НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних систем і технологій
Професор, к.е.н., М.З. Швиденко
підпис ініціали та прізвище
16 червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

“Веб–застосунок клавіатурного тренажеру”

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми

к.е.н., доцент Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи

д.т.н., професор Смолій Вікторія Миколаївна
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав Янцевич Антон Олександрович
(підпис) (ПІБ)

КИЇВ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри інформаційних
систем і технологій
_____ / Швиденко М.З.
підпис ініціали та прізвище
__ _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи
студенту(ці) Янцевичу Антону Олександровичу
Спеціальності 126 “Інформаційні системи та технології”

1. Тема роботи: **«Веб–застосунок клавіатурного тренажеру»**

Затверджена наказом ректора від 16.12.2024 №2245С

2. Термін подання завершеної роботи на кафедру – 10.06.2025р;

3. Вихідні дані розробити веб-застосунок клавіатурного тренажеру з можливістю авторизації, тренування, збереження результатів, перегляду лідерборду;

4. Перелік питань, що розглядаються:

1. Теоретико-методологічні засади дослідження систем управління виробничими замовленнями;
2. Архітектурно-технологічні аспекти проєктування веб-системи;
3. Програмна реалізація веб-системи.

5. Календарний план

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми, планування та підготовка	16.12.2024	успішно
2.	Дослідження та збір матеріалів	01.02.2025	успішно
3.	Аналіз та написання	10.03.2025	успішно
4.	Попередній перегляд та оцінка	27.05.2025	успішно
5.	Підготовка до захисту бакалаврської роботи	10.06.2025	успішно

Керівник кваліфікаційної роботи _____ / професор, д.т.н., В.М. Смолій
підпис ПІБ, вчене звання та ступінь

Завдання прийняв до виконання _____ / А.О. Янцевич
підпис ПІБ

Дата отримання завдання 16.12.2024

РЕФЕРАТ

Тема бакалаврської кваліфікаційної роботи: “Веб–застосунок клавіатурного тренажеру”.

Автор роботи: Янцевич Антон Олександрович.

Керівник роботи: Смолій Вікторія Миколаївна.

Пояснювальна записка: 73 с., 11 рис., 6 табл., 2 дод., 10 джерел.

Графічна частина: 17 презентаційних слайдів.

ВЕБ–ЗАСТОСУНОК КЛАВІАТУРНОГО ТРЕНАЖЕРУ.

Метою є розробка веб–додатку симуляції клавіатури, який підвищує ефективність навчання швидкому та точному друку. Для досягнення цієї мети були визначені такі завдання: проаналізувати сучасні технологічні рішення розробки веб–додатків, визначити характеристики сфери навчання клавіатурі, проаналізувати потреби користувачів, розробити систему з симулятором, авторизацією, здійснити програмну та апаратну реалізацію за допомогою стеку технологій: Vite, React, TypeScript та Firebase. У кінці кваліфікаційної роботи протестувати розроблене рішення.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – (User interface), інтерфейс користувача

БД – база даних

TS – мова програмування «Typescript»

WPM – (words per minute), кількість слів, які можуть бути введені, прочитані або передані за одну хвилину

SDK – (software Development Kit), набір із засобів розробки

SQL – Structured Query Language

UID – (user identifier), ідентифікатор користувача

SPA – (single–page application), також відомий як односторінковий інтерфейс

API – (application programming interface), інтерфейс прикладного програмування

WEB – інтернет-простір

URL – (Uniform Resource Locator), єдиний вказівник на ресурс

ER – (Entity-Relationship model), модель «сутність-зв'язок»

ВСТУП

У сучасному світі цифрових технологій зростає потреба цікавих та дієвих інструментах для покращення різних навичок, а саме друку тексту, що є невід'ємною складовою як професійної діяльності, так і повсякденного життя.

Актуальність теми «Веб–застосунок клавіатурного тренажеру» відповідає потребі створення інформаційних систем, що сприяють підвищенню продуктивності користувачів та адаптації до нових вимог інформаційного простору. Розробка такого інструменту може мати потенціал для комерційного використання та розвитку в секторі інформаційних технологій.

Метою дослідження є розробка веб–додатку для симуляції клавіатури, який підвищує ефективність навчання швидкому та точному друку. Для досягнення цієї мети були визначені такі завдання: проаналізувати сучасні технологічні рішення розробки веб–додатків, визначити характеристики сфери навчання клавіатурі, проаналізувати потреби користувачів, розробити систему з симулятором, авторизацією, здійснити програмну та апаратну реалізацію за допомогою стеку технологій: Vite, React, TypeScript та Firebase. У кінці кваліфікаційної роботи протестувати розроблене рішення.

Об'єктом дослідження є процес навчання та вдосконалення навичок друку тексту за допомогою інформаційних технологій. Предметом дослідження є розробка веб–застосунку з функціоналом тренажера, лідерборду, авторизації через Google Account і електронну пошту, профілю користувача та інших готових рішень.

Розроблений веб–застосунок використовує Vite як інструмент для збірки, React для створення інтерактивного інтерфейсу, TypeScript – мова програмування та Firebase для авторизації, збереження даних, хмарної інфраструктури, набору готових функціональних рішень, хостингу. Робота включає три розділи: аналіз предметної області, проектування та впровадження системи, а також реалізація та тестування.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

У сучасних умовах цифрової трансформації значна частина щоденних ділових і освітніх процесів базується на роботі з текстовою інформацією. Однією з ключових навичок, що забезпечують ефективну взаємодію з комп'ютером, є вміння швидко та точно вводити текст за допомогою клавіатури. Ця навичка необхідна не лише в професійній діяльності (програмування, журналістика, адміністрування, офісна робота), але й у навчанні та повсякденному житті.

Навчання друку на клавіатурі охоплює сучасні онлайн–інструменти, які дозволяють в інтерактивному форматі вдосконалювати точність і швидкість набору. Інформаційні технології в цій області надають можливості для створення адаптивних, персоналізованих і статистично–аналітичних систем, що забезпечують оцінку прогресу та інтерактивність.

З огляду на тенденції розвитку веб–технологій, оптимальним підходом є розробка веб–застосунку, який доступний з будь–якого пристрою через браузер, не потребує встановлення, має адаптивний інтерфейс та зберігає дані у хмарі. Важливим компонентом є підтримка користувацьких профілів, збереження статистики, надання змагального елементу (лідерборду) та можливість обрати різні режими тренування [7].

Створення веб–застосунку клавіатурного тренажеру, буде відповідати поставленим функціональним, нефункціональним, безпековим, та іншим вимогам. Результат виконаної роботи – повністю робочий, протестований продукт.

Проєкт організований у вигляді модульної структури з розділенням компонентів інтерфейсу (TypingArea.tsx, ResultModal.tsx), сторінок (pages), сервісів (firebase.ts), стилів та ресурсів (assets), що відповідає філософії React та принципам побудови масштабованих інформаційних системи. Структурний підхід у React орієнтований на створення додатків, розбитих на незалежні

компоненти. Це спрощує розробку, оптимізує код та полегшує подальше масштабування програми.

1.2 Аналіз наявних інформаційних технологій предметної області

Для побудови сучасного веб–застосунку важливо дослідити існуючі програмні продукти з подібним функціоналом, проаналізувати їх переваги та недоліки, інтерфейси, технічні рішення та можливість адаптації до власного проєкту (табл. 1.1).

Таблиця 1.1 – Існуючі програмні продукти друку тексту

Назва продукту	Призначення	Інтерфейс	Переваги	Недоліки
Monkeytype	Професійний тренажер для швидкісного друку	Сучасний, кастомізований, підтримка тем	Великий вибір режимів, підтримка профілів, лідерборд, WPM	Закритий код
10FastFingers	Онлайн–тест на швидкість друку	Мінімалістичний	Простота використання, популярність	Немає авторизації, профілів, мобільної адаптації
Keybr	Тренажер на основі штучного тексту	Аскетичний	Генерація тексту під користувача	Обмежений функціонал, відсутність гейміфікації
Ratatype	Освітній портал для навчання друку	Інтерактивний, дитячий стиль	Курси, сертифікація	Вузька цільова аудиторія, багато реклами

Найбільш наближеною до ідеї кваліфікаційної роботи є система «Monkeytype». Цей веб–тренажер виділяється своїм простим та водночас продуманим функціоналом, дизайном і зручним інтерфейсом. Він дозволяє користувачам проходити різні режими тренування, зберігати результати, переглядати статистику. Сайт швидкий, адаптивний і приємний у використанні навіть для початківця. Тому саме цей продукт був обраний як орієнтир для подальшого аналізу та проєктування системи.

1.3 Визначення вимог до інструментальних засобів створення і використання інформаційних систем та технологій, розробка технічного завдання

Для зручнішого аналізу функціональних можливостей майбутнього веб-застосунку та формалізації вимог до системи було застосовано діаграму варіантів використання UML (Use Case Diagram). Такий підхід дозволяє наочно відобразити основні дії, які може виконувати користувач, та взаємодію з системою на концептуальному рівні.

Ця діаграма (див. рис. 1.1) допомагає виділити ключові сценарії використання, що надалі слугують основою для формування як функціональних, так і нефункціональних вимог, а також сприяє структурованому підходу до побудови архітектури та інтерфейсу застосунку.

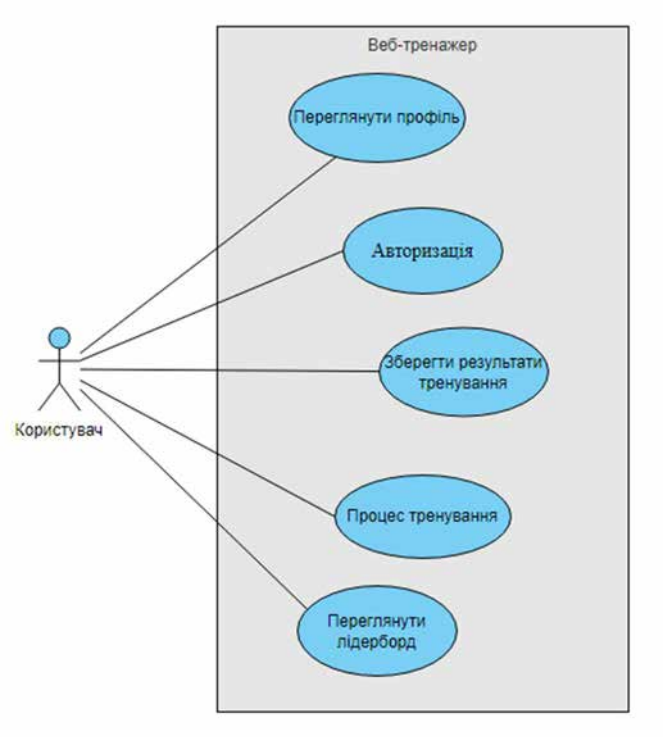

Рисунок 1.1 – Діаграма прецедентів

Загальні вимоги:

- Тип системи: веб–застосунок;
- Мета: покращення навичок друку за допомогою тренувального інтерфейсу;

- Цільова аудиторія: студенти, офісні працівники, початківці у роботі з ПК;
- Мова інтерфейсу: англійська (з можливістю локалізації).

Функціональні вимоги:

- Авторизація через Google та email/пароль;
- Вибір режиму тренування (15, 30, 60 секунд);
- Генерація текстів для тренування;
- Підрахунок WPM, точності, помилок;
- Візуальна підсвітка помилок у реальному часі;
- Лідерборд із фільтрацією за періодом (день, тиждень, всі часи);
- Профіль користувача з історією прогресу.

Нефункціональні вимоги:

- Висока продуктивність завдяки Vite;
- Надійне зберігання даних у Firebase;
- Підтримка мобільних пристроїв (адаптивна верстка);
- Безпечна робота з особистими даними;
- Швидка масштабованість проєкту.

Технологічний стек:

- React – побудова UI–компонентів;
- TypeScript – типізація, покращення якості коду;
- Firebase – автентифікація, Firestore, хостинг, функції;
- Vite – швидкий інструмент збірки;
- CSS–модулі – стилізація компонентів без inline–стилів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТВОРЕННЯ І ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

2.1 Аналіз та обґрунтування проєктних рішень для розробки веб–застосунку

На етапі проєктування інформаційної системи особливу увагу було приділено вибору архітектурної моделі, яка відповідала б сучасним вимогам до продуктивності, масштабованості, інтерактивності та зручності користування. З урахуванням специфіки задачі – створення адаптивного веб-тренажера, що працює без встановлення, зберігає результати у хмарі та надає швидкий зворотний зв’язок – було прийнято рішення реалізувати систему у вигляді односторінкового застосунку (SPA – Single Page Application) див. рис. 2.1.

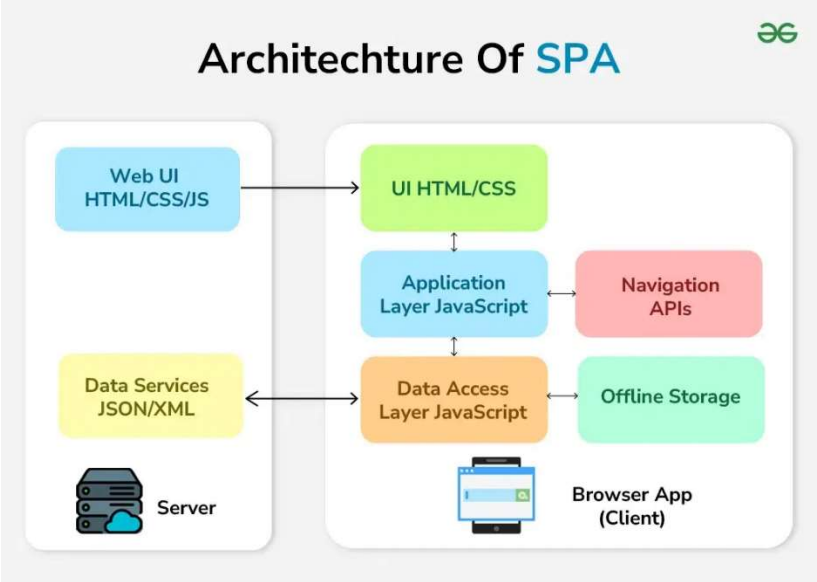

Рисунок 2.1 – Архітектура SPA

Такий тип архітектури передбачає завантаження всієї логіки застосунку на стороні клієнта, що дозволяє оновлювати контент без повного перезавантаження сторінки. Це забезпечує швидкодію, плавність взаємодії з інтерфейсом та покращений користувацький досвід.

На рис. 2.2 представлено загальну архітектуру SPA-застосунку, яка ілюструє взаємодію між основними складовими системи: інтерфейсом користувача, логікою обробки даних, сервісами авторизації та хмарною базою даних. Обрана архітектура дозволяє ефективно організувати роботу клієнтської

частини застосунку, централізовано керувати станом, маршрутизацією та доступом до даних.

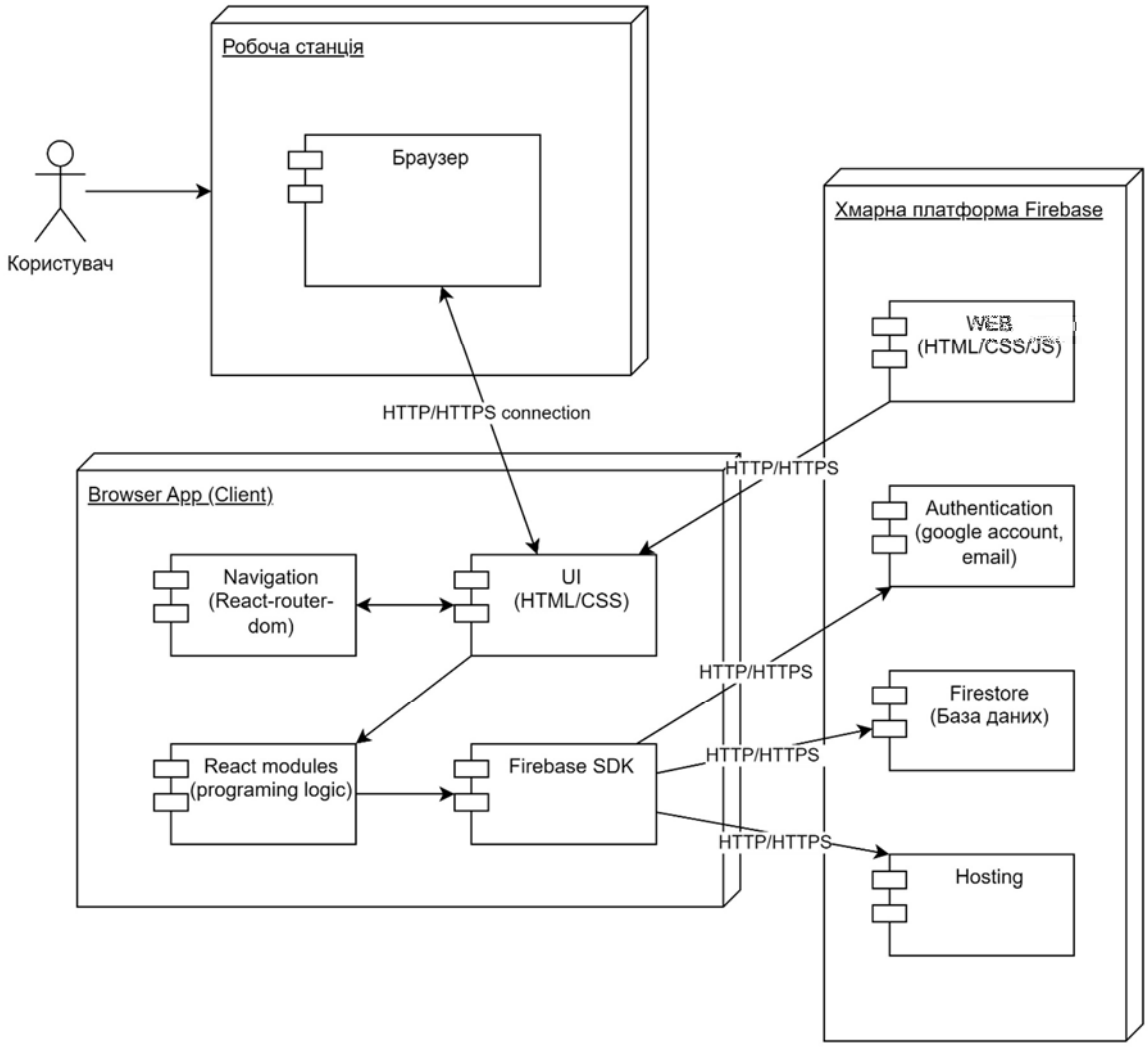

Рисунок 2.2 – архітектура клавіатурного тренажеру

Для побудови цієї схеми було використано діаграму розгортання (англ. deployment diagram), яка відображає фізичне розміщення компонентів програмної системи на вузлах (наприклад, браузер користувача та хмарна інфраструктура Firebase) і взаємозв’язки між ними. Такий тип діаграми дозволяє наочно представити взаємодію між клієнтом і серверною частиною та демонструє, як реалізується доступ до ресурсів через веб–технології.

Для побудови користувацького інтерфейсу обрано бібліотеку React, що базується на компонентному підході та дозволяє створювати масштабовані інтерфейси з повторно використовуваними елементами. У поєднанні з TypeScript

забезпечується сувора типізація, що знижує кількість помилок у коді на етапі розробки та полегшує підтримку проєкту.

Зберігання та обробка даних реалізовані на основі хмарної платформи Firebase, яка надає набір готових інструментів: автентифікацію, NoSQL–базу даних (Firestore), хостинг, систему авторизації та аналітики. Це дозволяє уникнути необхідності розгортання окремого серверного середовища та спрощує масштабування проєкту[4,8].

Враховуючи важливість швидкодії при роботі з інтерфейсом, для збірки застосунку використано Vite – сучасний інструмент розробки, який забезпечує миттєвий запуск, гаряче оновлення модулів і мінімальні затримки при розробці.

Стисло, обрані проєктні рішення: React + TypeScript + Firebase + Vite – є сучасним і продуктивним стеком для реалізації задач, поставлених у технічному завданні. Вони забезпечують швидкість розробки, масштабованість, кросплатформеність і відповідність сучасним стандартам веб–розробки.

2.2 Розробка логічної та фізичної структури системи

Логічна структура системи визначає, які компоненти складають інформаційну систему, які функції вони виконують і як взаємодіють між собою. Фізична структура відображає реальну реалізацію цих компонентів у вигляді програмних модулів, сервісів і зв’язків між ними.

У контексті веб–застосунку клавіатурного тренажера логічну структуру можна подати як сукупність таких основних компонентів:

- Інтерфейс користувача (UI) – забезпечує взаємодію з користувачем, відображає текст для друку, обробляє натискання клавіш, виводить результати та статистику;
- Модуль автентифікації – відповідає за вхід користувачів через Google або email/пароль;
- Модуль збереження та обробки статистики – підраховує WPM, точність, помилки, правильні/неправильні символи;

- Модуль історії тренувань – зберігає результати користувача в базі даних та формує лідерборд;
- База даних (Firestore) – зберігає профілі користувачів, результати, режим тренування, час та інші мета–дані.

Фізична структура системи реалізована у вигляді SPA–додатку на основі React. Компоненти логічної структури втілені у відповідних програмних модулях (див. табл. 2.1).

Таблиця 2.1 – Програмні модулі

Компонент	Реалізація	Призначення
UI	TypingArea.tsx, ResultModal.tsx	Відображення тренажера, модальних вікон
Авторизація	firebase.ts, AuthForm.tsx	Google/email-реєстрація
Обробка даних	TypingArea.tsx	Підрахунок WPM, точності, помилок
Профіль	Profile.tsx, UserInfo.tsx	Виведення й редагування даних користувача
БД	Firestore (колекції users, results)	Збереження результатів і профілів

Комунікація між клієнтською частиною та базою даних здійснюється через Firebase SDK, що дозволяє уникнути проміжного бекенду та забезпечує безпечну аутентифікацію, авторизацію й доступ до даних.

Вибір модульної структури та розділення обов’язків між компонентами підвищує супровід і масштабованість застосунку. Кожен модуль може бути протестований і вдосконалений окремо, що відповідає принципам розробки сучасних інформаційних систем.

2.3 Визначення підсистем та їх взаємодії

У структурі веб–застосунку клавіатурного тренажеру можна виокремити декілька логічних підсистем, кожна з яких виконує окремий набір функцій та відповідає за певну частину життєвого циклу роботи користувача із системою. Такий підхід дозволяє забезпечити модульність, спрощене тестування, майбутню підтримку й масштабування.

Основні підсистеми системи:

a) Підсистема автентифікації:
   1) Призначена для реєстрації та входу користувача до системи;
   2) Реалізована з використанням Firebase Authentication;
   3) Підтримує Google–авторизацію та email/пароль;
   4) Забезпечує захист доступу до персональних даних користувача;

b) Підсистема тренування:
   1) Ядро веб–застосунку;
   2) Відповідає за виведення тексту, підсвітку символів, обробку натискань клавіш;
   3) Формує метрики швидкості друку (WPM), точності, кількості правильних/неправильних символів;
   4) Реалізована у вигляді компонента TypingArea.tsx;

c) Підсистема обробки та збереження статистики:
   1) Після завершення тренування обчислює статистику;
   2) Результати передаються у Firestore, з прив'язкою до UID користувача;
   3) Статистика потім використовується для формування лідербордів та профілю;

d) Підсистема профілю користувача:
   1) Відповідає за відображення та редагування особистих даних (displayName, аватар);
   2) Виводить історію результатів тренувань;
   3) Дозволяє змінити ім’я раз на 7 днів;
   4) Реалізована через компоненти Profile.tsx і UserInfo.tsx;

e) Підсистема лідерборду
   1) Забезпечує ранжування результатів користувачів за різними фільтрами (час, складність, тривалість);
   2) Витягує дані з колекції results у Firestore та групує їх у таблиці;
   3) Сортує користувачів за WPM та точністю.

Для детальнішого відображення внутрішньої логіки роботи підсистем було використано діаграму діяльності UML (activity diagram). Такий тип діаграми дозволяє наочно змоделювати поведінку системи у відповідь на дії користувача, описуючи послідовність виконання дій, умови переходу між ними та точки завершення процесу.

На рис. 2.3 представлено діаграму активності підсистеми тренування, яка демонструє ключові етапи взаємодії користувача із системою під час набору тексту: від початку сесії, запуску таймера, посимвольного введення та перевірки коректності – до підрахунку метрик (WPM, точність) і завершення тренування. Ця діаграма дозволяє краще зрозуміти внутрішню логіку підсистеми тренування.

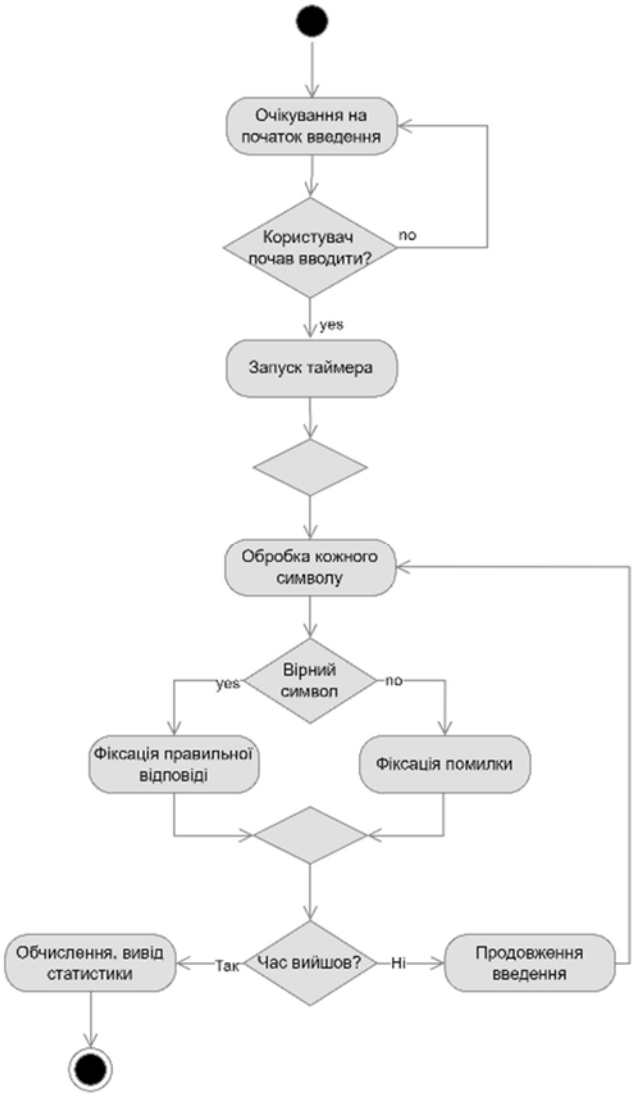

Рисунок 2.3 – Діаграма діяльності тренування

На рис. 2.4 наведено діаграму діяльності підсистеми автентифікації, що демонструє основні сценарії авторизації користувача – через Google або електронну пошту з паролем. Діаграма відображає логіку переходів залежно від вибраного способу входу, із наступною реєстрацією або входом до системи та перенаправленням користувача до тренувального модуля.

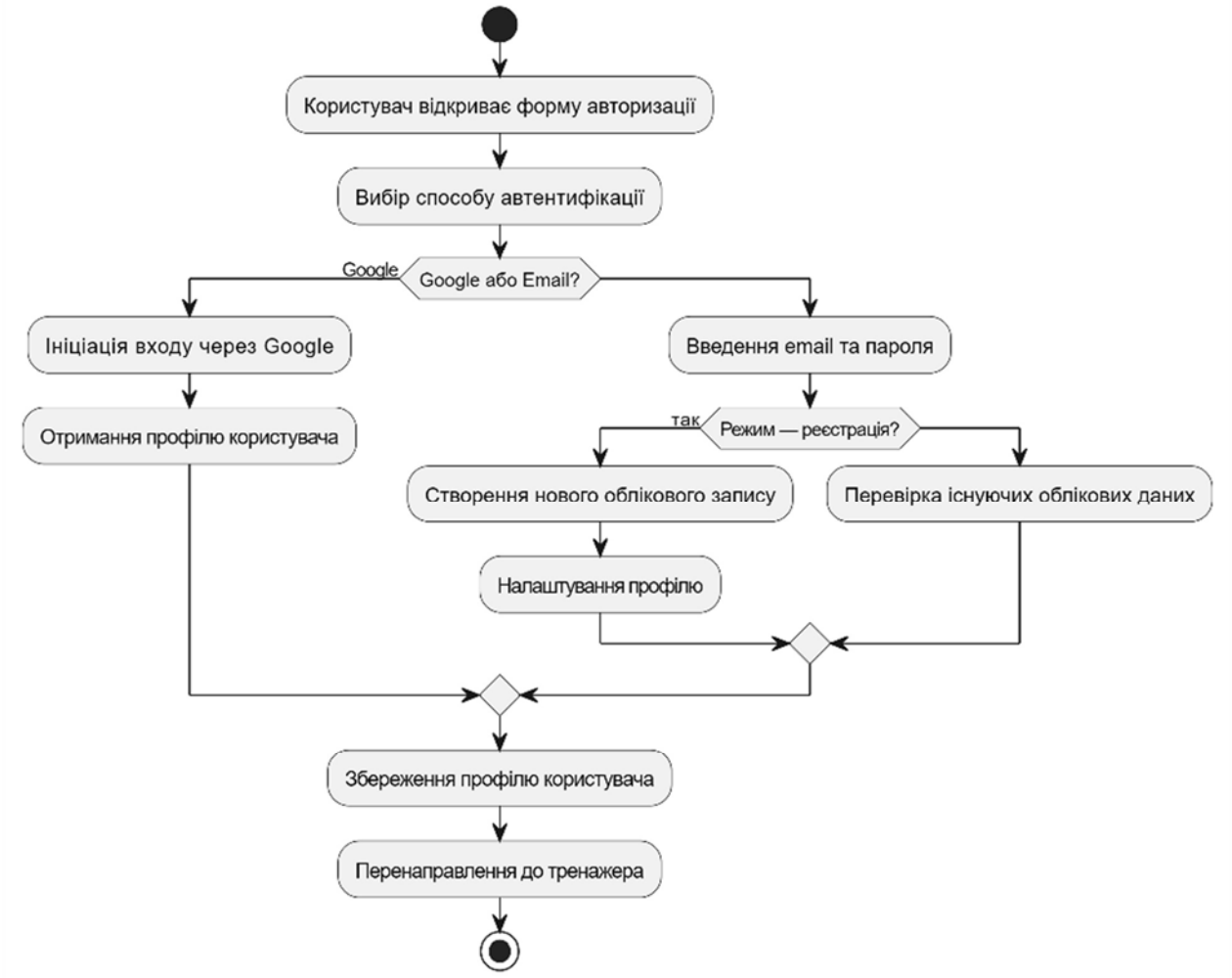

Рисунок 2.4 – Діграма діяльності авторизації

Всі підсистеми об’єднані у єдину клієнтську SPA–архітектуру, з якою взаємодіє користувач через веб–інтерфейс. Основний спосіб зв’язку між підсистемами – контекст React, передача props та централізоване зберігання інформації в Firebase.

Загальна схема взаємодії виглядає так:

- Користувач проходить автентифікацію;
- Успішна авторизація відкриває доступ до підсистеми тренування;

- Після проходження тесту дані передаються в підсистему збереження статистики;
- Підсистема лідерборду та профілю відображає ці дані в зручному форматі.

Такий підхід дозволяє уникнути дублювання логіки, забезпечити розмежування відповідальностей і досягти високої гнучкості застосунку.

2.4 Декомпозиція підсистем на модулі

Одним із ключових етапів розробки веб–застосунку є декомпозиція системи на окремі програмні модулі. Це дозволяє реалізувати принципи розділення обов'язків, повторного використання коду та спрощення майбутньої підтримки проєкту. Для реалізації клавіатурного тренажеру було обрано структуру, яка базується на сучасних підходах у React–розробці та відповідає архітектурі компонентно–орієнтованих SPA–додатків [1,6,9].

Загальна логіка побудови модулів передбачає розподіл функціональності за відповідними підсистемами, кожна з яких реалізує свою частину логіки та взаємодіє з іншими модулями через API, контекст або параметри компонентів.

Підсистема автентифікації відповідає за ідентифікацію користувача перед початком роботи з системою. Вона включає в себе такі компоненти:

- AuthForm.tsx – компонент, що забезпечує вхід і реєстрацію користувача за допомогою Google або email/паролю. Він містить валідацію, обробку помилок і зміну інтерфейсу залежно від режиму (вхід/реєстрація);
- firebase.ts – конфігураційний файл, у якому ініціалізується Firebase App і експортуються сервіси Firebase: аутентифікація, база даних, хостинг;
- firestore.ts – сервісний модуль, що містить функції для збереження та оновлення профілю користувача, зокрема його імені та аватара.

Центральна підсистема застосунку, яка реалізує безпосередній функціонал набору тексту, обробку введення та розрахунок результатів:

- TypingArea.tsx – ключовий модуль, що виводить текст для набору, підсвічує правильні/неправильні символи, опрацьовує натискання клавіш, відслідковує час і ініціює підрахунок статистики;
- ResultModal.tsx – модальне вікно, яке з'являється після завершення тренування. Воно відображає результати: WPM, точність, кількість набраних символів, а також дозволяє зберегти результат або розпочати тренування знову;
- Trainer.tsx – сторінка, яка відповідає за генерацію випадкового тексту, обрання режиму та передачу параметрів у TypingArea.

Підсистема профілю користувача дозволяє користувачу переглядати та редагувати особисті дані, зокрема ім'я та аватар:

- Profile.tsx – сторінка з інформацією про користувача та результатами тренувань.
- UserInfo.tsx – компонент, що виводить дані поточного користувача, дозволяє вийти з акаунту, а також один раз на тиждень змінити відображуване ім'я.

Для реалізації змагального елементу проєкту створено підсистему лідерборду, яка дозволяє переглядати найкращі результати користувачів за фільтрами:

- Leaderboard.tsx – сторінка, що формує таблицю лідерів на основі даних з Firestore;
- leaderboard.css – окремий файл стилів для таблиці результатів, що забезпечує її адаптивне відображення.

Для реалізації єдиної навігації між сторінками використовуються:

- NavBar.tsx – головне меню, реалізоване за допомогою react–router–dom, забезпечує переходи між головною, профілем, тренуванням і лідербордом;
- App.tsx – основний компонент маршрутизації, де через BrowserRouter задаються шляхи до сторінок (Home, Trainer, Profile, Leaderboard);

- main.tsx – точка входу в застосунок, де ReactDOM монтує головний компонент у DOM.

Статичні файли: index.html, favicon.ico зберігаються у папках public і assets.

CSS–файли: index.css, reset.css, typing.css, ResultModal.css, leaderboard.css забезпечують розділену стилізацію кожного модуля відповідно до їх функціонального призначення.

Конфігураційні файли: .env, firebase.json, vite.config.ts, tsconfig.json, firestore.rules – налаштовують середовище, правила доступу до бази та збірку застосунку.

Таким чином, декомпозиція застосунку на модулі забезпечує високу гнучкість архітектури, дозволяє працювати з кожним компонентом незалежно, а також спрощує розширення функціональності у майбутньому. Розділення на функціональні блоки дозволяє ефективно реалізувати складну логіку веб–застосунку із збереженням чистоти коду.

2.5 Проєктування інтерфейсу користувача

Проєктування інтерфейсу є одним із ключових етапів створення ефективної інформаційної системи, орієнтованої на кінцевого користувача. Інтерфейс користувача (UI, від англ. *User Interface*) повинен бути інтуїтивно зрозумілим, візуально привабливим, адаптивним до різних типів пристроїв та забезпечувати повну взаємодію користувача з основними функціональними можливостями системи. У контексті веб–застосунку клавіатурного тренажеру проєктування інтерфейсу набуває особливого значення, оскільки безпосередній процес навчання швидкому друку пов’язаний із візуальною стимуляцією, миттєвим зворотним зв’язком та динамічною зміною стану застосунку.

Інтерфейс системи реалізований за допомогою бібліотеки React із дотриманням компонентного підходу. Основні елементи UI згруповано в окремі

модулі у директорії components/, а повноцінні сторінки користувацького інтерфейсу – у Pages/. Нижче описані ключові компоненти інтерфейсу:

- NavBar.tsx – глобальна панель навігації, що забезпечує перемикання між основними сторінками (головна, тренажер, профіль, лідерборд). Реалізована через react–router–dom, використовує інтуїтивні назви розділів і адаптивне меню;
- TypingArea.tsx – центральний елемент інтерфейсу тренажера. Виводить згенерований текст, підсвічує правильні/неправильні символи в режимі реального часу, відображає таймер і зони для вводу. Є ядром UI–підсистеми навчання;
- ResultModal.tsx – модальне вікно з результатами, яке з'являється після завершення сесії. Відображає WPM, точність, кількість помилок, набраних символів, а також кнопки для повтору чи виходу;
- AuthForm.tsx – форма автентифікації/реєстрації з можливістю входу через Google або email/пароль. Має вкладки для перемикання режимів, валідацію полів, відображення помилок;
- UserInfo.tsx – компонент профілю користувача. Відображає ім’я, аватар, UID, дату останньої активності, а також дозволяє змінювати ім’я (раз на 7 днів) або вийти з акаунту;
- Leaderboard.tsx – таблиця лідерів з результатами користувачів. Має фільтрацію за періодом (день, тиждень, усі часи), сортування за WPM, точністю та підтримку адаптивного відображення.

Усі елементи інтерфейсу поєднані за допомогою маршрутизації React (App.tsx) та внутрішніх станів компонентів (useState, useEffect). Зовнішні дані (результати, профіль, лідерборд) підтягуються через Firebase SDK та обробляються у відповідних компонентах. Комунікація між модулями відбувається через:

- передачу параметрів (props);
- глобальний контекст аутентифікації (React Context API);

- обробники подій (callback–функції для переходу між станами інтерфейсу).

Особливу увагу приділено адаптивності інтерфейсу – компоненти оптимізовано для коректного відображення на десктопах, планшетах і мобільних пристроях. Верстка здійснюється з використанням CSS–модулів (index.css, typing.css, ResultModal.css тощо), що дозволяє ізолювати стилі окремих компонентів і уникнути конфліктів.

При створенні інтерфейсу застосовано принципи:

- Мінімалізм – прості шрифти, обмежена палітра кольорів (контраст кольорів для тексту), мінімум візуального шуму;
- Зворотний зв’язок у реальному часі – миттєве підсвічування помилок, анімації;
- Інтуїтивність – користувач бачить одразу, що потрібно робити, навіть без інструкцій;
- Гейміфікація – рейтинг, лідерборд, WPM та інші метрики мотивують на повторення вправ;
- Доступність – підтримка клавіатурного керування, помірні розміри елементів, відсутність дрібного тексту.

Кожен елемент інтерфейсу безпосередньо пов’язаний із функціональними вимогами системи (див. табл. 2.2)

Таблиця 2.2 – Компоненти та їх функції

Компонент	Функція
TypingArea	Реалізація тренування, облік натискань, підсвітка
ResultModal	Аналіз результату, мотиваційна функція
AuthForm	Вхід, реєстрація, збереження UID у Firebase
Leaderboard	Гейміфікація, статистика, соціальна взаємодія
Profile / UserInfo	Виведення та зміна особистих даних, управління сесією

Таке проєктне рішення забезпечує логічну відповідність між цілями системи та засобами їх реалізації. Інтерфейс користувача є ключовою складовою веб–застосунку клавіатурного тренажера, оскільки саме через нього користувач взаємодіє з системою, отримує візуальний фідбек і керує процесом навчання. Проєктування UI здійснюється з урахуванням принципів сучасної frontend–розробки, з використанням React–компонентів, адаптивної верстки, модульної стилізації та гнучкого управління станом. Результатом є інтерфейс, який є зручним, привабливим, адаптивним та логічно пов’язаним із усіма підсистемами тренажеру.

2.6 Розробка об’єктної моделі та структури бази даних

Проєктування об’єктної моделі та структури бази даних є важливим етапом у створенні інформаційної системи, що забезпечує ефективне зберігання, обробку, фільтрацію й аналіз даних, необхідних для реалізації функціоналу програмного продукту. У межах розробки веб–застосунку клавіатурного тренажера проєктування бази даних дозволяє забезпечити підтримку збереження користувацьких профілів, результатів тренувань, налаштувань, рейтингу та історичних даних у масштабованій, надійній та уніфікованій формі.

У сучасній практиці веб–розробки широко використовуються об’єктно–орієнтовані моделі даних, які забезпечують зручне відображення сутностей прикладної предметної області у вигляді об'єктів зі властивостями та зв’язками. Враховуючи це, в основі даного веб–застосунку закладено принципи об’єктного підходу до побудови логічної моделі бази даних, а також застосування безсхемної документо–орієнтованої бази Firebase Cloud Firestore, яка забезпечує підтримку гнучкої структури документів та колекцій без жорстких обмежень на типи або структуру збережених даних.

Об’єктна модель системи базується на двох основних сутностях: користувач та результат тренування. Ці сутності є ядром предметної області і забезпечують збереження персональних даних користувача та результатів його

взаємодії з системою. Кожен користувач є унікальним, має відповідні атрибути (ідентифікатор, ім’я, email, фото) та асоціюється з множиною результатів тренувань, які формуються після завершення кожної сесії роботи з тренажером. До основних атрибутів сутності «Користувач» належать:

- унікальний ідентифікатор (UID), що надається системою автентифікації;
- ім’я для відображення в інтерфейсі;
- електронна адреса;
- URL аватару користувача.

Сутність «Результат» охоплює:

- метрики швидкості введення (WPM – слів на хвилину);
- точність введення (% коректних символів);
- загальна кількість набраних символів;
- кількість правильних та помилкових натискань;
- режим тренування (легкий, середній, складний);
- тривалість сесії (у секундах);
- часову позначку збереження результату.

Між сутностями встановлено зв'язок типу «один–до–багатьох»: один користувач може мати багато результатів, кожен з яких зберігається окремо, що дозволяє здійснювати аналіз, формування статистики, побудову графіків прогресу та лідерборду.

На рис. 2.5 представлено загальну структуру бази даних, для цього було використано ER-модель (діаграма сутність–зв’язок), яка відображає ключові сутності системи, їх атрибути та тип зв’язку між ними.

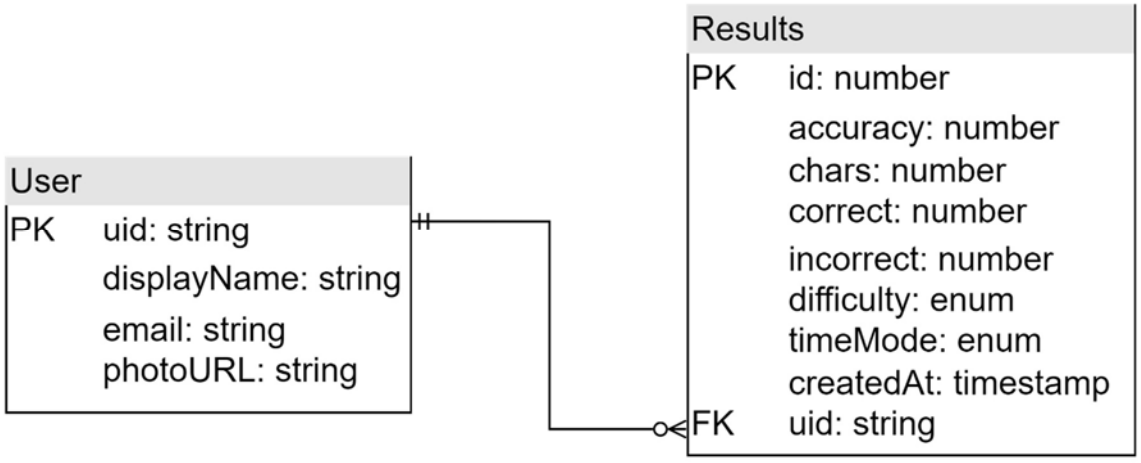

Рисунок 2.5 – ER-модель застосунку

Архітектура бази даних реалізована на основі ієрархії колекцій і документів, характерної для Firestore. Основними колекціями є:

- users – містить документи користувачів, де ключем є UID;
- results – окрема колекція, де кожен документ містить детальну інформацію про одну сесію тренування та пов’язаний з UID користувача.

Кожен документ у колекції users зберігає дані, які не змінюються часто, зокрема, дані облікового запису, що використовуються в інтерфейсі та для авторизації. У колекції results документи створюються після кожного проходження вправи, що дає змогу накопичувати статистику, визначати середні значення, виявляти прогрес та будувати загальні рейтинги.

Гнучкість структури бази Firestore дозволяє у майбутньому доповнювати модель новими колекціями або вкладеними підколекціями, зокрема:

- settings – для зберігання індивідуальних налаштувань користувача;
- achievements – для реалізації системи нагород;
- feedback – для збору оцінок і коментарів щодо якості тренажера.

Вибір Firestore як основного інструменту для зберігання даних ґрунтується на кількох ключових перевагах:

- повна інтеграція з Firebase Authentication, що спрощує реалізацію системи прав доступу;

- масштабованість – підтримка великої кількості одночасних клієнтів без зниження швидкодії;
- можливість роботи в реальному часі, що є критично важливим для оновлення рейтингу та статистики;
- відсутність необхідності у розгортанні власного серверного ПЗ, що спрощує обслуговування та знижує витрати.

Використання NoSQL–підходу дозволяє зберігати документи з різною структурою, що важливо при динамічному розширенні функціоналу проєкту. Це дає змогу зберігати тренувальні сесії в зручному форматі, гнучко фільтрувати й сортувати їх, а також легко впроваджувати додаткові властивості без змін у загальній архітектурі системи.

Розроблена об’єктна модель та структура бази даних повністю відповідає вимогам до сучасного веб–застосунку з функціоналом персоналізації, відстеження прогресу та рейтингової системи. Об’єктна модель базується на принципах логічної узгодженості, розширюваності та незалежності сутностей, а вибір Firebase Cloud Firestore забезпечує високу продуктивність, надійність і простоту впровадження. Структура бази є масштабованою, гнучкою та адаптованою до подальшого розширення функціоналу. У наступному етапі реалізації ця модель буде трансформована у відповідні програмні конструкції із застосуванням відповідних SDK-інструментів.

2.7 Вибір методів та алгоритмів для реалізації функціоналу

Розробка функціональної частини веб–застосунку вимагає ретельного підбору алгоритмів і методів, які забезпечують точність, надійність, швидкодію й масштабованість програмної системи. Важливим завданням на етапі проєктування є вибір найбільш доцільних рішень для реалізації таких функцій, як облік користувацької активності, оцінка результатів друку, сортування даних у лідербордах, обробка натискань клавіш, валідація введених значень та забезпечення інтерактивності у реальному часі.

Однією з ключових функцій системи є відслідковування дій користувача під час тренування, зокрема, натискань клавіш, посимвольного порівняння з еталонним текстом, підрахунку правильних і помилкових символів. Для реалізації цього застосовано подієву модель обробки введення (*event–driven input processing*), притаманну середовищам браузера. Введений символ миттєво порівнюється з відповідним символом рядка, після чого оновлюється стан інтерфейсу (візуальна підсвітка), а у внутрішній структурі даних зберігається результат порівняння.

Перевага такого підходу полягає у високій швидкодії й простоті реалізації. Альтернативою могло би бути попереднє збереження всього тексту у внутрішню структуру з наступним постобробленням, однак це ускладнює інтерфейсну логіку і знижує інтерактивність. Для вимірювання швидкості введення використовуються стандартні метрики:

- WPM (words per minute) – визначається як співвідношення кількості введених символів до 5 (умовна довжина слова), поділене на тривалість тренування у хвилинах;
- Accuracy (точність) – обчислюється як відсоткове відношення правильно введених символів до загальної кількості натискань.

Для їх обчислення використано прямі арифметичні операції, що не потребують складних алгоритмів, проте важливою є обробка виняткових випадків (наприклад, поділ на нуль або неповні введення), що враховано при проєктуванні.

Підсистема лідерборду реалізує динамічне формування таблиці найкращих результатів користувачів залежно від трьох основних критеріїв: тривалість сесії (15, 30 або 60 секунд), рівень складності (easy, medium, hard) та період часу (остання доба або останній тиждень).

Для цього буде використано механізми фільтрації безпосередньо на рівні запитів до бази даних Firestore, що дозволяє зменшити обсяг переданих даних і прискорити роботу клієнта. Зокрема, у запитах застосовуються такі методи:

- where(...) – для відбору результатів за полями timeMode, difficulty та createdAt;
- Timestamp.fromDate(...) – для визначення часової межі відповідно до обраного діапазону;
- orderBy('createdAt', 'desc') – для сортування результатів за датою додавання;
- limit(50) – для обмеження обсягу вибірки до 50 найновіших записів.

Отримані дані вже є відфільтрованими й відсортованими, що дозволяє уникнути додаткової обробки на стороні клієнта. Це особливо важливо для забезпечення високої швидкодії, коли кількість результатів у базі зростає.

Таким чином, реалізація сортування та фільтрації через механізми Firestore запитів відповідає сучасним практикам оптимізації роботи з хмарними NoSQL–сховищами та дозволяє досягти гнучкості в управлінні даними без ускладнення логіки інтерфейсу користувача.

2.8 Аналіз технологічного стеку та обґрунтування його використання

Для ефективної реалізації веб–застосунку клавіатурного тренажера було обрано сучасний і добре скомпонований технологічний стек, що поєднує засоби клієнтської розробки, інструменти типізації, хмарну інфраструктуру, модульну збірку та засоби керування стилями. Вибір кожного інструменту є результатом ретельного аналізу вимог до продуктивності, масштабованості, гнучкості у розробці та інтеграції з іншими компонентами системи.

Основні компоненти технологічного стеку:

а) React – JavaScript–бібліотека для створення інтерфейсів користувача. Обрана через:
   1. компонентно–орієнтовану архітектуру, що дозволяє будувати масштабовані UI;
   2. підтримку стану, маршрутизації та ефективного повторного рендерингу;
   3. активну спільноту, багатий екосистемний простір (React Router, Context API, Hooks);

4. просту інтеграцію з Firebase та іншими сторонніми сервісами.

b) TypeScript – надмножина JavaScript із статичною типізацією [2,3,10]. Впроваджена для: підвищення надійності коду; попередження помилок на етапі компіляції; покращення читаємості та автодоповнення в IDE; забезпечення масштабованості проєкту у міру його зростання.

c) Firebase (Firestore, Authentication, Hosting) – хмарна платформа Google для розробки додатків. Використовується як основа бекенду, що охоплює:
   1. автентифікацію користувачів (через Google або email/пароль);
   2. зберігання структурованих даних у Firestore;
   3. безпечне розгортання проєкту на Firebase Hosting;
   4. масштабованість, високу доступність і безперебійну інтеграцію з фронтендом.

d) Vite – сучасний інструмент збірки для фронтенд–проєктів. Обраний замість Webpack через: миттєвий запуск проєкту; швидку обробку змін у файлах; просту конфігурацію й підтримку TypeScript «із коробки».

e) CSS – модулі – підхід до стилізації компонентів, який передбачає ізоляцію стилів на рівні файлу. Застосовується для:
   1. запобігання конфліктам між класами;
   2. зручного повторного використання стилів;
   3. організованого зберігання CSS у відповідності до компонента.

Обраний стек повністю відповідає архітектурі SPA (Single Page Application), що була визначена як основа системи. React у поєднанні з React Router забезпечує маршрутизацію без перезавантаження сторінок, а контекст та хуки дозволяють централізовано керувати станом. Firebase служить як повноцінний безсерверний бекенд, що ідеально підходить для SPA–додатків, а Vite гарантує швидку та зручну розробку.

Цей стек дозволяє: реалізувати адаптивний інтерфейс; зберігати результати та профілі у реальному часі; швидко масштабувати систему за потреби; захищати дані користувача відповідно до вимог конфіденційності.

Для легшого відображення та розуміння, проведений аналіз інструментів було винесено у табл. 2.3.

Таблиця 2.3 – Аналіз інструментів

Завдання	Обраний інструмент	Можлива альтернатива	Причини відмови
Створення UI	React	Angular, Vue	Angular – складний у входженні; Vue – менш поширений у середовищі Firebase
Типізація	TypeScript	JavaScript	JS – динамічний, не дає гарантій типів
Бекенд	Firebase	Express.js + MongoDB	Потребує сервера, складна настройка, менша інтеграція
Збірка проєкту	Vite	Webpack, Parcel	Webpack – повільніший, потребує більше конфігурації
Стилізація	CSS–модулі	SCSS, Tailwind CSS	SCSS – важчий у масштабуванні, Tailwind – потребує окремого підходу

Як видно з табл. 2.3, обрані інструменти є оптимальними для невеликого, проте функціонального веб–застосунку, що має обмежені ресурси на розгортання, але високі вимоги до інтерактивності та простоти підтримки.

Проєктна реалізація веб–застосунку клавіатурного тренажера ґрунтується на сучасному, легкому й масштабованому технологічному стеку, який охоплює фронтенд (React + TypeScript), хмарну інфраструктуру (Firebase) та інструменти розробки (Vite, CSS–модулі). Обрані засоби забезпечують відповідність технічному завданню, прискорюють розробку, покращують зручність супроводу та дозволяють реалізувати повний життєвий цикл продукту – від створення до тестування й розгортання.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Опис модулів та їх функціонального призначення

Розроблений веб–застосунок реалізований відповідно до обраної архітектури односторінкового застосунку (SPA), що забезпечує динамічну зміну вмісту без перезавантаження сторінки. Основою реалізації є модульна структура, що передбачає поділ функціоналу на незалежні компоненти. Кожен модуль виконує окрему роль і взаємодіє з іншими через чітко визначені інтерфейси.

Загальна логіка застосунку базується на реалізації таких основних функціональних частин:

- автентифікація користувачів;
- виконання набору тексту з фіксацією статистики;
- збереження й відображення результатів;
- ведення особистого профілю;
- формування таблиці лідерів.

Компоненти, що реалізують основну функціональність, розташовано у відповідних директоріях проєкту src/components/, src/Pages/ та src/services/. Наведемо опис ключових модулів:

Компонент AuthForm.tsx реалізує механізм входу та реєстрації користувача. Він підтримує два методи автентифікації: через Google–акаунт або з використанням електронної пошти та пароля. У разі успішної реєстрації новий користувач отримує автоматично згенеровану аватарку (за допомогою API ui–avatars.com) та має можливість вказати своє ім’я. Дані зберігаються у Firebase Authentication, а також у базі Firestore.

Компонент NavBar.tsx відповідає за навігацію між основними сторінками програми. Реалізований з використанням бібліотеки react–router–dom, він дозволяє здійснювати переходи до розділів: «Головна», «Тренування», «Профіль» та «Лідерборд». Навігаційне меню адаптоване до різних розмірів екранів.

Компонент Trainer.tsx – це сторінка, з якої розпочинається основне тренування користувача. Тут реалізовано вибір складності, тривалості (15, 30, 60 секунд), а також генерується текст для набору. Саме цей компонент координує початок тренування та обробку фінальних результатів, які передаються до статистичного модуля та виводяться у вигляді модального вікна.

Компонент TypingArea.tsx є основою тренажера. Він обробляє натискання клавіш, порівнює введені символи з очікуваними, підраховує правильні та неправильні введення, обчислює показники WPM (words per minute) та точність. Особливу увагу приділено візуалізації: поточна літера підсвічується, помилки виділяються червоним, а пробіли замінюються на незламні символи для коректного рендерингу. При завершенні часу тренування обчислені показники передаються у модальне вікно результатів.

Модуль ResultModal.tsx реалізує діалогове вікно з підсумками тренування. Він відображає швидкість друку, точність, кількість правильних і неправильних символів, обраний режим та складність. Також у ньому реалізовано функціонал для перезапуску тесту або збереження результату в базу даних.

Компонент UserInfo.tsx, що відображається на сторінці Profile.tsx, показує відомості про користувача (displayName, email, аватар), а також дозволяє змінювати ім’я (не частіше ніж раз на 7 днів) та здійснити вихід із системи. Дані користувача зчитуються з Firebase Auth, а зміни оновлюються і в Firestore.

Компонент Leaderboard.tsx виводить таблицю лідерів, де можна побачити найкращі результати користувачів. Таблиця реалізує фільтрацію за періодом (день, тиждень, усі часи), складністю (easy, medium, hard) та тривалістю сесії (15, 30, 60 секунд). Дані витягуються з колекції results у Firestore із використанням складених запитів та індексів.

Комунікація між модулями реалізована через такі механізми:

- Props – передача параметрів між батьківськими та дочірніми компонентами;

- Контекст Firebase Auth – надає доступ до поточних даних користувача в будь–якому місці програми;
- Firestore – зберігання даних профілю користувача та результатів сесій. Використовуються запити setDoc, addDoc, getDocs, query, where, orderBy, що забезпечують асинхронну взаємодію із сервером;
- React Router – для маршрутизації між сторінками.

Кожен модуль взаємодіє лише з тими компонентами, що прямо залучені до його роботи, що відповідає принципу «низького зв'язування та високої зв'язаності» (Low Coupling – High Cohesion), що вважається найкращою практикою в розробці масштабованих SPA–додатків.

3.2 Реалізація програмних компонентів системи

Реалізація програмного забезпечення веб–застосунку «EasyTyping» здійснювалася за допомогою сучасних інструментів розробки інтерфейсів та сервісів: React, TypeScript, Firebase, а також модульної побудови компонентів. У цьому підрозділі наведено опис реалізації основних частин системи, що були зазначені у попередньому підрозділі.

3.2.1 Автентифікація

Компонент AuthForm.tsx відповідає за функціонал автентифікації користувача та є ключовим елементом взаємодії з Firebase Authentication. Його структура побудована з використанням хуків React, а логіка автентифікації поділена на дві гілки: вхід/реєстрація через email та Google Sign–In.

Основою є стан useState, який зберігає дані форми: email, пароль, ім'я користувача, а також прапор isSignUp, що визначає, у якому режимі перебуває форма – авторизації чи реєстрації. Також контролюється можливість відображення повідомлення про помилку (error).

При взаємодії з Firebase використовується auth (об'єкт, ініціалізований через getAuth(app) у firebase.ts), а для маршрутизації – useNavigate із react–router–dom.

Реєстрація реалізується за допомогою функції createUserWithEmailAndPassword. Успішне створення облікового запису супроводжується оновленням профілю через updateProfile, зокрема задається displayName та автоматично створюється аватарка користувача через публічний сервіс ui–avatars.com.

Код програмної реалізації updateProfile наведено в лістингу 3.1.

Лістинг 3.1

```
await updateProfile(auth.currentUser, {
  displayName,
  photoURL:  `${DEFAULT_AVATAR}${encodeURIComponent(displayName  ||
'User')}`
})
```

Для входу через email використовується signInWithEmailAndPassword, а для Google Sign–In – signInWithPopup у зв’язці з GoogleAuthProvider (див. лістинг 3.2).

Лістинг 3.2

```
const result = await signInWithPopup(auth, googleProvider)
await completeLogin(result.user)
```

В обох випадках після автентифікації викликається функція completeLogin, яка:

- Викликає функцію saveUserProfile, що зберігає базові дані користувача до Firestore (uid, displayName, email, photoURL);
- Очищає форму;
- Перенаправляє користувача на сторінку тренажера /trainer.

Функція saveUserProfile (лістинг 3.3). Вона записує інформацію про користувача до колекції users у Firestore. Для уніфікації профілів усі користувачі, незалежно від методу реєстрації, зберігаються з однаковою структурою.

Лістинг 3.3

```
export const saveUserProfile = async (user: UserProfile): Promise<void> => {
  try {
    await setDoc(doc(db, 'users', user.uid), {
      uid: user.uid,
      displayName: user.displayName,
      email: user.email,
      photoURL: user.photoURL,
    })
```

Це дозволяє в подальшому використовувати інформацію у профілі, лідерборді, історії результатів тощо.

Кожен блок автентифікації захищено конструкцією try/catch. У випадку помилок встановлюється текст помилки у стані error, який одразу виводиться на екран у вигляді повідомлення.

Код програмної реалізації обробки помилок наведено в лістингу 3.4.

Лістинг 3.4

```
catch (error) {
  if (error instanceof Error) {
    setError(error.message)
  } else {
    setError('Failed to sign in with Google')
  }
}
```

Отже компонент AuthForm є важливою частиною взаємодії користувача з системою, забезпечує зручний спосіб реєстрації/авторизації, а також дозволяє вести облік профілів у Firestore. Його реалізація поєднує простоту та надійність, а інтеграція з Firebase значно спрощує роботу з автентифікацією без необхідності створення власного бекенду. Повний лістинг коду компоненту AuthForm наведено у додатку А, лістинг А.1.

3.2.2 Профіль користувача

Компонент UserInfo.tsx реалізує логіку перегляду та оновлення персональної інформації користувача, а також дозволяє виконати вихід із системи. Він є важливою складовою людино–машинного інтерфейсу, оскільки

забезпечує зворотний зв’язок між користувачем і системою, дозволяючи переглядати та змінювати персональні дані.

Компонент використовує React–хуки для керування локальним станом:

- user – поточний користувач із Firebase;
- displayName – ім’я користувача, яке можна редагувати;
- editing – прапор, який вказує на режим редагування;
- lastChange – час останньої зміни імені (використовується для обмеження частоти змін);
- error – повідомлення про помилку (наприклад, при недотриманні обмеження зміни імені).

Ініціалізація користувача відбувається у useEffect (див. лістинг 3.5), де підписка onAuthStateChanged дозволяє відстежити поточний стан аутентифікації. Якщо користувач не авторизований – відбувається перенаправлення на головну сторінку.

Лістинг 3.5

```
useEffect(() => {
  const unsubscribe = onAuthStateChanged(auth, async (u) => {
    if (!u) {
      navigate('/')
      return
    }
    setUser(u)
    setDisplayName(u.displayName || '')

    const userDoc = await getDoc(doc(db, 'users', u.uid))
    const data = userDoc.data()
    if (data?.lastNameChange?.seconds) {
      setLastChange(new Date(data.lastNameChange.seconds * 1000))
    }
  })
  return () => unsubscribe()
}, [navigate])
```

Механізм виходу. Кнопка "Log out" викликає метод signOut(auth), після чого відбувається перенаправлення на головну сторінку. Це реалізує функціонал виходу із профілю.

Одна з функціональних особливостей – можливість зміни імені не частіше ніж раз на тиждень. Це забезпечується перевіркою дати останнього оновлення, яка зберігається у Firestore у полі lastNameChange. Якщо зміна відбувається раніше, ніж дозволено, користувач бачить відповідне повідомлення про помилку.

Якщо обмеження не порушене, зміна імені відбувається одночасно у Firebase Auth (через updateProfile) та у Firestore (через updateDoc). Для синхронізації використовуються сервіси.

Компонент UserInfo.tsx виконує функцію профільного модуля у веб-застосунку. Він дозволяє користувачу переглядати персональні дані, редагувати ім’я з урахуванням обмеження, а також виходити із системи. Використання Firebase забезпечує надійність, а локальний стан React – інтерактивність. Компонент реалізує важливу частину логіки взаємодії з користувачем та збереження даних, що відповідає вимогам до людино-машинного інтерфейсу та персоналізації системи. Повний лістинг коду компоненту UserInfo наведено у додатку А, лістинг А.2.

3.2.3 TypingArea

Компонент TypingArea є основним елементом тренувального процесу. Він відповідає за відображення тексту для набору, підрахунок введених символів, визначення правильності та неправильності кожного введеного символу, а також обчислення статистичних показників.

Структура логіки компонента:

a) Стани (useState):
   1. input – поточний рядок, який ввів користувач;
   2. timeLeft – час, що залишився до завершення сесії;
   3. isTyping – булеве значення, що вказує, чи триває набір;
   4. correctCount та incorrectCount – відповідно кількість правильних та неправильних символів;

b) Логіка часу:

Таймер запускається при першому натисканні клавіші користувачем. Обрахунок часу ведеться через setInterval у useEffect. Після завершення часу функція calculateStats обчислює точність (accuracy), швидкість друку (WPM) та викликає зворотній колбек onStatsUpdate;

c) Перевірка символів:

Кожен введений символ порівнюється з очікуваним символом з тексту. У разі збігу оновлюється correctCount, у разі помилки – incorrectCount (Лістинг 3.6);

d) Відображення тексту:

Текст розбивається на слова та символи. Кожному символу присвоюється клас стилю correct, incorrect, або current, залежно від поточного стану набору.

Лістинг 3.6

```
if (newChar === expectedChar) {
  setCorrectCount((c) => c + 1)
} else {
  setIncorrectCount((i) => i + 1)
}
```

Лістинг усього коду компоненту TypingArea наведено у Додатку А, лістинг А.3.

3.2.4 Сторінка тренажеру – Trainer

Компонент Trainer – це сторінка, що відповідає за формування умов для тренування. Він поєднує в собі логіку вибору режиму, відображення компоненту для набору (TypingArea), а також модального вікна результату (ResultModal).

Вибір режиму тренування реалізовано за допомогою кнопок, які змінюють difficulty (easy/medium/hard) та timeMode (15/30/60 секунд). При зміні складності генерується новий текст.

Функція generateText використовує випадковий вибір слів з відповідного словника. Текст формується динамічно при кожному запуску або перезапуску тесту. Після завершення сесії та показу модального вікна, користувач може

зберегти результат у базу Firestore. Для цього викликається addDoc (лістинг 3.7), що додає новий запис до колекції results.

Лістинг 3.7

```
addDoc(collection(db, 'results'), {
  uid: auth.currentUser.uid,
  displayName: auth.currentUser.displayName || 'Anonymous',
  wpm: result.wpm,
  accuracy: result.acc,
  chars: result.chars,
  correct: result.correct,
  ...
})
```

Лістинг усього коду компоненту Trainer наведено у Додатку А, лістинг А.4.

3.2.5 Лідерборд, відображення результатів

Leaderboard є сторінкою, яка виводить таблицю результатів користувачів у вигляді рейтингу. Вона дозволяє фільтрувати (див. ліст. 3.8) дані за трьома критеріями:

- Режим часу (15, 30, 60 секунд);
- Складність (easy, medium, hard);
- Період (за день або за тиждень).

Фільтрація реалізується за допомогою хуків useState, які передаються в Firebase-запит через умови where та orderBy. Запит формується динамічно при кожній зміні фільтрів.

Лістинг 3.8

```
const q = query(
  collection(db, 'results'),
  where('timeMode', '==', timeMode),
  where('difficulty', '==', difficulty),
  where('createdAt', '>=', Timestamp.fromDate(start)),
  orderBy('createdAt', 'desc'),
  limit(50)
)
```

Отримані результати обробляються та відображаються у таблиці, де вказано:

- порядковий номер у рейтингу;
- ім’я користувача;
- WPM;
- точність;
- кількість символів;
- кількість правильних та неправильних;
- дата створення результату.

Компонент використовує getDocs для отримання даних з Firestore та зберігає їх у локальному стані results. Після кожного оновлення фільтрів повторно викликається функція fetchResults.

Компонент ResultModal відповідає за виведення результатів тренування після завершення сесії. Він є модальним вікном, яке з’являється поверх основного інтерфейсу і подає узагальнену статистику сесії користувача.

До ключових параметрів, які відображаються, належать: WPM (кількість слів за хвилину); точність у відсотках; загальна кількість набраних символів; кількість правильних та неправильних символів; обрана складність (easy/medium/hard); час проходження сесії (15/30/60 секунд).

Окрім відображення, компонент містить кнопки керування: Save – збереження результату до бази даних Firestore; Restart – перезапуск тесту; Close – закриття модального вікна без збереження.

Таким чином, цей компонент реалізує логічне завершення тренувального процесу та забезпечує інтерактивність користувача після завершення сесії.

Лістинг усього коду компонентів Leaderboard та ResultModal наведено у Додатку А, лістинг А.5.

3.2.6 Навігаційна панель

Компонент Navbar реалізує основну навігаційну панель веб-застосунку. Він забезпечує користувачеві швидкий доступ до основних сторінок інтерфейсу: головної сторінки, тренажера, лідерборду та профілю. Для маршрутизації

використовується бібліотека react-router-dom, зокрема компонент NavLink, який дозволяє підсвічувати активну вкладку залежно від поточного шляху.

Основна логіка компонента побудована на відстеженні автентифікованого користувача за допомогою стану user, який оновлюється у useEffect при зміні автентифікаційного стану через auth.onAuthStateChanged. Якщо користувач не авторизований, посилання на профіль не відображається (лістинг 3.9).

Лістинг 3.9

```
{user && (
  <NavLink to="/profile">Profile</NavLink>
)}
```

Таким чином, Navbar є умовно динамічним компонентом, що адаптує свій вигляд відповідно до статусу користувача. Це підвищує зручність користування та забезпечує контекстну навігацію. Лістинг усього коду компоненту Navbar наведено у Додатку А, лістинг А.6.

3.3 Інтерфейс застосунку

Інтерфейс користувача реалізовано з використанням CSS-файлів, що розділені за функціональним призначенням. Такий підхід дозволяє зберігати логічну структуру та спрощує подальшу підтримку й розширення стилів. Для стилізації не застосовувалися інлайн-стилі, натомість перевага надавалась модульним файлам із чітким розмежуванням за компонентами.

Базове оформлення, що використовується глобально, міститься у файлі App.css. Тут визначено стилі для кореневого елементу #root, а також оформлення логотипу, анімації обертання та базових блоків. Наприклад, логотип React отримує ефект підсвічування при наведенні, що створює динамічність і підкреслює інтерактивний характер інтерфейсу. Цей файл також містить анімацію logo-spin, яка додає руху і візуальної привабливості головній сторінці або верхній панелі.

Файл typing.css відповідає за стилізацію інтерфейсу самого поля вводу тексту, тобто тренажеру (див. рис. 3.1, 3.2). Тут визначено змінні кольорів,

включно з основним кольором, кольором для коректних і некоректних символів, а також оформлення курсору.

У класі .typing-text застосовується масштабований шрифт із властивістю clamp, що забезпечує адаптивність на різних пристроях. Для кожного символу передбачено різну стилізацію залежно від його стану: .correct, .incorrect та .current. Також окремо оформлені блоки з вибором складності та режиму часу, що робить інтерфейс зрозумілим і зручним у використанні.

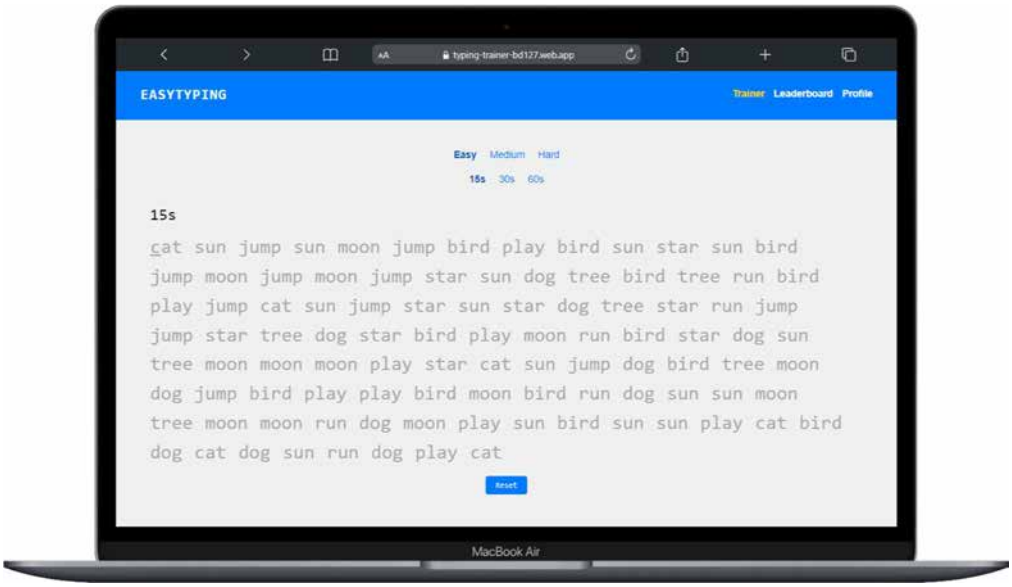

Рисунок 3.1 – Інтерфейс тренажеру (web)

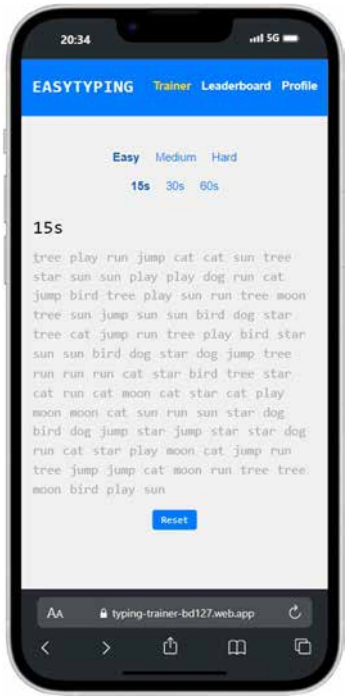

Рисунок 3.2 – Інтерфейс тренажеру (мобільна версія)

Файл leaderboard.css повністю присвячений стилям таблиці результатів. В ньому реалізовано гнучке оформлення за допомогою flex-контейнерів для фільтрів, адаптивне прокручування таблиці в горизонтальному напрямку, а також візуальне виділення перших трьох місць у рейтингу через зміну фону. Це надає лідерборду привабливого вигляду та дозволяє зручно переглядати дані навіть на мобільних пристроях.

Файл ResultModal.css містить стилі для модального вікна, яке з'являється після завершення тренування. Вікно оформлене таким чином, щоб бути максимально читабельним, із чіткими відступами, великою кнопкою збереження результату та акцентом на основних метриках користувача. Зображення вікна продемонстровано на рис. 3.3.

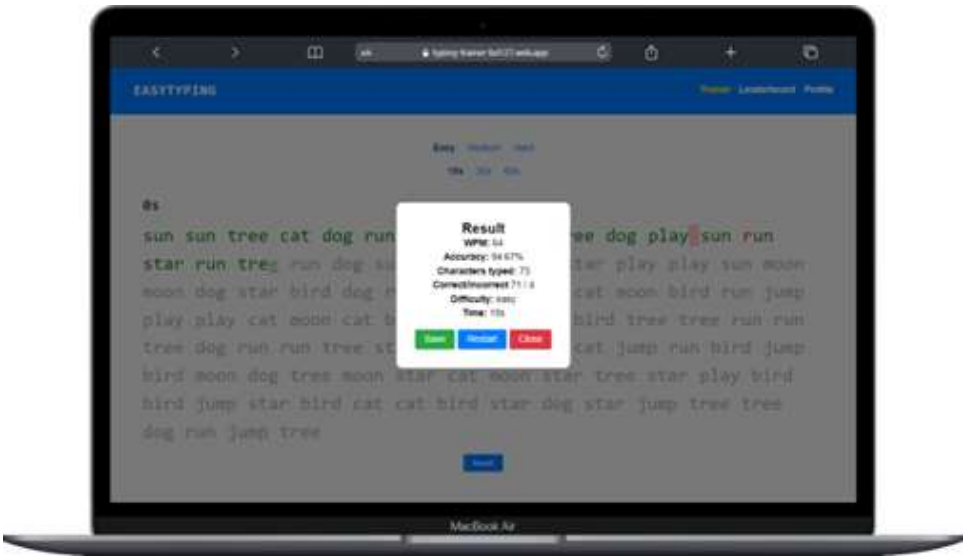

Рисунок 3.3 – Модальне вікно з результатами

Загалом, інтерфейс застосунку поєднує мінімалізм і функціональність. Для забезпечення консистентності стилів використано змінні CSS, а адаптивність досягається завдяки медіа-запитам. Стилізація окремих компонентів реалізована так, щоби користувач отримував миттєвий візуальний зворотний зв'язок при введенні тексту або взаємодії з елементами.

Таким чином, структура стилів в застосунку дозволяє легко орієнтуватися в коді, швидко вносити зміни та масштабувати дизайн у разі подальшого розвитку продукту.

Інтерфейс сторінки авторизації, профілю зображено на рис. 3.4 та 3.5.

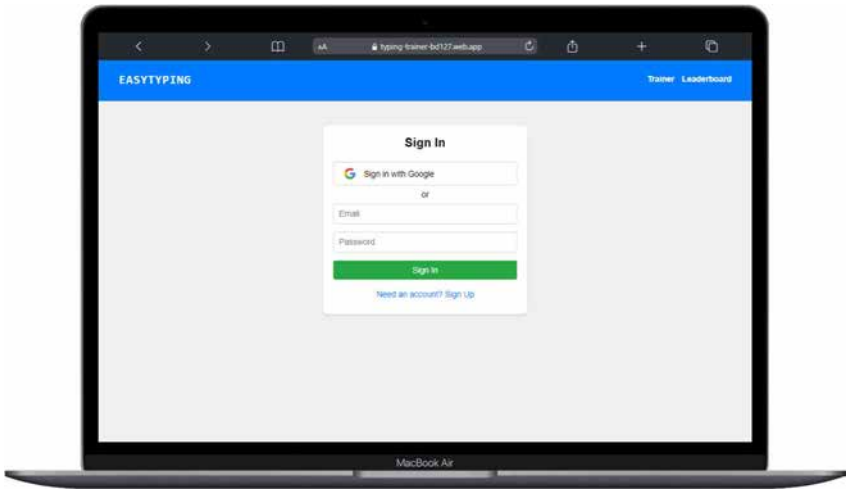

Рисунок 3.4 – сторінка аторизації (web/mobile)

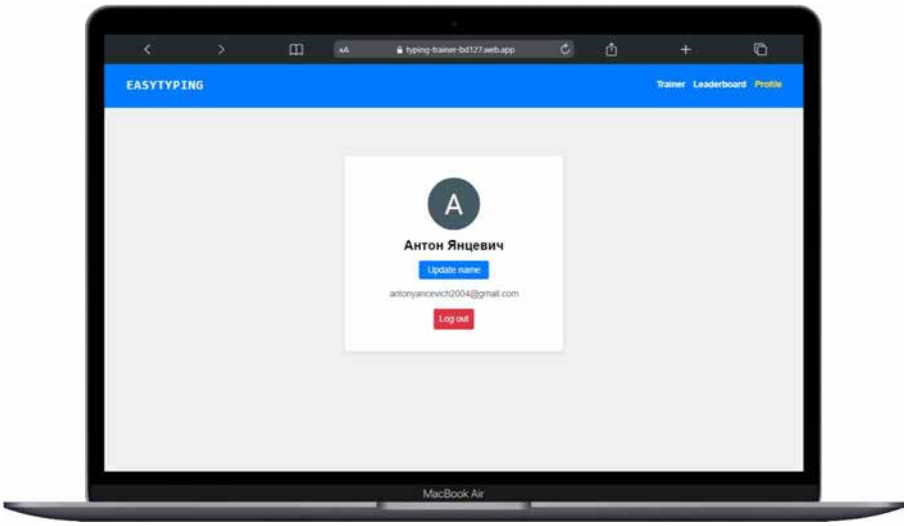

Рисунок 3.5 – сторінка профілю (web/mobile)

Лістинг css-файлів, стилі основних компонентів наведено в додатку Б.

3.4 Стратегія та методика тестування

Розробка веб-застосунку «EasyTyping» супроводжувалася поетапним тестуванням функціональності в умовах реального середовища. Враховуючи масштаби проєкту, динамічність змін і обмеження ресурсів, було прийнято рішення застосовувати мануальну стратегію тестування. Такий підхід дозволив не лише виявити технічні дефекти, а й оцінити зручність користування з точки зору звичайного користувача.

Короткі відомості про методику тестування. Для організації тестування було сформовано набір тест-кейсів (test cases) – сценаріїв перевірки конкретної функції або ситуації. Приклад тест-кейсу наведено у табл. 3.1. Кожен тест-кейс мав наступні атрибути:

- Ідентифікатор (TC-001, TC1…);
- Опис дії;
- Вхідні дані;
- Очікуваний результат;
- Фактичний результат;
- Статус: «успішно» / «невдача»;
- Примітки (у разі розбіжностей).

Таблиця 3.1 – приклад тест-кейсу

TC ID	Опис	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TC1	Вхід через Google	Аккаунт Google	Перенаправлення на /trainer, збереження профілю	Відкрилась сторінка з тренажером	Успішно

Мануальне тестування охопило ключові сценарії використання застосунку: автентифікацію через Google та email/пароль, проходження тренування, розрахунок швидкості та точності друку, збереження результатів у Firestore, перегляд лідерборду, зміну імені в профілі, адаптивність інтерфейсу. Особливу увагу було приділено перевірці роботи з клавіатурою на мобільних пристроях та відображенню статистики в модальному вікні. Усі ключові ситуації такого роду були вручну перевірені в браузерах Chrome, Firefox і на мобільному Android-пристрої.

Кожен знайдений дефект документувався коротким описом, зокрема вказувалися умови, за яких помилка виникла, очікувана та фактична поведінка, а також її критичність. Наприклад, на початковому етапі тестування було виявлено, що при швидкому наборі на Android вводяться зайві символи – ця помилка була оперативно локалізована та усунута.

Всі основні компоненти веб-застосунку від авторизації до лідерборду пройшли ручну перевірку на відповідність вимогам. За результатами тестування встановлено, що функціональність проєкту відповідає заявленим критеріям, а виявлені проблеми мають незначний характер і не заважають основному використанню програми.

3.5 Аналіз результатів тестування та звіт

Після реалізації всіх функціональних компонентів системи було проведено повне тестування з метою перевірки працездатності, відповідності вимогам та виявлення потенційних дефектів.

Тестування охоплювало функціональні та нефункціональні вимоги, включаючи:

- перевірку коректності автентифікації через email та Google;
- валідацію введення користувачем;
- перевірку режимів складності та тривалості тренування;
- оцінку підрахунку статистики (WPM, точність, символи);
- збереження результатів до Firestore;
- відображення даних у таблиці лідерів;
- оновлення профілю з обмеженням часу;
- адаптивність інтерфейсу на мобільних пристроях.

Для кожного функціонального модуля формувалися тест-кейси з описом очікуваних і фактичних результатів. Загалом було опрацьовано понад 20 тестових сценаріїв див. табл. 3.2.

Таблиця 3.2 Тест-кейси

ID	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
TC1	Реєстрація нового користувача	email: test@mail.com, password: 12345678	Користувача створено, відображено профіль	Створений новий користувач, занесений до колеції users	Успішно

Продовження табл. 3.2

TC2	Авторизація з помилкою пароля	email: test@mail.com, password: неправильний	Повідомлення про помилку	auth/invalid-password	Успішно
TC3	Підрахунок WPM	Текст: 30 слів, час: 30 сек	Результат у межах 55–65 WPM	60 WPM	Успішно
TC4	Зміна імені двічі за тиждень	2 спроби зміни displayName за 3 дні	Повідомлення про обмеження	Повідомлення коректно відображено	Успішно
TC5	Збереження результатів	Завершення тренування	Дані збережено в базі Firestore	Збережено у колекцію results	Успішно
TC6	Відображення лідерборду	Час: 15 сек, складність: easy	Таблиця з результатами відображається	Відображено списко користувачів	Успішно
TC7	Мобільна версія інтерфейсу	Екран 375px	Інтерфейс адаптується, не ламається	Усі елементи інтерфейсу коректно відображуються	Успішно

У ході тестування були виявлені незначні проблеми:

- Іноді аватарка не оновлювалась після першої реєстрації – виправлено повторним викликом updateProfile;
- У деяких браузерах мобільних пристроїв текст не друкувався.

Ці помилки були оперативно усунені, після чого повторні тест-кейси підтвердили їх успішне вирішення.

Тестування підтвердило, що розроблений веб-застосунок відповідає заявленим функціональним і нефункціональним вимогам. Усі критично важливі функції працюють стабільно, система готова до використання кінцевими користувачами. Базова функціональність автентифікація, тренування, обробка результатів, візуальний інтерфейс і база даних реалізовані повністю та успішно пройшли тестування.

ВИСНОВКИ

У межах виконання бакалаврської кваліфікаційної роботи на тему «Веб-застосунок клавіатурного тренажеру» було досягнуто поставлену мету – створено сучасний, адаптивний, функціональний веб-застосунок для навчання швидкому та точному набору тексту.

Для реалізації завдань, сформульованих у вступі, було здійснено комплексний аналіз предметної області, розглянуто наявні програмні рішення, визначено ключові функціональні та нефункціональні вимоги до системи. Особливу увагу приділено особливостям UI/UX у сфері навчальних застосунків. Застосовано сучасні інструменти та засоби розробки, зокрема: React, TypeScript, Firebase, Vite, що у поєднанні забезпечили високу швидкодію, масштабованість, модульність і безпеку застосунку. Було обґрунтовано вибір саме такого технологічного стеку. Спроєктовано архітектуру односторінкового застосунку (SPA) з чіткою декомпозицією на логічні підсистеми: автентифікація, тренування, облік статистики, профіль користувача, лідерборд. Реалізовано гнучкий і зручний користувацький інтерфейс, адаптований під різні пристрої. Впроваджено авторизацію через Google та email/пароль. Для кожного користувача ведеться профіль із можливістю зміни імені (з обмеженням за часом), збереженням результатів у базі Firestore, формуванням особистої історії тренувань. Реалізовано ключову функцію набору тексту з поетапною перевіркою введення, підсвіткою правильних/неправильних символів, підрахунком WPM, точності та повного обсягу введеної інформації. Всі результати можуть зберігатися та ранжуватися у загальному рейтингу. Розроблено та протестовано лідерборд, який дозволяє фільтрувати результати за тривалістю, складністю й часовим періодом. Це надає користувачу можливість порівнювати свій прогрес з іншими учасниками системи. Методика тестування базувалася на ручній перевірці функціоналу, що дозволило своєчасно виявити й усунути помилки. Основні модулі було перевірено на коректність відображення, відповідність введення, обчислення статистики, а також обробку граничних випадків.

Реалізований веб-застосунок повністю відповідає технічному завданню, є стабільним, функціональним і зручним у використанні. Його впровадження дозволяє:

- підвищити ефективність навчання набору тексту;
- заощадити час на встановлення додаткових програм;
- адаптувати процес тренування під індивідуальні потреби;
- додати ігрову складову у вигляді рейтингу;
- підвищити мотивацію користувачів через персоналізований фідбек.

Практична цінність полягає у можливості використання системи в освітніх установах, ІТ-курсах, а також для самостійного навчання. Система масштабована, легко доповнюється новими функціями (наприклад, словникові тести, нагороди, курси). У перспективі планується інтеграція штучного інтелекту для адаптивного формування тренувальних текстів, розширення системи статистики та можливість змагань між користувачами в реальному часі.

Посилання розробленого застосунку: URL: https://typing-trainer-bd127.web.app/

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kyrychek H. H., Tiahunova M. Yu., Balabukha O. M. Logistics web platform using React.js // *Publishing House "Baltija Publishing"*. – 2023.
2. Goldberg J. Learning TypeScript. – Sebastopol, CA : O'Reilly Media, Inc., 2022. – [Електронна книга].
3. Fenton S., Fenton, Spearing. Pro TypeScript. – Apress, 2014. – [Електронна книга].
4. Moroney L. An introduction to Firebase // In: *The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform*. – Berkeley, CA : Apress, 2017. – С. 1–24.
5. Jubilee Enterprise. HTML 5 Manual Book. – Jakarta : Elex Media Komputindo, 2015.
6. Gackenheimer C. Introduction to React. – Apress, 2015. – [Електронна книга].
7. Fowler S., Stanwick V. Web application design handbook: Best practices for web-based software. – Burlington, MA : Morgan Kaufmann, 2004.
8. Firebase Documentation [Електронний ресурс]. – Google Firebase. – Available from: https://firebase.google.com/docs (дата звернення: 10.04.2025).
9. React Documentation [Електронний ресурс]. – React.dev. – Available from: https://react.dev (дата звернення: 10.04.2025).
10. Bierman A. TypeScript Quickly. – Shelter Island, NY : Manning Publications, 2020. – [Електронна книга].

ДОДАТКИ

Додаток А.

Лістинг А.1

```
import { useState } from 'react'
import {
  type User,
  signInWithPopup,
  GoogleAuthProvider,
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword,
  updateProfile,
} from 'firebase/auth'
import { auth } from '../firebase'

import { useNavigate } from 'react-router-dom'
import { saveUserProfile } from '../services/firestore'
import googleLogo from '../assets/google/google-logo.svg'

const DEFAULT_AVATAR = 'https://ui-
avatars.com/api/?background=random&name='

const AuthForm = () => {
  const [email, setEmail] = useState('')
  const [password, setPassword] = useState('')
  const [displayName, setDisplayName] = useState('')
  const [isSignUp, setIsSignUp] = useState(false)
  const [error, setError] = useState('')
  const navigate = useNavigate()
  const googleProvider = new GoogleAuthProvider()

  const clearForm = () => {
    setEmail('')
    setPassword('')
    setDisplayName('')
  }

  const completeLogin = async (user: User) => {
    await saveUserProfile({
      uid: user.uid,
      displayName: user.displayName || displayName || 'Anonymous',
      email: user.email,
      photoURL:
        user.photoURL ||
            `${DEFAULT_AVATAR}${encodeURIComponent(displayName  ||
'User')}`,
    })
    clearForm()
    navigate('/trainer')
  }
```

```
  const handleGoogleSignIn = async () => {
    try {
      setError('')
      const result = await signInWithPopup(auth, googleProvider)
      await completeLogin(result.user)
    } catch (error) {
      if (error instanceof Error) {
        setError(error.message)
      } else {
        setError('Failed to sign in with Google')
      }
    }
  }

  const handleEmailAuth = async (e: React.FormEvent) => {
    e.preventDefault()
    try {
      setError('')
      const result = isSignUp
            ?  await  createUserWithEmailAndPassword(auth,  email,
password)
        : await signInWithEmailAndPassword(auth, email, password)

      if (isSignUp && auth.currentUser) {
        await updateProfile(auth.currentUser, {
          displayName,
          photoURL: `${DEFAULT_AVATAR}${encodeURIComponent(
            displayName || 'User'
          )}`,
        })
      }

      await completeLogin(result.user)
    } catch (error) {
      if (error instanceof Error) {
        setError(error.message)
      } else {
        setError('Failed to authenticate')
      }
    }
  }

  return (
    <div className="auth-form">
       <h2  className="auth-title">{isSignUp  ?  'Sign  Up'  :  'Sign
In'}</h2>
      {error && <p className="error">{error}</p>}

      <button className="google-btn" onClick={handleGoogleSignIn}>
        <img src={googleLogo} alt="g-logo" />
        Sign in with Google
      </button>
```

```
      <span className="or">or</span>
      <form onSubmit={handleEmailAuth}>
        <input
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Email"
          autoComplete="off"
        />
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Password"
          autoComplete="off"
        />
        {isSignUp && (
          <input
            type="text"
            value={displayName}
            onChange={(e) => setDisplayName(e.target.value)}
            placeholder="Your name"
            autoComplete="off"
          />
        )}
        <button type="submit" className="submit-btn">
          {isSignUp ? 'Sign Up' : 'Sign In'}
        </button>
      </form>
              <button   className="toggle-btn"   onClick={()   =>
setIsSignUp(!isSignUp)}>
        {isSignUp
          ? 'Already have an account? Sign In'
          : 'Need an account? Sign Up'}
      </button>
    </div>
  )
}

export default AuthForm
```

Лістинг А.2

```
import { useState, useEffect } from 'react'
import { useNavigate } from 'react-router-dom'
import { onAuthStateChanged, signOut, updateProfile } from
'firebase/auth'
import { auth } from '../firebase'
import { doc, getDoc, updateDoc, serverTimestamp } from
'firebase/firestore'
import { db } from '../firebase'

const ONE_WEEK_MS = 7 * 24 * 60 * 60 * 1000

const UserInfo = () => {
  const navigate = useNavigate()
  const [user, setUser] = useState(auth.currentUser)
  const [displayName, setDisplayName] = useState('')
  const [editing, setEditing] = useState(false)
  const [error, setError] = useState('')
  const [lastChange, setLastChange] = useState<Date | null>(null)

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (u) => {
      if (!u) {
        navigate('/')
        return
      }
      setUser(u)
      setDisplayName(u.displayName || '')

      const userDoc = await getDoc(doc(db, 'users', u.uid))
      const data = userDoc.data()
      if (data?.lastNameChange?.seconds) {
        setLastChange(new Date(data.lastNameChange.seconds * 1000))
      }
    })
    return () => unsubscribe()
  }, [navigate])

  const handleSignOut = async () => {
    await signOut(auth)
    navigate('/')
  }

  const handleSaveName = async () => {
    if (!user) return
    const now = Date.now()
    if (lastChange && now - lastChange.getTime() < ONE_WEEK_MS) {
      setError('You can only change your name once every 7 days.')
      return
    }

    try {
```

```
      await updateProfile(user, { displayName })
      await updateDoc(doc(db, 'users', user.uid), {
        displayName,
        lastNameChange: serverTimestamp(),
      })
      setEditing(false)
      setLastChange(new Date())
      setError('')
    } catch (err) {
      console.error(err)
      setError('Error updating name.')
    }
  }

  if (!user) return null

  return (
    <div className="profile">
      <img
        className="profile-avatar"
        src={user.photoURL || 'https://via.placeholder.com/150'}
        alt="Avatar"
      />
      {editing ? (
        <div className="edit-name">
          <input
            type="text"
            value={displayName}
            onChange={(e) => setDisplayName(e.target.value)}
            placeholder="New name"
          />
          <div>
                              <button    className="btn-modal--save"
onClick={handleSaveName}>
              Save
            </button>
            <button
              className="btn-modal--close"
              onClick={() => setEditing(false)}
            >
              Cancel
            </button>
          </div>
          {error && <p className="error">{error}</p>}
        </div>
      ) : (
        <>
          <h2>{user.displayName || 'User'}</h2>
                <button  className="btn-reset  "  onClick={()  =>
setEditing(true)}>
            Update name
          </button>
```

```
        </>
      )}
      <p>{user.email}</p>
      <button className="signout-btn" onClick={handleSignOut}>
        Log out
      </button>
    </div>
  )
}

export default UserInfo
```

Лістинг А.3

```
import React, { useState, useEffect, useCallback, useRef } from
'react'
import type { User } from 'firebase/auth'

interface Props {
  user: User | null
  text: string
  onStatsUpdate: (
    wpm: number,
    accuracy: number,
    chars: number,
    correct: number,
    incorrect: number
  ) => void
  timeMode: number
}

const TypingArea: React.FC<Props> = ({ text, onStatsUpdate, timeMode
}) => {
  const [input, setInput] = useState('')
  const [timeLeft, setTimeLeft] = useState(timeMode)
  const [isTyping, setIsTyping] = useState(false)
  const [correctCount, setCorrectCount] = useState(0)
  const [incorrectCount, setIncorrectCount] = useState(0)

  const inputRef = useRef<HTMLInputElement>(null)

  const reset = useCallback(() => {
    setInput('')
    setTimeLeft(timeMode)
    setIsTyping(false)
    setCorrectCount(0)
    setIncorrectCount(0)
  }, [timeMode])

  useEffect(() => {
    inputRef.current?.focus()
  }, [])
```

```
  useEffect(() => {
    reset()
    inputRef.current?.focus()
  }, [text, timeMode, reset])

  useEffect(() => {
    if (!isTyping || timeLeft <= 0) return
     const interval = setInterval(() => setTimeLeft((t) => t - 1),
1000)
    return () => clearInterval(interval)
  }, [isTyping, timeLeft])

  const calculateStats = useCallback(() => {
    const acc = (correctCount / (correctCount + incorrectCount)) *
100 || 0
    const wordsTyped = input.trim().split(/\s+/).length
    const wpm = wordsTyped / ((timeMode - timeLeft) / 60) || 0

    onStatsUpdate(
      parseFloat(wpm.toFixed(2)),
      parseFloat(acc.toFixed(2)),
      input.length,
      correctCount,
      incorrectCount
    )
   }, [input, timeLeft, timeMode, correctCount, incorrectCount,
onStatsUpdate])

  useEffect(() => {
    if (timeLeft <= 0 && isTyping) {
      calculateStats()
      setIsTyping(false)
    }
  }, [timeLeft, isTyping, calculateStats])

  const handleInput = (e: React.FormEvent<HTMLInputElement>) => {
    const value = e.currentTarget.value

    if (timeLeft <= 0) return
    if (!isTyping) {
      setIsTyping(true)
      setTimeLeft(timeMode)
    }

    if (value.length < input.length) {
      setInput(value)
      return
    }

    const newChar = value[value.length - 1]
    const expectedChar = text[input.length]
```

```
    if (newChar === expectedChar) {
      setCorrectCount((c) => c + 1)
    } else {
      setIncorrectCount((i) => i + 1)
    }

    setInput(value)
  }

  return (
          <div    className="typing-container"    onClick={()    =>
inputRef.current?.focus()}>
      <input
        ref={inputRef}
        value={input}
        onChange={handleInput}
        className="hidden-input"
        autoComplete="off"
        autoCorrect="off"
        spellCheck={false}
        autoFocus
      />
      <p className="info">{timeLeft}s</p>
      <div className="typing-text">
        {text.match(/\S+\s*/g)?.map((word, wordIdx, wordArray) => {
          const startIndex =
            wordArray
              ?.slice(0, wordIdx)
              .reduce((acc, w) => acc + w.length, 0) || 0
          return (
            <span className="word" key={wordIdx}>
              {word.split('').map((char, i) => {
                const charIndex = startIndex + i
                const typed = input[charIndex]
                const isCurrent = charIndex === input.length
                let className = 'char'
                if (charIndex < input.length) {
                    className += typed === char ? ' correct' : '
incorrect'
                }
                if (isCurrent) {
                  className += ' current'
                }
                return (
                  <span
                    key={charIndex}
                    className={className}
                    style={
                      char === ' '
                        ? {
```

```
                                                    backgroundColor:
classNameincludes('incorrect')
                                ? '#fbb'
                                : undefined,
                            }
                          : {}
                      }
                    >
                      {char === ' ' ? '\u00A0' : char}
                    </span>
                  )
                })}
              </span>
            )
          })}
        </div>
        <button onClick={reset} className="btn-reset">
          Reset
        </button>
      </div>
    )
}

export default TypingArea
```

Лістинг А.4

```
import { useEffect, useState } from 'react'
import { useNavigate } from 'react-router-dom'
import { onAuthStateChanged } from 'firebase/auth'
import { auth } from '../firebase'
import { db } from '../firebase'
import { collection, addDoc, serverTimestamp } from
'firebase/firestore'
import TypingArea from '../components/TypingArea'
import ResultModal from '../components/ResultModal'

const generateText = (words: string[], count: number) => {
  return Array.from(
    { length: count },
    () => words[Math.floor(Math.random() * words.length)]
  ).join(' ')
}

const wordLists = {
  easy: [
    'cat',
    'dog',
    'run',
    'jump',
    'play',
```

```
    'sun',
    'moon',
    'star',
    'tree',
    'bird',
  ],
  medium: [
    'quick',
    'brown',
    'fox',
    'jumps',
    'over',
    'lazy',
    'dog',
    'river',
    'mountain',
    'forest',
  ],
  hard: [
    'exquisite',
    'benevolent',
    'magnificent',
    'serendipity',
    'ephemeral',
    'resilient',
    'eloquent',
    'intricate',
    'profound',
    'vibrant',
  ],
}

const Trainer = () => {
  const navigate = useNavigate()

  const [difficulty, setDifficulty] = useState<'easy' | 'medium' |
'hard'>(
    'easy'
  )
  const [timeMode, setTimeMode] = useState<number>(15)
  const [text, setText] = useState('')
  const [result, setResult] = useState<{
    wpm: number
    acc: number
    chars: number
    correct: number
    incorrect: number
  } | null>(null)

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (user) => {
      if (!user) navigate('/')
```

```
    })
    return () => unsubscribe()
  }, [navigate])

  useEffect(() => {
    const wordCount = 100
    setText(generateText(wordLists[difficulty], wordCount))
  }, [difficulty])

  const handleRestart = () => {
    setResult(null)
    setText(generateText(wordLists[difficulty], 100))
  }

  return (
    <div className="trainer-page">
      {/* <h1 className="heading">Typing Trainer</h1> */}

      <div className="difficulty-select">
        <button
          className={`link-btn ${difficulty === 'easy' ? 'active' :
''}`}
          onClick={() => setDifficulty('easy')}
        >
          Easy
        </button>
        <button
          className={`link-btn ${difficulty === 'medium' ? 'active'
: ''}`}
          onClick={() => setDifficulty('medium')}
        >
          Medium
        </button>
        <button
          className={`link-btn ${difficulty === 'hard' ? 'active' :
''}`}
          onClick={() => setDifficulty('hard')}
        >
          Hard
        </button>
      </div>

      <div className="time-select">
        <button
          className={`link-btn ${timeMode === 15 ? 'active' : ''}`}
          onClick={() => setTimeMode(15)}
        >
          15s
        </button>
        <button
          className={`link-btn ${timeMode === 30 ? 'active' : ''}`}
          onClick={() => setTimeMode(30)}
```

```
          >
            30s
          </button>
          <button
            className={`link-btn ${timeMode === 60 ? 'active' : ''}`}
            onClick={() => setTimeMode(60)}
          >
            60s
          </button>
        </div>

        <TypingArea
          text={text}
          user={auth.currentUser}
          onStatsUpdate={(wpm, acc, chars, correct, incorrect) => {
            setResult({ wpm, acc, chars, correct, incorrect })
          }}
          timeMode={timeMode}
        />

        {result && (
          <ResultModal
            wpm={result.wpm}
            acc={result.acc}
            chars={result.chars}
            correct={result.correct}
            incorrect={result.incorrect}
            difficulty={difficulty}
            timeMode={timeMode}
            onClose={() => setResult(null)}
            onRestart={handleRestart}
            onSave={() => {
              if (!auth.currentUser || !result) return

              addDoc(collection(db, 'results'), {
                uid: auth.currentUser.uid,
                       displayName:  auth.currentUser.displayName  ||
'Anonymous',
                wpm: result.wpm,
                accuracy: result.acc,
                chars: result.chars,
                correct: result.correct,
                incorrect: result.incorrect,
                difficulty,
                timeMode,
                createdAt: serverTimestamp(),
              })
                .then(() => {
                  setResult(null)
                })
                .catch((error) => {
                  console.error('Error saving result:', error)
```

```
                alert('Saving error.')
              })
          }}
        />
      )}
    </div>
  )
}

export default Trainer
```

Лістинг А.5

```
import React from 'react'
import '../ResultModal.css'

interface Props {
  wpm: number
  acc: number
  chars: number
  difficulty: 'easy' | 'medium' | 'hard'
  timeMode: number
  correct: number
  incorrect: number
  onClose: () => void
  onRestart: () => void
  onSave: () => void
}

const ResultModal: React.FC<Props> = ({
  wpm,
  acc,
  chars,
  correct,
  incorrect,
  difficulty,
  timeMode,
  onClose,
  onRestart,
  onSave,
}) => {
  return (
    <div className="modal-backdrop">
      <div className="modal">
        <h2>Result</h2>
        <p>
          <strong>WPM:</strong> {wpm}
        </p>
        <p>
          <strong>Accuracy:</strong> {acc}%
        </p>
```

```
        <p>
          <strong>Characters typed:</strong> {chars}
        </p>
        <p>
         <strong>Correct/Incorrect</strong> {correct} / {incorrect}
        </p>

        <p>
          <strong>Difficulty:</strong> {difficulty}
        </p>
        <p>
          <strong>Time:</strong> {timeMode}s
        </p>

        <div className="modal-buttons">
                  <button   className="btn-modal   btn-modal--save"
onClick={onSave}>
            Save
          </button>
                 <button   className="btn-modal   btn-modal--reset"
onClick={onRestart}>
            Restart
          </button>
                 <button   className="btn-modal   btn-modal--close"
onClick={onClose}>
            Close
          </button>
        </div>
      </div>
    </div>
  )
}

export default ResultModal
```

Лістинг А.6

```
import { NavLink } from 'react-router-dom'
import { auth } from '../firebase'
import { useEffect, useState } from 'react'
import type { User } from 'firebase/auth'

const Navbar = () => {
  const [user, setUser] = useState<User | null>(auth.currentUser)

  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged((currentUser: User
| null) => {
      setUser(currentUser)
    })
    return () => unsubscribe()
  }, [])
```

```
  return (
    <header className="header">
      <div className="container">
        <NavLink to="/" className="logo">
          EasyTyping
        </NavLink>
        <nav className="nav-links">
          <NavLink
            to="/trainer"
            className={({ isActive }) => (isActive ? 'active' : '')}
          >
            Trainer
          </NavLink>
          <NavLink
            to="/leaderboard"
            className={({ isActive }) => (isActive ? 'active' : '')}
          >
            Leaderboard
          </NavLink>

          {user && (
            <NavLink
              to="/profile"
               className={({ isActive }) => (isActive ? 'active' :
'')}
            >
              Profile
            </NavLink>
          )}
        </nav>
      </div>
    </header>
  )
}

export default Navbar
```

Додаток Б

Лістинг Б.1 коду лідерборду leaderboard.css

Лістинг Б.1

```
.table-wrapper {
  overflow-x: auto;
  width: 100%;
}

.table-wrapper table {
  min-width: 600px;
}

.leaderboard {
  max-width: 960px;
  margin: 40px auto;
  padding: 20px;
  font-family: sans-serif;
  background-color: #fff;
  border-radius: 10px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

.leaderboard h1 {
  text-align: center;
  margin-bottom: 20px;
}

.filters {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
  gap: 1rem;
  margin-bottom: 20px;
}

.filters div {
  display: flex;
  align-items: center;
  gap: 0.5rem;
}

.filters span {
  font-weight: bold;
}

.filters button {
  padding: 6px 12px;
  border: 1px solid #ccc;
  border-radius: 4px;
  background-color: #f4f4f4;
```

```
  cursor: pointer;
  transition: background-color 0.2s ease;
}

.filters button.active {
  background-color: #007bff;
  color: white;
  border-color: #007bff;
}

.filters button:hover {
  background-color: #e0e0e0;
}

.leaderboard table {
  width: 100%;
  border-collapse: collapse;
  font-size: 14px;
}

.leaderboard th,
.leaderboard td {
  padding: 10px;
  border: 1px solid #ddd;
  text-align: center;
}

.leaderboard thead {
  background-color: #f9f9f9;
}

.leaderboard tbody tr:nth-child(1) {
  background-color: #ffeeba;
  font-weight: bold;
}

.leaderboard tbody tr:nth-child(2) {
  background-color: #e2f0cb;
}

.leaderboard tbody tr:nth-child(3) {
  background-color: #d1ecf1;
}
```

Лістинг Б.2 коду лідерборду typing.css

Лістинг Б.2

```
:root {
  --color-text: #aaa;
  --color-correct: green;
  --color-incorrect: red;
  --color-highlight: #fbb;
  --color-primary: #007bff;
  --color-primary-hover: #0056b3;
}

@keyframes blink {
  0%,
  100% {
    opacity: 1;
  }
  50% {
    opacity: 0;
  }
}

.cursor {
  color: black;
  animation: blink 1s steps(1) infinite;
}

.trainer-page {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
}

.typing-container {
  display: flex;
  flex-direction: column;
  gap: 15px;
  max-width: 1200px;
  outline: none;
  padding: 16px;
  font-family: monospace;
}

.typing-text {
  font-size: clamp(18px, 5vw, 32px);
  line-height: 1.5;
  white-space: pre-wrap;
  overflow-wrap: anywhere;
}

.word {
```

```
  display: inline-block;
  white-space: nowrap;
  margin-right: 2px;
}

.char {
  color: var(--color-text);
}
.char.correct {
  color: var(--color-correct);
}
.char.incorrect {
  color: var(--color-incorrect);
}
.char.current {
  text-decoration: underline;
}
.incorrect-space {
  background-color: var(--color-highlight);
}

.hidden-input {
  position: absolute;
  opacity: 0;
  pointer-events: none;
  user-select: none;
  height: 0;
}

.info {
  font-size: 26px;
  line-height: 100%;
}

.btn-reset,
.btn-modal {
  background-color: var(--color-primary);
  color: white;
  border: none;
  padding: 8px 16px;
  border-radius: 4px;
  cursor: pointer;
  width: auto;
  align-self: center;
}
.btn-reset:hover,
.btn-modal:hover {
  background-color: var(--color-primary-hover);
}

.difficulty-select,
.time-select {
```

```
  display: flex;
  align-items: center;
  gap: 10px;
}

.link-btn {
  background: none;
  border: none;
  color: var(--color-primary);
  cursor: pointer;
  font-size: 16px;
}
.link-btn:hover {
  text-decoration: underline;
}
.link-btn.active {
  font-weight: bold;
  color: var(--color-primary-hover);
}

.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 999;
}
.modal-content {
  background: white;
  padding: 24px 32px;
  border-radius: 8px;
  text-align: center;
}

@media (max-width: 600px) {
  .typing-text {
    font-size: 18px;
  }

  .btn-reset {
    padding: 6px 12px;
    font-size: 14px;
  }
}
```

Лістинг Б.3 коду лідерборду ResultModal.css

Лістинг Б.3

```
.modal-backdrop {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 100;
}

.modal {
  background: white;
  padding: 2rem;
  border-radius: 10px;
  width: fit-content;
  text-align: center;
}

.modal-buttons {
  display: flex;
  justify-content: space-between;
  margin-top: 1rem;
  gap: 5px;
}

.btn-modal--save {
  background-color: #28a745;
  color: white;
}

.btn-modal--restart {
  background-color: #ffc107;
  color: black;
}

.btn-modal--close {
  background-color: #dc3545;
  color: white;
}
```

Лістинг Б.4 коду лідерборду App.css

Лістинг Б.4

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}

.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}

@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}
```