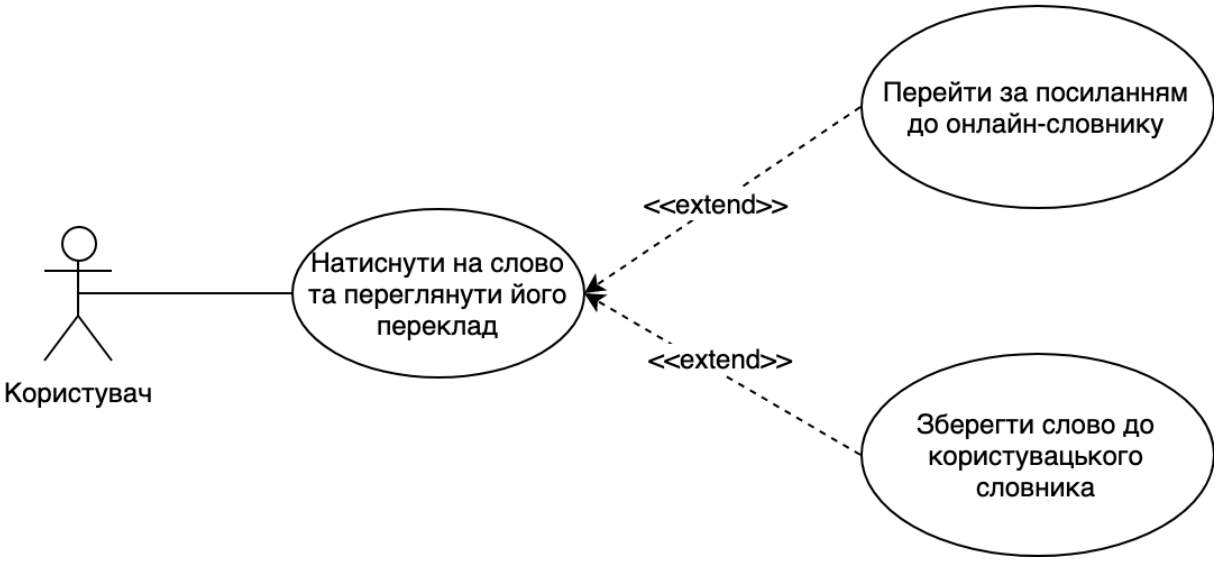



Рис. 3.3. Прецедент «Отримати переклад слова у браузері»

Прецедент 4: «Отримувати інформацію про потенційно незнайомі слова на сторінці».

1. Користувач активує режим підсвічування потенційно незнайомих слів на веб-сторінці.
2. Користувач може натиснути на слово та переглянути його переклад.

Прецедент 5: «Взаємодіяти зі списком збережених слів» (рис. 3.4).

1. Користувач переходить на нову вкладку браузера, де запущений додаток.
2. Користувач переходить до списку слів у пункті меню додатку.
3. Користувач переглядає список збережених ним слів.
4. Користувач може переглянути відомості про слово із списку.
5. Користувач може видалити слово із списку.
6. Користувач може знайти та додати нове слово до списку.

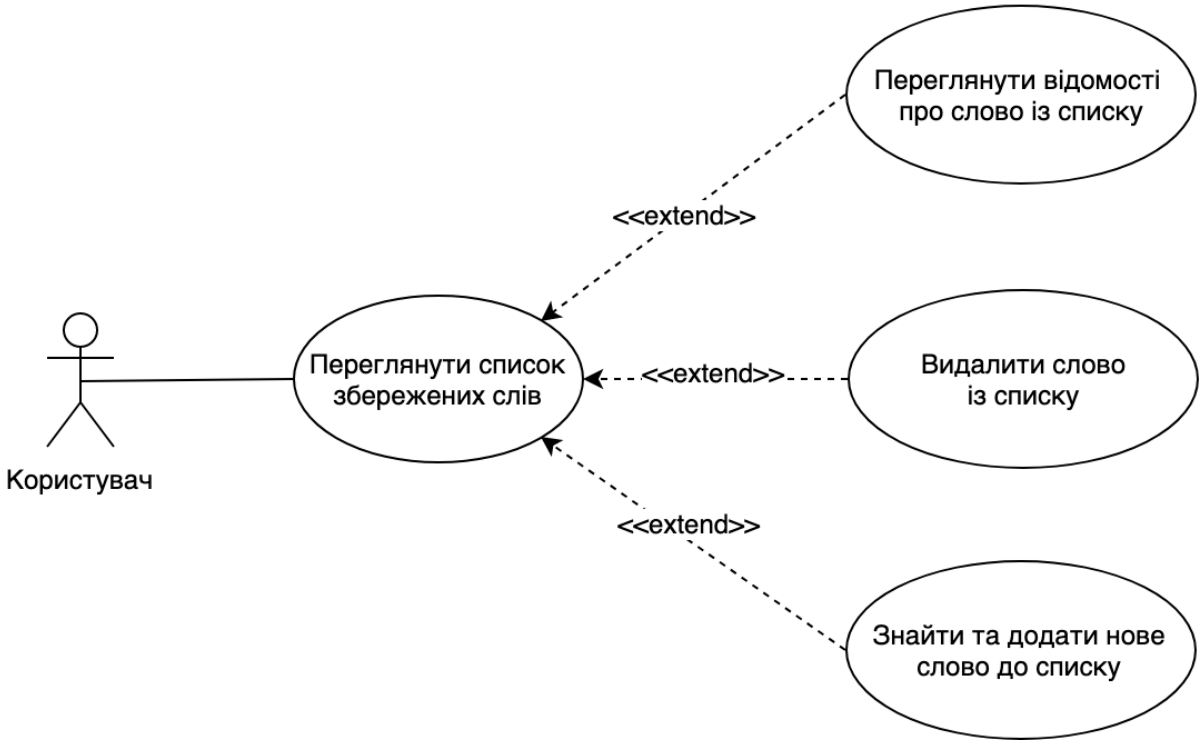



Рис. 3.4. Прецедент «Взаємодіяти зі списком збережених слів»

Прецедент 6: «Заучувати нові слова за методикою інтервальних повторень» (рис. 3.5).

1. Користувач відкриває нову вкладку із запущеним додатком, йому показується збережене ним раніше слово, обране системою за методикою інтервальних повторень.
2. Користувач може переглянути переклад та відомості про показуване слово.
3. Користувач оцінює та вказує рівень свого знання цього слова (наскільки швидко згадав, чи співпав переклад тощо).
4. Користувач може перейти до іншого слова зі списку раніше збережених.


Рис. 3.5. Заучувати нові слова за методикою інтервальних повторень»

Сформулюємо та опишемо нефункціональні вимоги до розроблюваного ПЗ, згрупувавши їх по категоріях: вимоги до безпеки описані в табл. 3.6, вимоги до продуктивності – в табл. 3.7, вимоги до реалізації – в табл. 3.8.

Таблиця 3.6

Вимоги до безпеки

| Код | Опис вимоги |
|---|---|
| SEC-1 | Паролі користувачів повинні зберігатися у захешованому вигляді. |
| SEC-2 | Система повинна проводити перевірку усіх даних, що надходять від користувача. |
| SEC-3 | Система повинна бути захищена від поширених видів атак (XSS, CSRF, SQL Injection). |

Таблиця 3.7

Вимоги до продуктивності

| Код | Опис вимоги |
|---|---|
| PRF-1 | Система повинна витримувати 1000 одночасних запитів. |
| PRF-2 | Максимально допустима затримка перекладу слова при запиті – 1 секунда |

Таблиця 3.8

Вимоги до реалізації

| Код | Опис вимоги |
|---|---|
| IMP-1 | Аналізований на предмет наявності потенційно незнайомих слів текст має включати в себе як мінімум основні блоки сторінки, наприклад текст статті на сайті-блозі. |
| IMP-2 | Переклад слів, які обирає користувач, має відбуватися як мінімум для 85% слів, що не є власними назвами. |
| IMP-3 | Розширення для веб-браузера повинно коректно працювати у браузері Google Chrome на операційних системах Windows, Ubuntu Linux, macOS. |
| IMP-4 | Серверна частина розширення має бути розгорнута на віддаленому сервері на платформі Ubuntu Linux. |

### 3.2. Опис архітектури системи

В результаті проведення аналізу популярних сучасних архітектур веб-додатків було обрано архітектуру Single Page Application (укр. «односторінковий додаток»), адже саме вона найкраще підходить для розроблення браузерного розширення [25].

Сучасні ефективні веб-додатки проектуються аби запитувати від сервера лише вкрай необхідну інформацію для генерації інтуїтивно зрозумілого та інтерактивного користувацького досвіду. Односторінкові веб-додатки взаємодіють із користувачем у більш привабливий спосіб, оновлюючи контент прямо на поточній сторінці, не перезавантажуючи сторінку повністю у відповідь на кожну виконану користувачем дію, як це було популярно раніше. Це дає змогу позбутися переривань користувацької взаємодії, трансформуючи поведінку додатку до такої, що є більш схожою на поведінку традиційного встановленого системі додатку. Створення односторінкових додатків стало можливим за появи технології AJAX, що дає змогу відправляти HTTP-запити до API-серверу, не перезавантажуючи веб-сторінку.

Після проведення дослідження способів організації компонентів системи в рамках обраної архітектури було спроектовано наступну архітектуру розроблюваної системи, зображено на рис. 3.9.

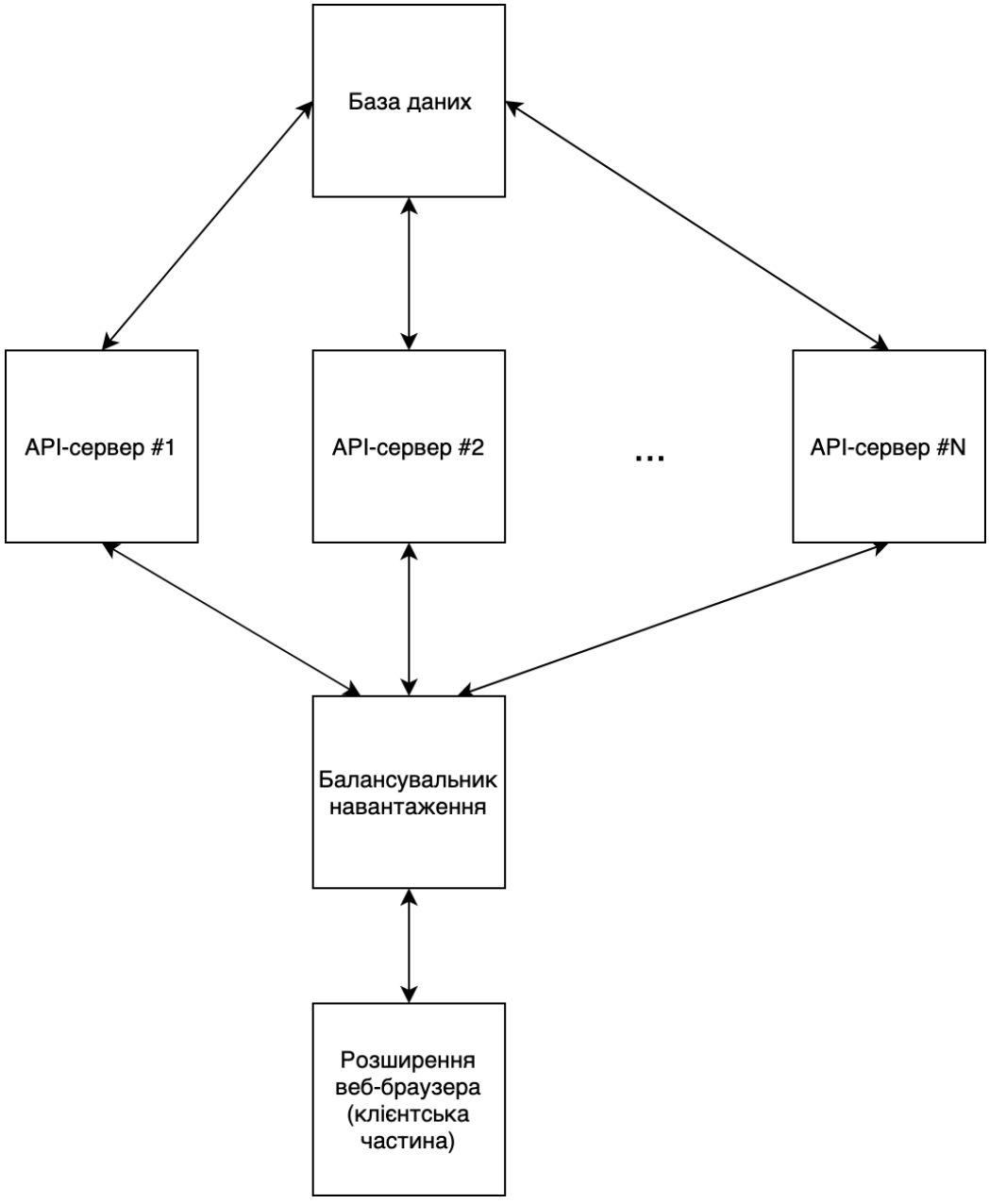



Рис. 3.9. Архітектура системи

Розглянемо розроблену архітектуру детальніше. Розширення браузера, що розгортається безпосередньо в браузері користувача і містить у собі клієнтський код односторінкового додатку (SPA), взаємодіє із API системи за допомогою технології AJAX, отримуючи усі необхідні для

коректної користувацької взаємодії дані. При цьому HTTP-запити від розширення виконуються не безпосередньо до серверу, на якому розгорнута серверна частина системи, а до спеціального балансувальника навантаження.

Балансувальник навантаження (англ. “load balancer”) – це сервіс (мережеве ПЗ), що розподіляє мережевий трафік між групою (кластером) серверів. Він встановлюється у мережі між клієнтами та серверами, що здатні обробити цей запит, та перенаправляє клієнтські запити таким чином аби максимізувати швидкість та пропускну спроможність серверів, унеможливлюючи ситуації, коли один з серверів перевантажений роботою у той час як інші простоюють. У випадку, якщо один сервер перестає відповідати на клієнтські запити (відмовляє в обслуговуванні), трафік розподіляється по серверам, що залишилися. У випадку додавання нового серверу балансувальник автоматично починає відправляти запити на нього.

Якщо система розгорнута в інфраструктурі хмарного провайдера (Microsoft Azure, Amazon AWS, Google Cloud [26]), балансувальник може автоматично створити та запустити додатковий сервер у випадку, якщо усі існуючі сервери перевантажені роботою. Таким чином досягається майже гарантована стійкість сервера до раптового підвищення обсягу користувацького трафіку.

Перевагами використання балансувальника навантаження є [27]:

1. Доступність. Балансувальник навантаження автоматично слідкує за станом підключених серверів та перенаправляє користувацькі запити лише на ті сервери, що готові їх приймати.
2. Гнучкість. Використання балансувальника навантаження в якості шлюзу надає можливість змінювати серверну інфраструктуру без впливу на доступність сервісів, надаючи можливість проводити горизонтальне розширення, «безшовні» розгортання, значні архітектурних перепроектування тощо.

3. Продуктивність. Розподілення оброблення навантаження на групу серверів замість одного сервера виключає можливість відмови усієї системи від переповнення користувацькими запитами.

Група серверів, розташована за балансувальником навантаження, виконує запити до єдиної реляційної бази даних. Можливість виконувати одночасні запити до бази даних на модифікацію, можливо, одних і тих самих даних, забезпечується за допомогою використання, де це потрібно, транзакційних запитів - запитів, що складаються із дії або серії дій, направлених на читання та/або оновлення бази даних, що виконуються серверною частиною і сприймаються базою даних як єдина логічна та нерозривна одиниця роботи, незалежна від інших запитів.

Метою використання транзакцій до бази даних в рамках розроблюваної архітектури є:

1. Забезпечення ізоляції між користувацькими запитами, що одночасно доступаються до одних і тих самих даних у БД.
2. Надання способу коректного відновлення стану бази даних в разі виникнення помилок при виконанні групи зв’язаних запитів.

### 3.3. Опис структур даних системи

В результаті аналізу сформульованих вимог було спроектовано такі сутності, що зберігатимуться у базі даних системи:

1. Користувач (User)
    a. Ідентифікатор (ID) – унікальний ідентифікатор користувача у системі.
    b. Ім’я (Name) – ім’я користувача для відображення у системі.
    c. Електронна пошта (Email) – використовуватиметься для виконання авторизації, має бути унікальним відносно інших користувачів.
    d. Пароль (Password) – відповідно до вимоги SEC-1 буде зберігатися у захешованому вигляді.

e. Рівень знання англійської лексики (Proficiency level) – кількісний показник, що характеризуватиме приблизний середній рівень лексичної компетенції користувача.
f. Дата створення (Created at) – дата реєстрації користувача.
g. Дата оновлення (Updated at) – дата останнього оновлення даних користувача.

2. Слово (Word)
    a. Ідентифікатор (ID) – унікальний ідентифікатор слова у таблиці.
    b. Слово (Word) – слово англійською мовою у лематизованій формі.
    c. Визначення (Definitions) – масив визначень слова із відповідними прикладами використання.
    d. Частина мова (Part of speech) – граматична категорія слова.
    e. Вимова слова (Pronunciation) – вимова слова, записана за правилами міжнародного транскрипційного алфавіту.
    f. Частота (Frequency) – частота використання слова в корпусі англійської мови.
    g. Дата створення (Created At) – дата додавання слова до БД.
3. Слово, що вивчається (Learned Word)
    a. Ідентифікатор (ID) – унікальний ідентифікатор слова, що вивчається.
    b. Ідентифікатор користувача (User ID) – унікальний ідентифікатор користувача (User), що додав слово до свого списку слів, що вивчаються.
    c. Ідентифікатор слова (Word ID) – унікальний ідентифікатор слова (Word), що вивчається.
    d. Прогрес вивчення (Learning Progress) – характеристика поточного рівня знання цього слова користувачем.

e. Наступна дата повторення (Next Repetition) – дата та час наступної взаємодії користувача зі словом за методикою інтервальних повторень.
f. Дата створення (Created At) – дата початку вивчення слова користувачем.

4. Запис про взаємодію зі словом (Word Interaction Record)
   a. Ідентифікатор (ID) – унікальний ідентифікатор взаємодії користувача зі словом за методом інтервальних повторень.
   b. Ідентифікатор слова, що вивчається (Learned Word Id).
   c. Рівень знання слова (Knowledge level) – рівень знання, вказаний користувачем після взаємодії.
   d. Дата взаємодії (Interacted At) – дата та час користувацької взаємодії із словом.

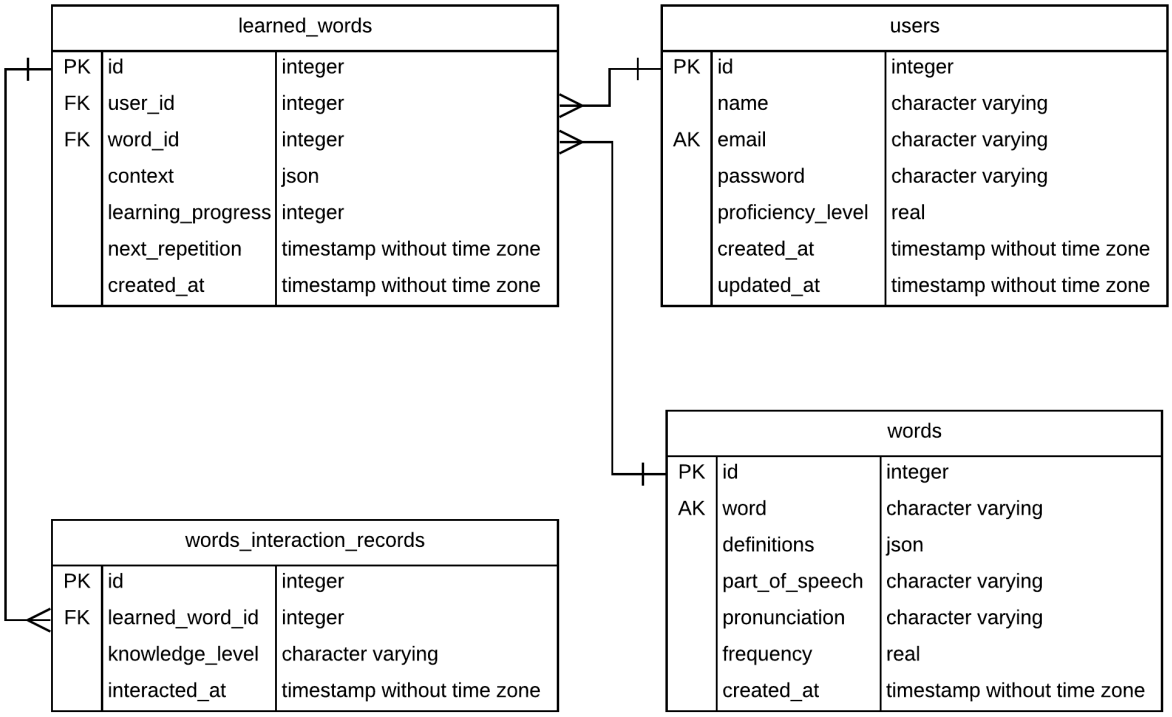



Рис. 3.10. Структура бази даних додатку

Опис розроблених сутностей було зведено до діаграми, на якій також позначено відповідні зв’язки між таблицями та типи даних колонок таблиць (див. рис. 3.10).

### 3.4. Алгоритм показу слів за методом інтервального повторення

Дослідження можливостей людини запам’ятовувати інформацію показали, що після того як наш мозок вперше «записує» якусь інформацію до довгострокової пам’яті, нам необхідно повернутися до цієї інформації ще кілька разів аби збільшити шанси на те, що потім ми зможемо згадати цю інформацію, коли вона нам насправді знадобиться. Ці дані лягли в основу концепції так званої «кривої забування», вперше сформульованої німецьким психологом Германом Еббінгаузом [28]. Ця крива (див. рис. 3.11) ілюструє частину нової інформації, яку людина може згадати, як функцію від часу, що пройшов із моменту заучування. Можна побачити, що вже після 24 години після заучування втрачається більше двох третин інформації, а за тиждень – більше 80% вивченого.

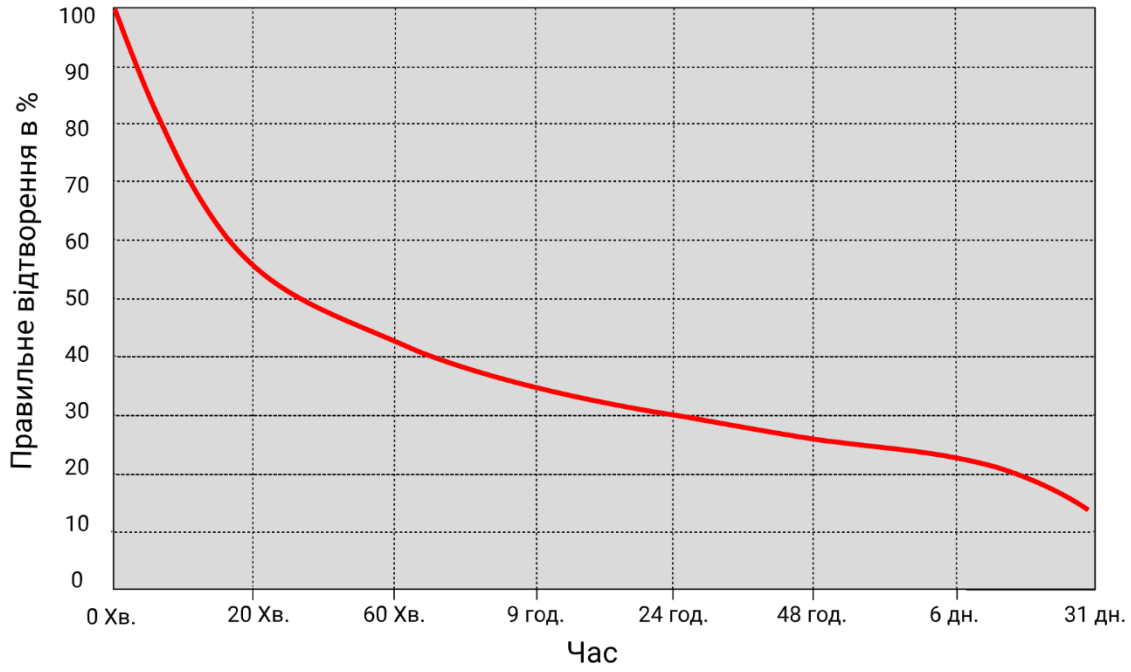



Рис. 3.11. Крива забування Еббінгауза

Аби подолати цю проблему, було створено систему інтервальних повторень [29]. Основна її ідея в тому, що якщо повертатися до заученої інформації через значний проміжок часу після заучування, але до того, як цю інформацію доведеться заучувати заново, ступінь її забуття надзвичайно різко зменшиться. Саме тому, аби досягти довгострокових результатів, необхідно «освіжати» знання у пам’яті з інтервалами, що поступово збільшуються: вплив цієї техніки на криву забування можна побачити на рисунку 3.12.

Описаний вище підхід є більш ефективним, ніж два інших, які часто практикуються учнями та студентами: намагатися запам’ятати інформацію, прочитавши її з десяток разів підряд без перерви або ж робити надкороткі перерви. Такі дії призводять лише до того, що мозок починає нудьгувати та ігнорувати нову інформацію, абсолютно не запам’ятовуючи її.

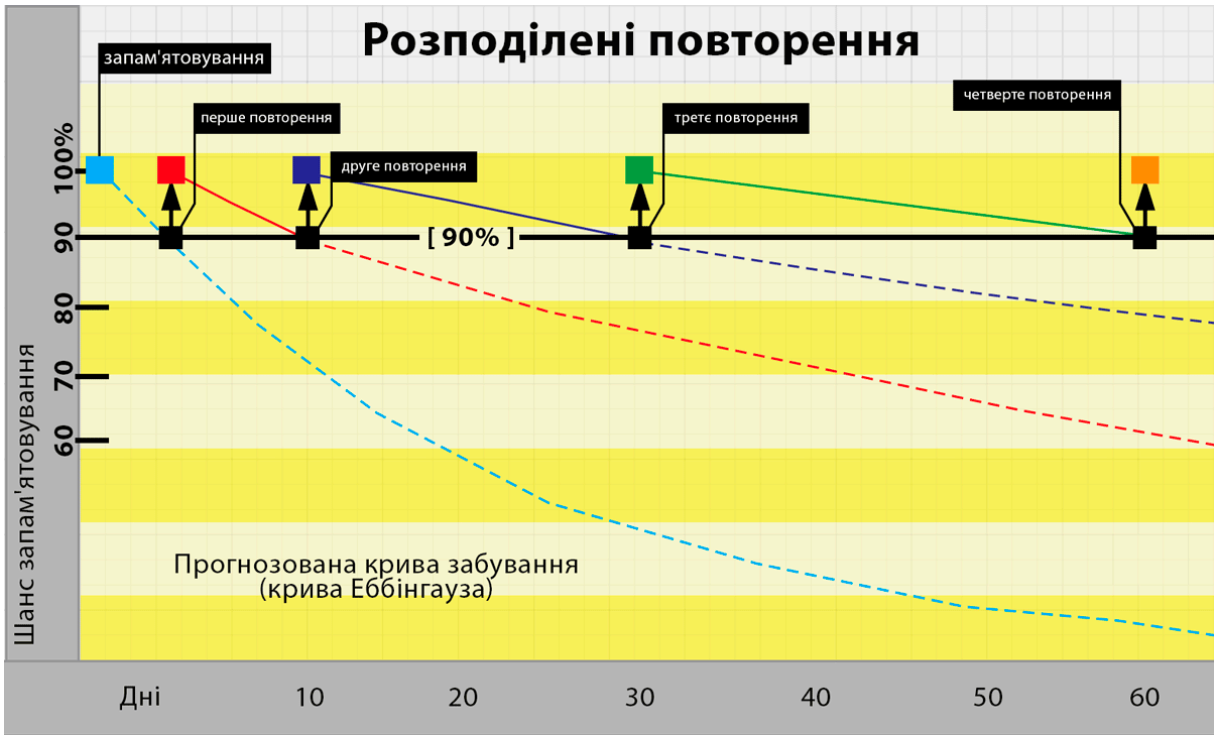



Рис. 3.12. Вплив розподілених повторень на криву забування

Отже, прикладна реалізація алгоритму полягає у показі слова користувачеві із інтервалами, що постійно збільшуються, при цьому в якості

інтервалів у днях було обрано числа Фібоначчі. Під час кожної взаємодії користувач може вказати рівень свого знання (швидкості згадування) цього слова: якщо він високий, наступна дата показу слова обирається як наступний член послідовності Фібоначчі для цього слова, якщо ж низький – слово переноситься для показу на наступний день. Після кількох послідовних позитивних відповідей (через приблизно місяць після початку заучування) слово вважається остаточно вивченим [30].

Опишемо реалізацію алгоритму детальніше.

Під час ініціалізації алгоритму (виконується окремо для кожного слова в кожного користувача) за словом закріплюється:

- progress (прогрес) – кількісна оцінка прогресу користувача у вивченні цього слова, при ініціалізації дорівнює 0;
- dueDate (дата показу) – дата наступного показу слова користувачеві, при ініціалізації дорівнює поточній даті.

Вхідними даними для ініціалізації алгоритму є:

- intervals (інтервали) – масив інтервалів – днів між кожним показом слова користувачеві, за замовчанням дорівнює першим дев’яти натуральним членам послідовності Фібоначчі;
- scoreToProgressChange (зміна прогресу за рівень) – масив цілих чисел, що додаються до progress при отриманні оцінки від користувача, при цьому індекс у масиві дорівнює номеру оцінки.

Після кожного етапу на вхід ініціалізованого алгоритму подається levelIndex (індекс рівня) - порядковий номер рівня, на який користувач оцінив знання слова, що вивчається. Розрахунок зміни прогресу та наступного дня показу слова відбувається за такою логікою, зображеною на рис. 3.13.

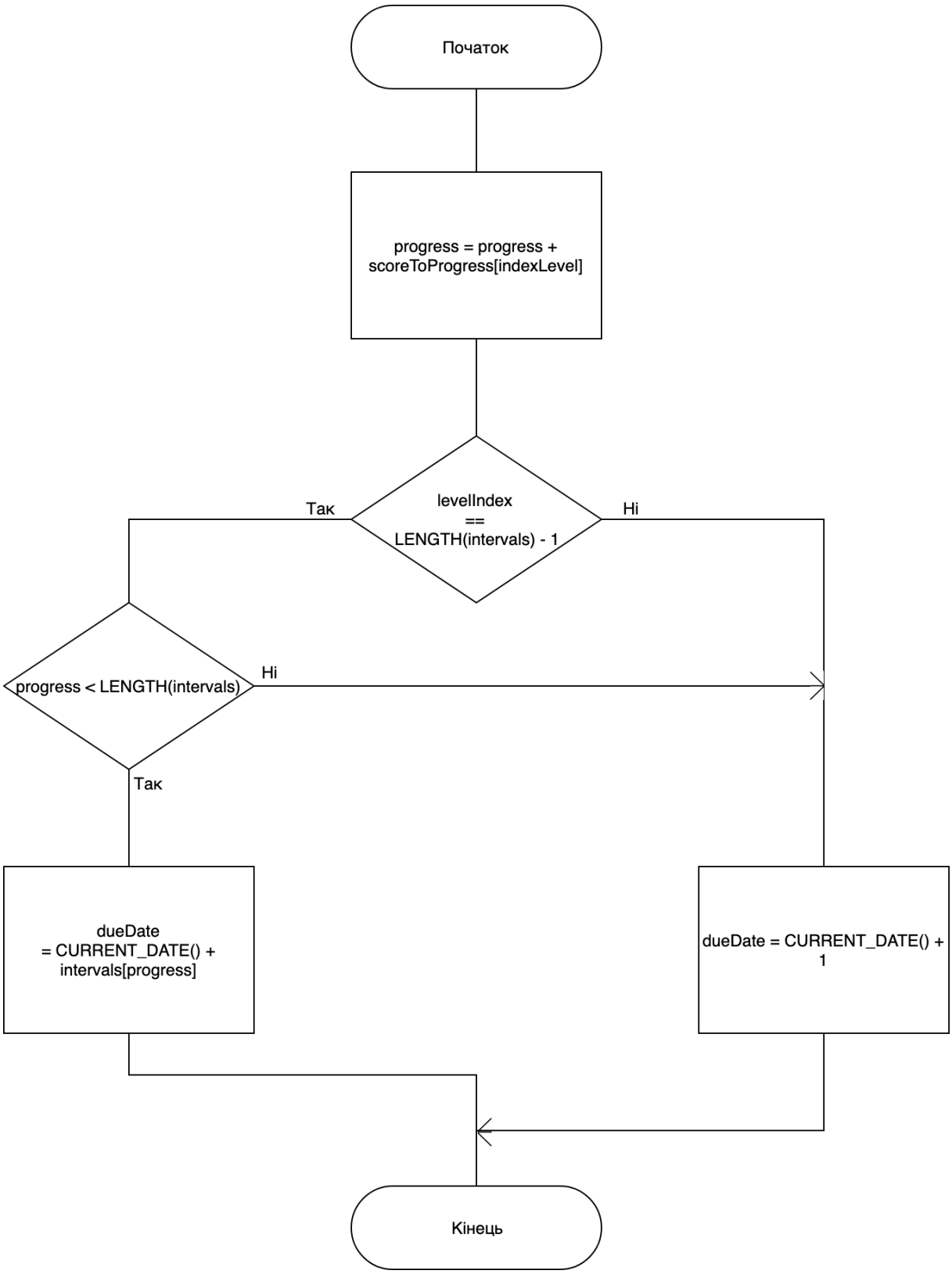



Рис. 3.13. Блок-схема алгоритму розрахунку ступеня нового прогресу та дня показу

Таким чином, якщо користувач постійно вказує рівень знання ним слова як максимальний, його прогрес збільшується, і через деякий час

досягне максимального значення (кількості інтервалів) – тоді слово вважатиметься вивченим.

### 3.5. Алгоритм визначення потенційно незнайомих слів у тексті

Загальновідомо, що абсолютна більшість програм із вивчення іноземних мов (у тому числі англійської) побудовані на такій стратегії, де студент мусить в першу чергу оволодіти словами, які дозволили б якомога швидше почати розуміти найпростіші тексти цією мовою, тобто початковий словниковий запас студента на початку формується із загальновживаних, а отже – найбільш популярних у використанні слів.

Дослідження доводять [31], що обсяг словникового запасу здобувача англійської мови дуже сильно пов’язаний із кількістю взаємодій, які вони мали з кожним конкретним словом (так званий «частотний ефект»). Це дозволяє стверджувати, що частота вживання слова у текстах, з якими стикається студент, а також загалом – в усіх друкованих текстах англійської мови (її корпусі) напряму корелює зі складністю цього слова (наскільки правильно студент може пояснити значення цього слова) [31]. Високочастотні слова, як правило, сприймаються та запам’ятовуються студентом значно швидше, аніж низькочастотні. З цього випливає, що і ймовірність знання студентом значення кожного окремого слова з тексту напряму залежить від частоти вживання цього слова в текстах мови, яка дозволяє передбачити, чи мав студент із конкретним обсягом наявного словникового запасу взаємодії із цим словом, і, якщо мав. чи було цих взаємодій потенційно достатньо для запам’ятовування ним цього слова.

Як наслідок, вхідними даними алгоритму визначення того, чи є слово із тексту потенційно незнайомим користувачеві, є:

- поточна оцінка обсягу словникового запасу користувача, приблизно визначена шляхом проведення відповідного тесту та виражена у вигляді деякого кількісного показника;
- слово англійської мови.

В якості джерела даних про частоту використання конкретного слова в корпусі англійської мови скористаємося агрегованими великими списками у форматі «слово – частота», наявними у вільному доступі в мережі Інтернет.

Таким чином, після проходження користувачем тесту для визначення приблизного рівня знання англійської лексики, встановимо відповідний показник (кількісна оцінка обсягу) як середню частоту вживання слів, правильно визначених користувачем під час тесту, в корпусі англійської мови [32]. Після цього, порівнюючи частоту використання в корпусі отриманого на вхід алгоритму слова із отриманою під час тесту середньою частотою, можемо оцінити ймовірність знання цього слова користувачем, яка і є результатом роботи алгоритму.

При цьому, якщо від користувача буде отримано зворотній зв’язок про те, що він вже знає значення слова, яке було визначено алгоритмом як потенційно незнайоме, кількісна оцінка обсягу словникового запасу користувача, що зберігається у зовнішньому сховищі даних, може бути перерахована в сторону підвищення. Таким чином, із часом алгоритм дозволить більш точно визначити обсяг словникового запасу кожного конкретного користувача та підвищити точність результуючого передбачення.

# 4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1. Особливості реалізації

Серверну частину системи було реалізовано у вигляді монолітного серверного додатку, який взаємодіє із клієнтською частиною за протоколом комунікації REST (Representational State Transfer), який забезпечує низьку зв'язність серверної та клієнтської частини, комунікація між якими відбувається за протоколом HTTP (запити GET, POST, PATCH і т.д.). Веб-сервер реалізує доступний для використання клієнтськими пристроями набір кінцевих точок (API).

Веб-сервіс було реалізовано мовою JavaScript із використанням фреймворків Node.js та Express. Оброблення користувацького запиту відбувається у ланцюжку Middleware (укр. «проміжний рівень»), кожен з яких відповідає за певну частину серверної логіки (див. рис. 4.1). Такий підхід дозволяє повністю відділити рівні абстракції із урахуванням потенційної можливості зміни або заміни реалізації одного з рівнів без необхідності вносити зміни до інших рівнів.

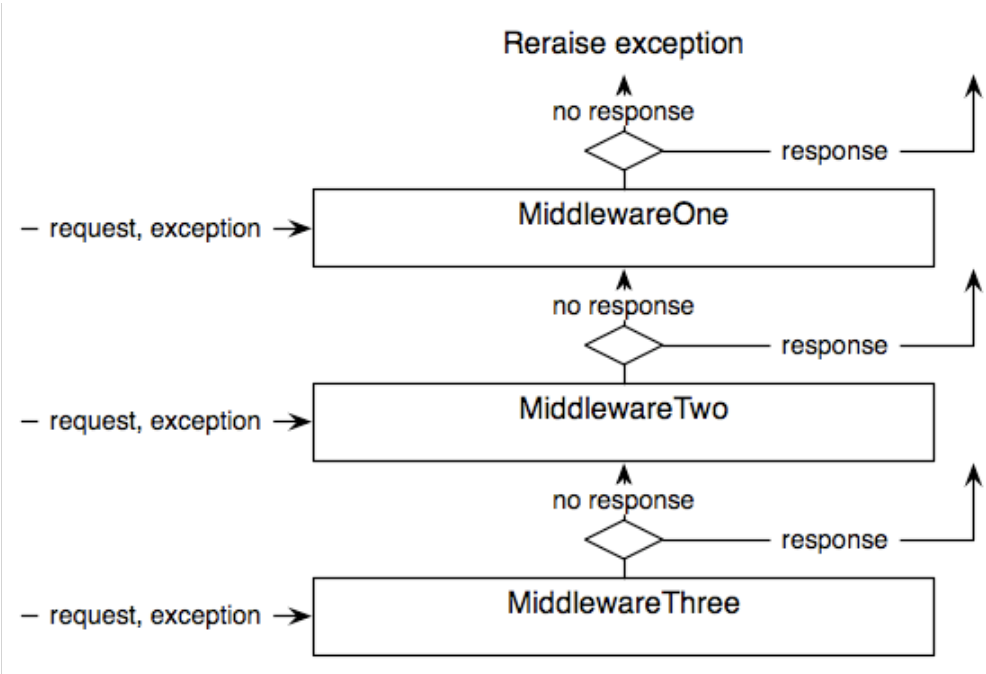



Рис. 4.1. Схема оброблення запитів на сервері

Кожен запит послідовно проходить через такі проміжні рівні оброблення:

1. Рівень оброблення запиту, на набір HTTP-заголовків запиту перетворюється у структурований об’єкт із даними для взаємодії на наступних рівнях.
2. Рівень ідентифікації, на якому унікальний автентифікаійний ідентифікатор користувача співставляється з наявними у БД даними про користувача. У випадку збігу останні додаються до запиту для використання на наступних рівнях.
3. Рівень авторизації, на якому перевіряються права доступу користувача до ресурсу, який він запитує.
4. Рівень бізнес-логіки, на якому власне і виконується бажана користувачем поведінка запитаного ним сервісу.

### *4.1.1. Реалізація сервісу ідентифікації*

Сервіс ідентифікації виконує дві основні задачі:

1. Автентифікація, під час якої виконується перевірка облікових даних користувача шляхом їх порівнювання із збереженими в базі.
2. Авторизація, під час якої виконується перевірка прав користувача на доступ до певних ресурсів.

Механізм авторизації було реалізовано із використанням стандарту JWT (JSON Web Token) – відкритого стандарту компактної та самодостатньої захищеної передачі інформації у вигляді JSON-об’єкту. Захищеність досягається за рахунок використання цифрового підпису (приватного ключу шифрування) на етапі створення повідомлення перед відправкою [33].

Сам JSON Web Token являє собою три блоки, розділених крапкою: заголовок (header), набір полів даних (payload) та сигнатура (signature). Перші два блоки представлені у JSON-форматі та додатково закодовані у формат base64. Набір полів містить довільні парі ключ-значення. Сигнатура

може генеруватися за допомогою як симетричних, так і асиметричних алгоритмів шифрування. Отриманий в результаті токен є засобом авторизації кожного запиту до серверу. Токени генеруються на сервері за допомогою приватного ключа (який зберігається на сервері) та видаються клієнту після успішної автентифікації. Клієнт, у свою чергу, зберігає їх локально та використовує їх (підписує ними) кожен запит, який потребує авторизації (див. рис. 4.2)

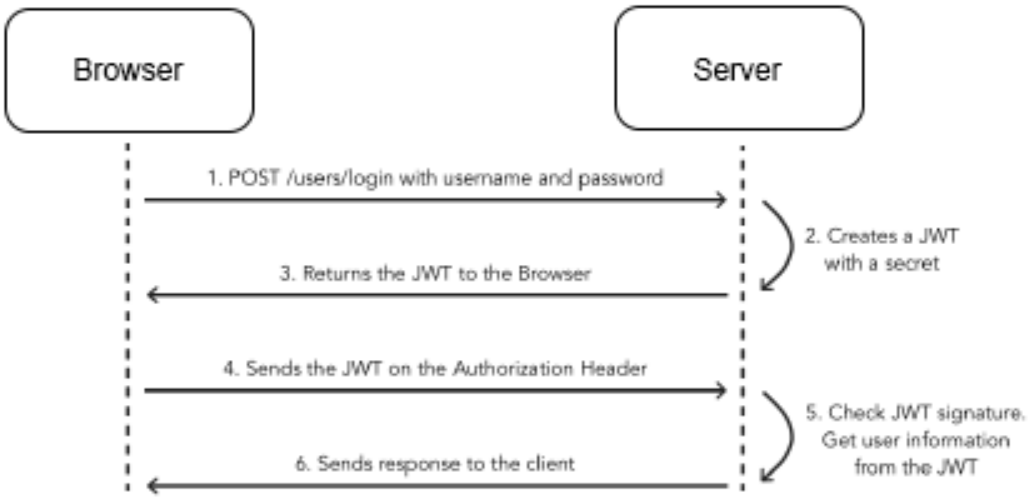



Рис. 4.2. Схема роботи стандарту JWT

Стандарт JWT є універсальним та може використовуватися як для взаємодії серверу із веб-додатками, так і з іншими типами: мобільні, десктопні тощо.

### *4.1.2. Реалізація сервісу визначення потенційно незнайомих слів*

Для реалізації алгоритму, описаного у підрозділі 3.5, було проведено дослідження наявних у вільному доступі в мережі Інтернет списків слів англійської мови, які містять інформацію про частоту використання цих слів в текстах, обрано найбільш повний та завантажено його до однієї з таблиць бази даних PostgreSQL для полегшення подальшої взаємодії [34].

Таким чином, при отриманні на вхід сервісу певного слова, відбувається:

1. Лематизація слова – приведення слова до його словникової форми, яка необхідна через те, що база даних також містить слова лише у лематизованій формі.
2. Витягування даних про слово з бази даних.
3. Розрахунок відхилення частоти використання цього слова із приблизною оцінкою рівня поточного словникового запасу користувача, визначеною для нього раніше з метою вирахування ймовірності знання цього слова користувачем.

### *4.1.3. Реалізація розширення веб-браузера для взаємодії із текстом веб-сторінки*

Розширення веб-браузера, яке виступає в ролі клієнта в рамках архітектури розроблюваної системи, можна умовно поділити на два модулі:

1. Модуль для підтримки взаємодії користувача із системою, що запускається на новій вкладці браузера.
2. Модуль для підтримки взаємодії розширення із веб-сайтом, який переглядає користувач.

Модуль для підтримки взаємодії користувача із системою являє собою повноцінний односторінковий додаток, реалізований мовою JavaScript із використанням бібліотеки React. Роблячи авторизовані HTTP-запити до API серверної частини, він отримує список збережених користувачем слів та дані про них, а також реалізує інтерфейс для взаємодії із цими словами за методом інтервальних повторень. Інформація про дії користувача у цьому інтерфейсі (оцінка знання значення слова, видалення слова тощо) відправляється на сервер за допомогою відповідних запитів до API, де записується до відповідних таблиць БД та використовується для подальшої користувацької взаємодії.

Модуль для підтримки взаємодії розширення із веб-сайтом після відкривання користувачем кожної нової веб-сторінки відправляє текстовий вміст її основної частини на сервер для подальшого аналізу та отримує від нього список слів, потенційно незнайомих користувачеві, із зазначенням відповідних ймовірностей. Крім того, модуль дозволяє користувачеві дізнатися значення та інші дані про слово, натиснувши два рази на бажане слово, що призводить до появи блоку з відповідною інформацією, отриманою з бази даних веб-серверу після виконання розширенням необхідного запиту до API.

### 4.2. Особливості тестування

Відповідно до вимог до розроблюваної системи, викладених у підрозділі 3.1, було сформовано набір тестових випадків (тест-кейсів), необхідних для проведення якісного димового (smoke) тестування.

Для тестування використовуємо ОС macOS версії 10.14.5 із запущеним браузером Google Chrome версії 74. Усі тести перевіряються у порядку їх опису нижче, тобто результати попереднього тест-кейсу можуть використовуватися у подальших тест-кейсах (див. табл. 4.3).

Таблиця 4.3

Тестові випадки (тест-кейси)

| № | Мета тест-кейсу | Опис дій тест-кейсу | Очікуваний результат |
|---|---|---|---|
| 1 | Перевірити можливість початкового запуску додатку. | Відкрити нову вкладку браузера. | Браузер відображає стартову сторінку додатку. Додаток відкривається на сторінці логіну. |

Продовження табл. 4.3

| № | Мета тест-кейсу | Опис дій тест-кейсу | Очікуваний результат |
|---|---|---|---|
| 2 | Перевірити можливість авторизації. | Ввести логін та пароль у форму входу.<br>Натиснути кнопку “Login”. | У полях “Login” та Password відображаються введені дані.<br>Показується інтерфейс користувача із наявними даними про користувача. |
| 3 | Перевірити можливість отримувати переклад слова на веб-сторінці. | Відкрити будь-яку сторонню сторінку на вибір користувача.<br>Натиснути двічі на будь-яке слово на сторінці. | Відкрита користувачем сторінка успішно завантажується.<br>Біля обраного слова з’являється блок з перекладом та іншими даними про це слово. |
| 4 | Перевірити можливість зберегти слово для подальшого вивчення. | У відкритому блоці із даними про слово натиснути кнопку “Save to learn”. | У відкритому блоці із даними про слово з’являється відповідна індикація про успішне збереження. |
| 5 | Перевірити наявність індикації потенційно незнайомих слів. | Натиснути на іконку розширення та увімкнути відповідний режим. | Слова на сторінці, що є потенційно незнайомими для поточного користувача, виділяються кольором. |
| 6 | Перевірити можливість перегляду списку збережених слів. | Відкрити нову вкладку.<br>Натиснути на елемент меню “Words”. | Відображається інтерфейс додатку.<br>Відображається список збережених раніше слів із даними про них. |

Продовження табл. 4.3

| № | Мета тест-кейсу | Опис дій тест-кейсу | Очікуваний результат |
|---|---|---|---|
| 7 | Перевірити можливість вивчення слів за методом інтервальних повторень | Відкрити нову вкладку.<br>Оцінити рівень знання показаного слова, натиснувши на відповідну кнопку.<br>Натиснути на кнопку “Next” для переходу до наступного слова. | Відображається запущений додаток із словом англійської мови та можливістю оцінити його знання.<br>Відображаються дані про слово.<br>Відображається наступне слово. |

# ВИСНОВКИ

Метою даного дипломного проекту є розроблення системи для підтримки процесу вивчення англійської мови, а саме складової, пов’язаної зі збільшенням словникового запасу.

Дослідження можливостей існуючих аналогів призвело до формулювання вимог до розроблюваного ПЗ, що поєднує у собі найкращі підходи у вирішенні проблеми, що розглядається у комбінації із новими та унікальними функціональними особливостями для взаємодії з незнайомими словами англійської мови.

Проведений аналіз доступних засобів, мов програмування та технологій виявив обґрунтовану доцільність використання мови програмування JavaScript та фреймворку Node.js для реалізації серверної частини та створення розширення для веб-браузера Google Chrome в якості клієнтської частини. В процесі проектування системи було розроблено та описано декілька важливих для роботи алгоритмів.

Розроблена система для підтримки процесу вивчення англійської мови:

- відображає переклад слова на веб-сторінці при натисканні на нього;
- дозволяє зберегти слово з веб-сторінки до списку для подальшого заучування;
- виділяє потенційно незнайомі користувачеві слова;
- дозволяє заучувати слова за методом інтервальних повторень.

Розроблення було виконано в повному обсязі згідно з положеннями Технічного завдання, проведено тестування у відповідності до затвердженої програми та методик тестування.

## СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ


1. 6 Essential Skills for Language Learning: [Електронний ресурс]. Режим доступу: http://powlyglot.com/6-essential-skills-for-language-learning/
2. 7 Tricks To Help Remember New Words Quickly & Easily: [Електронний ресурс]. Режим доступу: https://www.tefl.net/elt/articles/teacher-technique/7-tricks-to-help-remember-new-words/
3. Взять на заметку: как ассоциации помогают запоминать больше и лучше: [Електронний ресурс]. Режим доступу: https://theoryandpractice.ru/posts/14485-vzyat-na-zametku-kak-assotsiatsii-pomogayut-zapominat-bolshe-i-luchshe
4. The right time to learn: mechanisms and optimization of spaced learning: [Електронний ресурс]. Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5126970/
5. What is Anki and Why does it Work: [Електронний ресурс]. Режим доступу: https://japaneselevelup.com/what-is-anki-and-why-does-it-work/
6. Memrise Review: [Електронний ресурс]. Режим доступу: https://ltl-school.com/memrise-review/
7. Google Dictionary (by Google): [Електронний ресурс]. Режим доступу: https://chrome.google.com/webstore/detail/google-dictionary-by-goog/mgijmajocgfcbeboacabfgobmjgjcoja?hl=ru
8. 30 Seconds of Knowledge: [Електронний ресурс]. Режим доступу: https://30secondsofknowledge.com/
9. What is Python: [Електронний ресурс]. Режим доступу: https://www.pythonforbeginners.com/learn-python/what-is-python/
10. What is PHP: [Електронний ресурс]. Режим доступу: https://www.php.net/manual/en/intro-whatis.php

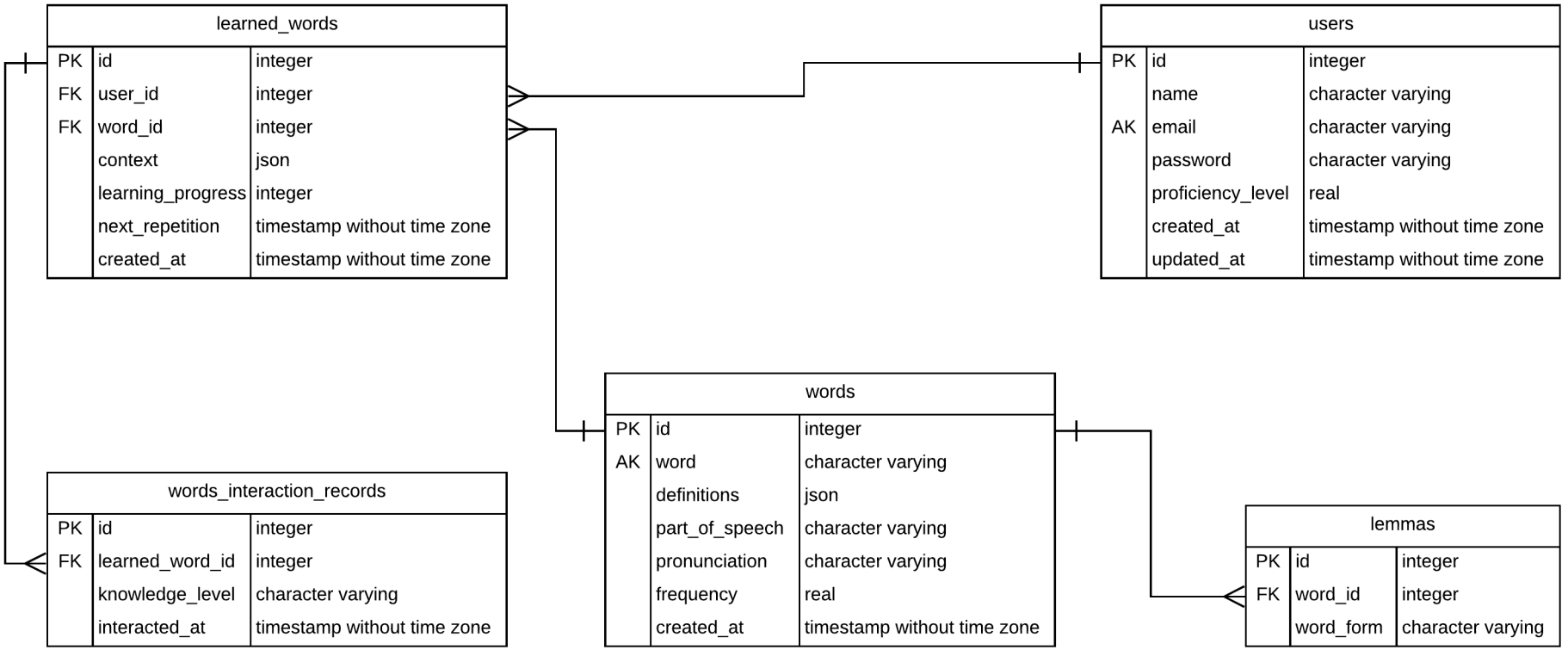
11. Usage of server-side programming languages for websites: [Електронний ресурс]. Режим доступу: https://w3techs.com/technologies/overview/programming_language/all
12. Introduction to the C# Language and the .NET Framework: [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework
13. What is JavaScript: [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
14. Javascript Engine & Performance Comparison (V8, Chakra, Chakra Core): [Електронний ресурс]. Режим доступу: https://developers.redhat.com/blog/2016/05/31/javascript-engine-performance-comparison-v8-charkra-chakra-core-2/
15. Node.js: [Електронний ресурс]. Режим доступу: https://nodejs.org/uk/
16. Angular: [Електронний ресурс]. Режим доступу: https://angular.io/
17. The Good and the Bad of Angular Development: [Електронний ресурс]. Режим доступу: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/
18. Long Term Support for Angular Announced at ng-conf 2017: [Електронний ресурс]. Режим доступу: https://www.infoq.com/news/2017/04/ng-conf-2017-keynote/
19. React: [Електронний ресурс]. Режим доступу: https://en.reactjs.org/
20. React, Angular – Аналіз – Google Trends: [Електронний ресурс]. Режим доступу: https://trends.google.com/trends/explore?cat=31&q=React,Angular
21. MySQL: [Електронний ресурс]. Режим доступу: https://www.mysql.com/
22. MySQL Pros and Cons | TrustRadius: [Електронний ресурс]. Режим доступу: https://www.trustradius.com/products/mysql/reviews/pros-and-cons?f=25
23. PostgreSQL: [Електронний ресурс]. Режим доступу: https://www.postgresql.org/

24. What Is the Cost of a Requirement Error: [Електронний ресурс]. Режим доступу: https://www.stickyminds.com/article/what-cost-requirement-error
25. Web Application Architecture: [Електронний ресурс]. Режим доступу: https://svitla.com/blog/web-application-architecture
26. Comparing AWS vs Azure vs Google Cloud Platforms For Enterprise App Development: [Електронний ресурс]. Режим доступу: https://medium.com/@distillerytech/comparing-aws-vs-azure-vs-google-cloud-platforms-for-enterprise-app-development-28ccf827381e
27. Load Balancers: [Електронний ресурс]. Режим доступу: https://www.digitalocean.com/docs/networking/load-balancers/
28. Ebbinghaus Forgetting Curve: [Електронний ресурс]. Режим доступу: https://www.psychestudy.com/cognitive/memory/ebbinghaus-forgetting-curve
29. Spaced Repetition: Learn Once, Remember Forever: [Електронний ресурс]. Режим доступу: https://www.themetalearners.com/spaced-repetition-learn-once-remember-forever/
30. A Simple but Effective Learning Algorithm: MS Syntax: [Електронний ресурс]. Режим доступу: http://blog.lotp.xyz/2018/08/12/A-Simple-But-Effective-Spaced-Repitition-Algorithm-MS/
31. Chen X. Characterizing Text Difficulty with Word Frequencies [Електронний ресурс] / X. Chen, D. Meurers. – 2016. – Режим доступу: https://aclweb.org/anthology/W16-0509.pdf.
32. A new and improved word frequency database for British English [Електронний ресурс] / W.van Heuven, P. Mandera, E. Keuleers, M. Brysbaert. – 2014. – Режим доступу: https://www.tandfonline.com/doi/full/10.1080/17470218.2013.850521.
33. Introduction to JSON Web Tokens: [Електронний ресурс]. Режим доступу: https://jwt.io/introduction/
34. Frequency Lists: [Електронний ресурс]. Режим доступу: http://ucrel.lancs.ac.uk/bncfreq/flists.html

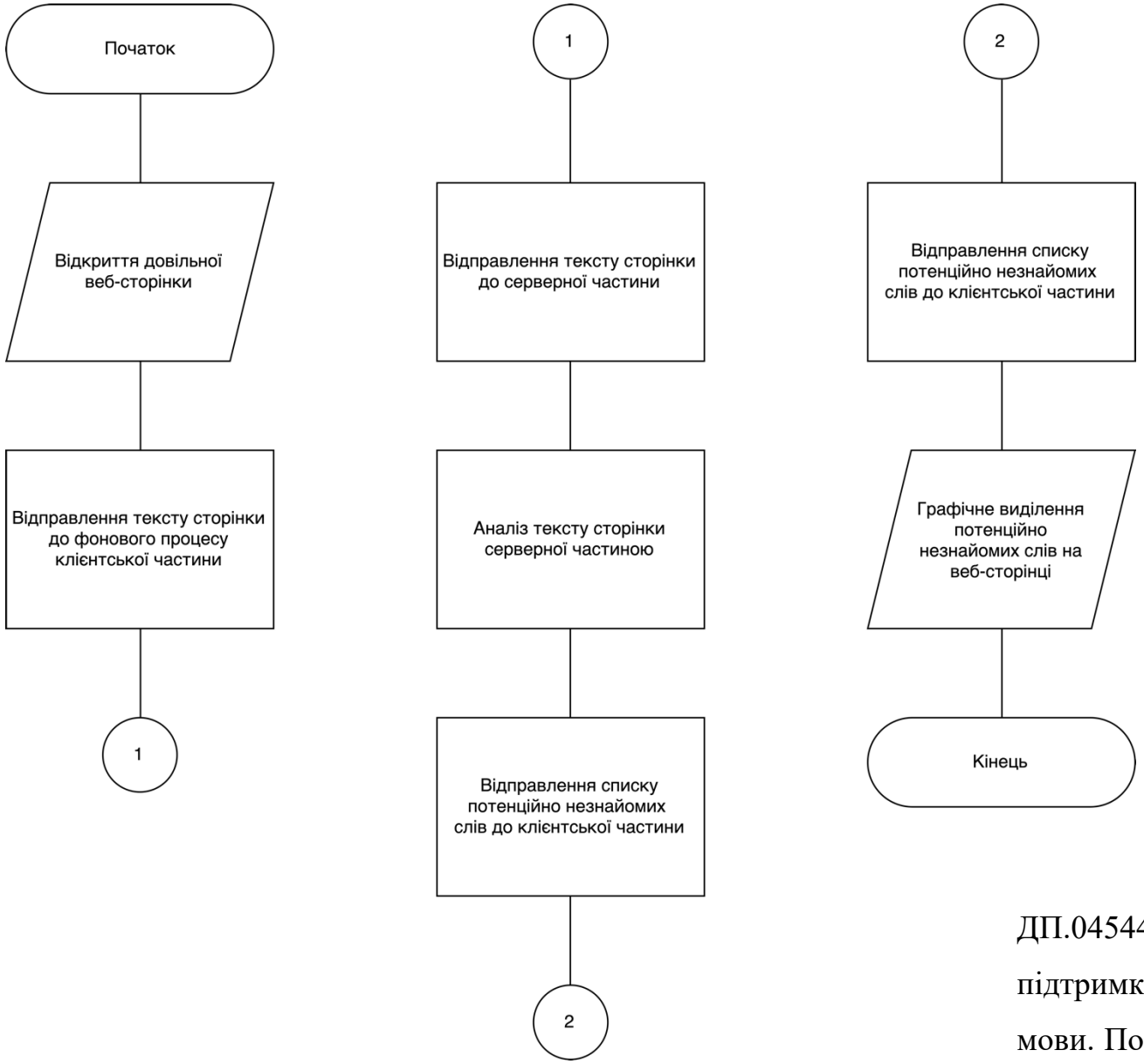
# ДОДАТКИ

**Додаток 1**

**Копії графічних матеріалів**

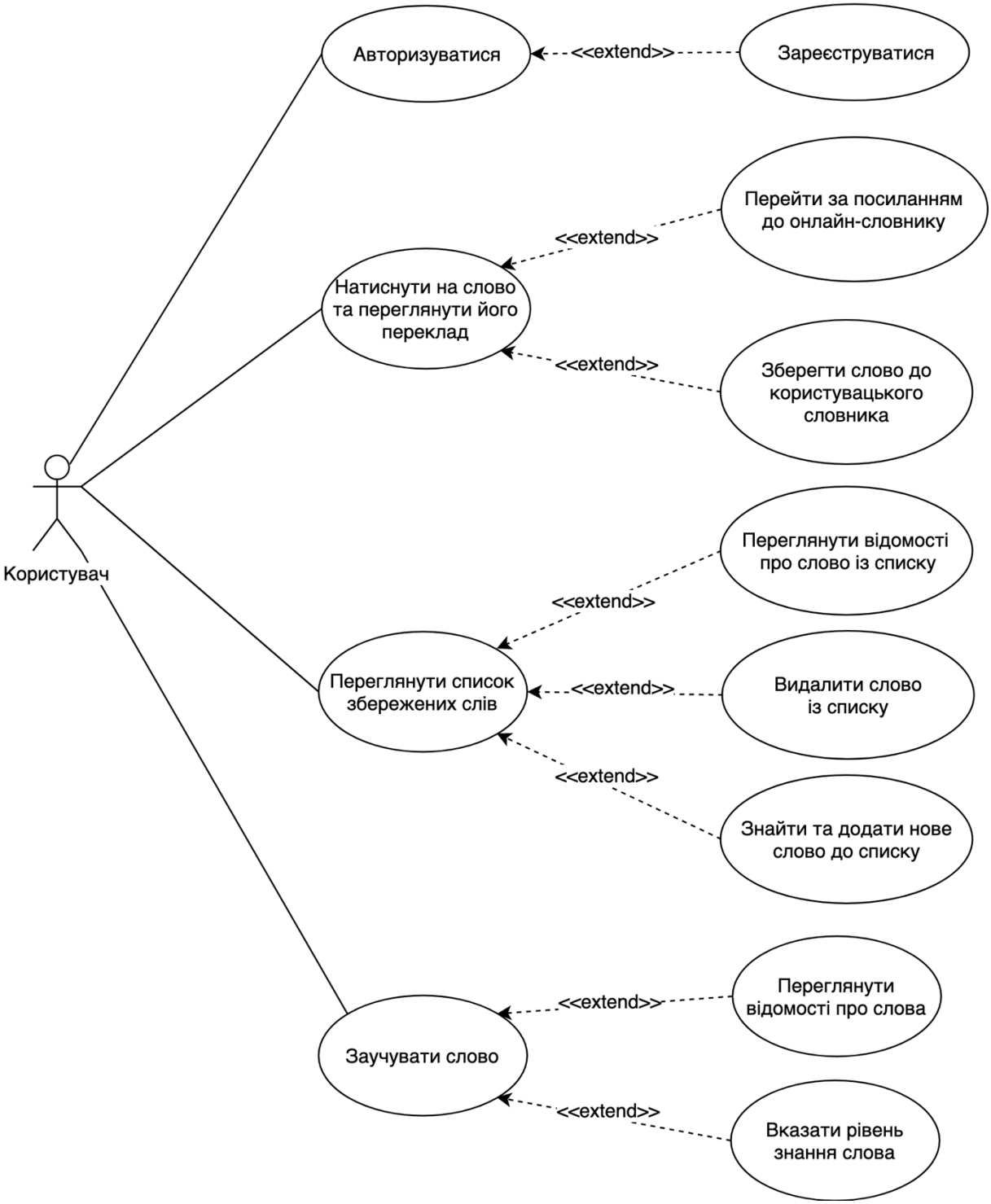



ДП.045440-06-99. Програмні засоби для підтримки процесу вивчення англійської мови. Структура бази даних. ERD-діаграма



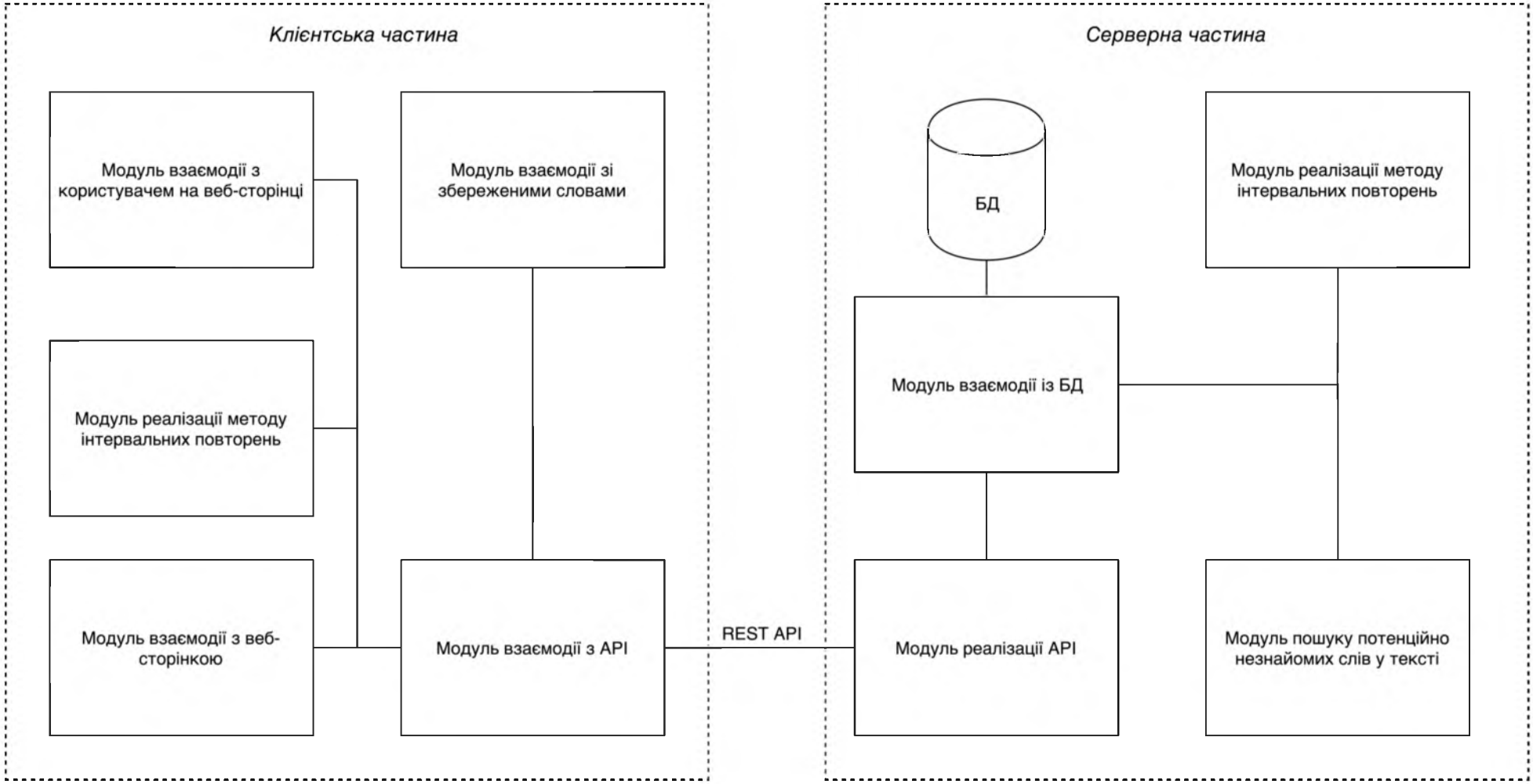

ДП.045440-07-99. Програмні засоби для підтримки процесу вивчення англійської мови. Потік даних сервісу аналізу потенційно незнайомих слів. Схема потоку

ДІАГРАМА ПРЕЦЕДЕНТІВ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ




Уруков Д.О., група КП-52

СТРУКТУРНА СХЕМА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ




Уруков Д.О., група КП-52

**Додаток 2**

**Копія презентації**

# Програмні засоби для підтримки процесу вивчення англійської мови

Уруков Д.О.
КП-52

Науковий керівник:
доцент кафедри ПЗКС, к.т.н., доцент Заболотня Т.М.

# Актуальність

1. Англійська мова – мова міжнародного спілкування.
2. Вивчення англійської мови - надзвичайно складна задача.
3. Вивчення англійської включає в себе декілька незалежних навичок, які опановуються окремо.
4. Ринок ПЗ для тренування навичок англійської постійно розвивається.

# Існуючі аналоги

Anki
(картки
для
запам’ятовування)

Memrise
(вивчення мов)

Google Dictionary
(браузерний
словник)


30 Seconds of Knowledge
(фрагменти коду на
новій вкладці)

# Мета розроблення

- Надання програмних засобів для формування пасивного словнику англійської мови.
- Покращення та часткова автоматизація процесу заучування нових слів англійської мови.

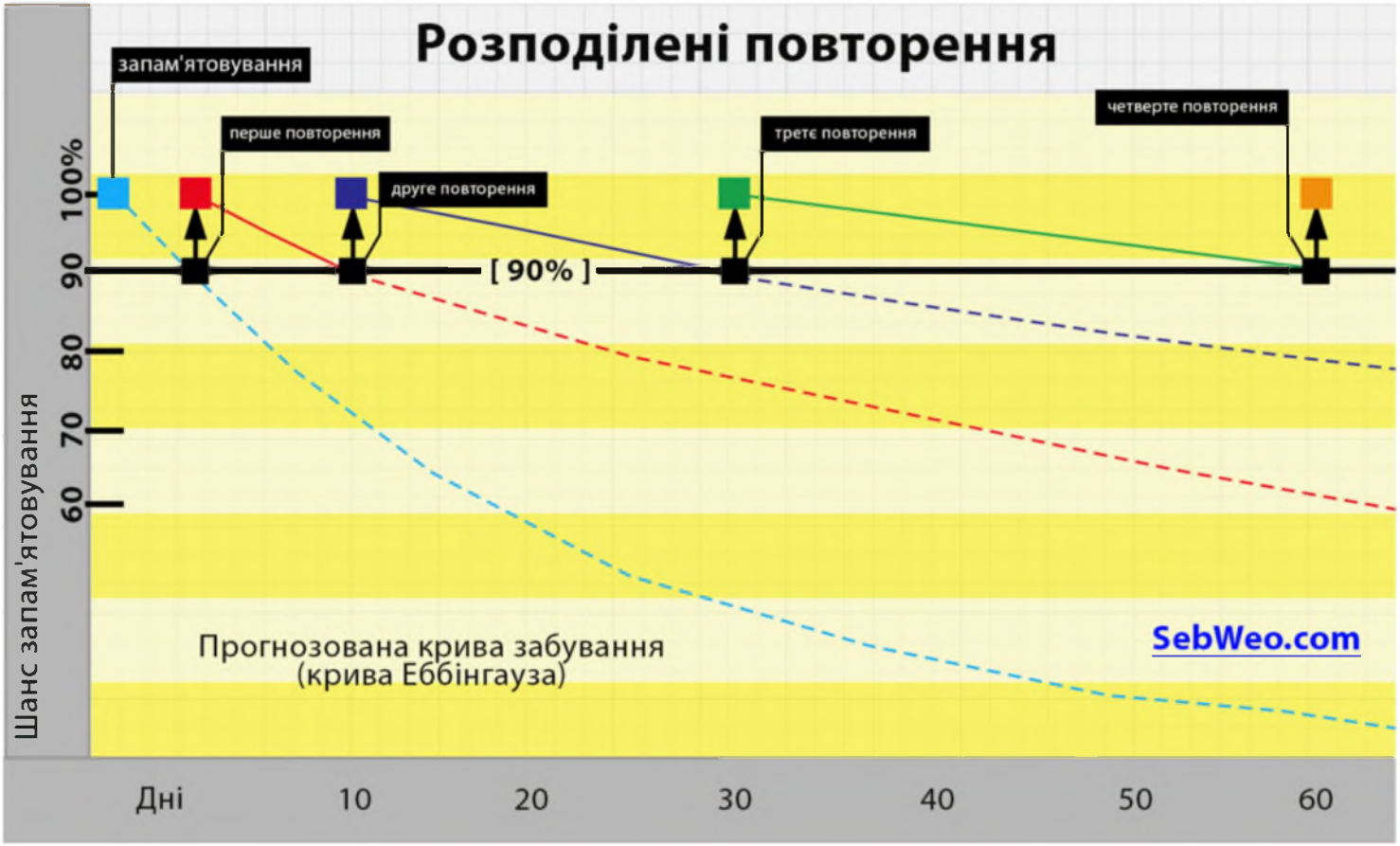
# Постановка завдання

1. Вибір та розроблення алгоритму для запам’ятовування нових слів англійської мови.
2. Створення модуля відображення значення слів на веб-сторінці, на якій знаходиться користувач.
3. Створення модуля для визначення потенційно незнайомих для користувача слів на веб-сторінці.
4. Створення користувацького модуля із збереження та вивчення нових слів.
5. Створення програмного застосунку, що може бути інтегрований до веб-браузера.
6. Тестування розробленого програмного застосунку

# Вимоги

1. Можливість використання у веб-браузері.
2. Можливість отримання значення довільного слова на веб-сторінці.
3. Можливість зберегти довільне слово з веб-сторінки для подальшого заучування засобами продукту.
4. Можливість переглядати та редагувати список збережених слів.
5. Можливість повторювати та заучувати нові слова за методом інтервальних повторень.

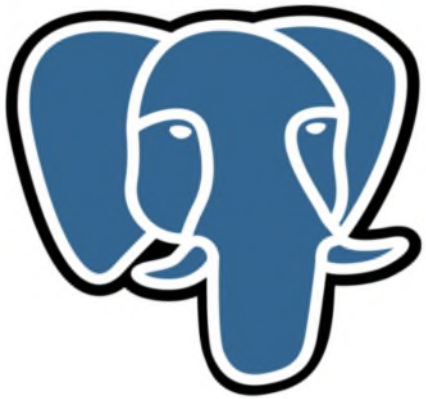
# Метод інтервальних повторень



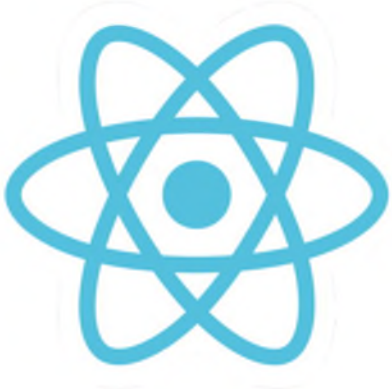
# Обрані засоби реалізації


Мова програмування JavaScript


Серверний фреймворк Node.js



СКБД PostgreSQL



Клієнтська бібліотека React

# Особливість клієнтської частини

Розширення браузера Google Chrome

# Структура БД

# Структура програми

# Діаграма прецедентів

# Потік даних

# Тестування

- Тестування виконується відповідно до техніки Gray Box Testing
- Використовуються наступні методи:
  1. Функціональне тестування, зокрема на рівні Integration testing (тестування взаємодії різних модулів програми) та Interface testing (тестування користувацької взаємодії).
  2. Нефункціональне тестування, зокрема на рівні Reliability testing (тестування надійності), Usability testing (тестування зручності використання) та Efficiency testing (тестування ефективності).

# Висновки

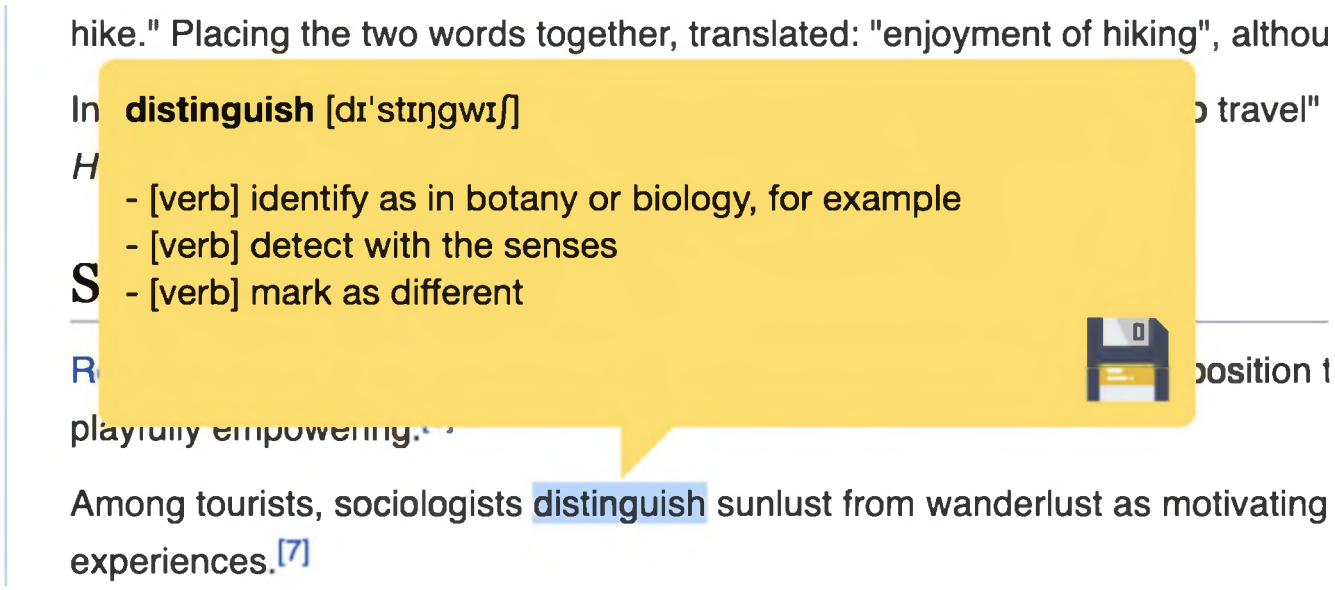
Проведений аналіз наявних схожих рішень виявив доцільність створення даного програмного продукту.

Розроблений програмний застосунок:

1. Працює як розширення для браузеру Google Chrome.
2. Надає можливість отримати значення слова на веб-сторінці, виділяє слова, що є потенційно незнайомими для користувача.
3. Надає можливість зберегти слово з веб-сторінки для його подальшого вивчення за допомогою реалізованого алгоритму за методом інтервальних повторень.

Розробку виконано в повному обсязі у відповідності до сформованих вимог, тестування проведено за затвердженою програмою та методикою тестування.

# Інтерфейс


hike." Placing the two words together, translated: "enjoyment of hiking", althou
In
H
S
R
playfully empowering.
Among tourists, sociologists distinguish sunlust from wanderlust as motivating experiences.[7]
distinguish [dɪ'stɪŋgwɪʃ]
- [verb] identify as in botany or biology, for example
- [verb] detect with the senses
- [verb] mark as different

# Інтерфейс

But dynamic resource allocation problems are not just concerned with giving humans what they want, when they want it. They will also be essential for tackling some of the world's most fundamental and complex issues, including climate change, as they help us allocate our planet's often scarce and depleted resources in the most efficient ways possible.

# Дякую за увагу!

## Додаток 3

## Лістинг програми

```
import Base from '../Base';
import { runService } from '../../tools';

import redis from '../../redis';
import { AsyncConcurrentMap } from '../../util';
import WordInfo from './Info';

export default class WordComplexityInfo extends Base {
  static get validationRules() {
    return {
      words: [
        'required',
        { list_of: ['required', 'string', 'trim', 'to_lc'] },
        'to_uniq_array',
      ],
    };
  }

  async isPresentInAlphaList(word) {
    return redis.sismember('english-words', word);
  }

  async execute({ words }) {
    const realWords = await AsyncConcurrentMap(
      words,
      async word => {
        const exists = await this.isPresentInAlphaList(word);
        return exists ? word : null;
      },
      {
        concurrency: 100,
      },
    );

    const wordsWithInfo = await AsyncConcurrentMap(
      realWords,
      async word => {
        try {
          const { word: wordInfo } = await runService(WordInfo, {
            params: { word },
          });
          return wordInfo;
        } catch (e) {
          return null;
        }
      },
      {
        concurrency: 50,
      },
    );
    const result = wordsWithInfo
      .filter(w => w.frequency)
      .map(w => ({
        word: w.word,
        easiness: w.frequency,
      }));
    return {
      words: result,
    };
```

```
  }
  static get paramsSecret() {
    return [];
  }
  static get resultSecret() {
    return [];
  }
}
```

---

```
import moment from 'moment';

import Base from '../Base';
import { runService } from '../../tools';

import WordInfo from '../words/Info';
import { LearnedWord } from '../../models';

export default class LearnedWordCreate extends Base {
  static get validationRules() {
    return {
      word: ['required', 'string', 'trim'],
    };
  }

  async execute({ word }) {
    const { word: existingWord } = await runService(WordInfo, {
      params: { word },
    });
    const existingLearnedWord = await LearnedWord.findOneWhere({
      word_id: existingWord.id,
    });
    if (existingLearnedWord) {
      return {
        learnedWord: existingLearnedWord.json(),
      };
    }
    const learnedWord = new LearnedWord({
      word_id: existingWord.id,
      learning_progress: 0,
      next_repetition_date: moment()
        .add(1, 'hours')
        .toDate(),
    });
    await learnedWord.save();
    return {
      learnedWord: learnedWord.json(),
    };
  }
  static get paramsSecret() {
    return [];
  }
  static get resultSecret() {
    return [];
  }
}


import lemmatize from 'wink-lemmatizer';
import _ from 'lodash';
```

```
import Base from '../Base';
import X from '../Exception';
import { runService } from '../../tools';

import { Word, Lemma } from '../../models';
import GetWordsAPIWord from '../words-api/Get';
import SaveWordsAPIWord from '../words-api/Save';

import redis from '../../redis';

export default class WordInfo extends Base {
  static get validationRules() {
    return {
      word: ['required', 'string', 'trim'],
    };
  }

  async getWordLemmatizedForm(word) {
    const existingLemmaForm = await Lemma.findOneWhere({ word });

    if (existingLemmaForm) {
      return existingLemmaForm.lemma;
    }

    const options = [
      word,
      lemmatize.adjective(word),
      lemmatize.verb(word),
      lemmatize.noun(word),
    ].filter(_.identity);

    return _.uniq(options).filter(form => form !== word)[0] || null;
  }

  async execute({ word }) {
    const existingWord = await Word.findOneWhere({
      word: word,
    });

    if (existingWord) {
      return {
        word: existingWord.json(),
      };
    }

    const lemmatizedWord = await this.getWordLemmatizedForm(word);
    if (lemmatizedWord) {
      const existingLemmatizedWord = await Word.findOneWhere({
        word: lemmatizedWord,
      });
      if (existingLemmatizedWord) {
        return {
          word: existingLemmatizedWord.json(),
        };
      }
    }

    let { data } = await runService(GetWordsAPIWord, {
      params: { word },
    });
```

```
    if (!data || !data.results) {
      if (!lemmatizedWord) {
        await Promise.all([redis.sadd('unavailable-words', word)]);
        throw new X({
          code: 'NOT_FOUND',
          fields: {
            word: 'NOT_FOUND',
          },
          statusCode: 404,
        });
      }

      ({ data } = await runService(GetWordsAPIWord, {
        params: { word: lemmatizedWord },
      }));

      if (!data || !data.results) {
        await Promise.all([
          redis.sadd('unavailable-words', word),
          redis.sadd('unavailable-words', lemmatizedWord),
        ]);
        throw new X({
          code: 'NOT_FOUND',
          fields: {
            word: 'NOT_FOUND',
          },
          statusCode: 404,
        });
      }

      const lemma = new Lemma({
        word,
        lemma: lemmatizedWord,
      });

      await lemma.save();
    }

    const { word: createdWord } = await runService(SaveWordsAPIWord, {
      params: { data },
    });

    return {
      word: createdWord,
    };
  }

  static get paramsSecret() {
    return [];
  }

  static get resultSecret() {
    return [];
  }
}

-----

import qrate from 'qrate';
import _ from 'lodash';
```

```
export async function AsyncConcurrentMap(
  array,
  func,
  { concurrency = 1, filter = _.identity } = {},
) {
  return new Promise((resolve, reject) => {
    const results = [];
    const queue = qrate(async ({ element, index }) => {
      try {
        results[index] = await func(element, index);
      } catch (e) {
        reject(e);
      }
    }, concurrency);
    for (const [index, element] of array.entries()) {
      queue.push({ element, index });
    }
    queue.drain = () => {
      if (filter) {
        return resolve(results.filter(filter));
      } else {
        return resolve(results);
      }
    };
  });
}

----

import FS from 'fs';
import path from 'path';
import _ from 'lodash';

import redis from '../redis';
import { AsyncConcurrentMap } from '../util';

const fs = FS.promises;

export const WORDS_WIKI_DICTIONARY_KEY = 'words-wiki';
export const WORDS_LANCS_DICTIONARY_KEY = 'words-lancs';

export async function initWikiWordsByComplexity() {
  const list = String(
    await fs.readFile(path.resolve('./src/dictionary/assets/wiki-
100k.txt')),
  );
  const words = list
    .split('\n')
    .map(w => w.trim())
    .filter(w => w && !w.startsWith('#'));

  await redis.del(WORDS_WIKI_DICTIONARY_KEY);

  await AsyncConcurrentMap(
    words,
    async (word, index) => {
      await redis.zadd(WORDS_WIKI_DICTIONARY_KEY, index, word);
    },
    {
      concurrency: 100,
    },
```

```
  );
}

export async function initLancsWordsByComplexity() {
  const list = String(
    await
fs.readFile(path.resolve('./src/dictionary/assets/1_2_all_freq.txt')),
  );
  const words = list
    .split('\n')
    .slice(1)
    .map(w => w.trim().split('\t'))
    .map(([word, , frequency]) => ({
      word,
      frequency: Number(frequency),
    }))
    .filter(({ frequency }) => !_.isNaN(frequency));

  const wordsInCorpusCount = _.sumBy(words, 'frequency');

  const wordsWithZipf = words.map(({ word, frequency }) => ({
    word,
    frequency,
    zipf: Math.log10(frequency / (wordsInCorpusCount / 1e6)),
  }));

  await redis.del(WORDS_LANCS_DICTIONARY_KEY);

  await AsyncConcurrentMap(
    wordsWithZipf,
    async ({ word, zipf }) => {
      await redis.zadd(WORDS_LANCS_DICTIONARY_KEY, zipf, word);
    },
    {
      concurrency: 100,
    },
  );
}


----

($x = 0), ($y = 0);

var CURRENT_LEMMA = "";

// Add bubble to the top of the page.
$box = $("<div>", { class: "box_overlay" });
$content = $("<div>", { class: "meaning_content" });
$moreButton = $("<div>", { class: "more_button" });
$moreLink = $("<img>", {
  class: "save_icon",
  src:

"https://static.wixstatic.com/media/53c332_b0dbc76b0fcb4771b214422261246cd9
~mv2.png",
  alt: "save"
});
$savedCheck = $("<img>", {
  class: "check_icon",
  src:
```

```
"https://static.wixstatic.com/media/53c332_a0b0f0ab70eb4d93bb05c4fbab65f84b
~mv2.png",
  alt: "ok"
});
$moreButton.append($savedCheck);
$moreButton.append($moreLink);
$moreLink.click(onSaveWordButtonClick);
$box.append($content);
$box.append($moreButton);
$savedCheck.hide();

$highlightButton = $("<img>", {
  class: "highlight_icon",
  src:

"https://static.wixstatic.com/media/53c332_ef05ed15960d40af9648dcb3717b6b12
~mv2.png",
  alt: "highlight"
});
$highlightButton.click(onHighlightButtonClicked);

$("body").append($box);
$("body").append($highlightButton);

$("body").dblclick(function(e) {
  var selectedText = window
    .getSelection()
    .toString()
    .trim();
  if (selectedText.length > 0) {
    selectedText = pluralize(selectedText, 1);
    $x = $(window).scrollLeft() + e.clientX;
    $y = $(window).scrollTop() + e.clientY;
    chrome.runtime.sendMessage({ message: selectedText });
  }
});

// Close the bubble when we click on the screen.
$("body :not(.box_overlay, .box_overlay *)").on("click", function() {
  changeContent("", "");
  $box.removeClass("visible");
});

chrome.runtime.onMessage.addListener(function(request, sender) {
  if (!sender.tab) {
    if (request.active) {
      changeContent({
        word: request.word,
        meaning: request.meaning,
        lemma: request.lemma
      });
    }
  }
});

function debounce(f, ms) {
  let timer = null;
  return function(...args) {
    const onComplete = () => {
      f.apply(this, args);
```

```
      timer = null;
    };
    if (timer) {
      clearTimeout(timer);
    }
    timer = setTimeout(onComplete, ms);
  };
}

$box.bind(
  "DOMSubtreeModified",
  debounce(function() {
    $box.addClass("visible");
    $x -= $(this).width() / 2;
    $y -= $(this).height() + 50;
    if ($x < 0) $x = 0;
    if ($y < 0) $y += $(this).height() + 60;
    $(this).offset({ top: $y > 0 ? $y : 0, left: $x });
  }, 100)
);

function changeContent({ word, meaning, lemma }) {
  if (!meaning || meaning.length === 0) {
    $savedCheck.hide();
    $moreLink.hide();
    $moreButton.hide();
  } else {
    if (meaning.length > 500) {
      meaning = meaning.substring(0, 500).concat("...");
    }
    $savedCheck.hide();
    $moreLink.show();
    $moreButton.show();
  }
  CURRENT_LEMMA = lemma;
  $content.html(meaning);
}

function onSaveWordButtonClick() {
  console.log(CURRENT_LEMMA);
  $.ajax({
    url: "http://localhost:9000/api/v1/learned-words",
    type: "POST",
    data: JSON.stringify({
      word: CURRENT_LEMMA
    }),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    async: true,
    success: onWordSaved
  });
}

function onWordSaved(msg) {
  $moreLink.hide();
  $savedCheck.show();
}

function onHighlightButtonClicked() {
  var words = [];
  var walkDOM = function(node, func) {
```

```
    func(node);
    node = node.firstChild;
    while (node) {
      walkDOM(node, func);
      node = node.nextSibling;
    }
  };
  walkDOM(document.body, function(node) {
    if (node.nodeName === "#text") {
      var text = node.textContent;
      text = text.replace(/[^A-Za-z]/g, " ");
      text = text.split(" ");
      if (text.length) {
        for (var i = 0, length = text.length; i < length; i += 1) {
          var matched = false,
            word = text[i];
          for (
            var j = 0, numberOfWords = words.length;
            j < numberOfWords;
            j += 1
          ) {
            if (words[j][0] === word) {
              matched = true;
              words[j][1] += 1;
            }
          }
          if (!matched) {
            words.push([word, 1]);
          }
        }
      }
    }
  });

  const wordsList = words.map(([w]) => w).filter(w => w && w.length > 2);
  $.ajax({
    url: "http://localhost:9000/api/v1/words/complexity",
    type: "POST",
    data: JSON.stringify({
      words: wordsList
    }),
    contentType: "application/json; charset=utf-8",
    dataType: "json",
    async: true,
    success: msg => onWordsEasinessReceived(msg.words)
  });
}

-----

var result = null;
var active = true;

chrome.browserAction.onClicked.addListener(function(tab) {
  if (active) {
    chrome.browserAction.setTitle({ title: "Click to Enable" });
    chrome.browserAction.setIcon({ path: "images/icon48_disabled.png" });
  } else {
    chrome.browserAction.setTitle({ title: "Click to Disable" });
    chrome.browserAction.setIcon({ path: "images/icon48.png" });
  }
  active = !active;
```

```
});

chrome.extension.onMessage.addListener(async function(request, sender) {
  if (active) {
    try {
      const word = await getWordData(request.message);
      console.log(word);
      const { word: lemma, definitions, pronunciation } = word;
      const html = `
      <div class="word__word" ><b>${lemma}</b> ${
        pronunciation ? `[${pronunciation}]` : ""
      }</div>
      <br>
      <ol>
        ${definitions
          .slice(0, 3)
          .map(
            ({ definition, partOfSpeech }) =>
              `<li class="word__definition">- ${
                partOfSpeech ? `[${partOfSpeech}]` : ""
              } ${definition}</li>`
          )
          .join("\n")}
      </ol>
    `;
      sendResult({
        meaning: html,
        lemma,
        active: active
      });
    } catch (e) {
      sendResult({
        active: active
      });
    }
  } else {
    result = {};
    result["active"] = active;
    sendResult(result);
  }
});

async function getWordData(word) {
  console.log("word", word);
  const data = await (await fetch(
    `http://localhost:9000/api/v1/words/${word}`
  )).json();
  return data.word;
}

function sendResult(result) {
  chrome.tabs.query({ active: true, currentWindow: true }, function(tabs) {
    chrome.tabs.sendMessage(tabs[0].id, result);
  });
}
```

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018 р.

# ПРОГРАМНІ ЗАСОБИ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ

**Програма та методика тестування**

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Д.О. Уруков

2018

**ЗМІСТ**

## 1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмні засоби для підтримки процесу вивчення англійської мови, яке представлено у вигляді розширення для веб-браузеру Google Chrome.

## 2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1. Можливість коректного встановлення браузерного розширення.
2. Коректність та стабільність користувацької взаємодії з розширенням.
3. Відповідність інтерфейсу нефункціональним вимогам (час відгуку, зрозумілість, зручність використання тощо).
4. Забезпечення необхідного рівня безпеки даних.

## 3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується відповідно до техніки Gray Box Testing, в ході якої на відповідність висунутим вимогам перевіряється як зовнішня поведінка системи, так і її внутрішня реалізація.

Використовуються наступні методи:

1. Функціональне тестування, зокрема на рівні Integration testing (тестування взаємодії різних модулів програми) та Interface testing (тестування користувацької взаємодії).
2. Нефункціональне тестування, зокрема на рівні Reliability testing (тестування надійності), Usability testing (тестування зручності використання) та Efficiency testing (тестування ефективності).

# 4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність програмного продукту перевіряється шляхом:

1. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які доступні для використання у рамках даного програмного проекту.
2. Динамічного ручного тестування на відповідність функціональним вимогам.
3. Статичного тестування коду.
4. Тестування програмного застосунку у веб-браузері Google Chrome.
5. Тестування стабільності роботі при різних зовнішніх умовах.
6. Тестування зручності використання.
7. Тестування інтерфейсу.

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп'ютерних систем**

"ЗАТВЕРДЖЕНО"

Науковий керівник кафедри

_____ І.А. Дичка

"___" _____ 2019 р.

# ПРОГРАМНІ ЗАСОБИ ДЛЯ ПІДТРИМКИ ПРОЦЕСУ ВИВЧЕННЯ АНГЛІЙСЬКОЇ МОВИ

**Керівництво користувача**

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проекту:

_____ Т.М. Заболотня

Нормоконтроль:

_____ М.В. Онай

Виконавець:

_____ Д.О. Уруков

2019

**ЗМІСТ**

## 1. Опис структури програмного застосунку

Програмні засоби для підтримки процесу вивчення англійської мови являють собою розширення для веб-браузера Google Chrome, яке встановлюється користувачем через відповідне меню «Розширення» браузера. Після установки значок розширення з’являється у верхній навігаційній панелі браузера серед інших іконок браузерних розширень.

Активоване користувацьке розширення аналізує текстовий вміст поточної відкритої веб-сторінки, не змінюючи його. Крім того, розширення підміняє інтерфейс нової вкладки своїм інтерфейсом, у якому реалізовані можливості для взаємодії із клієнтською частиною системи.

## 2. Опис процедури налаштування системи для роботи

Після встановлення розширення воно є повністю готовим до роботи, але варто зазначити, що для коректної роботи розширення необхідно вручну перезавантажити вкладки, відкриті до моменту встановлення розширення.

Розширення може бути деактивовано або активовано шляхом натискання на відповідну іконку розширення у верхній навігаційній панелі браузера (див. рис. 2.1).

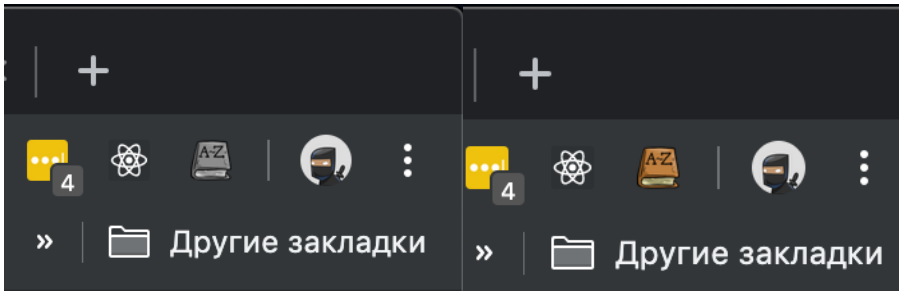



Рис. 2.1. Ілюстрація процесу деактивації та активації розширення

## 3. Користування програмними засобами

Розширення самостійно інтегрується до веб-сторінок, що переглядає користувач, тому для початку його використання після відкриття нової веб-сторінки не треба виконувати ніяких додаткових дій.

Натиснувши два рази на довільне слово на веб-сторінці, користувач може дізнатися значення обраного слова, що відображається у спеціальному блоці, який з’являється над словом (див. рис. 3.1).

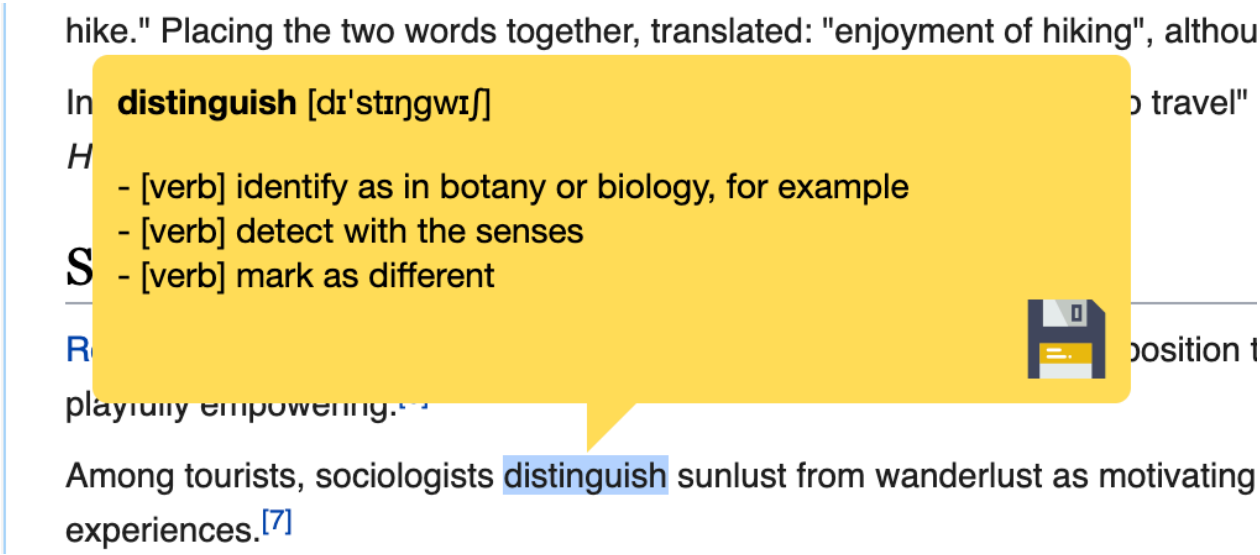



Рис. 3.1. Ілюстрація відображення значення довільного слова

Окрім власне перегляду значення користувач має можливість зберегти слово для подальшого вивчення, натиснувши на відповідну кнопку у правому нижньому куті блоку. Після цього слово з’являється в списку для заучування в інтерфейсі інтервальних повторень, що автоматично відкривається при відкритті нової вкладки браузера.

Користувач має змогу ввімкнути індикацію слів, що є потенційно незнайомими для нього, натиснувши на відповідну кнопку в правому нижньому кутку веб-сторінки. За деякий час усі такі слова будуть підкреслені на веб-сторінці (див. рис. 3.2).

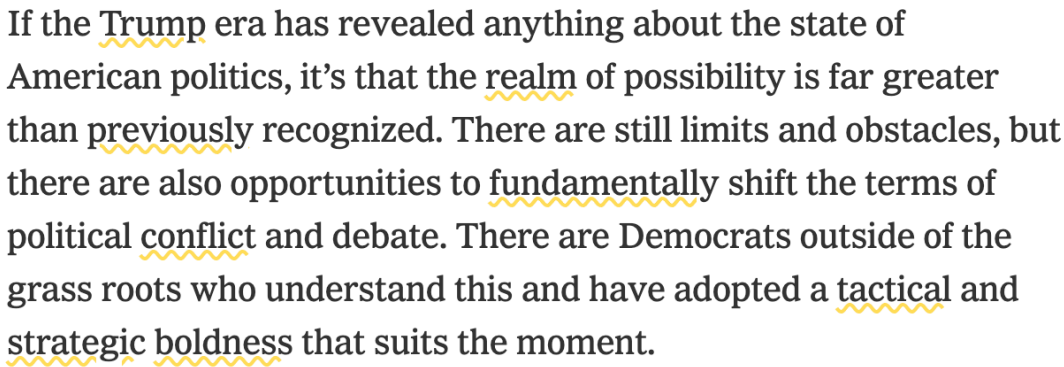



Рис. 3.2. Зображення потенційно незнайомих для користувача слів, підкреслених розширенням

Користувач має змогу за вищеописаним алгоритмом переглянути значення підкреслених слів, а також за бажанням зберегти їх для подальшого заучування.