Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБДОДАТКУ «КЛАВІАТУРНИЙ ТРЕНАЖЕР» З ВИКОРИСТАННЯМ REACT

DEVELOPMENT OF THE "KEYBOARD TRAINER" WEB APPLICATION USING REACT

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗс-21
Умша Костянтин Сергійович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Умші Костянтину Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебдодатку "Клавіатурний тренажер" з використанням React»*
Керівник роботи: *доцент Ящук А.А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *React, JavaScript, HTML5, CSS3, Node.js, Visual Studio Code, Netlify, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз аналогів, формування вимог, побудова UML-діаграм, вибір технологій розробки, розробка інтерфейсу, реалізація функціоналу, тестування, створення документації, SEO-оптимізація, забезпечення безпеки*
5. Перелік графічного матеріалу: *18 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Яцук А. А.*		
Специфікація вимог до розробленої системи	*Яцук А. А.*		
Розробка об'єкта проектування	*Яцук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	9,78%		
Академічна доброчесність	*Яцук А. А.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Вик.
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Вик.

Здобувач вищої освіти _____ Костянтин УМША

Керівник кваліфікаційної роботи _____ Андрій ЯЩУК

АНОТАЦІЯ

Умша К. С. Розробка вебдодатку «Клавіатурний тренажер» з використанням React. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз існуючих клавіатурних тренажерів та сформульовано завдання на кваліфікаційну роботу. У другому розділі описано архітектуру, структуру даних і взаємодію компонентів та обґрунтовано вибір технологій розробки. У третьому розділі описано практичну реалізацію вебдодатку, результати тестування, інструкцію з розгортання, SEO-оптимізацію та заходи з безпеки. У висновках узагальнено результати роботи та підтверджено досягнення поставленої мети.

Ключові слова: клавіатурний тренажер, вебдодаток, React, сліпий метод друку, SPA, JSON, LocalStorage.

ABSTRACT

Umsha K. Development of the "Keyboard Trainer" Web Application Using React. Manuscript. Qualification work of the bachelor's program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references and appendices.

The first chapter analyzes existing keyboard trainers and formulates the objectives of the qualification work. The second chapter describes the system architecture, data structure, and component interaction, and substantiates the choice of development technologies. The third chapter presents the practical implementation of the web application, testing results, deployment instructions, SEO optimization, and security measures. The conclusions summarize the outcomes of the work and confirm the achievement of the stated goal.

Keywords: keyboard trainer, web application, React, touch typing, SPA, JSON, LocalStorage.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному цифровому суспільстві навички швидкого і правильного набору тексту набувають все більшого значення. Вміння швидко друкувати без помилок є важливим як у професійній діяльності, так і в повсякденному житті. З цією метою розробляються спеціальні тренажери для навчання сліпого методу друку.

Актуальність даної роботи полягає у створенні сучасного і багатомовного вебдодатку для тренування навичок набору тексту. Більшість існуючих рішень мають обмежений функціонал або не підтримують українську мову, що створює потребу у розробці нового продукту.

Метою кваліфікаційної роботи є розробка вебдодатку «Клавіатурний тренажер» з використанням сучасних вебтехнологій, а саме React, HTML5, CSS3, JavaScript.

Об’єктом кваліфікаційної роботи є процес навчання сліпому методу друку за допомогою інтерактивного інструменту.

Предметом кваліфікаційної роботи виступає методика реалізації вебдодатку для тренування навичок набору тексту з використанням компонентної архітектури.

Завданням кваліфікаційної роботи є поетапна реалізація розробки вебдодатку, що передбачає виконання таких дій:

– здійснити аналіз існуючих рішень для тренування навичок друку;

– сформувати функціональні і нефункціональні вимоги до системи;

– побудувати UML-діаграми, які відображають структуру системи та взаємодію її компонентів;

– обґрунтувати вибір архітектурних рішень та програмних засобів;

– створити інтуїтивно зрозумілий, зручний інтерфейс;

– створити структуру для зберігання слів та речень у форматі JSON;

– розробити можливість вибору режиму тренування;

– провести всебічне тестування функціональності вебдодатку з метою виявлення і усунення можливих недоліків;

– підготувати користувацьку документацію;

– оптимізувати вебдодаток відповідно до вимог пошукових систем;

– реалізувати заходи інформаційної безпеки клієнтської частини системи.

Практична значущість роботи полягає у можливості використання розробленого тренажера широким колом користувачів для вдосконалення своїх навичок набору тексту.

Робота базується на аналізі сучасних тенденцій у розробці SPA-додатків і використовує передові рішення у сфері UX/UI-дизайну для забезпечення найкращого користувацького досвіду.

Результати дослідження і розробки були представлені на X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025), м. Луцьк, 23-24 травня 2025 р. (додаток А).

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ВЕБДОДАТКУ «КЛАВІАТУРНИЙ ТРЕНАЖЕР» З ВИКОРИСТАННЯМ REACT І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Вебдодаток – це програмне забезпечення, яке запускається у веббраузері і використовує його в якості клієнта. Компанії повинні обмінюватися інформацією та надавати послуги віддалено. Вони використовують вебпрограми для зручного та безпечного зв'язку з клієнтами. Найбільш поширені функції вебсайту, такі як кошики, пошук та фільтрація товарів, обмін миттєвими повідомленнями та стрічки новин соціальних мереж, за своєю структурою є вебдодатками. Вони дозволяють отримати доступ до складних функціональних можливостей без встановлення або налаштування програмного забезпечення [1].

У кожному вебдодатку є три елементи: вебсервер для обробки запитів від клієнта, сервер додатків для виконання запитуваних завдань і база даних для зберігання інформації. Для роботи вебдодатка використовується комбінація сценарію на стороні сервера та сценарію на стороні клієнта.

Сценарій на стороні сервера займається зберіганням і отриманням інформації та потребує спеціальних мов кодування. Розробники програмують на стороні сервера для створення сценаріїв, які вебпрограма може використовувати для відповіді на запити. Сценарій на стороні клієнта займається представленням інформації користувачеві та використовує власні мови кодування.

Вебдодатки зазвичай використовують комбінацію серверного сценарію (ASP, PHP тощо) і клієнтського сценарію (HTML, Javascript тощо). Сценарій на стороні клієнта займається представленням інформації, тоді як сценарій на стороні сервера займається всіма складними речами, такими як зберігання та отримання інформації [2].

Архітектура вебдодатків – це структура, яка складається із зв'язків і взаємодії між компонентами додатків, такими як системи проміжного

програмного забезпечення, інтерфейси користувача та бази даних. Загальна концепція архітектури вебдодатків відповідає концепції користувача браузера, який запускає програму, здатну працювати на кількох вебсайтах [3].

По суті, архітектури вебдодатків можна визначити за допомогою зображення цього процесу:

– користувач переглядає певну URL-адресу, яку браузер знаходить і запитує;

– по мережі дані надсилаються із сервера до браузера, а потім виконуються браузером, щоб він міг відобразити запитану сторінку;

– користувач переглядає сторінку та взаємодіє з нею.

У сучасному цифровому світі програмні додатки відіграють вирішальну роль у трансформації інформаційного середовища. З розвитком технологій внутрішня логіка програми та спосіб її взаємодії з користувачами постійно вдосконалюються. Архітектура програмних рішень розвивається зі зростаючим попитом на швидкість, масштабованість, адаптивність та кросплатформні можливості.

Протягом останнього десятиліття Інтернет став основною платформою для розповсюдження контенту та надання послуг, що робить вкрай важливим для компаній усіх рівнів забезпечення своєї присутності в кіберпросторі. У сучасних умовах ця присутність також означає мобільну доступність, оскільки все більше користувачів отримують доступ до Інтернету через мобільні пристрої. У цьому відношенні архітектура мобільних та вебдодатків відіграє важливу роль у задоволенні потреб користувачів та забезпеченні єдиного досвіду на різних платформах.

Оскільки структура додатків стає складнішою, роль розробника змінюється. Наразі все більшої популярності набуває повноцінний підхід до розробки, де експерти відповідають як за клієнтську, так і за серверну частини системи. Повностекова архітектура охоплює набір знань та інструментів, які забезпечують цілісність вебдодатку – від інтерфейсу до обробки даних на

сервері. У такій моделі REST API відіграє ключову роль, забезпечуючи взаємодію між фронтендом та бекендом.

Вебсайт, з іншого боку – це група взаємопов’язаних вебсторінок, які мають спільне доменне ім’я та доступні з будь-якої точки світу. Він може бути створений окремою особою, компанією чи організацією та може використовуватися для різних цілей – від інформаційних до комерційних. Сучасні вебсайти відіграють багато важливих ролей: вони допомагають просувати товари та послуги, будувати бренди, покращувати обслуговування клієнтів та досягати бізнес-цілей [4].

Відмінність вебдодатків від нативних полягає в їх універсальності, оскільки вони можуть працювати на будь-яких пристроях та операційних системах. Вебдодаток розробляється таким чином, щоб належним чином адаптуватися до платформ iOS, Android, Windows Phone та інших.

Одна з переваг вебдодатків полягає в їх незалежності від ресурсів та апаратної платформи. Вони не ставлять вимог до потужності обладнання та не вимагають підтримки старих версій програм. У випадку з нативними додатками, користувачам часто доводиться вирішувати проблеми, пов’язані з оновленням вже встановленої копії програми на їх пристроях. У разі вебдодатків таких проблем не виникає, оскільки існує лише одна версія, яка працює для всіх користувачів. Коли випускається нова версія, всі користувачі, без винятку, переходять на неї, іноді навіть не помічаючи цього.

Вебдодатки відрізняються тим, що їх не потрібно встановлювати на пристрій. Вони працюють у вбудованому веббраузері пристрою, за допомогою простої URL-адреси. Немає необхідності завантажувати і встановлювати їх з магазинів додатків, таких як Google Play або Apple App Store. Це приносить економічну вигоду, оскільки використання вебдодатків через пряме посилання є безкоштовним.

Однією з основних вимог до вебдодатків є наявність постійного інтернет-з’єднання. Оскільки вебдодатки працюють через веббраузер, вони вимагають постійного доступу до Інтернету для завантаження вмісту та взаємодії з

сервером. Це означає, що без наявності стабільного інтернет-з’єднання користувачі не зможуть повноцінно користуватись вебдодатком.

Вебдодатки мають обмежений доступ до функцій та ресурсів пристрою порівняно з нативними додатками. Наприклад, вебдодатки можуть мати обмежені можливості доступу до камери, мікрофона, геолокації та інших апаратних компонентів пристрою.

Більшість користувачів досі друкують переважно двома пальцями, що є звичкою, що виробилася через брак систематичних тренувань. Цей метод неефективний, оскільки не має характеристик високої швидкості та високої точності. Натомість, метод сліпого друку може збільшити швидкість друку в 2-3 рази, знижує навантаження на зір, а також значно підвищує ефективність роботи. Ось чому багато IT-компаній, редакційно-видавничих агентств та суміжних галузей наполягають на оволодінні цим методом як базовою навичкою.

Існує багато способів опанувати сліпий друк, але одним з найефективніших вважається використання клавіатурного тренажера – програмного забезпечення, спеціально розробленого для розвитку навичок десятипальцевого друку та підвищення швидкості друку. Хоча інші методи все ще можливі (наприклад, механічне натискання на клавіатуру або виконання інструкцій на роздрукованій таблиці розташування пальців), вони поступаються симуляторам з точки зору гнучкості, інтерактивності та ефективності навчання.

Клавіатурний тренажер – це програма з функціоналом текстового редактора, яка дозволяє вводити заданий фрагмент тексту відповідно до тренувального зразка, а також може автоматично відстежувати та аналізувати деякі ключові показники: швидкість друку, кількість помилок тощо. Онлайн-версії симуляторів особливо популярні, оскільки вони не потребують встановлення, доступні з будь-якого пристрою та забезпечують зручний і практичний процес навчання як для початківців, так і для досвідчених користувачів, які прагнуть покращити свої навички друку. У наведеному нижче списку ми розглянемо кілька таких сервісів, які, при правильному підході, гарантують досягнення позитивних результатів. Кожен з них добре

зарекомендував себе серед користувачів, які успішно засвоїли всім відомий метод набору тексту.

Keybr – цей тренажер буде корисним, якщо ви зважилися навчитися сліпому друку або збільшити швидкість набору тексту. З одного боку, сервіс схожий на тренажери, що існують вже не перший рік, з іншого – має зовсім інший підхід до навчання (рис. 1.1).

Поширена проблема тренажерів зі збільшення швидкості друку – набір комбінацій символів, які навряд чи зустрінуться в щоденному наборі тексту. Наприклад, «tx» чи «kq» – настільки неприродні поєднання, що придумати слова, у яких ці літери стояли б поруч, практично нереально. Розробники Keybr розглядають іншу модель навчання. Головна увага приділяється набору послідовностей, частин слів, які найчастіше зустрічаються у мові. Завдяки цьому тренується м’язова пам’ять аналогічно пальцям музиканта, який грає на фортепіано.

Рисунок 1.1 – Інтерфейс вебдодатку «Keybr» [5]

Серед переваг цього сервісу є штучний інтелект, який аналізує ваші помилки, статистику швидкості та точності, коригуючи наступні уроки. Незвичайною фішкою є так звані «Гонки» – змагання на швидкість набору тексту з іншими людьми в реальному часі.

Значним недоліком є відсутність української мови, наявність всього двох кольорових тем – світлої і темної, а також занадто затягнуті вправи.

Typing Speed Test – ще один з представників онлайн-тренажерів сліпого друку. Розробником цього додатку є Shrey Agarwal. Сам додаток представляє з себе невелике вікно з набором випадкових слів, які користувач має набирати в зазначене для цього поле вводу тексту (рис. 1.2). Додаток має невеликий набір налаштувань. Користувач може вибрати час, за який він має якомога швидше набирати слова, а також можна вибрати один з двох рівнів складності – «Початківець» і «Досвідчений». Ці рівні відрізняються між собою складністю слів, які надає додаток для набору.

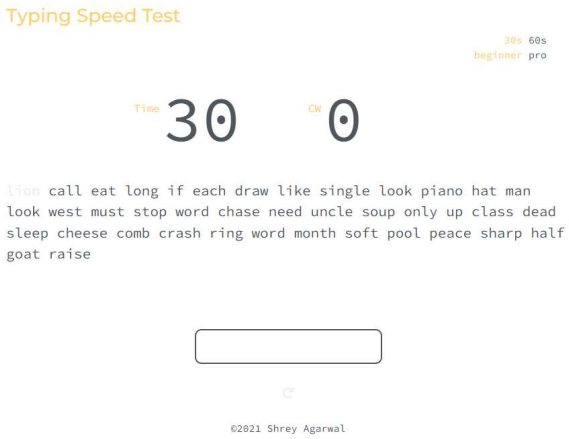

Рисунок 1.2 – Інтерфейс вебдодатку «Typing Speed Test» [6]

Відмінними якостями додатку є інтуїтивний та приємний інтерфейс і статистика після проведення вправи, а з недоліків можна зазначити невеликий набір налаштувань тренування і невеликий функціонал.

Тренажер друку, як онлайн-сервіс, не вимагає установки на комп’ютер користувача. Доступ до навчання можливий з будь-якої операційної системи за допомогою сучасного браузера. Займатися на клавіатурному тренажері можна з будь-якого комп’ютера, що має вихід в мережу Інтернет. Результати виконання завдань зберігаються в хмарі і ніколи не загубляться.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Враховуючи обрані технології та загальні цілі розробки вебдодатку для тренування навичок сліпого методу друку, було поставлено конкретні завдання, які дозволяють досягти успішного виконання кваліфікаційної роботи бакалавра. Зазначені цілі визначають основні напрямки розробки та вимоги до функціональності вебдодатку:

– здійснити аналіз існуючих рішень для тренування навичок друку, зокрема вебдодатків «Keybr» та «Typing Speed Test», визначити їх сильні і слабкі сторони;

– сформувати функціональні і нефункціональні вимоги до системи з урахуванням потреб цільової аудиторії та особливостей інтерфейсу;

– побудувати UML-діаграми, які відображають структуру системи та взаємодію її компонентів (діаграми прецедентів, послідовності, діяльності, розгортання);

– обґрунтувати вибір архітектурних рішень та програмних засобів, що використовуються у розробці (зокрема бібліотеки React і підходу SPA);

– створити інтуїтивно зрозумілий, зручний інтерфейс, який забезпечить комфортну взаємодію користувачів із системою;

– створити структуру для зберігання слів та речень у форматі JSON з підтримкою багатомовності (українська та англійська мови) для тренування користувачів;

– розробити можливість вибору режиму тренування (набір слів, речень, вільний текст), налаштування тривалості сесії та складності вправ відповідно до потреб користувача;

– провести всебічне тестування функціональності вебдодатку з метою виявлення і усунення можливих недоліків у роботі інтерфейсу, адаптивності та правильності обробки даних;

– підготувати користувацьку документацію (опис налаштування, запуску та використання вебдодатку);

– оптимізувати вебдодаток відповідно до вимог пошукових систем – забезпечити семантичну розмітку HTML, meta-теги, швидке завантаження, адаптивність;

– реалізувати заходи інформаційної безпеки клієнтської частини системи – уникнути XSS-уразливостей, реалізувати безпечні політики Content-Security-Policy.

Виконання цих завдань є ключовим для успішної реалізації вебдодатку «Клавіатурний тренажер» та спрямоване на створення сучасного, доступного і ефективного інструменту для розвитку навичок сліпого набору тексту серед користувачів.

Висновки до розділу 1

У результаті аналізу сучасного стану проблеми визначено основні особливості та тенденції розвитку вебдодатків, що використовуються для тренування навичок сліпого методу друку. Було встановлено, що існуючі рішення мають ряд обмежень, серед яких відсутність повноцінної підтримки української мови, недостатня адаптивність інтерфейсу та обмежені можливості налаштування вправ.

Аналіз архітектури сучасних вебдодатків показав доцільність використання компонентного підходу на базі бібліотеки React та архітектури Single Page Application (SPA), що дозволяє забезпечити високу швидкодію, гнучкість і масштабованість системи.

Постановка завдання на кваліфікаційну роботу визначила конкретні цілі та завдання розробки нового вебдодатку «Клавіатурний тренажер».

Таким чином, обґрунтована актуальність розробки сучасного вебдодатку для тренування навичок сліпого методу друку з урахуванням актуальних технологічних підходів та потреб користувачів.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Вебдодаток «Клавіатурний тренажер» створюється як навчальний інструмент для вдосконалення навичок сліпого методу друку. Основним завданням системи є надання користувачеві можливості ефективного тренування швидкості та точності набору тексту у комфортному та інтуїтивно зрозумілому середовищі. Система повинна підтримувати українську та англійську мови та зберігати налаштування користувача локально без використання серверної бази даних.

Вимоги до розроблюваного вебдодатку визначалися за допомогою кількох методів:

– аналіз аналогічних продуктів (Keybr, Typing Speed Test);

– вивчення відгуків користувачів про існуючі системи;

– аналітичне дослідження світових тенденцій у сфері навчальних вебдодатків.

На основі отриманої інформації сформульовано як функціональні, так і нефункціональні вимоги до системи.

Функціональні вимоги:

– можливість вибору режиму вправ, тобто набір окремих слів, речень або вільний текст;

– підтримка української та англійської мов для вправ;

– вибір теми оформлення інтерфейсу;

– відображення підсумкової статистики, в яку входить швидкість друку, кількість помилок тощо;

– збереження налаштувань користувача у LocalStorage.

Нефункціональні вимоги:

– висока швидкодія додатку без затримок взаємодії;

– надійність роботи при нестабільному з’єднанні з мережею;

– забезпечення доступності додатку для людей з різними рівнями підготовки.

Для формалізації вимог та проектування архітектури системи було обрано наступні діаграми UML:

– діаграма прецедентів – дозволяє відобразити основні сценарії використання додатку з боку користувача;

– діаграма діяльності – ілюструє логічні потоки процесів у системі;

– діаграма послідовності – моделює динамічну взаємодію об’єктів під час виконання основних сценаріїв роботи.

Використання цих діаграм сприяє структурованому представленню вимог і підвищує якість подальшої розробки.

Діаграма прецедентів (рис. 2.1) демонструє ключові взаємодії користувача із системою: вибір режиму тренування, зміну мови інтерфейсу, налаштування теми оформлення, запуск вправи, перегляд результатів.

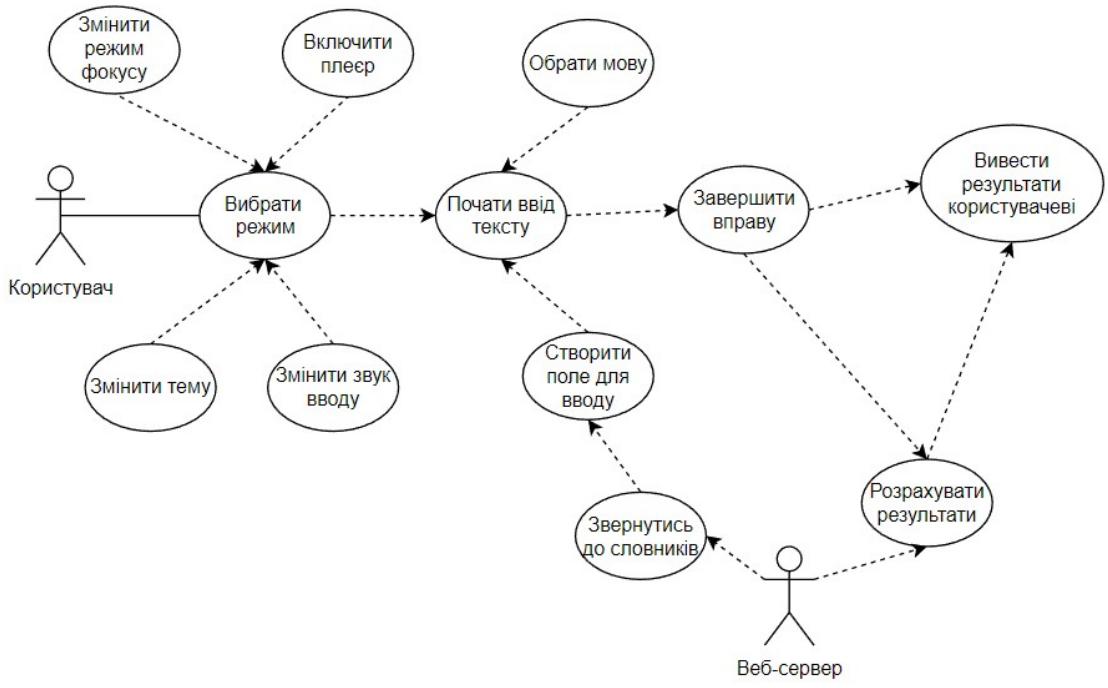

Рисунок 2.1 – Діаграма прецедентів

Діаграма діяльності (рис. 2.2) демонструє послідовність основних процесів взаємодії користувача з вебдодатком. Користувач починає з відкриття додатку та вибору режиму роботи – «режим слів» або «режим речень». Далі відбувається введення тексту вправи, після чого дані надсилаються на вебсайт для обробки. Отримавши результати, користувач має змогу повторити вправу. Така діаграма наочно ілюструє логіку роботи системи та взаємозв’язок між клієнтською та серверною частинами.

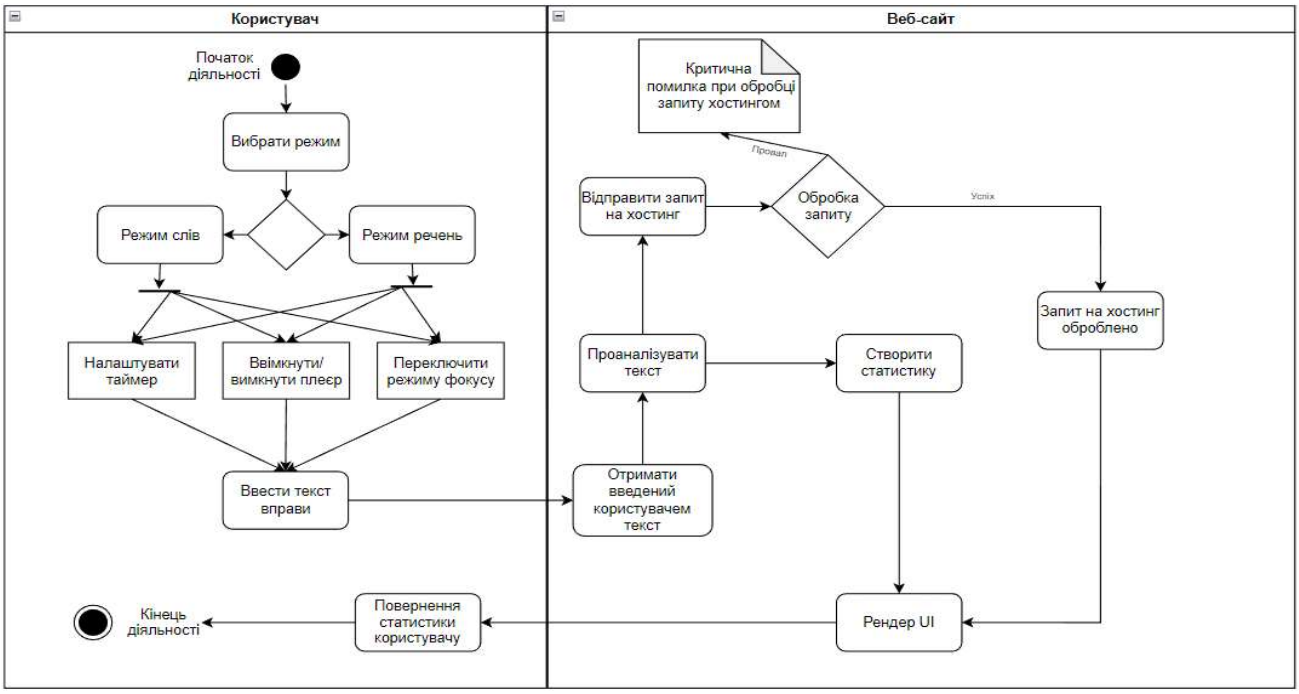

Рисунок 2.2 – Діаграма діяльності

Діаграма послідовності (рис. 2.3) моделює послідовність обміну повідомленнями між об’єктами під час тренування: від початку вправи до збереження проміжних результатів у локальному сховищі. Особливу увагу приділено реакції системи на введення користувача та оперативному оновленню інтерфейсу. Умовні розгалуження демонструють підтримку декількох режимів взаємодії (друк слів або речень), можливість вмикання фокус-режиму або звукового супроводу.

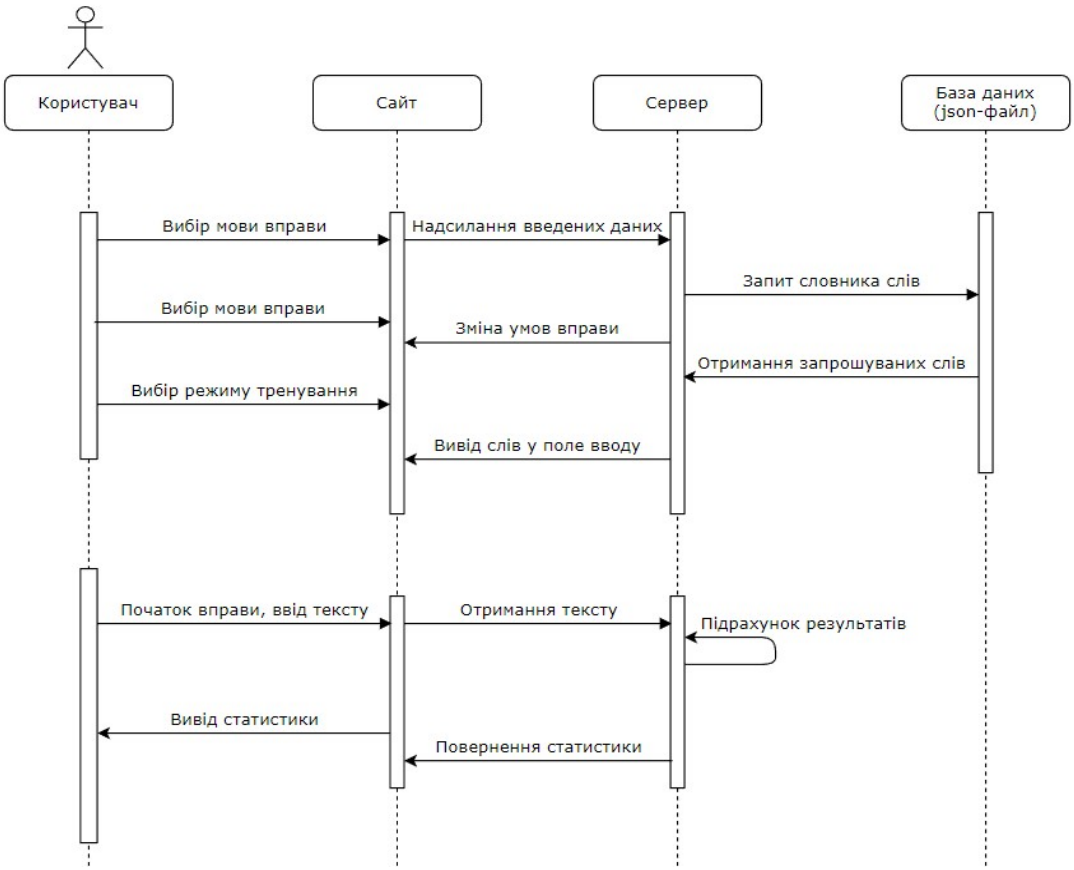

Рисунок 2.3 – Діаграма послідовності

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Сайти можуть бути створені з використанням різноманітних технологій та інструментів. Основними складовими будь-якого вебсайту є мови програмування та розмітки, такі як HTML, CSS, JavaScript тощо.

HTML (HyperText Markup Language, мова гіпертекстової розмітки) – мова, яка використовується для створення документів вебсторінок. В даний час використовується HTML5, що давно набув популярності і отримує все більшу підтримку у браузерах. У цій версії є більш вузька реалізація, так звана XHTML (extensible HTML; HTML, що розширюється). Це, по суті, та сама мова HTML, тільки з суворішими правилами синтаксису [7].

На відміну від HTML, який базується на специфікації SGML (Standard Generalized Markup Language, метамова для визначення мов розмітки

документів), XHTML побудовано на правилах XML, більш суворої підмножини SGML. Це означає, що документи XHTML повинні відповідати вимогам до структури XML, щоб їх могли обробляти стандартні парсери XML. HTML, у свою чергу, вимагає складніших та ресурсоємних інструментів парсингу. Отже, XHTML можна вважати поєднанням HTML та XML, оскільки він фактично є версією HTML, переписаною в термінах XML. Стандарт XHTML 1.0 був схвалений консорціумом W3C 26 січня 2000 року, а XHTML 1.1 – 31 травня 2001 року [8].

У минулому більша частина візуального форматування вебсторінок забезпечувалася елементами розмітки, прямо змішуючи концепції логічної та фізичної розмітки в безлад, яким є класичний HTML. У строгих варіантах (X)HTML застаріли елементи й атрибути, які зосереджені на презентації, забезпечуючи чітке розмежування між структурою, яку забезпечує розмітка, і виглядом, який диктує таблиця стилів, написана в синтаксисі каскадних таблиць стилів (CSS). Чіткий розподіл обов'язків між розміткою та стилем може забезпечити численні переваги у виробництві, обслуговуванні та навіть продуктивності, що робить його набагато кращим рішенням для презентації, ніж лише розмітка.

Концепцію каскадних таблиць стилів (CSS) запропонував Хокон Віум Лі у 1994 році. У грудні 1996 року альянс W3C опублікував офіційну специфікацію CSS, яка стала основою для подальшого розвитку технології вебдизайну. CSS дозволяє розробникам контролювати зовнішній вигляд елементів HTML-документа – змінювати шрифти, кольори, розміри, відступи та інші атрибути. Однією з головних переваг CSS є можливість підключити один файл стилів до кількох вебсторінок, що дозволяє централізовано змінювати дизайн усього вебсайту [9].

Перша версія, CSS1, була схвалена як рекомендація W3C у грудні 1996 року. Вона описує базову структуру мови та просту модель візуального представлення елементів HTML. Наступна версія, CSS2, стала рекомендованою у травні 1998 року. Вона розширила функціональність таблиць стилів, додавши

підтримку стилів для різних типів пристроїв (таких як принтери та аудіопристрої), шрифти для завантаження, систему позиціонування елементів та форматування таблиць [10].

Іноді звичайної, статичної вебсторінки користувачеві буде недостатньо, тому необхідно змусити сайт інтеркативно і динамічно змінюватись в залежності від дій користувача. Для цього використовується мова програмування JavaScript.

JavaScript є мовою програмування вебмережі. Переважна більшість вебсайтів використовує JavaScript, а всі сучасні веббраузери – у настільних комп'ютерах, планшетах і смартфонах – включають інтерпретатори JavaScript, роблячи JavaScript найпоширенішою мовою програмування в історії. Назва «JavaScript» є зареєстрованою торговою маркою компанії Sun Microsystems, Inc.

З моменту випуску в 1995 році JavaScript зазнав багато змін. Спочатку це значно спростило додавання інтерактивних елементів на вебсторінки. Потім він став більш надійним з DHTML і AJAX. Тепер, з Node.js, JavaScript став мовою, яка використовується для створення повноцінних програм. Зміни в мові здійснюються спільнотою. Вони походять із пропозицій, які пишуть учасники спільноти. Кожен може подати пропозицію до комітету ECMA. Відповідальність комітету ECMA полягає в тому, щоб керувати цими пропозиціями та визначати їх пріоритетність, щоб вирішити, що включено до кожної специфікації. Пропозиції розглядаються через чітко визначені етапи: від етапу 0, який представляє найновіші пропозиції, до етапу 4, який представляє готові пропозиції [11].

Існує кілька версій JavaScript. Версія ECMAScript 3 була широко підтримуваною версією під час сходження JavaScript до домінування, приблизно між 2000 і 2010 роками. Протягом цього часу тривала робота над амбітною версією 4, яка планувала низку радикальних покращень і розширень мови. Змінити живу, широко вживану мову таким радикальним способом виявилося політично важко, і робота над версією 4 була припинена в 2008 році, що призвело до набагато менш амбітної версії 5, яка вносила лише деякі покращення, яка вийшла в 2009 році. Потім у 2015 році вийшла версія 6, велике оновлення, яке

включало деякі ідеї, заплановані для версії 4. Відтоді ми щороку отримували нові невеликі оновлення [12].

Для створення дійсно якісного додатку використання одних лиш мов програмування недостатньо. Для максимально гнучких налаштувань додатку використовуються фреймворки.

Фреймворк – це програмний інструмент, який може значно спростити процес розробки та підтримки технічно складних програмних проєктів. Зазвичай фреймворк містить лише набір базових модулів та структур, тоді як конкретна реалізація функціональних компонентів залишається на розсуд розробника. Такий підхід сприяє швидкій розробці та підвищує продуктивність і надійність створених рішень.

Вебфреймворк – це спеціалізована платформа, орієнтована на створення вебсторінок та вебдодатків. Він забезпечує інтеграцію основних компонентів проєкту, сприяє реалізації бізнес-логіки, а також дозволяє ефективно розширювати та підтримувати масштабні вебсистеми. Завдяки широкому вбудованому інструментарію, вебфреймворки особливо корисні в складних проєктах, де важливі гнучкість, швидкість та підтримка сучасних стандартів безпеки.

З практичної точки зору, використання фреймворку є більш економічним та доцільним, ніж розробка проєкту з нуля на чистій мові програмування без використання готової платформи. Єдиними винятками можуть бути дуже прості проєкти або специфічні системи, які потребують розширеної оптимізації (наприклад, обробка десятків тисяч запитів за секунду). У переважній більшості випадків фреймворки дозволяють досягти кращих результатів з меншими витратами часу та фінансів.

Порівняно з іншими типовими платформами (такими як SaaS, CMS або CMF), фреймворки демонструють вищу ефективність у реалізації складної бізнес-логіки та систем, а також мають вищі вимоги до масштабованості, продуктивності та надійності. Водночас, для простіших проєктів з меншою

кількістю функціональних вимог, використання SaaS або CMS може бути більш доцільним з точки зору вартості та швидкості впровадження [13].

Однією з ключових переваг використання фреймворку є наявність чітко визначеної структури інтеграції для створення застосунків. Завдяки цьому структура додатків, створених за допомогою фреймворку, зрозуміла іншим розробникам, що значно спрощує підтримку, вдосконалення та розширення системи. Типові шаблони побудови класів та логічного розміщення дозволяють навіть експертам, які не брали участі в початковій розробці, швидко орієнтуватися в архітектурі. Більшість сучасних вебфреймворків реалізують принципи шаблону MVC (Model-View-Controller), який чітко розділяє бізнес-логіку, презентацію та обробку подій, тим самим сприяючи більш організованій та простішій для розуміння структурі коду.

Розробка програмного забезпечення за допомогою фреймворків спрощує архітектурне проєктування завдяки використанню найкращих практик програмної інженерії. Фреймворк зазвичай складається з набору конкретних та абстрактних класів, які взаємодіють один з одним відповідно до логіки фреймворку. Абстрактні класи виступають точками розширення, де можна створювати власні функції або використовувати базову логіку, не змінюючи її. Конкретні класи визначають зв’язки між компонентами системи. Більшість фреймворків побудовані на об’єктно-орієнтованому програмуванні, що дозволяє успадковувати функціональність, повторно використовувати код та організовувати систему в незалежні, але взаємопов’язані модулі.

Крім того, сучасна вебекосистема пропонує різноманітні готові модулі та компоненти, розроблені спільнотою. Це дозволяє значно зменшити ваше щоденне робоче навантаження: розробнику не потрібно створювати типові функції «з нуля», оскільки він може використовувати перевірені та протестовані рішення. Повторне використання відкритих компонентів не лише економить ресурси, але й покращує стабільність та якість програмних продуктів, оскільки такі компоненти зазвичай активно підтримуються, добре документовані та тестуються в багатьох проєктах.

React – популярна бібліотека, яка використовується для створення інтерфейсів користувача. Він був створений у Facebook для вирішення деяких проблем, пов’язаних із великомасштабними вебсайтами.

React надає можливість розробникам створювати великі вебдодатки, які можуть змінювати дані без перезавантаження сторінки. Основна мета React – бути швидким, таким, що масштабується, і простим. Працює тільки на призначених для користувача інтерфейсів в додатку. Це відповідає уявленню в шаблоні MVC. Він може використовуватися з комбінацією інших бібліотек JavaScript або середовищ, таких як Angular JS, Redux.

React – це бібліотека, призначена для оновлення DOM браузера. Нам більше не потрібно турбуватися про складнощі, пов’язані зі створенням ефективних Single Page Application (SPA), оскільки React може зробити це за нас. За допомогою React ми не взаємодіємо безпосередньо з DOM API. Замість цього ми взаємодіємо з Virtual DOM або набором інструкцій, які React використовуватиме для створення інтерфейсу користувача та взаємодії з браузером. Віртуальна DOM складається з елементів React, які концептуально здаються схожими на елементи HTML, але насправді є об’єктами JavaScript. Набагато швидше працювати безпосередньо з об’єктами JavaScript, ніж працювати з DOM API. Ми вносимо зміни в об’єкт JavaScript, virtual DOM, і React максимально ефективно рендерить ці зміни за допомогою DOM API.

При необхідності змінити елементи на вебсторінці, React використовує підхід з віртуальним DOM. Зміни спочатку застосовуються до віртуального DOM, а потім порівнюються з поточним станом. Якщо виявляються різниці, React знаходить найменшу кількість операцій, необхідних для оновлення реального DOM до нового стану, і виконує їх. Цей підхід дозволяє працювати з елементами вебсторінки набагато швидше та ефективніше, ніж пряма маніпуляція JavaScript з DOM.

Раніше, коли браузери були менш потужними, а продуктивність JavaScript була низькою, кожна сторінка завантажувалася з сервера. При кожній взаємодії зі сторінкою, був виконаний новий запит на сервер, і браузер

перезавантажувався, щоб відобразити нову сторінку. Тільки деякі інноваційні продукти використовували альтернативні підходи та експериментували з новими методами.

Односторінковий застосунок (SPA, англ. Single Page Application) – це тип вебдодатку або вебсайту, який взаємодіє з користувачем шляхом динамічного оновлення вмісту на поточній сторінці без її повного перезавантаження. Замість традиційного механізму, за якого кожна дія користувача ініціює повне завантаження нової HTML-сторінки з сервера, SPA здійснює вибіркове підзавантаження необхідних даних і відображає їх у межах вже завантаженого інтерфейсу. Такий підхід забезпечує швидший відгук системи та покращений користувацький досвід, наближаючи вебдодаток за поведінкою до нативного програмного забезпечення [14].

У рамках SPA модель передбачає, що вся структура HTML, JavaScript і CSS завантажується при першому відкритті сайту або динамічно дозавантажується при потребі – у відповідь на дії користувача. Наприклад, у більшості сучасних систем електронної пошти навігація між папками не призводить до повного оновлення сторінки – інтерфейс залишається сталим, а змінюється лише вміст основної області. Такий механізм дозволяє надсилати на сервер лише ті запити, які стосуються нових даних, а не повного HTML-документа, що значно зменшує навантаження на сервер та оптимізує споживання мережевих ресурсів.

Використання SPA сприяє зменшенню затримок при взаємодії, оскільки клієнт отримує оновлення миттєво, без тривалих циклів «запит-відповідь». Це позитивно впливає на швидкість роботи застосунку, знижує вимоги до серверної інфраструктури та скорочує загальні витрати на її підтримку. Крім того, відокремлення візуального представлення (presentation) від логіки роботи з даними (content and data) дозволяє розробникам frontend і backend частин працювати паралельно, ефективніше координуючи спільну розробку. SPA також легко адаптується до різних типів пристроїв – десктопів, планшетів, мобільних телефонів – і є популярним рішенням у розробці сучасних адаптивних вебінтерфейсів [15].

Таким чином, архітектура SPA забезпечує переваги як для кінцевих користувачів – у вигляді швидкого та зручного доступу до функціоналу – так і для розробників, пропонуючи гнучкість, узгодженість структури, покращену продуктивність і зниження витрат на підтримку та масштабування.

JSX – це абревіатура від JavaScript Syntax Extension. Це синтаксис для поєднання коду JavaScript і HTML, або, точніше, написання HTML-коду в коді JavaScript. JSX – це мова шаблонів, яка містить усі можливості JavaScript [16].

Синтаксис: «const element = <h1>Hello World!</h1>».

Як видно з наведеного синтаксису, код HTML і JS написаний в одному рядку. Це не код HTML і не JS, це JSX, компіляція обох мов в одному коді. Писати код React у JSX – чудова практика, але це не обов'язково. У деяких випадках можливо використовувати звичайні функції створення елементів, проте JSX значно полегшує процес розробки. Ось деякі переваги написання коду в JSX:

– написання коду React у JSX полегшує його читання та розуміння;

– є можливість просто написати HTML-код всередині JS, і це буде називатися кодом React;

– відтворення коду відбувається швидше порівняно зі звичайним JavaScript;

– написання коду в JSX не тільки робить його чистим і зрозумілим, але навіть може допомогти у виправленні помилок.

Основна проблема JSX полягала в тому, що браузер не міг прочитати його синтаксис, оскільки він містить і HTML, і JavaScript. Щоб вирішити цю проблему, у 2014 році було представлено Babel, який допомагає браузерам читати код JSX, перетворюючи його на звичайний JS.

Babel – це компілятор JavaScript, який в основному використовується для перетворення ECMAScript2015 / ES6 та його новіших версій у звичайний JavaScript, який розуміють браузери.

ECMAScript 2015 або ES6 – це шоста версія JavaScript, яка повністю оснащена новими функціями, зокрема стрілковими функціями, ключовими словами let і const, класами тощо.

Хуки дозволяють використовувати стани компонентів та інші функції React без написання класу. Хуки не замінюють знань про концепції React. Натомість хуки надають більш прямий API для концепцій React, які вже відомі: пропси, стан, контекст, посилання та життєвий цикл. Хуки також пропонують новий потужний спосіб їх поєднання.

React не пропонує способу «приєднати» багаторазову поведінку до компонента (наприклад, підключити його до магазину). Якщо деякий час працювати з React, можливо стикнутись з такими шаблонами, як рендеринг-реквізити та компоненти вищого порядку, які намагаються вирішити цю проблему. Але ці шаблони вимагають реструктуризації компонентів під час їх використання, що може бути громіздким і ускладнює дотримання коду. Якщо поглянути на типову програму React у React DevTools, швидше за все, знайдеться «пекло оболонки» компонентів, оточених шарами провайдерів, споживачів, компонентів вищого порядку, рендеринг-пропсів та інших абстракцій. Хоча варіант відфільтрувати їх у DevTools можливий, та це вказує на глибшу проблему: React потребує кращого примітиву для спільного використання логіки стану.

За допомогою хуків можна витягти логіку стану з компонента, щоб її можна було протестувати незалежно та повторно використовувати. Хуки дозволяють повторно використовувати логіку стану без зміни ієрархії компонентів. Це полегшує спільний доступ до хуків між багатьма компонентами або спільнотою.

У багатьох випадках неможливо розбити компоненти на більш дрібні, тому що логіка стану повсюдна. Тестувати їх теж важко. Це одна з причин, чому багато людей вважають за краще поєднувати React з окремою бібліотекою керування станом. Однак це часто створює занадто багато абстракцій, вимагає

переходу між різними файлами та ускладнює повторне використання компонентів.

Щоб вирішити цю проблему, хуки дозволяють розділити один компонент на менші функції на основі того, які частини пов’язані (наприклад, налаштування підписки чи отримання даних), а не примусово розділяти на основі методів життєвого циклу. Ще одним з можливих варіантів є вибрати керування локальним станом компонента за допомогою редуктора, щоб зробити його більш передбачуваним.

Платформа Node.js необхідна для створення веборієнтованих систем, написаних на JavaScript. Вона необхідна для створення серверної частини веборієнтованої системи, підключення бібліотек, запуска сервера і взаємодії користувача з додатком через введення запитів та виведення результатів. Також вона є необхідною, тому що такі системи у використанні обробляють велику кількість запитів, при чому одночасно. З метою забезпечення обробки всіх викликів використовується асинхронна модель запуску коду. Система Node.js виконує код додатку окремо від браузера. До переваг Node.js можна віднести наступні особливості:

– легкість створення систем;

– велика кількість доступних бібліотек;

– можливість застосування лише однієї мови програмування для написання як клієнтської, так і серверної частини додатку;

– асинхронність роботи і обробки запитів.

Node.js – це середовище для виконання JavaScript коду та транслювання його в машинний код. В цьому середовищі створюється серверна частина веборієнтованої системи. При створенні сервера використовується модуль http, який забезпечує використання протоколу http. Викликавши функцію створення сервера необхідно вказати номер порта, що буде використовуватися у створеній системі.

Створивши сервер, необхідно забезпечити обробку запитів до сервера. Для цього існують два параметри:

– request – отримання та зберігання інформації про запити;

– response – відправлення відповіді.

Основними недоліками вважається те, що в зв’язку з постійними змінами в цій платформі, необхідно ретельно стежити за оновленнями та вона має низьку продуктивність при роботі зі складними задачами.

Висновки до розділу 2

У цьому розділі було проведено всебічний аналіз вимог до розроблюваного програмного забезпечення, а також здійснено проектування основних елементів вебдодатку «Клавіатурний тренажер». В результаті аналізу аналогічних рішень, вивчення відгуків користувачів та консультацій з потенційними споживачами сформульовано чіткі функціональні та нефункціональні вимоги до системи.

Проектування архітектури було здійснено із застосуванням діаграм UML, зокрема діаграми прецедентів, діаграми діяльності та діаграми послідовності, що дозволило наочно відобразити сценарії використання системи, її логіку роботи та взаємодію об’єктів.

У процесі вибору технологій для розробки було обґрунтовано застосування HTML5, CSS3 та JavaScript як основних мов для створення вебдодатку, а також використання сучасних фреймворків і бібліотек, таких як React і Node.js, що дозволяють реалізувати адаптивну, динамічну і масштабовану архітектуру додатку.

Таким чином, проведений аналіз і проектування створили надійну основу для подальшої розробки функціонального, зручного та ефективного інструменту для навчання сліпому методу друку, що відповідає сучасним вимогам до якості вебдодатків.

РОЗДІЛ 3
РОЗРОБКА ВЕБДОДАТКУ «КЛАВІАТУРНИЙ ТРЕНАЖЕР» З ВИКОРИСТАННЯМ REACT

3.1 Практична реалізація об’єкта проєктування

Розробка вебдодатку «Клавіатурний тренажер» здійснювалась з використанням бібліотеки React, яка забезпечує ефективну побудову інтерфейсу за допомогою компонентної архітектури. Додаток реалізовано як односторінковий (SPA), що дозволяє уникати перезавантаження сторінок при зміні станів та підвищує швидкодію і зручність взаємодії користувача з додатком.

Основна логіка додатку реалізована за допомогою функціональних компонентів React. Взаємодія користувача з додатком побудована на обробці подій введення з клавіатури, що дозволяє в реальному часі оцінювати точність набору тексту. Додаток зберігає налаштування користувача, обрану мову, тему та результати вправ у локальному сховищі браузера (LocalStorage), що забезпечує автономність і безпеку персональних даних.

Основними вимогами до вебдодатку є:

- зручний мінімалістичний інтерфейс;
- інтерактивний режим набору тексту в реальному часі;
- можливість вибору рівня складності та мови;
- підтримка зміни темного та світлого оформлення;
- адаптивність для мобільних пристроїв;
- збереження локальних налаштувань користувача через LocalStorage.

Розробка вебдодатку здійснювалась із використанням бібліотеки React. Для запуску проєкту була створена його структура через команду «npx create-react-app», після чого були встановлені необхідні залежності для управління станом та маршрутизації. Основна логіка розміщена у функціональних компонентах із використанням React Hooks (useState, useEffect, useContext).

Компоненти та об’єкти, що виконуються або використовуються при роботі розроблюваного застосунку, наведено на діаграмі розгортання (рис. 3.1). Роботу застосунку можна описати так, що користувач працює з додатком, використовуючи веббраузер. Надсилаючи запити, він звертається до вебсервера, всередині якого відбуваються необхідні розрахунки, а в кінцевому результаті створює користувацький інтерфейс і виводить його на екран.

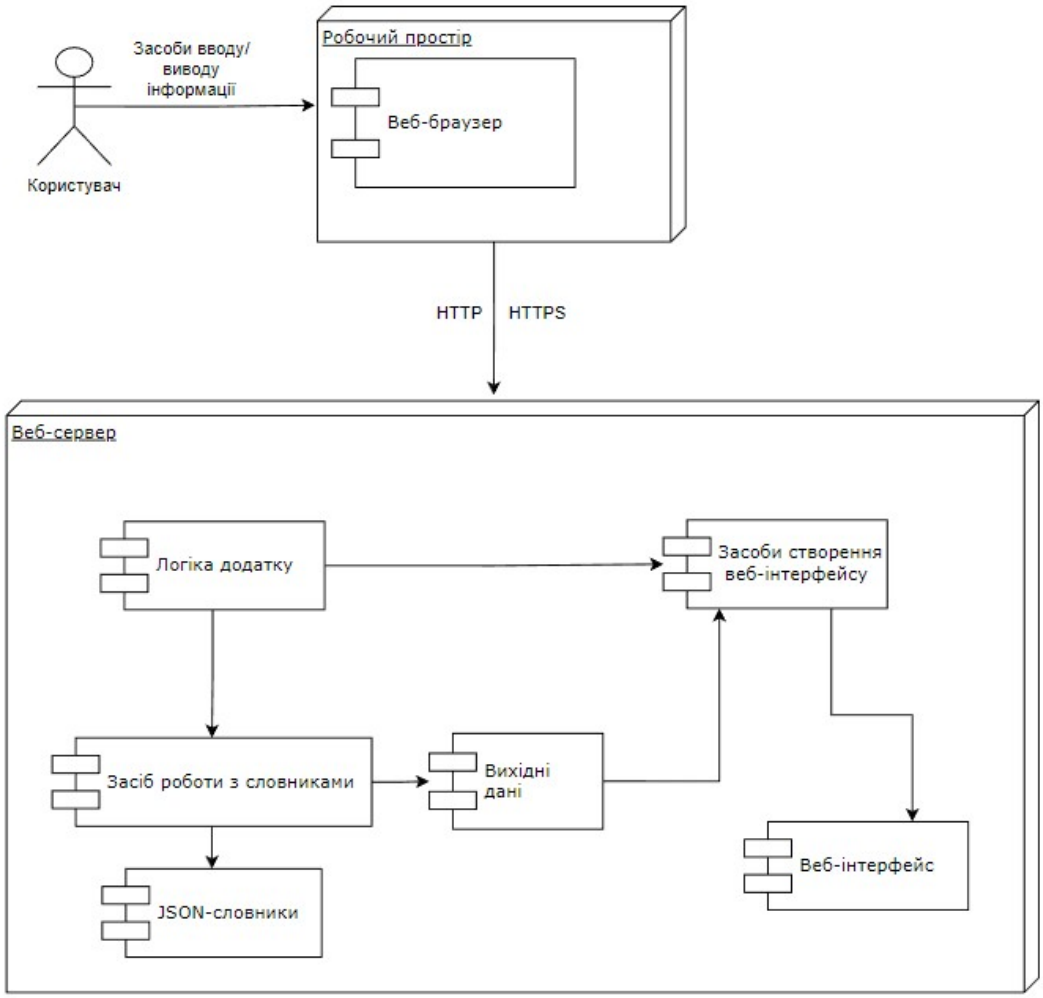

Рисунок 3.1 – Діаграма розгортання

Основні компоненти вебдодатку:

– App – кореневий компонент, що відповідає за загальне структурування сторінки, маршрутизацію та глобальні налаштування;

– FooterMenu – реалізує нижню панель керування інтерфейсом користувача у вебдодатку. Він надає доступ до основних налаштувань програми

та виконує функції навігації, вибору режиму тренувань, керування звуком, темою оформлення та іншими параметрами взаємодії користувача з додатком. FooterMenu побудовано як функціональний компонент з використанням бібліотеки React та компонентів з бібліотек @mui/material і @mui/icons-material. Він містить адаптивну структуру, що організована за допомогою компонента Grid із системи Material UI;

– SentenseBox (рис. 3.2) – основний інструмент у режимі вправ з реченнями (Sentence Mode) у вебдодатку. Його функціональне призначення – створення інтерактивного середовища, в якому користувач тренує навички сліпого друку, вводячи цілі речення. Він об’єднує в собі логіку генерації вправ, обробку користувацького введення, ведення статистики, а також керування мовними та технічними налаштуваннями;

Рисунок 3.2 – Вигляд компонента SentenseBox

– Stats – відповідає за відображення основної статистичної інформації під час і після виконання вправи з набору тексту. Він надає користувачу зворотний зв’язок про результати тренування у реальному часі та після завершення таймеру;

– TypeBox (рис. 3.3) – є основною частиною функціоналу вебдодатку у режимі тренування набору слів. Його призначення – організувати інтерактивне середовище для тренування сліпого друку окремих слів протягом обмеженого часу із підрахунком швидкості набору, точності та кількості правильних/помилкових натискань.

Рисунок 3.3 – Вигляд компонента TypeBox

Додаток структуровано за принципом ізольованих компонентів. Кожен компонент виконує свою роль у загальній логіці програми, що дозволяє легко масштабувати або змінювати окремі частини без впливу на інші. Компоненти обмінюються даними через props або за допомогою глобального стану (React Context або hooks).

Для підвищення функціональності та покращення користувацького досвіду у додатку кваліфікаційної роботи були впроваджені низка нестандартних і цікавих рішень.

Одним із таких рішень є інтеграція модулю відтворення музики безпосередньо в інтерфейс тренажера. Це дозволяє створити комфортну атмосферу для користувача під час набору тексту, що позитивно впливає на концентрацію та знижує рівень стресу під час проходження вправ.

Для реалізації цієї можливості створено окремий компонент MusicPlayer, який вбудовує у сторінку інтерактивний віджет Spotify. Компонент динамічно підлаштовується під активний режим роботи додатку: у режимі фокусу (Zen Mode) музичний плеєр автоматично змінює свої розміри для збереження мінімалістичності інтерфейсу, а при вимкненні – розгортається повністю.

Також реалізовано анімаційне з’явлення плеєра за допомогою компонента Slide з бібліотеки Material UI, що забезпечує плавність переходів і покращує загальне враження від взаємодії.

Кодова реалізація компонента MusicPlayer наведена у лістингу 3.1.

Лістинг 3.1 – Компонент MusicPlayer

```
import React from "react";
import { Slide } from "@mui/material";
const MusicPlayer = ({ disabled, isZenMode }) => {
  const height = isZenMode ? "240" : "400";
  const players = {
    spotify:
      '<iframe style="border-radius:12px" src="" width="100%" height=' +
height+
      ' frameBorder="0" allow="autoplay; clipboard-write; encrypted-
media; fullscreen; picture-in-picture"></iframe>',
  };
  const IframePlayer = (props) => {
    if (disabled) {return null;}
    return (<div dangerouslySetInnerHTML={{ __html: props.iframe ?
props.iframe : "" }}/>);
  };
  return (
      <Slide direction="up" style={{ transitionDelay: disabled ? "200ms"
: "0ms" }} in={!disabled} mountOnEnter unmountOnExit>
      <div><IframePlayer iframe={players["spotify"]} /></div>
      </Slide>);};
export default MusicPlayer;
```

кінець лістингу 3.1

Таким чином, впровадження мультимедійної підтримки у додатку дозволяє не тільки урізноманітнити процес тренування, але й забезпечує гнучкість налаштувань відповідно до індивідуальних потреб користувачів.

Ще одним важливим удосконаленням функціональності вебдодатку є реалізація системи оповіщення про активований режим Caps Lock.

У процесі тренування друку неправильне включення Caps Lock може суттєво вплинути на результати вправи, оскільки призводить до некоректного введення символів. Для вирішення цієї проблеми був розроблений окремий компонент CapsLockSnackbar, який відповідає за відображення попереджувального повідомлення при активації клавіші Caps Lock. Вигляд спливаючого повідомлення зображено на рисунку 3.4.

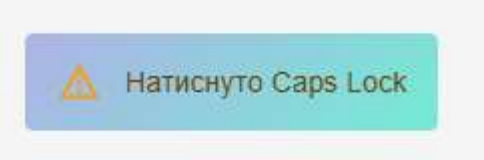

Рисунок 3.4 – Спливаюче повідомлення CapsLockSnackbar

Компонент реалізований із використанням бібліотеки Material UI та поєднує у собі такі елементи:

– Snackbar – для створення попередження у верхній частині екрану;

– Alert – для акцентування уваги користувача на важливому повідомленні;

– Slide – для додання анімаційного ефекту плавної появи сповіщення.

При активації стану open, який передається у вигляді пропса, компонент відображає попередження «Натиснуто Caps Lock», тим самим миттєво інформуючи користувача про помилку. Візуальне оформлення повідомлення обрано у стилі «warning», що підсилює ефект привернення уваги.

Кодова реалізація компонента наведена у лістингу 3.2.

Лістинг 3.2 – компонент CapsLockSnackbar

```
import React from "react";
import { Slide } from "@mui/material";
import { Alert } from "@mui/material";
import { Snackbar } from "@mui/material";
const CapsLockSnackbar = ({open}) => {
  return (
    <div>
```

```
    <Snackbar open={open} anchorOrigin={{vertical: "top", horizontal:
"left",}}>
      <Slide in={open} mountOnEnter unmountOnExit><Alert
className="alert" severity="warning" sx={{ width: "100%" }}>Натиснуто
Caps Lock</Alert></Slide></Snackbar>
  </div>);};
export default CapsLockSnackbar;
```

кінець лістингу 3.2

Як результат, впровадження миттєвого оповіщення про активований Caps Lock суттєво покращує зручність використання сайтом та дозволяє зменшити кількість помилок під час виконання вправ.

Словникова база реалізована у вигляді JSON-масиву, що містить перелік слів і речень двома мовами. Залежно від обраної мови, додаток формує вправу відповідно до заданих параметрів. Слова обираються випадковим чином із відповідного словника. Невеликий фрагмент з словника наведено у лістингу 3.3.

Лістинг 3.3 – Словник слів

```
{
"21": {
    "val": "Своїм успіхом я зобов’язана тому, що ніколи не
виправдовувалася і не приймала виправдань від інших."
},
"22": {
    "val": "Логіка може привести вас. від пункту “А” до пункту “Б”, а
уява – куди завгодно"
},
"23": {
    "val": "Неосмислене життя не варте того, щоб його прожити."
},
"24": {
    "val": "Наука – це організовані знання, мудрість – це організоване
життя."
},
"25": {
    "val": "Ви ніколи не перетнете океан, якщо не наберетеся мужності
втратити берег з виду."
},
"26": {
    "val": "Або ви керуєте вашим днем, або день управляє вами."
},
}
```

кінець лістингу 3.3

Інтерфейс додатку (рис. 3.5) побудований за принципами UX-дизайну: всі налаштування зібрані в одній зоні, інтерфейс адаптивний, а навігація інтуїтивно зрозуміла. Оформлення враховує темний та світлий режими роботи, що важливо для зручності користувачів.

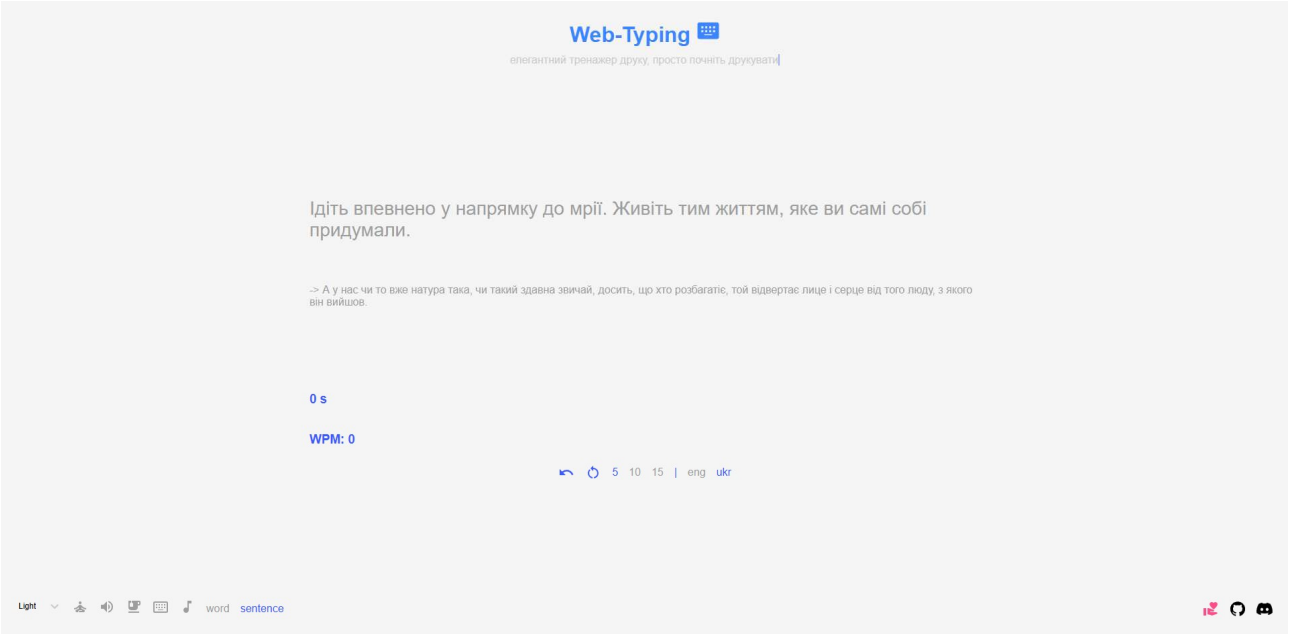

Рисунок 3.5 – Інтерфейс додатку

Діаграми діяльності використовуються в багатьох галузях, включаючи програмування, бізнес-аналітику, проектування програмного забезпечення, управління проектами та багато іншого. Вони допомагають розібратися зі складними процесами та візуалізувати послідовність дій, що дозволяє зрозуміти, як можна оптимізувати ці процеси або знайти їхні проблеми.

Для кращого розуміння внутрішньої логіки роботи застосунку були побудовані діаграми діяльності для окремих процесів.

На рисунку 3.6 зображена діаграма діяльності детальної роботи генерації слів, взаємодії додатку з словником, та вивід необхідних для вводу слів користувачу. Після вибору мови та рівня складності викликається відповідна функція генерації. Вона отримує необхідні параметри (кількість слів, складність) та випадковим чином обирає слова з попередньо підготовленого JSON-словника.

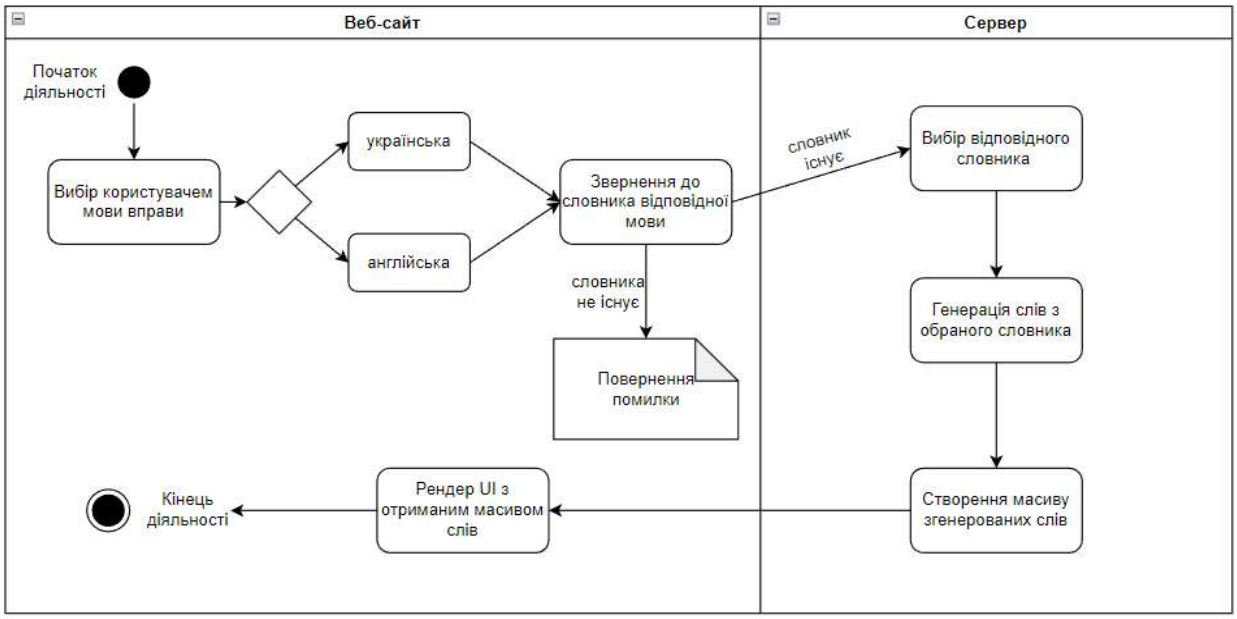

Рисунок 3.6 – Діаграма діяльності генерації слів

Наступна діаграма (рис. 3.7) показує детальний процес виконання самої вправи з боку інтерфейсу користувача та його дій, відображає повний цикл обробки введення користувачем, а також обробки даних з боку серверу. Під час вправи автоматично підраховуються кількість правильних і неправильних натискань та точність набору.

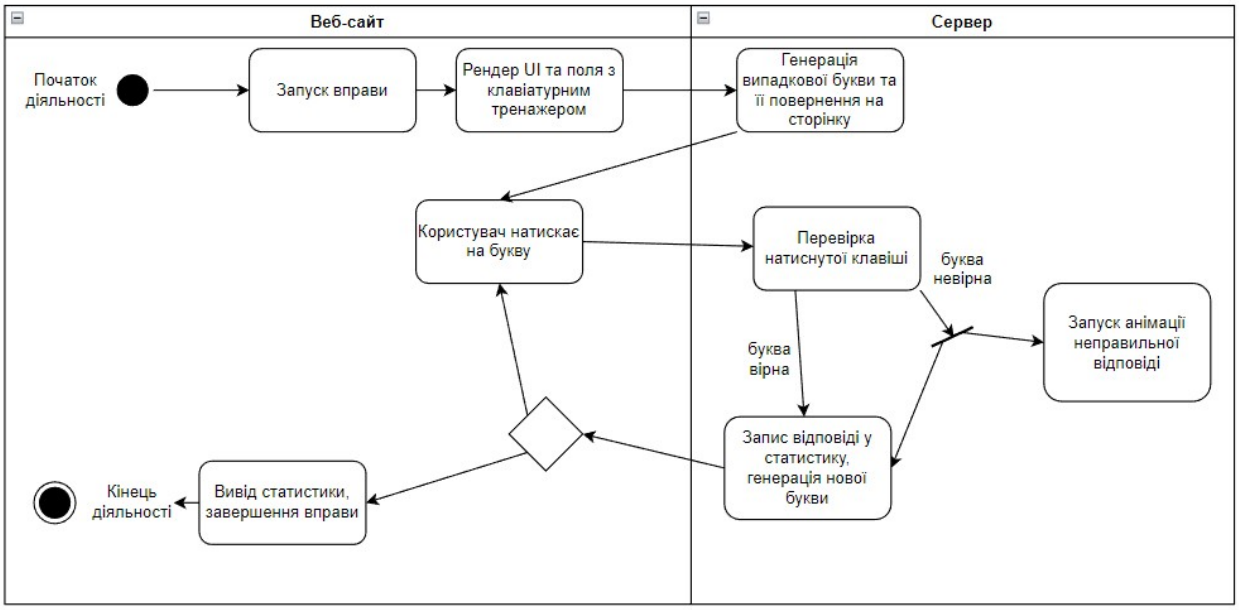

Рисунок 3.7 – Діаграма діяльності клавіатурного тренажеру

На рисунку 3.8 зображена діаграма діяльності яка ілюструє процес вибору користувачем режиму оформлення інтерфейсу: світлого або темного. Після натискання на перемикач теми в інтерфейсі сайту ініціюється запит до контекстного провайдера теми, який змінює поточний стан оформлення на протилежний.

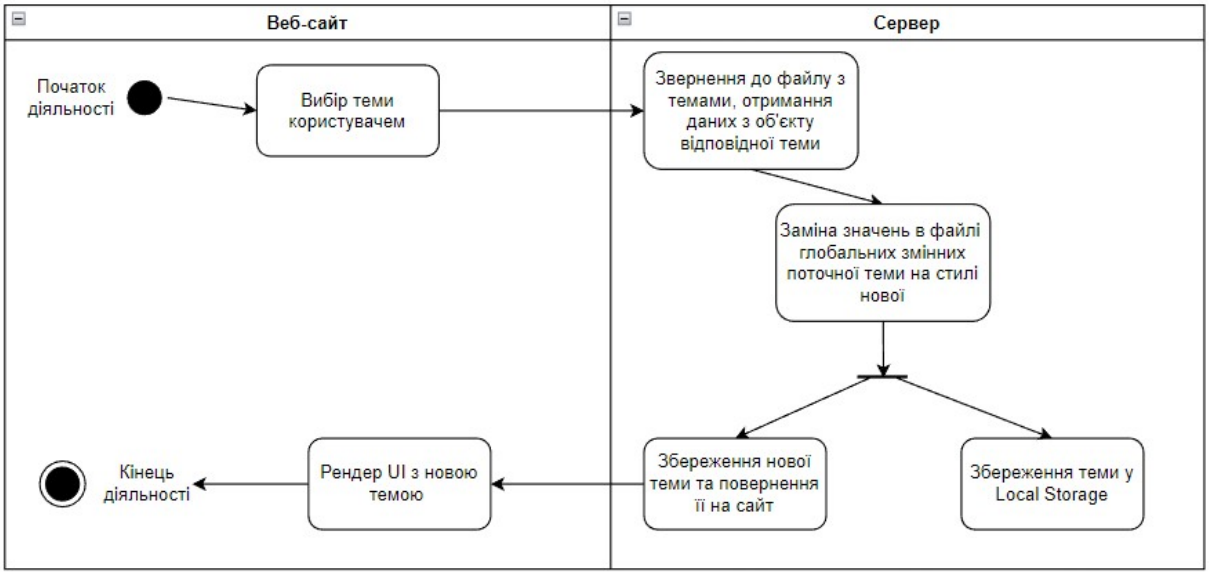

Рисунок 3.8 – Діаграма діяльності зміни теми

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі розробки вебдодатку було особливу увагу приділено процесу тестування та налагодження системи. Це дозволяє виявити помилки, покращити якість програмного забезпечення та забезпечити стабільність функціонування на різних пристроях.

Для тестування використовувалися такі підходи: модульне тестування компонентів інтерфейсу, інтеграційне тестування взаємодії між модулями та ручне тестування з точки зору кінцевого користувача. Тестування проводилось у різних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge) для перевірки кросбраузерної сумісності.

Одним із прикладів автоматизованого тестування є тест компонента, що відповідає за генерацію випадкових символів для друку. Цей компонент має забезпечити коректне відображення символів на екрані, їх зміну при помилковому вводу та підрахунок правильних і помилкових натискань.

Нижче представлено приклад тесту, написаного з використанням бібліотеки Jest (ліст. 3.4). Цей тест перевіряє правильність генерації рядків символів відповідно до вказаного рівня складності.

Лістинг 3.4 – Тест перевірки генерації рядків

```
import { generateText } from '../utils/generateText';

describe('generateText', () => {
  test('коректно генерує текст для різних рівнів складності', () => {
    const easy = generateText('easy');
    const medium = generateText('medium');
    const hard = generateText('hard');

    expect(typeof easy).toBe('string');
    expect(easy.length).toBeGreaterThan(0);

    expect(medium).toMatch(/[A-Z.,!?]/); // має великі літери або
розділові знаки
    expect(hard).toMatch(/[0-9!@#$%^&*]/); // має цифри або спецсимволи
  });

  test('генерує різні результати при повторному виклику', () => {
    expect(generateText('easy')).not.toBe(generateText('easy'));
  });
});
```

кінець лістингу 3.4

У процесі тестування було виявлено ряд незначних недоліків у валідації вводу та обробці пробілів. Зокрема, у старих версіях браузера Safari виникали затримки при відображенні тексту. Було внесено відповідні зміни до коду, які усунули ці проблеми шляхом оптимізації DOM-операцій і використання асинхронної обробки подій.

Мінімальні системні вимоги для коректної роботи вебдодатку наступні:

– наявність сучасного веббраузера з підтримкою ES6;

– процесор не нижче Intel Core i3;

– не менше 4 ГБ оперативної пам’яті;

– стабільне інтернет-з’єднання.

Рекомендовані параметри:

– Google Chrome (версія 100 і вище);

– 8 ГБ оперативної пам’яті;

– роздільна здатність екрану не менше 1280×720 пікселів.

Таким чином, реалізоване тестування дозволило виявити і своєчасно усунути критичні помилки, підвищити стабільність та якість продукту.

Загалом, система була протестована вручну та автоматизовано, що відповідає сучасним вимогам до якості програмного забезпечення.

3.3 Інструкція користувача з розгортання вебдодатку

Після завершення розробки програмного забезпечення постає завдання завантажити сайт на хостинг. Оскільки вебдодаток написано з використанням фреймворку React, то для хостингу доцільно використати платформу Netlify.

Netlify – одна з найдосконаліших платформ для веброзробки, яка допомагає програмістам публікувати свої проекти в Інтернеті. Цей сервіс спрощує життя користувачів завдяки автоматизованим інструментам для тестування, створення та розміщення додатків і сайтів в Інтернеті.

Netlify використовує модульний підхід до розробки, допомагаючи підприємцям і програмістам забути про налаштування значної частини бекенд-аспектів і впровадження власних серверних рішень.

Для використання Netlify файли розроблюваного додатку повинні знаходитись у git-репозиторії. Щоб завантажити їх необхідно увійти в обліковий запис вебсервісу GitHub, створити репозиторій (рис. 3.9).

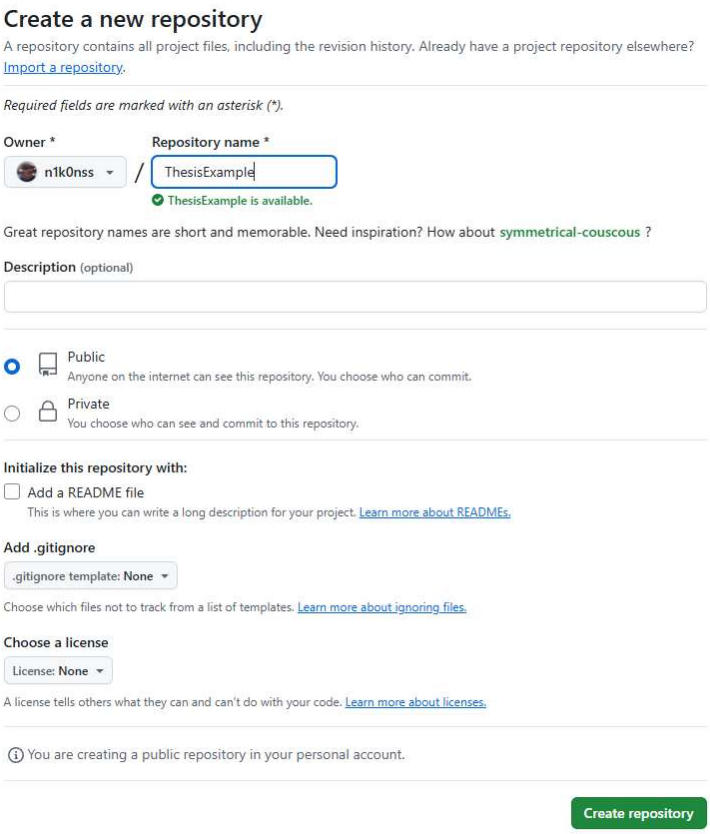

Рисунок 3.9 – Створення нового git-репозиторію

Після створення репозиторію необхідно завантажити до нього всі основні файли розробленого вебдодатку, що включають файли вихідного коду, конфігураційні файли та медіафайли (рис. 3.10).

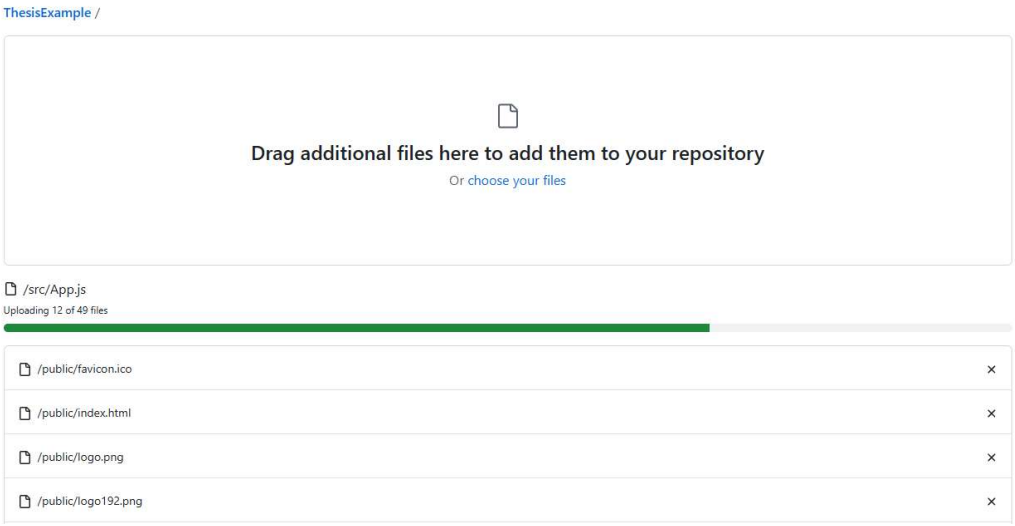

Рисунок 3.10 – Завантаження файлів проекту в репозиторій

Після завершення розробки вебдодатку всі файли проєкту були підготовлені до розгортання на хостинг-платформі Netlify, яка забезпечує просте та швидке публікування статичних сайтів і односторінкових додатків (SPA).

Першим кроком після завантаження всіх файлів є ініціалізація нового проєкту в Netlify та імпорт коду з відповідного git-репозиторію, де зберігається остання версія застосунку (рис. 3.11).

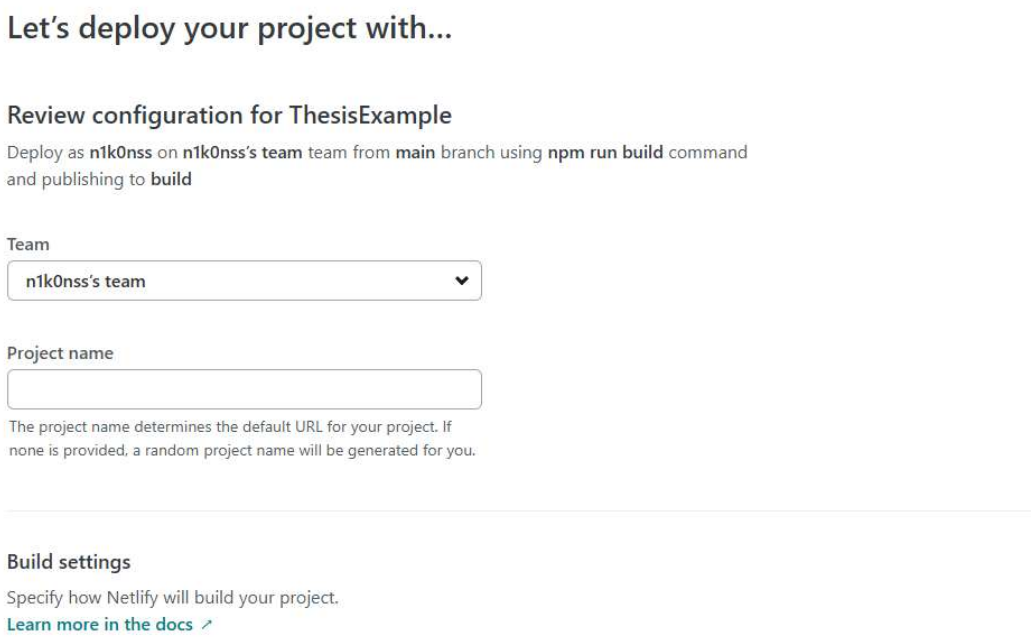

Рисунок 3.11 – Імпорт git-репозиторію в Netlify

Під час імпорту репозиторію Netlify автоматично виконує аналіз структури проєкту, встановлює необхідні залежності, налаштовує середовище збірки та створює сервер для публічного доступу до застосунку. У процесі обробки користувач має можливість вказати специфічні налаштування збірки, такі як гілка репозиторію, команда збірки та каталог для публікації.

Після успішного розгортання та обробки файлів платформа автоматично генерує посилання на готовий вебдодаток, яке доступне для будь-якого користувача через Інтернет (рис. 3.12).

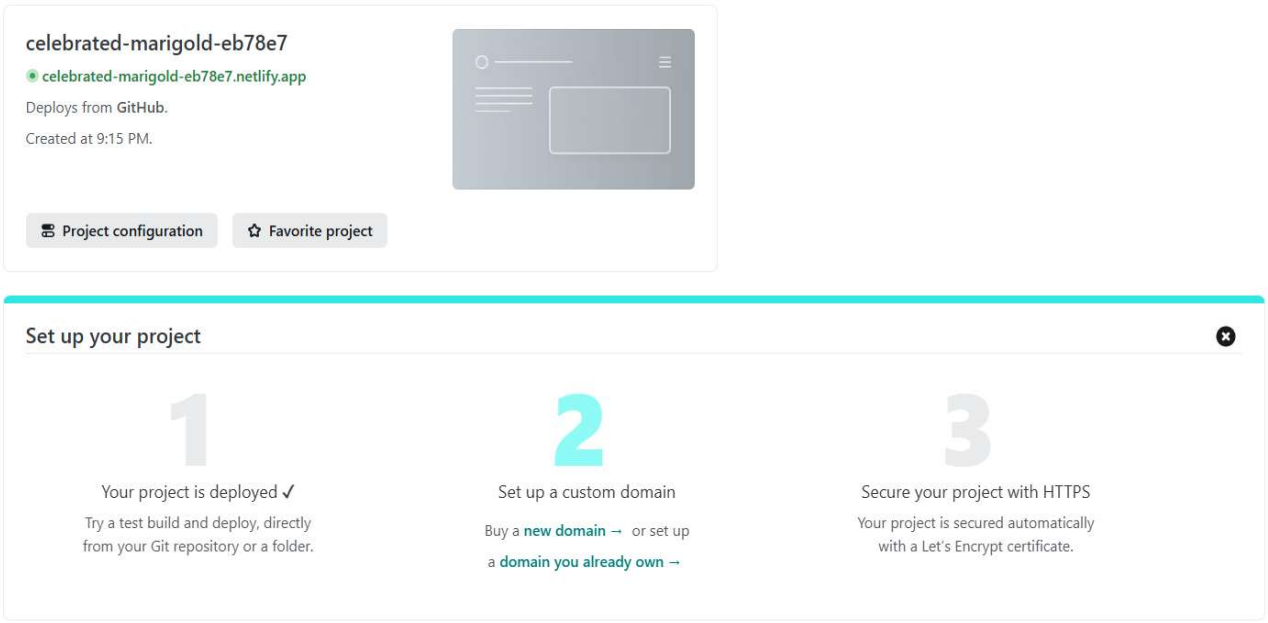

Рисунок 3.12 – Створений хостинг Netlify

За цим посиланням кінцеві користувачі мають можливість безперешкодно отримати доступ до всіх функцій вебдодатку. Інтерфейс залишається стабільним та адаптивним на різних пристроях завдяки механізмам автоматичної оптимізації, що надаються платформою Netlify. Таким чином, процес розгортання став максимально простим, швидким і безпечним, що відповідає сучасним стандартам DevOps-підходу для невеликих та середніх проєктів.

3.4 SEO інформаційно-комп'ютерної системи

Оптимізація вебдодатку для пошукових систем (SEO) є важливим етапом, який сприяє підвищенню його видимості у результатах пошуку, збільшенню відвідуваності та покращенню досвіду користувачів. У межах цього проєкту були реалізовані основні принципи технічного та контентного SEO.

Усі сторінки вебдодатку мають семантично коректну HTML-структуру. Використовуються теги <header>, <main>, <section>, <article>, <footer>, які дозволяють пошуковим системам краще інтерпретувати вміст сторінки. Заголовки структуровано за ієрархією: <h1> для основного заголовка, <h2> для

підрозділів тощо. У розділі <head> кожної сторінки наявні ключові meta-теги, зокрема:

– <meta name="description" content="Елегантний тренажер сліпого друку.">;

– <meta name="viewport" content="width=device-width, initial-scale=1.0">;

– <title>WebTyping – Елегантний тренажер сліпого друку.</title>.

Велика увага приділялася оптимізації швидкості завантаження сайту, що є важливим фактором SEO. Застосунок було протестовано за допомогою Google Lighthouse. Було досягнуто наступних показників (рис. 3.13).

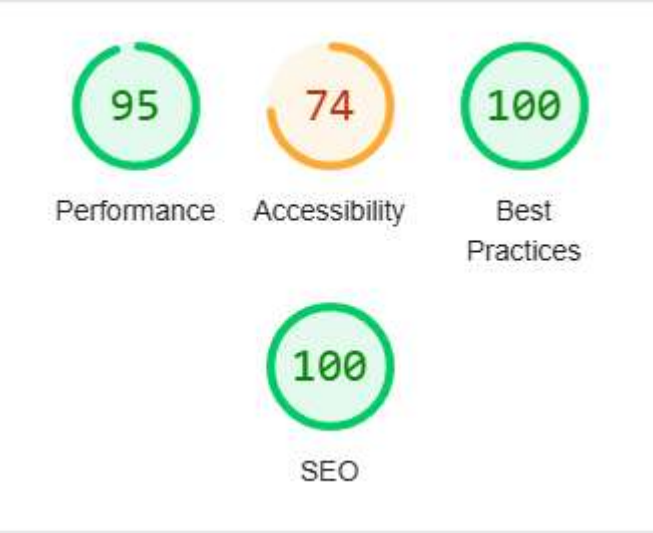

Рисунок 3.13 – Результат тестування Lighthouse

З метою оптимізації SEO також були вжиті такі заходи:

– використання семантичних тегів для контенту;

– додавання favicon, що покращує візуальне сприйняття ресурсу;

– видалення непотрібного JavaScript і стилів для пришвидшення рендерингу.

У результаті реалізації заходів SEO-оптимізації, вебдодаток повністю відповідає сучасним рекомендаціям Google з технічної оптимізації. Це дозволяє не лише покращити видимість сайту в пошукових системах, а й зробити його більш зручним, швидким і привабливим для кінцевого користувача.

3.5 Захист інформаційно-комп’ютерної системи

Інформаційна безпека є важливим аспектом розробки навіть для простих вебдодатків. У рамках створення програмного забезпечення були враховані базові принципи захисту вебдодатків, які дозволяють мінімізувати потенційні вектори атак та забезпечити безпечну взаємодію з користувачем.

Оскільки розроблюваний додаток є односторінковим застосунком без серверної логіки, основні ризики пов’язані з обробкою даних на клієнті та відображенням контенту. Було реалізовано такі заходи:

– увесь трафік здійснюється через захищене з’єднання HTTPS, що забезпечується платформою Netlify;

– у застосунку не передбачено автентифікації, що виключає ризики, пов’язані з обробкою персональних даних;

– усі дані (результати вправ, помилки, час) обробляються тільки в оперативній пам’яті браузера, що виключає передачу через мережу;

– реалізовано обробку подій клавіатури з перевіркою допустимих символів, щоб уникнути некоректного або шкідливого введення;

– відсутні динамічно вставлені HTML-елементи, що мінімізує ймовірність XSS-атак (кроссайтового скриптингу);

– усі текстові повідомлення користувачу обробляються через React, який автоматично екранує HTML, тим самим забезпечуючи захист від ін’єкцій.

У додатку також враховано налаштування політик безпеки браузера. Зокрема, для розгортання на Netlify було активовано заголовки безпеки (Security Headers), зокрема Content-Security-Policy, X-Content-Type-Options, Referrer-Policy та інші. Це дозволяє обмежити виконання скриптів та завантаження сторонніх ресурсів.

Крім того, система захищена від таких типових загроз:

– clickjacking: через заголовок X-Frame-Options: DENY;

– MIME-sniffing: через X-Content-Type-Options: nosniff;

– витік реферера: обмежено за допомогою Referrer-Policy: no-referrer-when-downgrade.

Оскільки вебдодаток не має доступу до зовнішніх баз даних, API або користувацьких сесій, ризики витоку або зміни критичних даних практично відсутні. Водночас структура коду підтримує можливість розширення із збереженням безпечної архітектури.

У майбутньому, при розширенні функціоналу (додання збереження результатів, особистого кабінету), рекомендується реалізувати:

– авторизацію через OAuth2 (наприклад, Google Sign-In);

– захист від CSRF при формуванні запитів;

– використання HTTPS-only cookies і JWT з підписом.

Таким чином, застосовані заходи захисту в додатку повністю відповідають специфіці проєкту, зберігаючи баланс між простотою розробки і вимогами безпеки.

Висновки до розділу 3

У третьому розділі описано процес розробки вебдодатку «Клавіатурний тренажер» на основі React. Реалізовано основні функціональні компоненти для різних режимів тренування, зміни мови, теми та складності. Додано музичний супровід, сповіщення про Caps Lock та інші елементи, що покращують користувацький досвід. Внутрішню логіку підтримано відповідними діаграмами діяльності та розгортання.

Проведено ручне й автоматизоване тестування, що забезпечило виявлення і усунення помилок та підвищення стабільності роботи. Здійснено SEO-оптимізацію шляхом додавання семантичної HTML-структури, meta-тегів і прискорення завантаження. Впроваджено базові засоби клієнтської безпеки – контроль введення, уникнення XSS, політики CSP. Підготовлено інструкцію з розгортання додатку на платформі Netlify, що робить застосунок зручним для кінцевого користувача.

ВИСНОВКИ

У результаті формулювання та реалізації завдань, що стосуються розробки вебдодатку «Клавіатурний тренажер» з використанням React, було закладено цілісну методологічну та технологічну основу кваліфікаційної роботи. Кожне із завдань мало на меті забезпечити послідовний перехід від аналітичного етапу до реалізації, перевірки та впровадження програмного продукту.

Першочергово було здійснено аналіз існуючих рішень, таких як «Keybr» та «Typing Speed Test», цей етап дав змогу визначити загальні тенденції у сфері цифрових освітніх продуктів для формування навичок друку, виявити недоліки наявних рішень та сформулювати власні вимоги до майбутнього додатку.

Наступним етапом стало формування функціональних і нефункціональних вимог. Було визначено такі функції як: підтримка кількох мов, режимів вправ, вибір складності, тривалості, а також вимоги до продуктивності, швидкості завантаження та адаптивності інтерфейсу.

Для систематизації логіки програми створено UML-діаграми прецедентів, послідовності, діяльності та розгортання, що відображають ключові сценарії взаємодії користувача із застосунком, структуру компонентів, їхню взаємодію та порядок виконання.

Обґрунтування вибору технологій базувалося на критеріях масштабованості, швидкодії та зручності у підтримці проєкту. Обрана бібліотека React забезпечує компонентну архітектуру та дозволяє будувати односторінкові застосунки (SPA), що значно покращує продуктивність і користувацький досвід.

Завдання щодо розробки користувацького інтерфейсу (UI) було реалізоване з урахуванням принципів UX-дизайну. У проєкті передбачено перемикання тем, чітку візуалізацію прогресу, доступність з мобільних пристроїв, а також інтуїтивно зрозумілу панель налаштувань.

Інтеграція словникової бази даних у форматі JSON дала змогу динамічно підвантажувати мовні конструкції відповідно до обраної мови, а також легко

масштабувати набір вправ без потреби у серверній інфраструктурі. Вибір структури JSON забезпечив гнучкість і зручність при оновленні словника.

Під час реалізації тренувальних сценаріїв було враховано різні стилі навчання: від фокусованого друку одного рядка до повноцінного набору речень. Користувач має можливість налаштовувати вправи, вмикати або вимикати звуковий супровід, обирати режим складності – усе це підвищує ефективність навчання.

Завдання з тестування виконано на кількох рівнях: модульне тестування логіки генерації тексту (на базі Jest), функціональне тестування взаємодії компонентів, а також візуальна перевірка адаптивності і кросбраузерної сумісності.

Підготовлено коротку користувацьку інструкцію, яка містить послідовність дій для розгортання вебдодатку на платформі Netlify, опис налаштування середовища та запуску проєкту у виробничому режимі, що забезпечує зручність користування додатком навіть для осіб без технічної підготовки.

Пошукова оптимізація (SEO) реалізована через правильну структуру HTML, додавання meta-тегів, favicon та адаптивну верстку. Застосунок проходить аудит Lighthouse із високими показниками у категоріях Performance, Best Practices та SEO.

У ході роботи реалізовано основні принципи клієнтської безпеки. Застосунок не виконує динамічне вставлення HTML-коду, що унеможливлює XSS-атаки. Також додано CSP-заголовки на рівні хостингу для обмеження зовнішніх підключень та виконання сторонніх скриптів.

Таким чином, виконання поставлених завдань дало змогу розробити повнофункціональний вебдодаток, що відповідає сучасним технічним і ергономічним вимогам. Робота має потенціал для подальшого розвитку, включаючи розширення функціоналу, адаптацію під інші мови або інтеграцію із системами онлайн-навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Flanagan D. JavaScript. The Definitive Guide. O'Reilly Media, Inc., 2020. 706 p.

2. Browser How-Tos, Help & Tips. Lifewire. URL: https://www.lifewire.com/what-is-a-web-application-3486637 (date of access: 10.02.2025).

3. Web App Architecture. Fundamentals & Design by Svitla Systems. Svitla Systems. URL: https://svitla.com/blog/web-application-architecture (date of access: 10.02.2025).

4. What is a Web App? Web Application Explained – AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/web-application (date of access: 18.02.2025).

5. Keybr.com – Typing lessons. URL: https://www.keybr.com/ (date of access: 18.02.2025).

6. Typing Test. GitHub Pages. URL: https://shreyagarwal13.github.io/typing-speed-test/ (date of access: 20.02.2025).

7. Касабона Дж. HTML and CSS: Visual QuickStart Guide. Pearson Education, Limited, 2021. 352 с.

8. Учасники проектів Вікімедіа. XHTML. Вікіпедія. URL: https://uk.wikipedia.org/wiki/XHTML (дата звернення: 05.03.2025).

9. What is CSS. Computer Hope. URL: https://www.computerhope.com/jargon/c/css.htm (date of access: 06.03.2025).

10. Phang C. L. HTML, Bootstrap, CSS, Tailwind, and Cordova. A Complete Guide. Independently Published, 2022. 155 p.

11. Stefanov S. React. Up and Running. O'Reilly Media, Incorporated, 2021. 230 с.

12. Simpson K. You Don't Know JS Yet: Get Started. Primedia eLaunch LLC, 2020. 145 p.

13. Введення у React. Krypton. URL: https://krypton.com.ua/rozdil-1-vvedennya-u-react (дата звернення: 15.03.2025).

14. Contributors to Wikimedia projects. Single-page application. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Single-page_application (date of access: 29.03.2025).

15. Narayn H. Just React! Learn React the React Way. Apress L. P., 2022. 369 c.

16. Основи JSX. Krypton. URL: https://krypton.com.ua/rozdil-1-vvedennya-u-react/osnovy-jsx (дата звернення: 02.04.2025).

ДОДАТКИ

Додаток А – Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025)

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ІТОНВ

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

УДК 004.42

ВЕБДОДАТОК ДЛЯ ТРЕНУВАННЯ СЛІПОГО ДРУКУ З ВИКОРИСТАННЯМ REACT

Умша Костянтин Сергійович
Луцький національний технічний університет, здобувач вищої освіти спеціальності ІПЗ, kostyantin.umsha@gmail.com
Ящук Андрій Анатолійович
Луцький національний технічний університет, к.т.н., доцент кафедри інженерії програмного забезпечення, a.yashchuk@lutsk-ntu.com.ua

WEB APPLICATION FOR TOUCH TYPING TRAINING USING REACT

Umsha Kostiantyn Serhiiovych
Lutsk National Technical University, higher education student in the educational program of Software Engineering, kostyantin.umsha@gmail.com
Yashchuk Andrii Anatoliiovych
Lutsk National Technical University, PhD, Associate Professor of the Department of Software Engineering, a.yashchuk@lutsk-ntu.com.ua

This paper examines the role of modern educational technologies in learning touch typing, analyzes common issues in existing systems, and presents a React-based application as an adaptive digital learning solution.

Keywords: *typing, education technology, React, SPA, motivation, interactivity.*

Навички сліпого друку залишаються одним із фундаментальних інструментів цифрової грамотності. У професійному середовищі, де швидкість обробки інформації прямо впливає на ефективність праці, вміння друкувати без погляду на клавіатуру перетворюється з додаткової переваги на необхідність. Попри це, у більшості освітніх закладів ця компетенція не закріплюється на достатньому рівні. Однією з головних причин є відсутність сучасних, зручних і персоналізованих засобів цифрового навчання.

На ринку наявні численні сервіси та застосунки для тренування друку, однак більшість із них мають низку суттєвих недоліків: морально застарілий інтерфейс, обмежений функціонал, надмірна кількість реклами або ж платний доступ до базових функцій. Часто такі інструменти не адаптовані під

сучасні освітні потреби й не враховують індивідуальні стилі навчання, а деякі навіть не мають мультимовної підтримки або не дають зворотного зв'язку в реальному часі.

З огляду на це постала ідея створити власний вебдодаток для тренування сліпого друку, який би об'єднував у собі всі сучасні вимоги до цифрових освітніх рішень: інтуїтивний інтерфейс, адаптивність, багатомовність, легкість у доступі та налаштуванні, а головне – відкритість і безкоштовність. Основою для реалізації було обрано React – JavaScript-бібліотеку, що дозволяє створювати швидкі й масштабовані односторінкові додатки. Саме це рішення забезпечило плавну роботу інтерфейсу, динамічне оновлення контенту і гнучкість при розширенні функціональності [1].

Щоб краще усвідомити переваги розробленого рішення, варто розглянути існуючі аналоги. Одним із найбільш відомих є онлайн-сервіс Keybr (рис. 1). Цей інструмент активно використовується у навчанні сліпому друку завдяки простому інтерфейсу, автоматичній генерації слів та адаптації складності залежно від рівня користувача. Keybr акцентує увагу на точності та ритмічності друку, надаючи користувачу статистику та рекомендації щодо вдосконалення навичок.

Рисунок 1 – Аналог Keybr [2]

На рисунку 2 продемонстровано інтерфейс власної розробки. У порівнянні з Keybr, дана система реалізована як

повноцінний SPA-додаток на React та орієнтована не лише на розвиток технічної швидкості друку, але й на естетичне та мотиваційне сприйняття. Підтримка темної теми, інтуїтивна навігація, мультимовність та адаптивність до різних пристроїв дозволяють користувачу відчути зручність, завдяки якій він залишається сфокусованим на самій меті – розвитку навички сліпого друку. Окрім того, додаток пропонує гнучке налаштування сценаріїв тренування, підтримку різних режимів і персоналізацію досвіду.

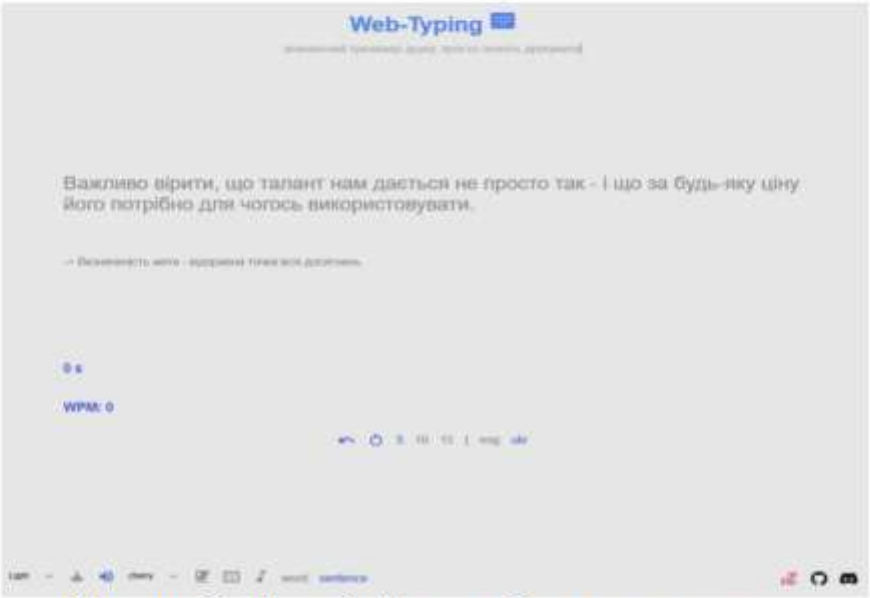

Рисунок 2 – Інтерфейс розроблюваного додатку

Перевага розробленого інструменту полягає в тому, що він не є простою симуляцією друку – це система, що адаптується до користувача. Використання словникових структур у форматі JSON дозволило реалізувати різні режими тренування – від простого набору окремих слів до повноцінних речень з розділовими знаками та різним рівнем складності. Збереження параметрів і результатів здійснюється на боці клієнта, що гарантує конфіденційність і автономність без залучення серверної частини. Крім того, для зручності користувача

реалізовано зміну теми оформлення, звуковий супровід, а також попередження про активований Caps Lock.

Важливим акцентом у розробці стало забезпечення мотиваційного ефекту. Усі елементи системи створено з урахуванням принципів UX-дизайну: легкість навігації, кольорове підсвічування помилок, індикатори прогресу, вибір режиму навчання, що дозволяє користувачу поступово збільшувати складність вправ. Відсутність рекламних блоків та обов'язкової реєстрації також позитивно впливає на концентрацію та частоту повторного використання ресурсу.

Застосунок розгорнуто на платформі Netlify [3], що спрощує доступ до нього через будь-який сучасний браузер. Було проведено ручне та автоматизоване тестування логіки генерації символів, підрахунку помилок і розрахунку швидкості набору. Тестування показало стабільну роботу інтерфейсу, незалежно від пристрою або браузера. Додатково було враховано основи пошукової оптимізації (SEO): сторінки містять структуровану HTML-розмітку, мета-теги, favicon, а також адаптовані для мобільних пристроїв.

Таким чином, створений вебдодаток є не лише інструментом для розвитку навички друку, а й прикладом того, як сучасні вебтехнології можуть ефективно вирішувати актуальні освітні задачі. Він поєднує в собі гнучкість, ефективність, доступність і простоту розгортання, що дозволяє використовувати його як частину інтерактивного освітнього середовища або як самостійний ресурс для самонавчання у будь-який зручний момент.

Список використаних джерел

1. Quick Start – React. React. URL: https://react.dev/learn (дата звернення: 11.05.2025).

2. keybr.com – Typing lessons. keybr.com – Typing lessons. URL: https://www.keybr.com/ (дата звернення: 11.05.2025).

3. Netlify Documentation. https://docs.netlify.com/ (дата звернення: 11.05.2025).

СЕРТИФІКАТ

Даний сертифікат засвідчує, що

Умша Костянтин Сергійович

брав(-ла) участь у

X Міжнародній науково-практичній конференції
з проблем вищої освіти і науки
"Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"
загальним обсягом 15 годин (0,5 кредитів ECTS),

яка проводилася 23-24 травня 2025 року у м.Луцьк
на базі Луцького національного технічного університету

Ректор ЛНТУ **Ірина ВАХОВИЧ**

Кафедра цифрових освітніх технологій

Кафедра інженерії програмного забезпечення

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

№ ІТОНВ-2025-160

Програма конференції: https://itonv.lntu.edu.ua/files/2025/program-itonv-2025.pdf