МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАТИКИ ТА ФІЗИКО-МАТЕМАТИЧНИХ ДИСЦИПЛІН

ЗВІТ

з переддипломної практики

Реєстраційний № \_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Параска Богдан Володимирович

студент 4 курсу

денної форми навчання

залікова книжка № \_\_\_

ЗВІТ ПРО ПРОХОДЖЕННЯ ПЕРЕДДИПЛОМНОЇ ПРАКТИКИ

Спеціальність "Комп’ютерні науки (Інформатика)"

Рекомендовано до захисту з оцінкою:

"\_\_\_\_\_\_\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Підпис керівника практики

"\_\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 р.

Керівник

практики від ВНЗ

доктор технічних наук,

професор Ніколенко В. В.

Ужгород – 2025

ЗМІСТ

[ВСТУП 3](#_Toc198486331)

[РОЗДІЛ 1. ВІДОМОСТІ ПРО БАЗУ ПРАКТИКИ ТОВ «ПЕТТЕРСОНАПС» 5](#_Toc198486332)

[1.1. Відомості про ТОВ «Петтерсонапс» 5](#_Toc198486333)

[РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАВДАННЯ 7](#_Toc198486334)

[2.1. Клавіатурні тренажери та їх можливості 7](#_Toc198486335)

[2.2. Аналіз існуючих додатків клавіатурних тренажерів 9](#_Toc198486336)

[2.3. Використання фреймворку ReactJS для розробки веб-додатків 14](#_Toc198486337)

[РОЗДІЛ 3. ПРАКТИЧНЕ ЗАВДАННЯ 18](#_Toc198486338)

[3.1. Вибір програмних засобів 18](#_Toc198486339)

[3.2. Оцінка основних бібліотек та фреймворків 21](#_Toc198486340)

[3.3. Створення Google client ID 24](#_Toc198486341)

[3.4. Структура проєкту 26](#_Toc198486342)

[3.5. Реалізація перевірки введених даних 28](#_Toc198486343)

[3.6. Реалізація функції лічильника часу 30](#_Toc198486344)

[3.7. Реалізація швидкості та точності набору тексту 32](#_Toc198486345)

[3.8. Реалізація Google авторизації 33](#_Toc198486346)

[3.9. Реалізація секцій 34](#_Toc198486347)

[3.9. Реалізація створення користувача 37](#_Toc198486348)

[3.10. Інструкція для користувача 39](#_Toc198486349)

[ВИСНОВКИ 45](#_Toc198486350)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ 46](#_Toc198486351)

[ДОДАТКИ 49](#_Toc198486352)

# ВСТУП

Прогнозування цін є одним із ключових завдань у фінансовій аналітиці, торгівлі та економіці загалом. У сучасних умовах, коли обсяг і складність даних зростають, все більшого значення набувають методи штучного інтелекту, зокрема нейронні мережі. Вони демонструють високу здатність до виявлення складних залежностей у часових рядах і дозволяють досягати точніших результатів у порівнянні з класичними статистичними методами.

Метою цього проєкту є дослідження ефективності різних архітектур нейронних мереж у задачах прогнозування цін (на прикладі криптовалют Bitcoin та Ethereum). Передбачається реалізація, тестування та порівняння таких моделей, як багатошаровий перцептрон (MLP), рекурентна нейронна мережа LSTM, згорткова нейронна мережа (CNN), XGBoost та трансформери. Для досягнення цієї мети заплановано виконання наступних завдань:

**Аналіз вимог до дипломного проєкту:** буде сформульовано чіткі цілі дослідження, обсяг задач і обґрунтовано вибір теми з урахуванням актуальності прогнозування цін у сучасному цифровому середовищі.

**Огляд існуючих підходів та технологій:** проведено аналіз сучасних методів машинного навчання, що застосовуються для аналізу часових рядів, зокрема в контексті прогнозування фінансових показників. Обрані методи буде обґрунтовано на основі їх точності, складності реалізації та практичної придатності.

**Підготовка даних та реалізація моделей:** виконано обробку часових рядів, побудовано архітектури обраних моделей у середовищі Python з використанням бібліотек PyTorch, scikit-learn, XGBoost та інших.

**Навчання та тестування моделей:** буде проведено тренування моделей на історичних даних із подальшим тестуванням їхньої точності, стійкості та здатності до генералізації.

**Візуалізація та порівняльний аналіз результатів:** створено графіки реальних і прогнозованих значень, виконано оцінювання ефективності моделей за метриками RMSE, MAE та R².

# РОЗДІЛ 1. ВІДОМОСТІ ПРО БАЗУ ПРАКТИКИ ТОВ «ЦБО СКАЙТЕК»

## Відомості про ТОВ «ЦБО СКАЙТЕК»

Базою проходження практики є Центр бізнес-обслуговування **«СкайТек»** – компанія, що надає широкий спектр ІТ-послуг як для бізнесу, так і для приватних клієнтів. Основні напрямки діяльності центру охоплюють комп’ютерне обслуговування, автоматизацію бізнес-процесів, розробку веб-сайтів, бухгалтерський супровід та створення програмного забезпечення.

**«СкайТек»** відзначається гнучким підходом до клієнтів, індивідуальними консультаціями, а також можливістю виїзду спеціаліста або віддаленого усунення технічних проблем. Працівники компанії мають глибокі знання у сфері електронного документообігу, касового обладнання та інтеграції різноманітних систем обліку.

Рис 1.1 Основні послуги компанії

Центр бізнес-обслуговування "СкайТек" – це компанія, що стрімко розвивається, активно розширюючи свою діяльність у декількох ключових секторах ІТ-індустрії. Вона поєднує в собі різноманітні напрями роботи: від технічного обслуговування комп'ютерів до розробки складних програмних продуктів та реалізації інноваційних рішень для автоматизації бізнес-процесів. Це робить "СкайТек" не просто сервісною компанією, а й надійним ІТ-партнером для малого, середнього та великого бізнесу. Унікальність полягає в наданні повного спектру послуг: від первинної консультації та аналізу потреб клієнта до розробки, впровадження та технічної підтримки ІТ-продуктів або рішень.

Для студентів, які проходять практику в "СкайТек", фірма створює сприятливі умови для отримання практичного досвіду у професійній сфері. Практиканти мають змогу не лише ознайомитися з реаліями корпоративної інфраструктури, але й брати безпосередню участь у вирішенні практичних завдань: налаштуванні та обслуговуванні техніки, роботі над проєктами з розробки веб-сайтів, створенні або тестуванні програмного забезпечення, консультуванні клієнтів, інтеграції облікових систем та багато іншого. Цей досвід дозволяє краще зрозуміти специфіку роботи в ІТ, вдосконалити професійні навички, покращити комунікативні здібності та освоїти практичні інструменти, що використовуються в сучасному ІТ-середовищі.

Багатопрофільність компанії та її орієнтація на реальні бізнес-потреби дають практикантам можливість працювати з актуальними проєктами, вирішувати нестандартні завдання, аналізувати функціонування сучасних систем автоматизації, працювати в команді над проєктами, а також бачити весь процес роботи з клієнтом – від першого звернення до впровадження рішення. Це дозволяє сформувати комплексне розуміння роботи в інформаційних технологіях, покращити професійну підготовку та підвищити конкурентоспроможність на ринку праці після завершення навчання.

# РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАВДАННЯ

## 2.1 Поняття штучних нейронних мереж

Штучна нейронна мережа (ШНМ) - це математична конструкція, що копіює принципи роботи біологічного мозку. Вона базується на великій кількості взаємопов'язаних вузлів, так званих "нейронів", які організовані у шари: вхідний, один чи декілька прихованих та вихідний. Така мережа здатна вчитися на прикладах, виявляючи закономірності у даних, та згодом робити прогнози або класифікувати нову інформацію.

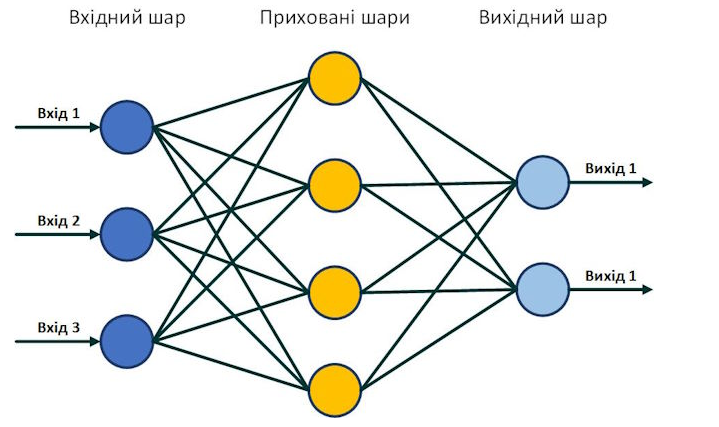
Загалом, структура ШНМ має високу гнучкість і може бути пристосована для різноманітних задач - від розпізнавання зображень до передбачення часових рядів. Її ефективність значною мірою залежить від правильно обраної архітектури та методу навчання. Основними компонентами ШНМ є нейрони, вагові коефіцієнти, функція активації та алгоритм оптимізації. Кожен нейрон отримує сигнали від попередніх шарів, зважує їх, підсумовує та передає результат далі.

Рис. 2.1. Структура найпростішої нейронної мережі

На Рис. 2.1 представлено одношарову нейронну мережу, яка складається з одного входу, одного прихованого шару та одного виходу. Кожен зв'язок має вагу, яка змінюється в процесі навчання.

Навчання ШНМ є процесом налаштування ваг, у ході якого помилка між фактичним та очікуваним результатом мінімізується. Це може досягатися за допомогою різних методів – найпоширенішими з них є градієнтний спуск та його варіації. Під час зворотного поширення помилки (backpropagation) мережа "навчається", змінюючи ваги у напрямку зменшення помилки.

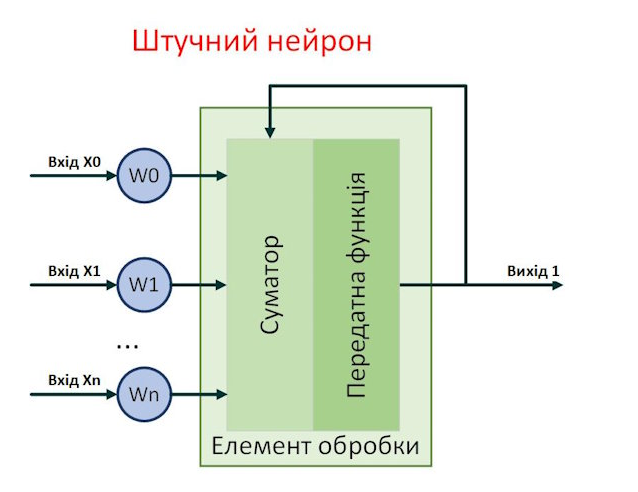
На Рис. 2.2 продемонстровано, як нейронна мережа обчислює похибку, а потім повертається шарами, змінюючи ваги для покращення точності.

Рис. 2.2 – Схематичне зображення процесу навчання мережі за допомогою зворотного поширення помилки

Ключовим аспектом при побудові ефективної моделі є вибір архітектури нейромережі. Просте збільшення кількості шарів або нейронів не завжди приносить кращі результати – важливо уникати як недонавчання (underfitting), так і перенавчання (overfitting). Для цього використовуються спеціальні техніки, серед яких регуляризація, дропаут або крос-валідація.

ШНМ активно застосовуються у задачах прогнозування цін через їх здатність розпізнавати складні нелінійні залежності в даних. Особливо ефективними є такі модифікації як рекурентні нейронні мережі (RNN), довготривала короткострокова пам'ять (LSTM) та трансформери.

Отже, штучні нейронні мережі – потужний інструмент в аналітиці та прогнозуванні. У наступних розділах буде розглянуто класифікацію типів ШНМ, методи їх навчання, а також їх застосування у задачах прогнозування економічних показників.

## 2.2 Основи машинного навчання як фундаменту для нейромереж

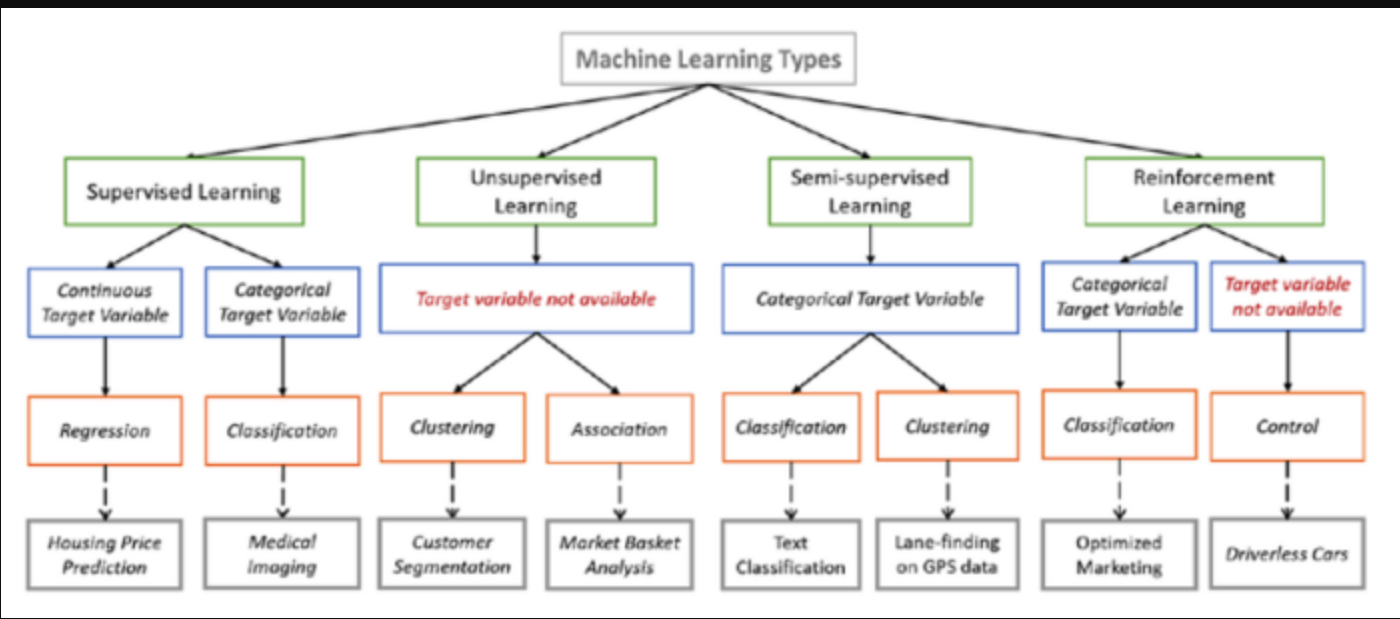
Машинне навчання (ML) є невід’ємною складовою штучного інтелекту та одночасно основою для побудови та навчання штучних нейронних мереж. Цей підхід дозволяє алгоритмам «вчитися» на основі даних, без явного програмування кожного кроку розв’язання задачі. У загальному сенсі, машинне навчання – це здатність комп’ютерних систем знаходити закономірності у даних і використовувати ці закономірності для прийняття рішень, прогнозів чи класифікацій.

машинне навчання визначається як: Метод аналізу даних, який автоматично будує аналітичну модель, використовуючи алгоритми, що ітеративно навчаються на даних. Це дозволяє комп’ютерам знаходити приховані інсайти без явного програмування, де шукати.

Це визначення добре підкреслює ключову перевагу ML – здатність до самостійного вдосконалення на основі досвіду.

Класифікація видів машинного навчання агалом машинне навчання поділяється на три основні типи:

* **Навчання з учителем (Supervised Learning)**
* **Навчання без учителя (Unsupervised Learning)**
* **Підкріплювальне навчання (Reinforcement Learning)**

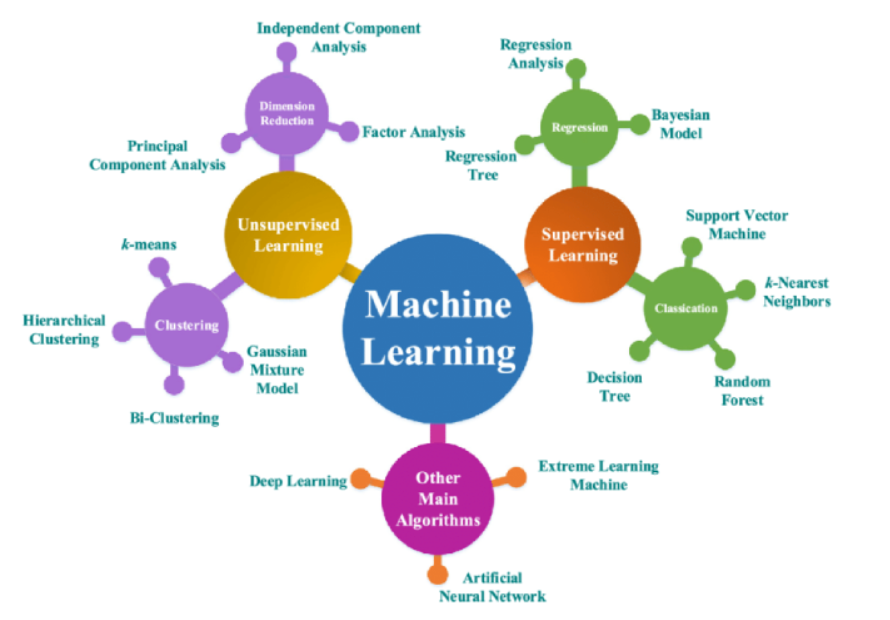
На **Рис. 2.3** зображено порівняння основних підходів ML. У таблиці наведено їхні характеристики, застосування та приклади.

**Рис. 2.3 – Схема типів машинного навчання**

Навчання з учителем. Це найбільш поширений тип ML, при якому алгоритм навчається на заздалегідь маркованих даних. Іншими словами, кожному прикладу у вхідному наборі дано правильну відповідь. Система вивчає зв’язки між вхідними даними та виходами, щоб у майбутньому застосовувати ці зв’язки до нових випадків.

**Приклади задач:**

* Класифікація електронних листів як «спам» або «не спам»
* Прогнозування вартості нерухомості за площею, розташуванням тощо

На **Рис. 2.4** показано, як система використовує позначені дані для побудови моделі.

**Рис. 2.4 – Схема роботи алгоритму з учителем**

Навчання без наглядача (Unsupervised learning). Це різновид машинного навчання, де алгоритм опановує інформацію на основі масиву даних без маркування. Основна ціль алгоритму – самостійно виявити шаблони й віднайти структури в даних, згрупувати об'єкти за їхньою подібністю та, можливо, передбачити майбутні тенденції.

Навчання без наглядача часто використовується для розподілу даних на кластери за спільними характеристиками (кластеризація), скорочення кількості властивостей даних без втрати значущої інформації (зменшення розмірності) та виявлення нетипових даних або відхилень від загальної тенденції (виявлення аномалій).

До алгоритмів навчання без наглядача зараховують:

* Метод k-середніх - алгоритм кластеризації, який розділяє дані на наперед задану кількість груп.
* Алгоритм наближених k-середніх. Більш оптимізована версія попереднього алгоритму, призначена для обробки великих обсягів даних.
* Аналіз головних компонент. Метод зменшення розмірності, що відбирає найважливіші атрибути з масивів даних.

Також можливе поєднання двох вищезгаданих методів (з учителем та без нього). Такий гібрид, умовно, може бути позначений як "напівнаглядне навчання" (або змішане навчання).

У цьому випадку алгоритм вивчає дані, що містять як розмічені, так і не розмічені властивості. Позначені приклади служать для тренування алгоритму, як у випадку навчання з учителем, тоді як не розмічені використовуються для пошуку закономірностей у даних, подібно до навчання без наглядача.

"Напівнаглядне навчання" може показати високу ефективність у випадках, коли наявний незначний набір розмічених даних та великий обсяг не розмічених. Наприклад, під час самонавчання або взаємодії ML-моделей.

**Навчання з підкріпленням (Reinforcement Learning, RL)** – це потужна галузь машинного навчання, де інтелектуальний агент навчається ухвалювати оптимальні рішення шляхом безпосередньої взаємодії з динамічним **довкіллям**. Уявіть собі дресирування домашнього улюбленця: правильна поведінка заохочується ласощами (позитивне підкріплення), а небажана – ігнорується або викликає несхвалення (негативне підкріплення чи його відсутність). Подібним чином, в RL агент виконує певні **дії** у відповідь на поточний **стан** довкілля і отримує чисельний сигнал – **"винагороду"** або **"штраф"**. Головна мета агента – не просто отримати миттєву винагороду, а розробити таку **стратегію (політику)** дій, яка максимізує сукупну, довгострокову винагороду. Цей процес відбувається методом проб і помилок, де агент поступово "розуміє", які дії призводять до кращих результатів у різних ситуаціях. На відміну від інших парадигм машинного навчання, таких як навчання з учителем, де алгоритм отримує готові пари "вхід-правильний вихід", або навчання без учителя, де алгоритм шукає приховані структури в нерозмічених даних, RL агент вчиться самостійно, спираючись виключно на зворотний зв'язок від своїх дій.

Ключові переваги навчання з підкріпленням:

Навчання з підкріпленням пропонує низку унікальних переваг, що роблять його незамінним інструментом для вирішення широкого кола завдань:

* **Здатність до самонавчання та автономність:** Мабуть, найважливішою перевагою є те, що RL-моделі не потребують величезних, ретельно розмічених наборів даних. Агент вчиться, досліджуючи довкілля та отримуючи зворотний зв'язок. Це особливо цінно в ситуаціях, де збір або розмітка даних є надто дорогим, трудомістким або просто неможливим. **Наприклад**, ігровий ШІ може зіграти мільйони партій проти самого себе, поступово вдосконалюючи свою стратегію, не потребуючи даних про ігри людей-чемпіонів (хоча такі дані можуть прискорити навчання). Ще один приклад – робот, що вчиться ходити: він може починати з хаотичних рухів, але з часом, отримуючи позитивну винагороду за кожен успішний крок і негативну за падіння, він самостійно оптимізує свою ходу.
* **Ефективність у надзвичайно складних задачах:** RL демонструє вражаючі результати в задачах, де простір можливих рішень є гігантським, а оптимальну стратегію складно або й неможливо формалізувати за допомогою традиційних алгоритмів. Це стосується як ігрових середовищ, так і реальних застосунків. **Наприклад**, система AlphaGo від DeepMind, навчена за допомогою RL, змогла перемогти найкращих у світі гравців у Го – гру з астрономічною кількістю можливих ходів. Іншим прикладом є оптимізація роботи центрів обробки даних, де RL-агенти можуть керувати системами охолодження для зменшення енергоспоживання, враховуючи безліч динамічних факторів. Також RL використовується в робототехніці для навчання маніпуляторів складним діям, таким як збирання об'єктів або виконання точних операцій.
* **Адаптивність та гнучкість до змін:** RL-алгоритми здатні динамічно адаптуватися до змін у довкіллі або в самій задачі. Якщо умови змінюються, агент може скоригувати свою стратегію на основі нового досвіду, щоб продовжувати досягати максимальної винагороди. Це робить їх ідеальними для систем, що функціонують у непередбачуваних або еволюціонуючих середовищах. **Наприклад**, система управління трафіком на основі RL може адаптуватися до змін у потоках автомобілів протягом дня або до непередбачених подій, таких як дорожні роботи чи аварії, оптимізуючи роботу світлофорів для мінімізації заторів. Персоналізовані системи рекомендацій також можуть використовувати RL, щоб адаптуватися до мінливих інтересів користувача в реальному часі, пропонуючи більш релевантний контент.

Таким чином, навчання з підкріпленням відкриває шлях до створення по-справжньому інтелектуальних систем, здатних навчатися, адаптуватися та вирішувати завдання, які раніше вважалися прерогативою людського інтелекту. Його застосування охоплює все ширший спектр галузей, від ігор та робототехніки до фінансів, медицини та управління складними системами.

Етапи побудови моделі машинного навчання

Якщо узагальнити процес побудови системи машинного навчання, він складається з наступних етапів:

* **Збір даних** – отримання максимально повного та репрезентативного набору даних.
* **Попередня обробка** – очищення, нормалізація, кодування.
* **Вибір моделі** – визначення алгоритму навчання (дерева рішень, нейронні мережі тощо).
* **Навчання моделі** – власне побудова зв’язків.
* **Оцінка точності** – перевірка на тестових даних.
* **Тестування та впровадження** – реальна перевірка роботи системи.

## 2.3 Типи архітектур нейронних мереж у задачах прогнозування

# РОЗДІЛ 3. ПРАКТИЧНЕ ЗАВДАННЯ

## 3.1. Вибір програмних засобів

Одним із найважливіших моментів при створенні проекту є вибір середовища розробки. З огляду на це, було обрано програмне забезпечення Visual Studio Code (VS Code).

Visual Studio Code (VS Code) – це текстовий редактор, розроблений Microsoft для операційних систем Windows, Linux та macOS. Він позиціонується як "легкий" інструмент для написання коду, призначений для розробки веб-додатків і хмарних сервісів на різних платформах. Редактор містить вбудований налагоджувач, інтеграцію з Git, підсвічування синтаксису, технологію IntelliSense та інструменти для рефакторингу коду. VS Code надає широкі можливості для персоналізації, включаючи теми оформлення, комбінації клавіш і файли конфігурації. Цей редактор розповсюджується безкоштовно і розробляється як програмне забезпечення з відкритим вихідним кодом, хоча його готові версії розповсюджуються під певною ліцензією [25].

Visual Studio Code – це редактор вихідного коду, який підтримує різні мови програмування, такі як C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, а також Rust. Крім того, VS Code має підтримку IntelliSense для JavaScript, TypeScript, JSON, CSS та HTML, а також засоби для налагодження Node.js [25].

Замість традиційної системи проектів, редактор дозволяє користувачам відкривати один або кілька каталогів, які потім можна зберегти як робочі області для подальшої роботи. Це робить його універсальним інструментом для написання коду будь-якою мовою програмування. Він підтримує велику кількість мов та надає різні функції, які варіюються в залежності від конкретної мови. Непотрібні файли та папки можна виключити з структури проекту за допомогою налаштувань. Інтерфейс програмного забезпечення VS Code зображений на рисунку 3.1.

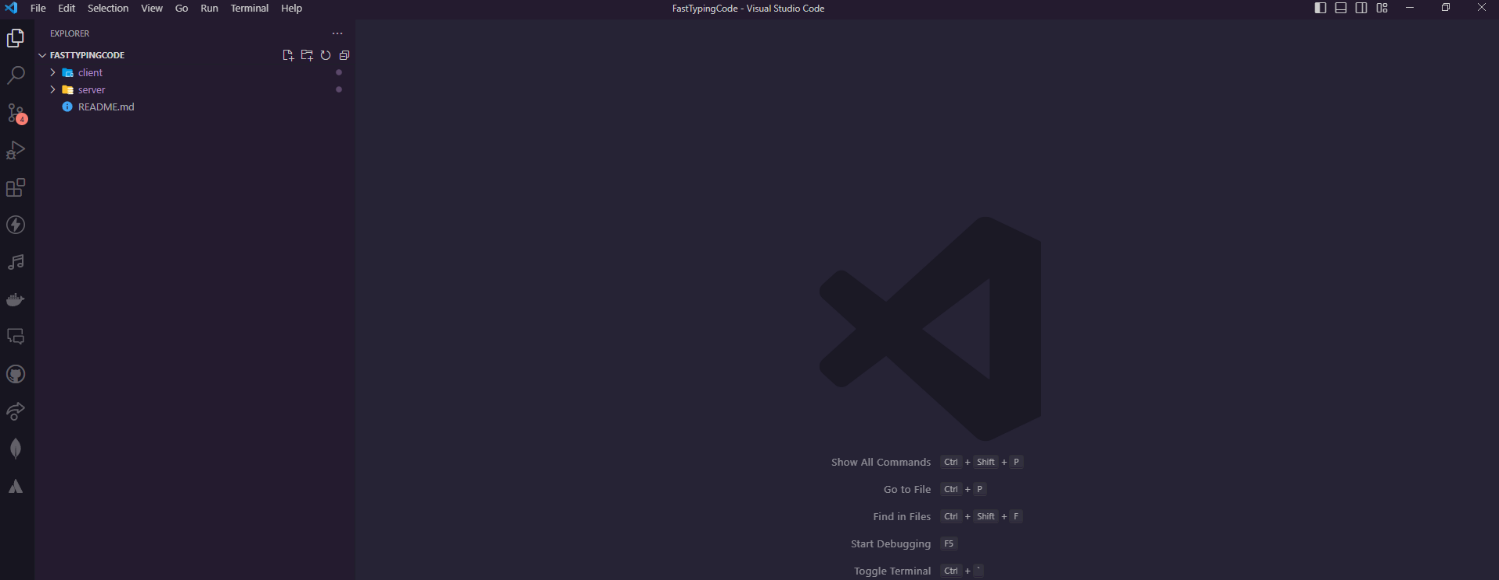


Рисунок 3.1 – Програмне середовище *Visual Studio Code*

Обраною мовою для створення дипломного проєкту було використано JavaScript.

JavaScript — це мова програмування, що слугує інструментом для розробників, що створюють інтерактивні веб-сторінки. Вона дає змогу покращити взаємодію користувача з сайтом, від оновлення стрічок новин у соціальних мережах до демонстрації анімації та інтерактивних карт [26].

JavaScript активно використовується як мова програмування в розробці скриптів, що виконуються на клієнтській стороні. Це робить JavaScript однією з ключових технологій в інтернеті. За допомогою JavaScript можна створювати різні елементи, що супроводжують перегляд веб-сторінок, наприклад, каруселі зображень, випадаючі меню або динамічне змінювання кольору елементів.

JavaScript дозволяє розробникам створювати динамічні, інтерактивні та реактивні елементи веб-сторінок, що значно розширює функціональність і покращує привабливість веб-додатків. Він виступає важливим інструментом для веб-розробки, дозволяючи реалізовувати різні функції та покращувати взаємодію користувача з веб-додатками [26].

Усі мови програмування працюють через трансляцію синтаксису, написаного англійською мовою, у машинний код, який потім виконується операційною системою. JavaScript можна віднести до скриптових або інтерпретованих мов загалом. У випадку з JavaScript, його код інтерпретується, що означає його безпосереднє перетворення у машинний код за допомогою движка JavaScript. На противагу цьому, у багатьох інших мовах програмування компілятор обробляє весь код і перетворює його у машинний код на окремому етапі, перед виконанням. Отже, всі скриптові мови є мовами програмування, але не всі мови програмування є скриптовими [26].

Двигун JavaScript — це програмне забезпечення, що відповідає за виконання коду JavaScript. Спочатку двигуни JavaScript були лише інтерпретаторами, проте сучасні двигуни застосовують JIT-компіляцію або компіляцію під час виконання для покращення продуктивності [26].

З метою розширення можливостей JavaScript, зокрема, для додавання типізації, також використовувався TypeScript.

TypeScript — це мова програмування, що є розширенням JavaScript. Вона надає статичну типізацію, яка дозволяє визначити типи для змінних, параметрів функцій та їх результатів. TypeScript компілюється в чистий JavaScript, отже, код, написаний на TypeScript, можна виконати в будь-якому середовищі, що підтримує JavaScript.

Основні риси TypeScript:

1. Статична типізація: TypeScript дає можливість визначати типи для змінних, функцій, об'єктів та інших фрагментів коду. Це сприяє виявленню помилок на етапі розробки та полегшує рефакторинг коду.

2. Підтримка класів і об'єктно-орієнтованого програмування: TypeScript підтримує класи, наслідування, інтерфейси та інші концепції ООП, що дозволяє створювати більш структурований код.

3. Додаткові функції: TypeScript надає деякі додаткові можливості порівняно з чистим JavaScript, наприклад, перерахування (enums), типи-об'єднання (union types), типи-перетини (intersection types) та інші.

4. Інструменти розробки: TypeScript інтегрується з різними редакторами коду та інструментами розробки, які забезпечують автодоповнення коду, перевірку типів у реальному часі та інші корисні функції.

## 3.2. Оцінка основних бібліотек та фреймворків

Основною бібліотекою для реалізації дипломного проєкту є ReactJS, оскільки це продукт від компанії Facebook, який суттєво полегшує процес конструювання та керування елементами DOM. Спочатку цю бібліотеку створили та використовували винятково для внутрішніх потреб компанії (наприклад, для розробки їх соціальної мережі). З часом вона стала доступною широкому колу розробників.

У своєму класичному вигляді React є всього лиш бібліотекою для створення нескладних клієнтських застосунків, навіть не лише для односторінкових застосунків (Single Page Application), бо вона не має вбудованих механізмів маршрутизації (Routing) та інших додаткових можливостей. Нещодавно React отримав офіційну можливість використовувати Context API для реалізації доволі простого управління станом (state management) [27].

Головні переваги *ReactJS* [27]:

1. Синтаксис React надзвичайно зрозумілий для розробників, які вже знайомі з JavaScript та HTML.
2. Розробка у React підтримує функціональний підхід до написання компонентів.
3. Важливим принципом, що активно використовується у React, є Immutability - заборона безпосередньої зміни стану або даних у компонентах, а замість цього - створення нових екземплярів на основі вже наявних [27].
4. Підхід до написання додатків у React є компоненто-орієнтованим, де кожен елемент – це окремий компонент.
5. Підхід до створення екосистеми додатків у React полягає у поступовому додаванні необхідних бібліотек і пакетів.
6. React зберігає та опрацьовує стан DOM у віртуальному DOM, що робить його швидким при відображенні [27].
7. Розробка у React супроводжується великою кількістю інформації про помилки та ключові проблеми.
8. У React можна інтегрувати будь-яку JavaScript-бібліотеку разом із великою кількістю спеціалізованих бібліотек.
9. React має документацію з багатьма прикладами та детальним описом, включно з українською версією.
10. У React існує активна спільнота розробників.
11. Є багато інструментів, що спрощують процес роботи та відлагодження у React*.*
12. React регулярно оновлюється та удосконалюється.
13. React підтримується Facebook [27].

Недоліки бібліотеки *ReactJS*:

1. *React* "з коробки" не містить практично жодних додаткових можливостей, крім роботи з *DOM*, важливо мати знання або вміти ефективно використовувати пошук потрібних та надійних бібліотек для вирішення різних завдань [27].
2. У простоті React може критися його головний недолік – компонування, структурування файлів у React-додатках, масштабування додатку загалом та інших нюансів, є ризик неправильної реалізації. Це може призвести до складної структури додатку з плином часу (і збільшенням обсягу коду), що вимагатиме багато зусиль для підтримки та подальшого розширення [27].

Для управління станом програми застосовується бібліотека Redux Toolkit.

Redux Toolkit – це офіційний набір інструментів, що допомагає спростити розробку з використанням Redux, надаючи додаткові інструменти та утиліти. Його створили з метою зробити роботу з Redux простішою, продуктивнішою та ефективнішою.

Основні риси та переваги Redux Toolkit:

1. Спрощений синтаксис: Redux Toolkit надає спеціальні функції, як-от createSlice та createAsyncThunk, що допомагають зменшити обсяг коду та полегшити створення reducer.
2. Автоматичне керування станом: Redux Toolkit автоматично генерує ідентифікатори дій та створює редьюсери з вбудованою обробкою сполучень (reducers), що дозволяє зосередитись на логіці додатку, а не на деталях редукторів.
3. Вбудована підтримка асинхронних операцій: Redux Toolkit має вбудовану підтримку асинхронних операцій через функцію createAsyncThunk, що дозволяє виконувати асинхронні запити та опрацьовувати результати [3].
4. Для реалізації маршрутизації в проекті використовується бібліотека React Router. Вона дає змогу керувати навігацією в додатку, визначати, які компоненти мають відображатися під час зміни URL, та робить розробку багатосторінкових додатків в React простішою.

Головні причини використання React Router:

1. Маршрутизація: React Router дає змогу визначити набір маршрутів (routes), пов'язаних з певними компонентами.
2. Навігація: React Router надає компоненти, як-от Link та NavLink, які дають змогу створювати посилання, що відповідають маршрутам в додатку.
3. React Router підтримує параметризовані маршрути, що дають змогу передавати динамічні дані у шляху.
4. Вкладені маршрути: React Router дає змогу створювати вкладені маршрути, коли компоненти можуть мати власну маршрутизацію [6].

При створенні серверної частини використано фреймворк NestJS. Він пропонує розширені можливості для побудови масштабованих та ефективних серверних додатків.

Основні переваги використання NestJS полягають у модульному підході для розробки, що дає змогу розбити функціонал на окремі модулі. Також фреймворк пропонує однорідний стиль коду, що сприяє кращій зрозумілості та співпраці між розробниками. Ще він побудований на основі TypeScript, що дозволяє використовувати строгу типізацію. NestJS має велику кількість розширень, які дозволяють розширювати його функціональність та додавати підтримку різних інструментів та технологій.

## 3.3. Створення Google client ID

Щоб впровадити авторизацію через Google, необхідно отримати унікальний ідентифікатор клієнта (client ID). Для цього слід спершу відвідати веб-сайт console cloud google та пройти процедуру аутентифікації, використовуючи власну електронну адресу. Наступним кроком буде створення нового проекту. Процес створення нового проекту візуалізовано на зображенні, що позначене як рисунок 3.2.

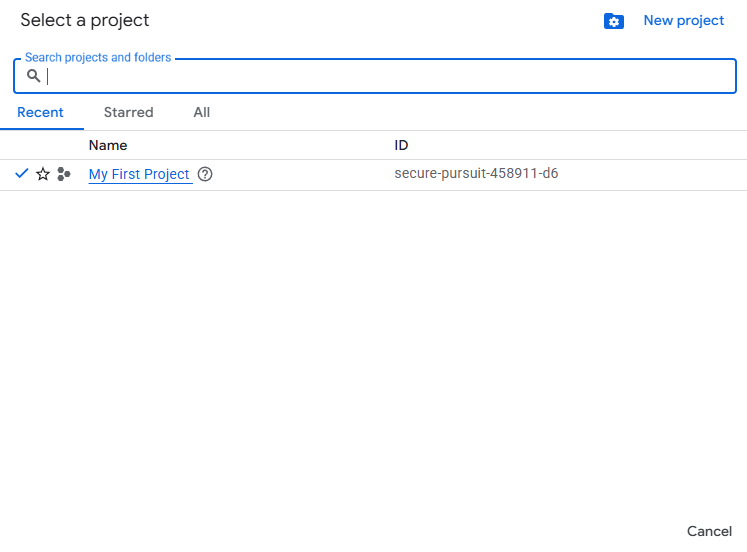


Рисунок 3.2 – Створення нового проекту

Заповнення контактної інформації легко бачити на рисунку 3.3 та на рисунку 3.4

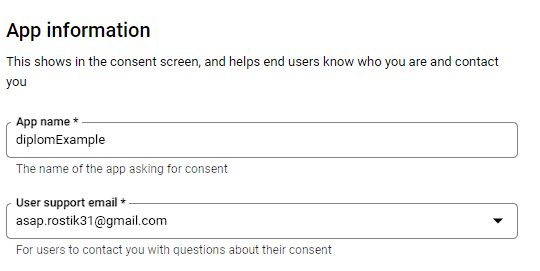


Рисунок 3.3 – Назва проекту, та власна електронна пошта для підтримки користувачів

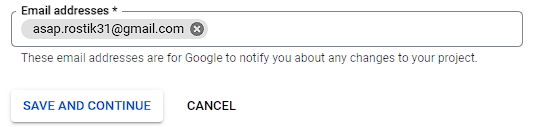


Рисунок 3.4 – Контактна інформація розробника

Вибір потрібних *API* (*Application* *Programming* *Interface*) для створення *Google* авторизації зображено на рисунку 3.5

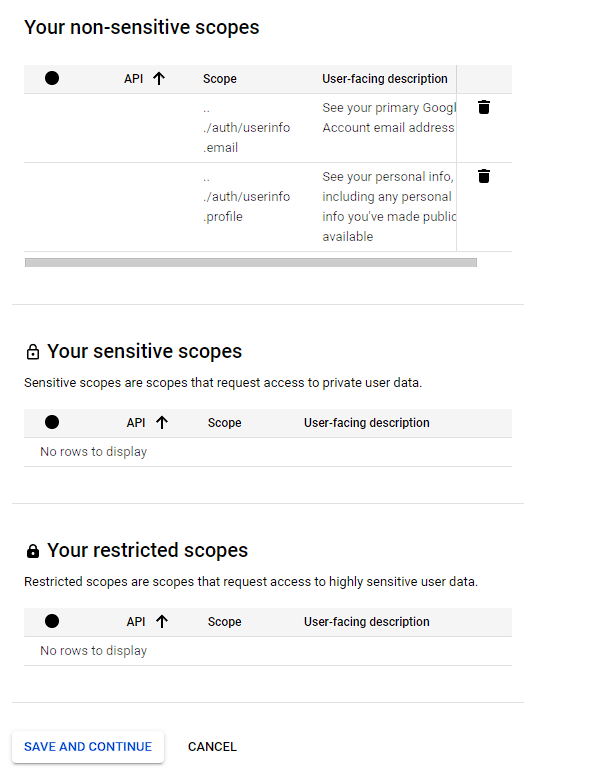


Рисунок 3.5 – Вибір потрібних *API (Application Programming Interface)* для створення *Google* авторизації

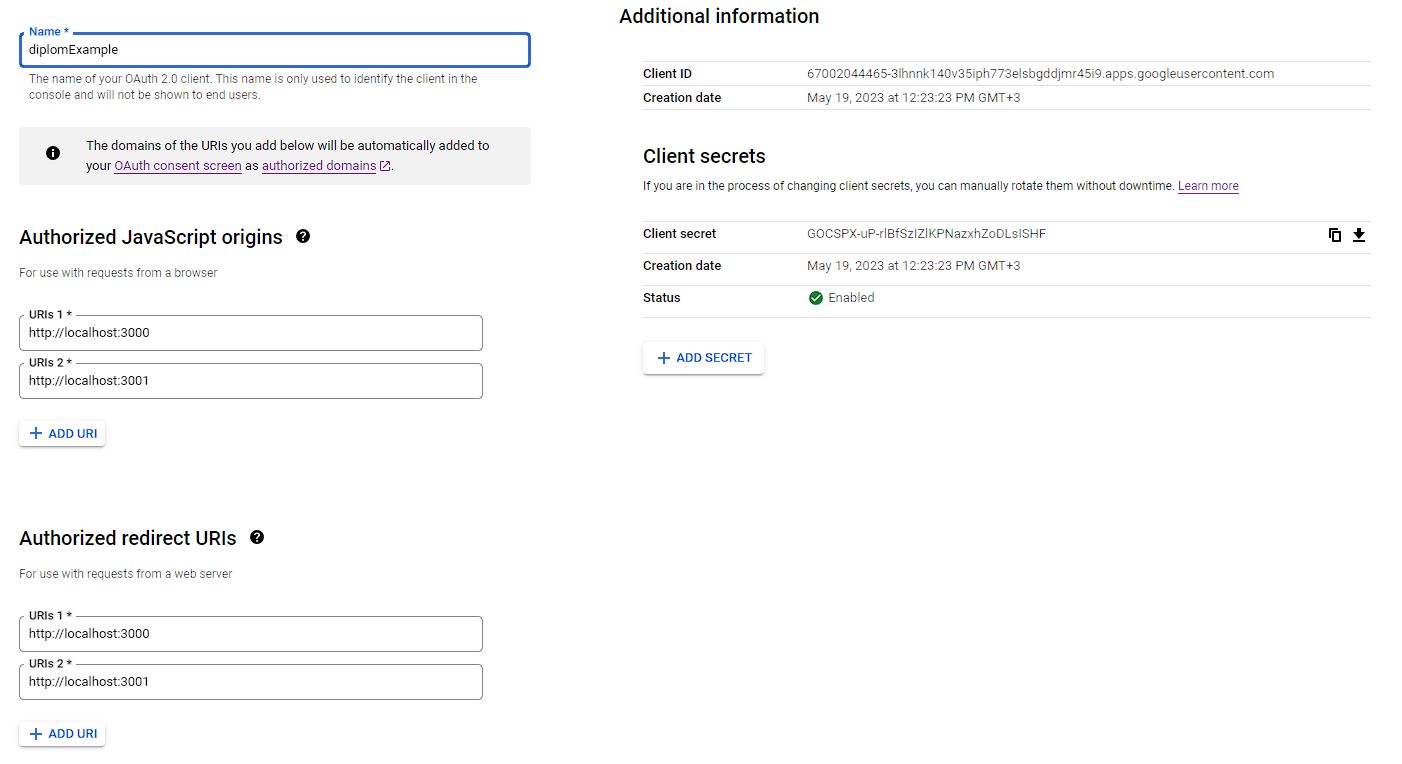
Легко помітити наявність ідентифікатора клієнта (*client ID*) на рисунку 3.6.

Рисунок 3.6 – Стоврення *сlient ID*

## 3.4. Структура проєкту

Структура проєкту розпадається на дві ключові складові: клієнтську та серверну. Структура клієнтської частини продемонстрована на малюнку 3.7, тоді як структура серверної частини представлена на малюнку 3.8.

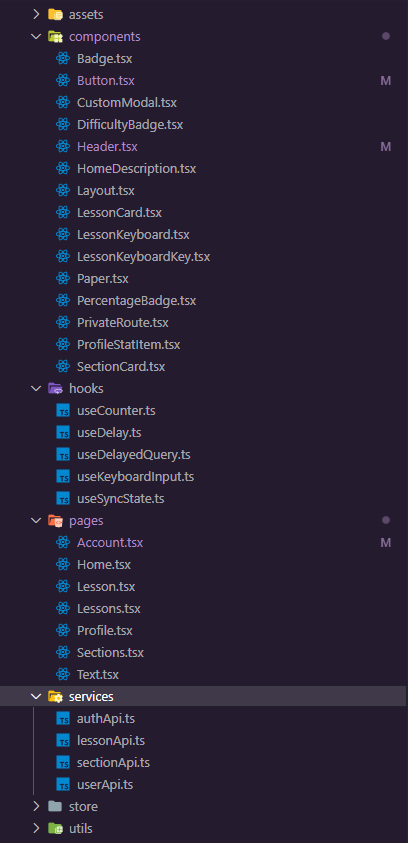


Рисунок 3.7 – Структура клієнтської частини

Клієнтська частина містить такі ключові теки:

1. assets – тут зберігаються картинки, що безпосередньо використовуються у проєкті;
2. components – вміщує різні компоненти, залучені на головних сторінках проєкту;
3. hooks – тут розміщуються функції, що мають доступ до внутрішніх властивостей ReactJS;
4. services – призначено для зберігання API;
5. store – відповідає за управління станом застосунку (реалізовано за допомогою Redux Toolkit).

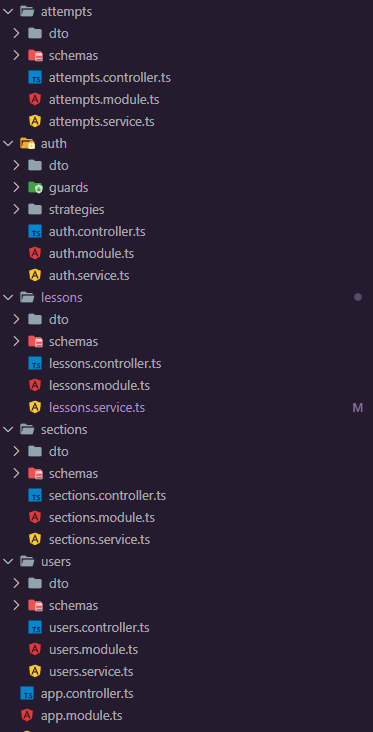


Рисунок 3.8 – Структура серверної частини

Серверна частина організована навколо ключових директорій:

1. attempts – відповідає за обробку та пошук спроб, реалізованих в системі;
2. auth – забезпечує механізми аутентифікації, інтегровані з Google;
3. lesson – керує операціями створення та пошуку лекцій в системі;
4. sections – відповідає за створення та пошук розділів контенту;
5. users – забезпечує функціональність для створення користувачів, пошуку користувачів та отримання їхніх профілів.

## 3.5. Реалізація перевірки введених даних

Для забезпечення зручності основні функції додатку було розбито на спеціальні хуки. Одним з них є хук useKeyboardInput, що відповідає за обробку введення з клавіатури. Головне завдання цього хука – відстежувати символ, який зараз вводить користувач, та порівнювати його з передбачуваними символами.

Всередині хука працюють дві функції: changeCurrentChar та cleanUp. changeCurrentChar відповідає за оновлення поточного символу та реалізацію візуальних змін, що з ним пов'язані. Функція cleanUp видаляє деякі класи з поточного символу.

Опис функції useKeyboardInput:

export const useKeyboardInput = (lessonCode: React.MutableRefObject<HTMLPreElement | null>) => {

const [currentChar, setCurrentChar] = useState(lessonCode.current?.children[0]);

const [typed, setTyped] = useState({

total: 0,

wrong: 0,

});

const [isLessonEnded, setIsLessonEnded] = useState(false);

const changeCurrentChar = (newVal: Element) => {

//this check if next item === undefined, end lesson

if (!newVal && lessonCode.current) {

lessonCode.current.classList.add('overflow-y-hidden');

return setIsLessonEnded(true);

}

setCurrentChar(newVal);

newVal.classList.add(`bg-green-500`);

newVal.classList.add(`text-white`);

newVal.scrollIntoView({ block: 'center', behavior: 'smooth' });

};

const cleanUp = (s: string[]) => s.forEach((i) => currentChar?.classList.remove(i));

useEffect(() => {

setCurrentChar(lessonCode.current?.children[0]);

}, [lessonCode]);

Метод handleInput відповідає за реагування на події, які надходять від клавіатури. Він визначає, яку клавішу було натиснуто, порівнює її з поточним літерним знаком, та коригує класи HTML-елементів, керуючись правильністю введеного символу.

Опис функції *handleInput*:

const handleInput = (e: KeyboardEvent<HTMLInputElement>) => {

if (!currentChar || !lessonCode.current) return;

const chars = Array.from(lessonCode.current.children);

const currCharIndex = chars.indexOf(currentChar);

let isError = !!currentChar.classList.contains('bg-red-500');

if (e.key === 'Backspace' && isError) {

currentChar.classList.add(`bg-green-500`);

cleanUp(['bg-red-500']);

return;

} else if (e.key === 'Backspace' && currCharIndex !== 0) {

cleanUp(['text-white', 'text-black', 'bg-red-500', 'bg-green-500']);

let skippedChars = 1;

while (chars[currCharIndex - skippedChars].textContent === '\t') skippedChars++;

return changeCurrentChar(chars[currCharIndex - skippedChars]);

} else if (e.key === 'Backspace' && currCharIndex === 0) {

return;

}

if (isError || e.key === 'Shift') return;

if (e.key === 'Enter' && currentChar.textContent === '\n') {

cleanUp(['bg-green-500']);

let skippedChars = 1;

while (chars[currCharIndex + skippedChars].textContent === '\t') skippedChars++;

return changeCurrentChar(chars[currCharIndex + skippedChars]);

}

setTyped((prev) => ({ ...prev, total: prev.total + 1 }));

if (e.key === currentChar.textContent) {

cleanUp(['text-white', 'bg-red-500', 'bg-green-500']);

currentChar.classList.add(`text-black`);

changeCurrentChar(chars[currCharIndex + 1]);

} else {

cleanUp(['bg-green-500']);

currentChar.classList.add(`bg-red-500`);

setTyped((prev) => ({ ...prev, wrong: prev.wrong + 1 }));

}

};

return { currentChar, setCurrentChar, handleInput, typed, isLessonEnded };

};

## 3.6. Реалізація функції лічильника часу

Щоб користувач бачив тривалість виконання вправи, було реалізовано функцію, що показує час, котрий минув з початку роботи над нею.

Опис функції *useCounter*:

import { useEffect, useState } from 'react';

export const useCounter = (isActive: boolean) => {

const [counter, setCounter] = useState({

second: '00',

minute: '00',

time: 0,

});

const handleChangeCounter = (newTime: number) => {

const second = String(newTime % 60);

const minute = String(Math.floor(newTime / 60));

const computedSecond = second.length === 1 ? `0${second}` : second;

const computedMinute = minute.length === 1 ? `0${minute}` : minute;

setCounter({

second: computedSecond,

minute: computedMinute,

time: newTime,

});

};

useEffect(() => {

let timerId: NodeJS.Timeout;

if (isActive) {

timerId = setTimeout(() => handleChangeCounter(counter.time + 1), 1000);

}

return () => clearTimeout(timerId);

}, [isActive, counter]);

return {

counter,

restartCounter: () => handleChangeCounter(0),

};

};

Функція handleChangeCounter отримує новий час (newTime) та змінює стан лічильника. Вона обчислює секунди та хвилини, далі форматує їх у рядки, застосовуючи умовні оператори для додавання початкових нулів. Після цього вона оновлює стан лічильника з новими значеннями.

За допомогою useEffect створюється таймер, що кожну секунду активує функцію handleChangeCounter, якщо лічильник активований (isActive дорівнює true). Таймер збільшує поточне значення counter.time на одну секунду. Таймер повертає унікальний ідентифікатор, який використовується для зупинки таймера за допомогою clearTimeout, коли змінюється активність або значення лічильника.

## 3.7. Реалізація швидкості та точності набору тексту

Використано хук useEffect для обчислення швидкості набору тексту (WPM) та точності. Слова за хвилину (wpm) розраховуються як кількість коректно набраних символів, поділена на 5 (передбачається, що слово містить 5 літер), помножена на 60 (для приведення до показника за хвилину).

Точність (accuracy) обчислюється як відношення кількості правильно введених символів до загальної кількості натискань, помножене на 100, що дозволяє отримати відсотковий показник.

Приклад коду:

useEffect(() => {

const wpm = +(

((typed.total - typed.wrong) / 5 / (counter.time + 0.1)) \*

60

).toFixed(0);

const accuracy = +(

((typed.total - typed.wrong) / (typed.total + 0.00001)) \*

100

).toFixed(0);

indicators.set({

wpm: wpm > 0 ? wpm : 0,

accuracy: accuracy === 0 ? 100 : accuracy,

});

// eslint-disable-next-line react-hooks/exhaustive-deps

}, [typed, counter.time]);

## 3.8. Реалізація Google авторизації

Клас AuthService, що виконує функції автентифікації користувачів, у тому числі забезпечує вхід за допомогою Google. Застосування декоратора @Injectable() для визначення AuthService як сервісу, який можна впроваджувати, ілюстровано на малюнку 3.1.

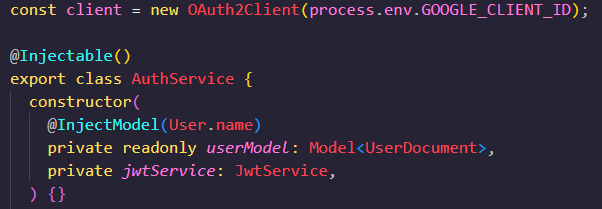


Рисунок 3.9 – Реалізація класу *SectionsService*

Асинхронний метод googleLogin, що здійснює аутентифікацію користувача за допомогою Google OAuth, спираючись на отриманий dto-об'єкт (який містить tokenId) та об'єкт response (що репрезентує HTTP-відповідь).

Видобувається об'єкт із інформацією про користувача (payloadUser) з результату верифікації токену.

Приклад коду функції *googleLogin*:

async googleLogin(dto: GoogleLoginDto, response: Response) {

const OAuthResponse = await client.verifyIdToken({

idToken: dto.tokenId,

audience: process.env.GOOGLE\_CLIENT\_ID,

});

const payloadUser = OAuthResponse.getPayload();

const user = await this.userModel

.findOne({ email: payloadUser.email })

.select('-attempts');

if (user) {

const token = this.jwtService.sign({ \_id: user.\_id });

response.cookie('token', token);

return user;

} else {

const newUser = await this.userModel.create({

googleId: payloadUser.sub,

email: payloadUser.email,

name: payloadUser.name,

imageUrl: payloadUser.picture,

});

const token = this.jwtService.sign({ \_id: newUser.\_id });

response.cookie('token', token);

return user;

}

}

}

Основний функціонал класу включає:

Відбувається пошук користувача в базі даних за адресою електронної пошти (email), застосовуючи метод findOne з userModel, і виключаючи поле "attempts" з отриманих даних.

У випадку успішного пошуку користувача, створюється JWT-токен за допомогою jwtService.sign, куди передається \_id користувача. Цей токен додається до відповіді, використовуючи метод response.cookie.

## 3.9. Реалізація секцій

Сервіс володіє двома приватними полями: sectionModel та attemptModel. Вони відповідають за моделі, котрі необхідні для роботи з колекціями Section і Attempt у базі даних. Ці поля створюються за допомогою @InjectModel, щоб їх можна було застосовувати як залежності у конструкторі.

Конструктор приймає об'єкти sectionModel і attemptModel. Вони зберігаються у приватних полях для наступного застосування, як показано на малюнку 3.2.

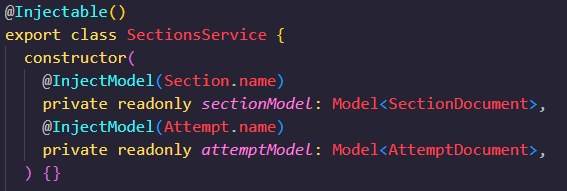


Рисунок 3.10 – Реалізація класу *SectionsService*

Метод create приймає об'єкт createSectionDto типу CreateSectionDto та генерує новий розділ у базі даних. Значення route для новоствореного розділу визначається як властивість name з createSectionDto. Задля одночасної генерації кількох розділів, метод використовує sectionsSeed (масив), вдаючись до Promise.all.

Реалізація функції *create*:

async create(createSectionDto: CreateSectionDto): Promise<any> {

createSectionDto.route = createSectionDto.name;

return Promise.all(

sectionsSeed.map(async (section) => {

await this.sectionModel.create(section);

}),

);

}

Метод findAll повертає всі секції (Section) з бази даних, виключаючи при цьому поле уроків (lessons) з результату.

Опис функції *findAll*:

async findAll(): Promise<Section[]> {

return this.sectionModel.find().select('-lessons');

}

Метод findOne приймає параметри route (шлях до розділу) та reqUser (користувач, що надіслав запит) і повертає відомості про конкретний розділ. Метод здійснює пошук розділу за значенням route, враховуючи пов'язані з ним уроки, але не включаючи взаємозв'язки section та attempts. Після цього, для кожного уроку в розділі, метод витягує всі спроби (attempts) користувача для даного уроку та обчислює середню точність (avgAccuracy) на основі значень accuracy (точність) з цих спроб.

Опис функції *findOne*:

async findOne(route: string, reqUser: User & { \_id: any }): Promise<any> {

const section = await this.sectionModel

.findOne({ route })

.populate('lessons', { section: false, attempts: false });

if (!section) {

throw new NotFoundException('Section not found');

}

await Promise.all(

section.lessons.map(

async (

lesson: Lesson & {

\_id: any;

},

index,

) => {

const attempts = await this.attemptModel.find({

lesson: lesson.\_id,

user: reqUser.\_id,

});

let avgAccuracy = 0;

attempts.forEach((item) => {

avgAccuracy += item.accuracy / attempts.length;

});

section.lessons[index].avgAccuracy = +avgAccuracy.toFixed(2);

},

),

);

return section;

}

}

## 3.9. Реалізація створення користувача

Декоратор @Injectable() застосовано для маркування класу UsersService як сервісу, що може бути впровадженим.

Конструктор класу, що приймає userModel типу Model<UserDocument>, отримує залежність User.name з моделі User за допомогою декоратора @InjectModel, як показано на малюнку 3.3.

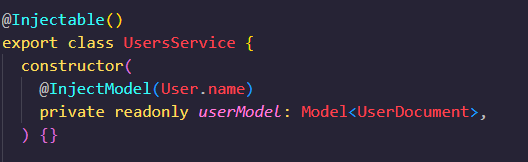


Рисунок 3.11 – Реалізація класу *UsersService*

Асинхронний метод create, що відповідає за створення нового користувача в базі даних. Він базується на об'єкті createUserDto, який отримує як вхідний параметр. Безпосереднє створення користувача відбувається з використанням методу create моделі userModel.

Реалізація функції *create*:

async create(createUserDto: CreateUserDto): Promise<User> {

const createdUser = await this.userModel.create(createUserDto);

return createdUser;

}

Асинхронний метод findAll отримує усіх користувачів з бази даних, використовуючи метод find моделі userModel.

Реалізація функції *findAll*:

async findAll(): Promise<User[]> {

return this.userModel.find().exec();

}

Метод findOne відшукує користувача за його унікальним ідентифікатором id у базі даних. Він додатково наповнює пов'язане поле "attempts" використовуючи метод populate, і наостанок повертає результат виконання цього запиту.

Опис функції *findOne*:

async findOne(id: string): Promise<User> {

return this.userModel.findById(id).populate('attempts').exec();

}

Метод getProfile дістає профіль користувача, орієнтуючись на його ідентифікатор reqUser.\_id з бази даних, одночасно оновлюючи поле "attempts". Після цього він визначає ключові характеристики профілю, зокрема загальний час, кількість уроків, максимальну швидкість, середню швидкість та графік набору тексту. Підсумки обчислень формуються в об'єкті profile, котрий і повертається.

Опис функції *getProfile*:

async getProfile(reqUser: User & { \_id: string }): Promise<any> {

const user = await this.userModel

.findById(reqUser.\_id)

.populate('attempts');

let totalTime = 0;

const profile = {

totalTime: '',

totalLessons: user.attempts.length,

topSpeed: 0,

averageSpeed: 0,

typingChart: [] as { lessonNumber: number; wpm: number }[],

};

user.attempts.forEach((attempt, index) => {

totalTime += attempt.time;

profile.topSpeed = Math.max(profile.topSpeed, attempt.wpm);

profile.averageSpeed += Math.round(attempt.wpm / user.attempts.length);

profile.typingChart.push({ lessonNumber: index + 1, wpm: attempt.wpm });

});

profile.totalTime = [

(totalTime / 60 / 60).toFixed(0),

((totalTime / 60) % 60).toFixed(0),

(totalTime % 60).toFixed(0),

]

.join(':')

.replace(/\b(\d)\b/g, '0$1');

return profile;

}

}

## 3.10. Інструкція для користувача

Коли користувач вперше потрапляє на веб-додаток, він бачить стартову сторінку. Щоб продовжити роботу з тренажером, йому необхідно перейти на сторінку авторизації. Для цього треба натиснути кнопку «SIGN IN», розташовану у верхній частині додатку. Зовнішній вигляд головної сторінки показано на рисунку 3.12.



Рисунок 3.12 – Головна сторінка

Після того, як ви натиснете кнопку *«SIGN IN»*, відкриється сторінка *«Account»*, На ній слід обрати кнопку *«Sign in with Google»,* це показано на рисунку 3.13.

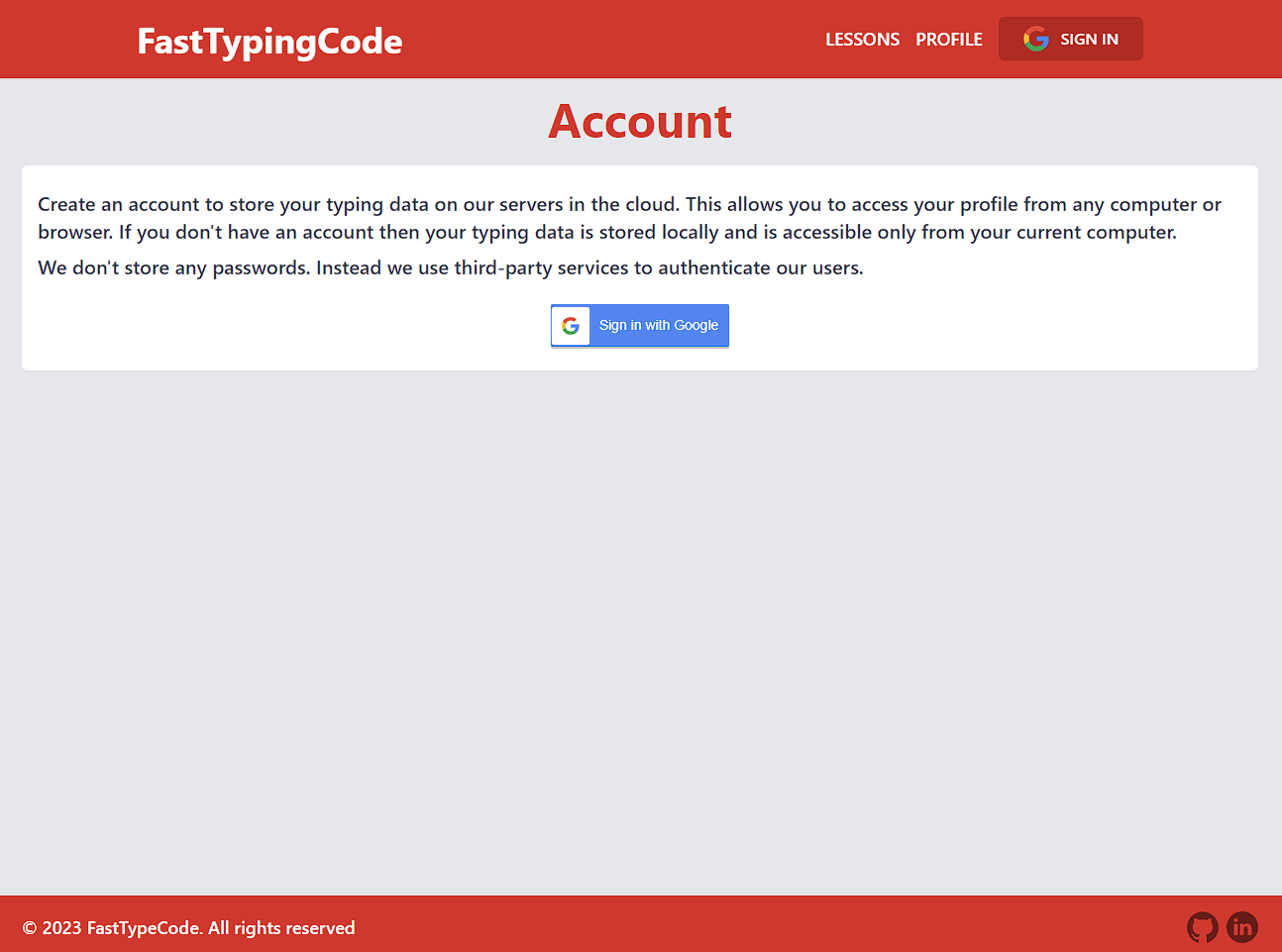


Рисунок 3.13 – Сторінка авторизації

Далі потрібно визначитися з акаунтом, через який буде взаємодіяти користувач з програмою, і де зберігатиметься його особиста статистика. Вибір акаунту показано на рисунку 3.14

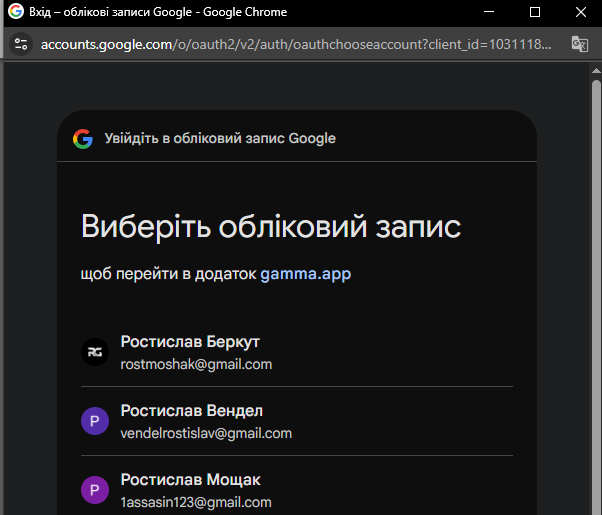


Рисунок 3.14 – Вибір облікового запису

Після успішної авторизації користувач бачить оновлення на сторінці *«Account»*. Там з'являється кнопка *«SING OUT»*, відображається його ім'я та електронна адреса. Відтепер він має доступ до сторінок *«Profile»* та *«Lessons».* Зміни на сторінці *«Account»* показано на рисунку 3.15.

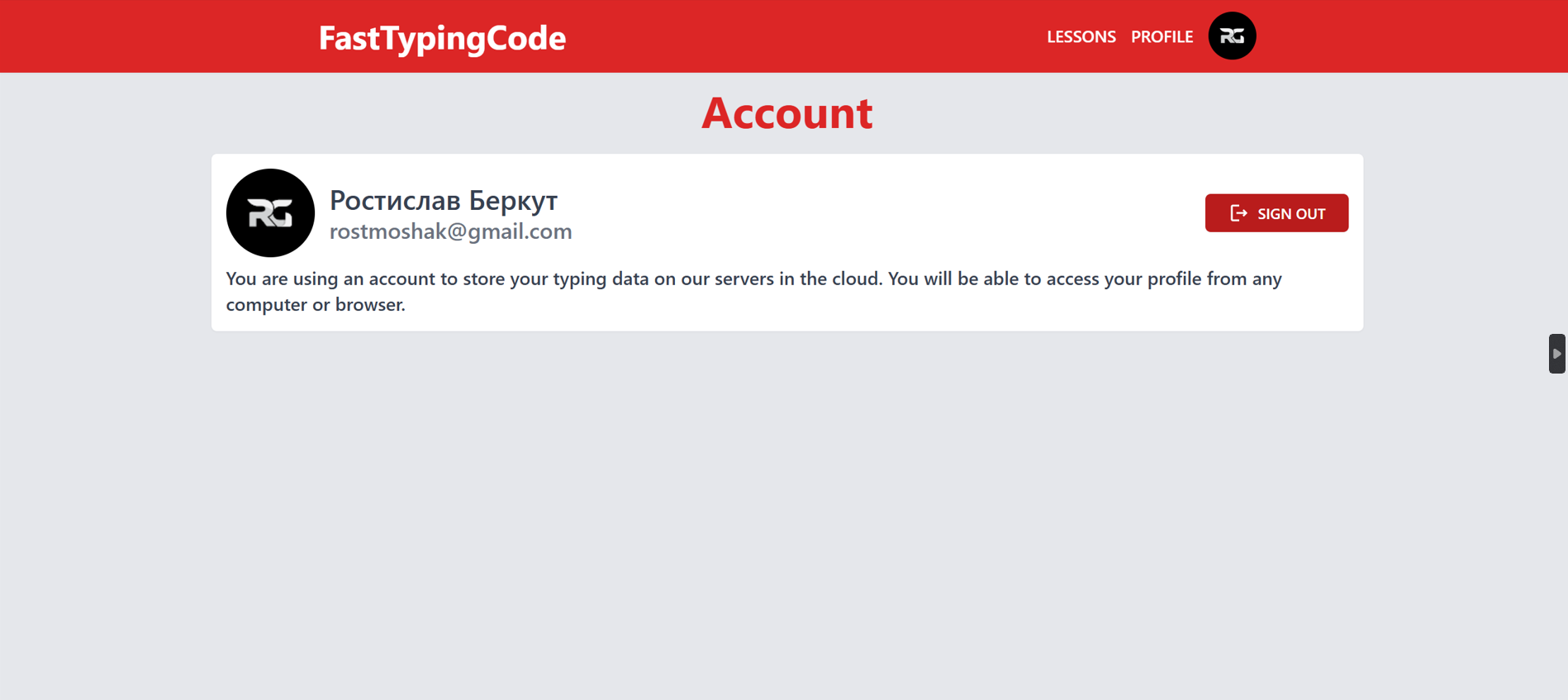


Рисунок 3.15 – Авторизація

Якщо користувач відвідав сторінку *«Lessons»*, йому надається вибір розділу, що його зацікавив. Сторінка *«Lessons»* зображена на рисунку  3.16.



Рисунок 3.16 – Сторінка секцій

Після того, як користувач визначиться з секцією, він/вона може переглянути доступні вправи та обрати ту, що найбільше припаде до душі. Кожна вправа має власний рівень складності. Зовнішній вигляд сторінки конкретної секції демонструє рисунок 3.17.

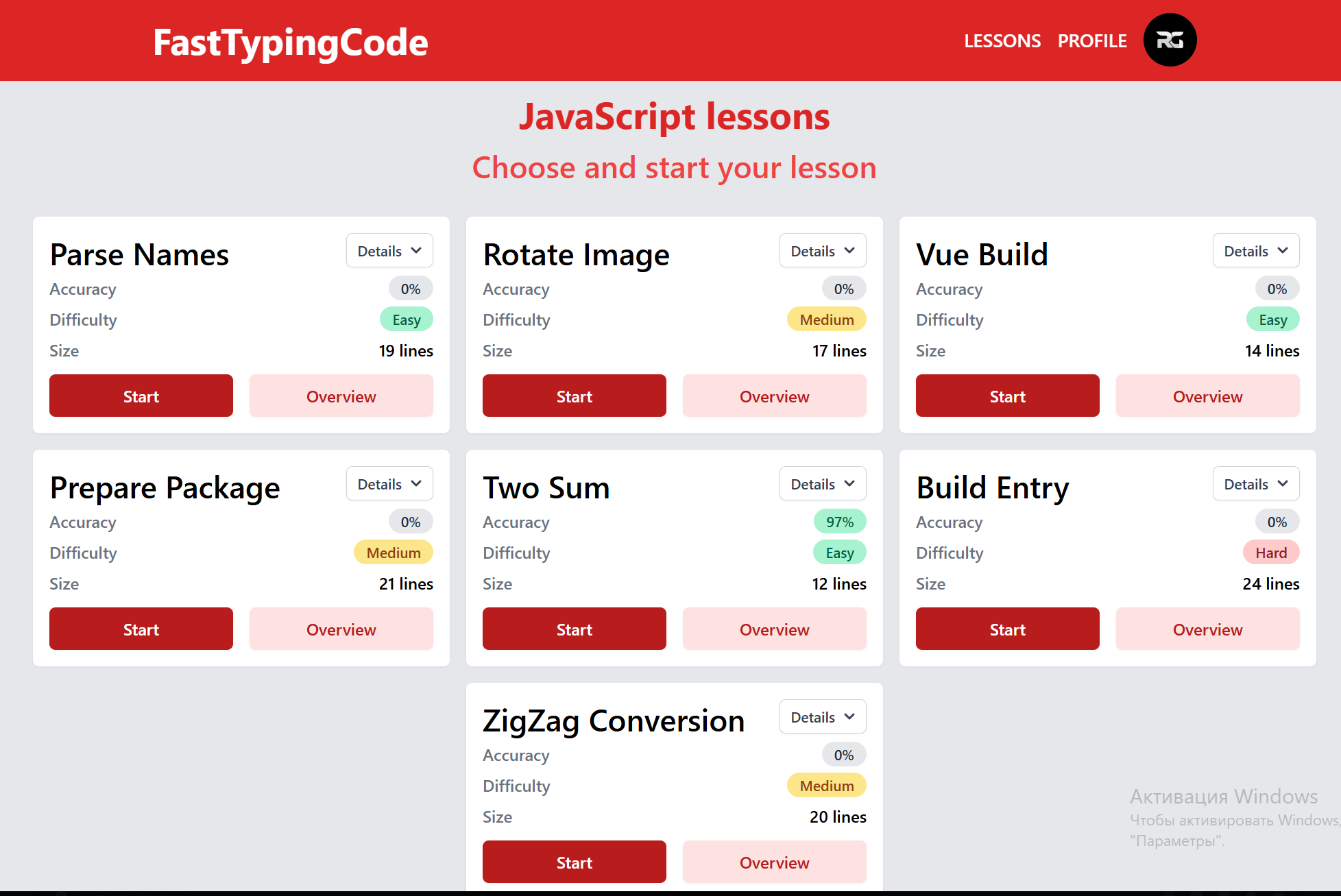


Рисунок 3.17 – Сторінка певної секції

Після вибору вправи, на екрані користувача з'являється текст, який необхідно ввести. Одночасно відображаються: швидкість друку, точність набору, загальна кількість набраних символів, кількість помилок та час, витрачений на завдання. Для повторного проходження вправи з самого початку передбачена кнопка «Restart». Візуальне представлення вправи продемонстровано на рисунку 3.18.

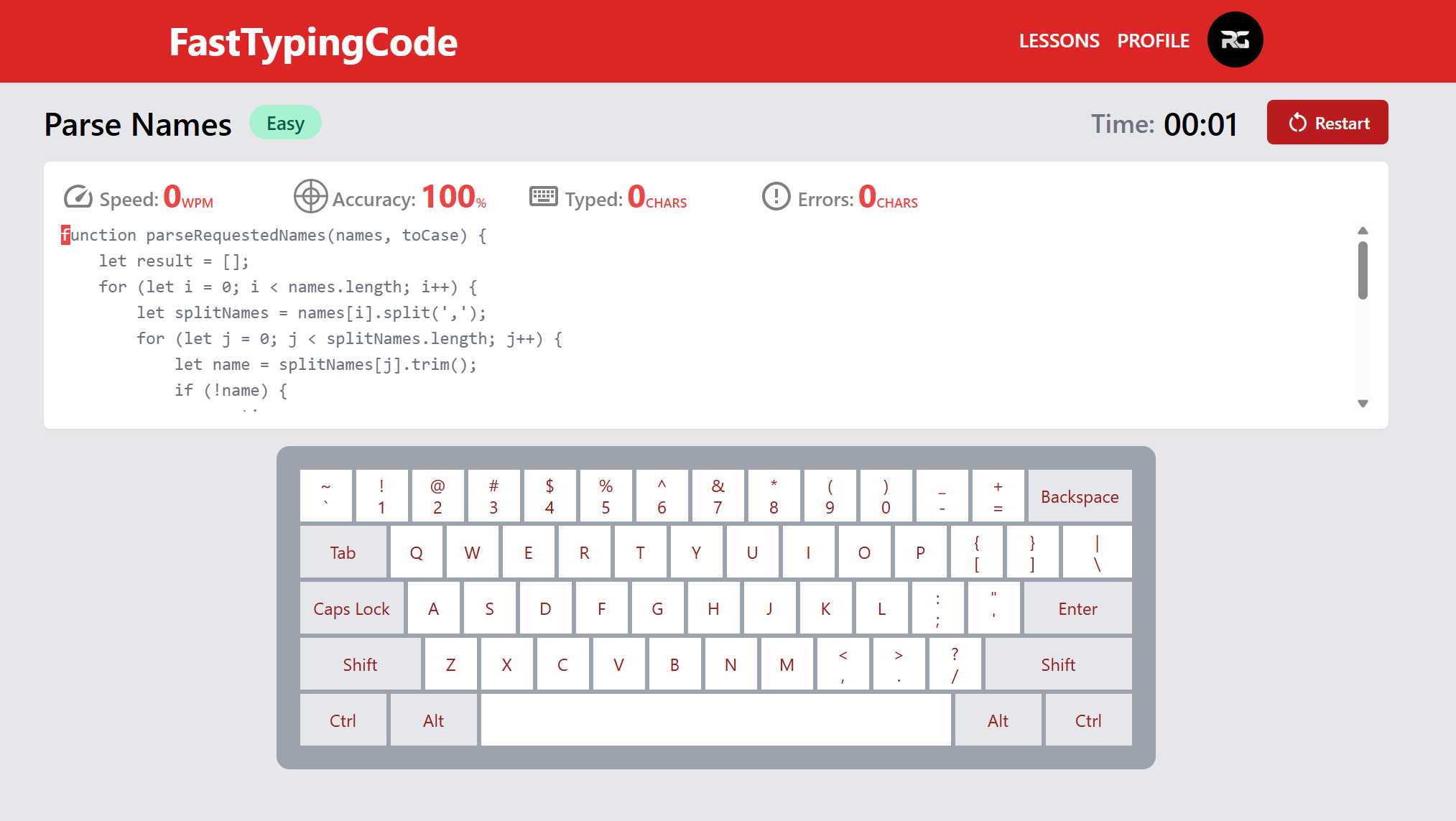


Рисунок 3.18 – Сторінка певної вправи

Після завершення вправи користувачеві буде відображено повідомлення про її успішне виконання, а також запрошення перейти до інших лекцій. Перехід до інших уроків здійснюється натисканням на кнопку «See more lessons», що ілюструється на рисунку 3.19.

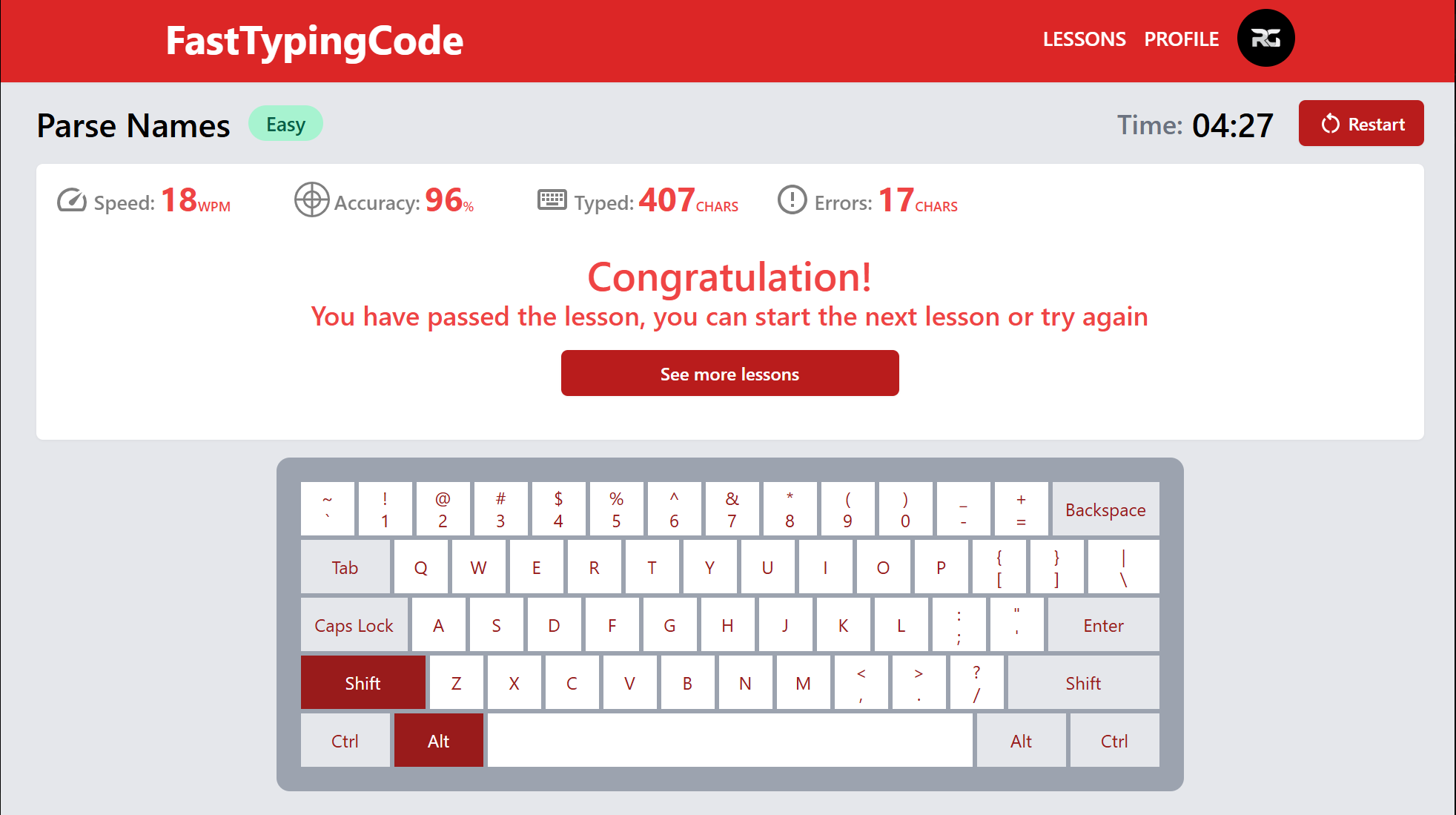


Рисунок 3.19 – Повідомлення після виконання лекції

Відвідавши розділ «Profile», користувач матиме змогу ознайомитися зі власною статистикою за весь період користування платформою. Зокрема, буде відображено загальний час, витрачений на виконання завдань, кількість переглянутих лекцій, зафіксовану максимальну швидкість друку, а також середню швидкість набору тексту за весь час. Зображення сторінки профілю користувача представлено на рисунку 3.20.

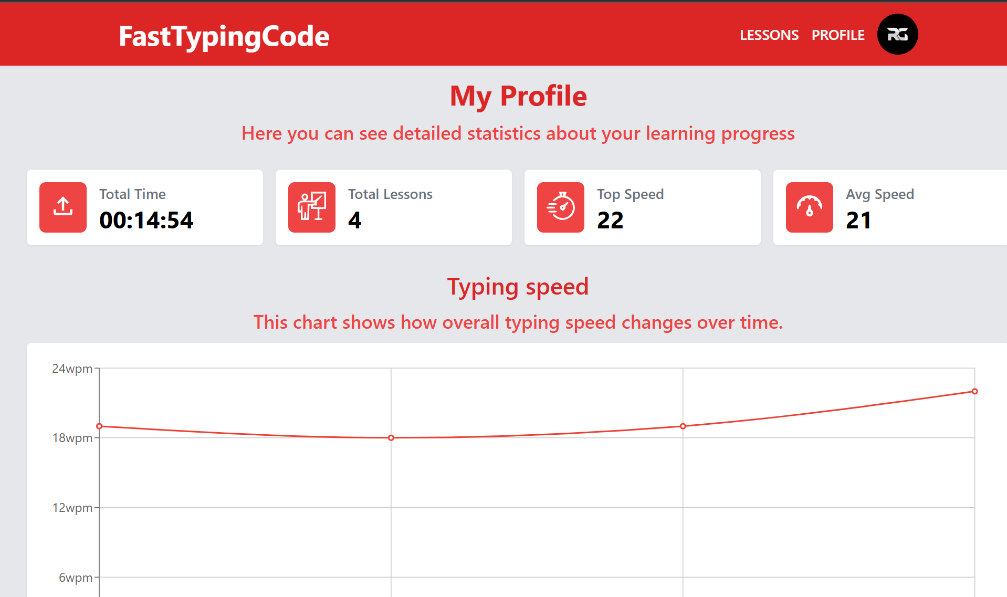


Рисунок 3.20 – Сторінка профілю користувача

# ВИСНОВКИ

Під час проходження проектно-технологічної (виробничої) практики було реалізовано теоретичні та практичне завдання. Було проведено ознайомлення з історією та діяльністю ТОВ «Петтерсонапс», досліджено впроваджені компанією підходи до розробки програмного забезпечення для різних систем, у тому числі для фреймворку ReactJS.

При розробці програмного забезпечення для навчання "сліпого" друку, інтерфейс користувача було створено з використанням ReactJS. Це включало створення складових частин, управління станом застосунку, реалізацію взаємодії з користувачем та інші функції.

Проведено аналіз та порівняння аналогів, визначено переваги та недоліки клавіатурних тренажерів для розробників. Розглянуто існуючі підходи до розв'язання проблеми.

Вибрано програмні засоби для реалізації веб-додатку на мові TypeScript, а саме: React Router, Redux, NestJS, Axios, TailwindCSS, Passport-JWT, Mongoose, та базу даних MongoDB. Також створено Google client ID для реалізації Google авторизації. Архітектура проекту поділяється на серверну та клієнтську частини. У клієнтській частині розміщені основні компоненти, такі як кнопки, поля для введення даних, заголовок застосунку, головна сторінка, сторінка акаунту, профілю, лекцій та вправ. Для серверної частини розроблено компоненти users, auth, attempts, lessons, sections.

Розроблено основні модулі проєкту, такі як перевірка введених даних з клавіатури, функція лічильника часу, розрахунок швидкості та точності набору тексту, реалізація Google авторизації, створення секцій з мовами програмування, створення облікового запису користувача, відображення повної статистики на сторінці профілю, а також реалізація різних рівнів складності вправ. Створено покрокову інструкцію для користувача.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ

 Штучна нейронна мережа [Електронний ресурс] – Режим доступу: [https://uk.wikipedia.org/wiki/Штучна\_нейронна\_мережа](https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%83%D1%87%D0%BD%D0%B0_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0) – Назва з екрана. – Дата звернення: 27.05.2025.

 SkyTek – центр бізнес обслуговування [Електронний ресурс] – Режим доступу: <https://skytek.in.ua/> – Назва з екрана. – Дата звернення: 27.05.2025.

 Штучні нейронні мережі – вступ та основи [Електронний ресурс] – Режим доступу: <https://itmaster.biz.ua/programming/vision/neural-networks.html> – Назва з екрана. – Дата звернення: 27.05.2025.

 The Transmitted. Що таке MLP у машинному навчанні? [Електронний ресурс]. – Режим доступу: <https://thetransmitted.com/adlucem/shho-take-mlp-u-mashynnomu-navchanni/> – Назва з екрана.

 IT Wiki. Long Short-Term Memory (LSTM) [Електронний ресурс]. – Режим доступу: <https://itwiki.dev/data-science/ml-reference/ml-glossary/long-short-term-memory-lstm> – Назва з екрана. Дата звернення: 27.05.2025.

 Speka. Як працює згорткова нейронна мережа: просте пояснення [Електронний ресурс]. – Режим доступу: <https://speka.media/yak-pracyuje-zgortkova-neironna-mereza-proste-poyasnennya-9er7j1> – Назва з екрана. Дата звернення: 27.05.2025.

 NVIDIA Blog. What is a Transformer Model? [Електронний ресурс]. – Режим доступу: <https://blogs.nvidia.com/blog/what-is-a-transformer-model/> – Назва з екрана. Дата звернення: 27.05.2025.

 IT Wiki. XGBoost [Електронний ресурс]. – Режим доступу: <https://itwiki.dev/data-science/ml-reference/ml-glossary/xgboost> – Назва з екрана. Дата звернення: 27.05.2025.

# ДОДАТКИ

Додаток А. Реалізація структури Account.tsx

import { Button } from "components/Button";

import { Paper } from "components/Paper";

import React from "react";

import GoogleLogin, { GoogleLogout } from "react-google-login";

import signoutIcon from "assets/signout.svg";

import { useAppDispatch } from "store/hooks";

import { useSignInMutation } from "services/authApi";

import {

  selectUserData,

  setUserData,

  setUserLoadingState,

} from "store/slices/userSlice";

import { useSelector } from "react-redux";

import { hasOwnProperty } from "utils/hasOwnProperty";

import Cookies from "js-cookie";

import { IUser, LoadingState } from "store/types";

export const Account = () => {

  const [signIn] = useSignInMutation();

  const dispatch = useAppDispatch();

  const user = useSelector(selectUserData);

  const responseSuccessGoogle = async (response: any) => {

    console.log(response);

    try {

      const result: any = await signIn({ tokenId: response.tokenId });

      console.log(result);

      if (hasOwnProperty(result, "data")) {

        dispatch(setUserData(result.data));

        dispatch(setUserLoadingState(LoadingState.LOADED));

      }

    } catch (err) {

      console.log(err);

      dispatch(setUserLoadingState(LoadingState.ERROR));

    }

  };

  const responseErrorGoogle = (response: any) => {};

  const handleLogoutSuccess = () => {

    Cookies.remove("token");

    dispatch(setUserData(null));

    dispatch(setUserLoadingState(LoadingState.CLEARED));

  };

  return (

    <div data-testid="home">

      <h1 className="mt-4 mb-6 text-center text-5xl font-bold text-red-600">

        Account

      </h1>

      <Paper>

        {user ? (

          <div>

            <div className="flex items-center justify-between">

              <div className="flex items-center">

                <img

                  className="rounded-full"

                  width={96}

                  height={96}

                  src={user.imageUrl}

                  alt="google profile icon"

                />

                <div className="ml-4">

                  <h4 className="text-3xl font-semibold text-gray-700">

                    {user.name}

                  </h4>

                  <h6 className="text-2xl font-semibold text-gray-500">

                    {user.email}

                  </h6>

                </div>

              </div>

              <div>

                <GoogleLogout

                  clientId="122216114574-6vvglitrjul91ksp3mn462c5ou95n0ft.apps.googleusercontent.com"

                  onLogoutSuccess={handleLogoutSuccess}

                  render={(renderProps) => (

                    <Button

                      onClick={renderProps.onClick}

                      disabled={renderProps.disabled}

                      className="px-6"

                    >

                      <img src={signoutIcon} alt="signout icon" />

                      <span className="ml-2">SIGN OUT</span>

                    </Button>

                  )}

                />

              </div>

            </div>

            <p className="text-xl font-medium text-gray-700 pt-2">

              You are using an account to store your typing data on our servers

              in the cloud. You will be able to access your profile from any

              computer or browser.

            </p>

          </div>

        ) : (

          <div>

            <p className="text-xl font-medium text-gray-700 py-2">

              Create an account to store your typing data on our servers in the

              cloud. This allows you to access your profile from any computer or

              browser. If you don't have an account then your typing data is

              stored locally and is accessible only from your current computer.

            </p>

            <p className="text-xl font-medium text-gray-700">

              We don't store any passwords. Instead we use third-party services

              to authenticate our users.

            </p>

            <div className="flex mb-2 mt-6 justify-center">

              <GoogleLogin

                clientId="122216114574-6vvglitrjul91ksp3mn462c5ou95n0ft.apps.googleusercontent.com"

                buttonText="Sign in with Google"

                theme="dark"

                onSuccess={responseSuccessGoogle}

                onFailure={responseErrorGoogle}

                cookiePolicy={"single\_host\_origin"}

                // onSuccess={() => console.log("biba")}

                // onFailure={(boba) => console.log(boba)}

              />

            </div>

          </div>

        )}

      </Paper>

    </div>

  );

};

Додаток Б. Реалізація структури Home.tsx

import React from "react";

import laptopImg from "assets/laptop.png";

import { HomeDescription } from "components/HomeDescription";

import codeIcon from "assets/code.svg";

import chartIcon from "assets/chart.svg";

import { Button } from "components/Button";

import { Link } from "react-router-dom";

export const Home = () => {

return (

<div data-testid="home">

<h1 className="mt-4 mb-2 text-center text-5xl font-bold text-red-600">

Typing Practice for Frontend Developers

</h1>

<img src={laptopImg} alt="preview img" width="60%" className="m-auto" />

<div className="flex gap-4">

<HomeDescription

title="Source code"

value="Choose from: JavaScript, TypeScript, CSS, React"

icon={codeIcon}

/>

<HomeDescription

title="Typing speed"

value="Increase your Typing Speed & Accuracy"

icon={chartIcon}

/>

</div>

<div className="mt-6 flex flex-col items-center">

<h3 className="text-3xl font-bold text-red-500 text-center">

It's free, just sign in.

</h3>

<Link to="/account">

<Button className="px-6 my-3 text-xl">Sign in with Google</Button>

</Link>

</div>

</div>

);

};

Додаток В. Реалізація структури Lesson.tsx

import accuracyIcon from "assets/accuracy.svg";

import errorIcon from "assets/error.svg";

import keyboardIcon from "assets/keyboard.svg";

import restartIcon from "assets/restart.svg";

import speedIcon from "assets/speed.svg";

import startIcon from "assets/start.svg";

import { CustomModal } from "components/CustomModal";

import { DifficultyBadge } from "components/DifficultyBadge";

import { LessonKeyboard } from "components/LessonKeyboard";

import { useCounter } from "hooks/useCounter";

import { useKeyboardInput } from "hooks/useKeyboardInput";

import { useSyncState } from "hooks/useSyncState";

import React, { useEffect, useRef, useState } from "react";

import { Link, useParams } from "react-router-dom";

import {

useCreateAttemptMutation,

useGetLessonQuery,

} from "services/lessonApi";

import { sectionApi } from "services/sectionApi";

import { useAppDispatch } from "store/hooks";

import { ICreateAttempt } from "store/types";

import { escapeSpecialChars } from "utils/escapeSpecialChars";

import { Button } from "../components/Button";

import { Paper } from "../components/Paper";

export const Lesson: React.FC = () => {

const params: { lessonId: string; sectionId: string } = useParams();

const { data: lesson } = useGetLessonQuery(params.lessonId);

const [createAttempt] = useCreateAttemptMutation();

const dispatch = useAppDispatch();

const invisibleInput = useRef<null | HTMLInputElement>(null);

const lessonCode = useRef<null | HTMLPreElement>(null);

const [isOpenModal, setIsOpenModal] = useState(true);

const indicators = useSyncState({ wpm: 0, accuracy: 100 });

const { currentChar, setCurrentChar, typed, isLessonEnded, handleInput } =

useKeyboardInput(lessonCode);

const { counter } = useCounter(!isOpenModal && !isLessonEnded);

const startLesson = () => {

if (currentChar) {

setIsOpenModal(false);

invisibleInput?.current?.focus();

currentChar.classList.add(`bg-red-500`);

currentChar.classList.add(`text-white`);

}

};

const restartLesson = () => {

//restartCounter();

window.location.reload();

};

useEffect(() => {

// here i use custom wpm formula, because with original f. when you hit error wpm falls heavily

const wpm = +(

((typed.total - typed.wrong) / 5 / (counter.time + 0.1)) \*

60

).toFixed(0);

const accuracy = +(

((typed.total - typed.wrong) / (typed.total + 0.00001)) \*

100

).toFixed(0);

indicators.set({

wpm: wpm > 0 ? wpm : 0,

accuracy: accuracy === 0 ? 100 : accuracy,

});

// eslint-disable-next-line react-hooks/exhaustive-deps

}, [typed, counter.time]);

useEffect(() => {

// when component is mounted, it doesn`t have lesson, here we sync and rerender

if (lesson) setCurrentChar(lessonCode.current?.children[0]);

}, [lesson, setCurrentChar]);

useEffect(() => {

if (isLessonEnded && lesson) {

const data: ICreateAttempt = {

lesson: lesson.\_id,

accuracy: indicators.get().accuracy,

wpm: indicators.get().wpm,

time: counter.time,

numberOfErrors: typed.wrong,

};

createAttempt(data).then(() => {

dispatch(

sectionApi.endpoints.getSectionById.initiate(params.sectionId, {

subscribe: false,

forceRefetch: true,

})

);

});

}

// eslint-disable-next-line react-hooks/exhaustive-deps

}, [isLessonEnded]);

return (

<div data-testid="lesson" className="my-4">

<div className="flex items-center justify-between">

<div className="flex items-center">

{!!lesson && (

<>

<h3 className="text-3xl font-semibold mr-4">{lesson.name}</h3>

<DifficultyBadge difficulty={lesson.difficulty} size="large" />

</>

)}

</div>

<div className="flex items-center">

<h4 className="text-2xl font-semibold mr-2 text-gray-500">Time:</h4>

<h4 className="text-3xl font-semibold mr-4 w-20">{`${counter.minute}:${counter.second}`}</h4>

<Button onClick={restartLesson}>

<img

className="mr-1"

src={restartIcon}

width={24}

height={24}

alt="restart icon"

/>

Restart

</Button>

</div>

</div>

<LessonKeyboard isLessonEnded={isLessonEnded}>

<Paper className="my-4">

<div className="flex gap-6">

<Indicator

name="Speed"

measure="WPM"

value={indicators.get().wpm}

icon={speedIcon}

/>

<Indicator

name="Accuracy"

measure="%"

value={indicators.get().accuracy}

icon={accuracyIcon}

/>

<Indicator

name="Typed"

measure="CHARS"

value={typed.total}

icon={keyboardIcon}

/>

<Indicator

name="Errors"

measure="CHARS"

value={typed.wrong}

icon={errorIcon}

/>

</div>

<div className="relative mt-2 ">

<pre

ref={lessonCode}

className="text-base text-gray-500 font-medium font-mono h-44 max-h-44 overflow-y-scroll"

>

{lesson &&

escapeSpecialChars(lesson.code)

.split("")

.map((char, i) => (

<span

className={

char === "\n" ? "before:enter text-white px-1" : ""

}

key={i}

>

{char}

</span>

))}

</pre>

<div

className={`transition-all absolute top-0 min-w-full min-h-full bg-white opacity-0 ${

isLessonEnded ? "visible opacity-100" : "invisible"

} flex items-center justify-center flex-col`}

>

<h3 className="text-4xl text-red-500 font-semibold">

Congratulation!

</h3>

<h4 className="text-2xl text-red-500 font-semibold">

You have passed the lesson, you can start the next lesson or try

again

</h4>

<div className="flex gap-4 w-1/4 mt-4">

<Link className="w-full" to={`/lessons/${params.sectionId}`}>

<Button fullWidth>See more lessons</Button>

</Link>

</div>

</div>

</div>

{!isLessonEnded && (

<input

ref={invisibleInput}

onBlur={() => invisibleInput?.current?.focus()}

onKeyDown={handleInput}

className="fixed -left-full"

/>

)}

</Paper>

</LessonKeyboard>

<CustomModal

isOpen={isOpenModal}

setIsOpen={setIsOpenModal}

shouldCloseOnOverlayClick={false}

maxWidth={500}

>

<div className="flex flex-col items-center">

<div className="bg-greens-200 p-3 rounded-lg w-20">

<img width={64} height={64} src={startIcon} alt="start icon" />

</div>

<h4 className="text-3xl font-semibold py-4">

Please be prepared. Good luck!

</h4>

<Button fullWidth onClick={startLesson} disabled={!!!lesson}>

{!!lesson ? "Start Typing Now" : "Loading..."}

</Button>

</div>

</CustomModal>

</div>

);

};

type IndicatorPropsType = {

name: string;

measure: string;

value: number;

icon: string;

};

const Indicator: React.FC<IndicatorPropsType> = ({

name,

measure,

value,

icon,

}) => {

return (

<div className="flex items-end max-h-8 max-w-s w-full">

<div className="flex items-end opacity-50 mr-1">

<img className="mr-1" src={icon} width={32} height={32} alt={icon} />

<span className="text-lg font-semibold">{name}:</span>

</div>

<div className="text-red-500">

<span className="text-3xl font-bold ">{value}</span>

<span className="text-xs font-semibold">{measure}</span>

</div>

</div>

);

};

Додаток Г. Реалізація структури Lessons.tsx

import { LessonCard, LessonCardPreloader } from "components/LessonCard";

import { useDelayedQuery } from "hooks/useDelayedQuery";

import React from "react";

import { useParams } from "react-router";

import { useGetSectionByIdQuery } from "services/sectionApi";

import emptyImg from "assets/empty.png";

import { Link } from "react-router-dom";

import { Button } from "components/Button";

export const Lessons = () => {

const params: { sectionId: string } = useParams();

const section = useDelayedQuery(params.sectionId, useGetSectionByIdQuery);

return (

<div className="my-4" data-testid="lessons">

<h1

className={`text-center text-4xl font-bold text-red-600 ${

section ? "visible" : "invisible"

}`}

>{`${section ? section.name : "Section"} lessons`}</h1>

<h2 className="my-3 text-center text-3xl font-semibold text-red-500">

Choose and start your lesson

</h2>

<div className="my-8 w-full flex flex-wrap justify-center gap-y-4 gap-x-12 px-4">

{section ? (

section.lessons.length ? (

section.lessons.map((lesson, i) => (

<LessonCard

{...lesson}

key={lesson.\_id}

sectionId={params.sectionId}

/>

))

) : (

<div className="flex flex-col items-center">

<img width={256} height={256} src={emptyImg} alt="empty icon" />

<h2 className="my-2 text-2xl font-semibold text-gray-500">

Oops! No lessons

</h2>

<Link to="/">

<Button>Go to homepage</Button>

</Link>

</div>

)

) : (

[...Array(6)].map((v, i) => <LessonCardPreloader key={i} />)

)}

</div>

</div>

);

};

Додаток Ґ. Реалізація структури Profile.tsx

import avgSeedIcon from "assets/avgSpeed.svg";

import lessonIcon from "assets/lesson.svg";

import topSpeedIcon from "assets/topSpeed.svg";

import uploadIcon from "assets/upload.svg";

import { Paper } from "components/Paper";

import {

ProfileStatItem,

ProfileStatItemPreloader,

} from "components/ProfileStatItem";

import { useDelayedQuery } from "hooks/useDelayedQuery";

import React from "react";

import {

CartesianGrid,

Line,

LineChart,

ResponsiveContainer,

Tooltip,

XAxis,

YAxis,

} from "recharts";

import { useGetUserProfileQuery } from "services/userApi";

export const Profile = () => {

const profile = useDelayedQuery(null, useGetUserProfileQuery);

return (

<div className="my-4" data-testid="sections">

<h1 className="text-center text-4xl font-bold text-red-600">

My Profile

</h1>

<h2 className="my-3 text-center text-2xl font-semibold text-red-500">

Here you can see detailed statistics about your learning progress

</h2>

<div className="my-8">

<div className="flex gap-4 my-4">

{profile ? (

<>

<ProfileStatItem

title="Total Time"

value={profile.totalTime}

icon={uploadIcon}

/>

<ProfileStatItem

title="Total Lessons"

value={profile.totalLessons}

icon={lessonIcon}

/>

<ProfileStatItem

title="Top Speed"

value={profile.topSpeed}

icon={topSpeedIcon}

/>

<ProfileStatItem

title="Avg Speed"

value={profile.averageSpeed}

icon={avgSeedIcon}

/>

</>

) : (

[...Array(4)].map((v, i) => <ProfileStatItemPreloader key={i} />)

)}

</div>

</div>

<div>

<h3 className="text-center text-3xl font-semibold text-red-600">

Typing speed

</h3>

<h4 className="my-3 text-center text-2xl font-medium text-red-500">

This chart shows how overall typing speed changes over time.

</h4>

{profile ? (

<Paper className="min-h-98 ">

<ResponsiveContainer width={"99%"} height={400}>

<LineChart

width={1200}

height={400}

data={profile.typingChart}

margin={{

top: 16,

right: 16,

left: 16,

}}

style={{ cursor: "pointer" }}

>

<CartesianGrid />

<XAxis dataKey="lessonNumber" />

<YAxis dataKey="wpm" unit="wpm" />

<Tooltip />

<Line

type="monotone"

dataKey="wpm"

stroke="#eb4034"

strokeWidth={2}

/>

</LineChart>

</ResponsiveContainer>

<h4 className="text-center text-red-700">

Horizontal axis: lesson number. Vertical axis: wpm

</h4>

</Paper>

) : (

<div className="flex justify-center items-center mt-16">

<div

className="

animate-spin

rounded-full

h-32

w-32

border-t-4 border-b-4 border-red-700

"

></div>

</div>

)}

</div>

</div>

);

};

Додаток Д. Реалізація структури Sections.tsx

import cssIcon from "assets/css.svg";

import jsIcon from "assets/js.svg";

import reactIcon from "assets/react.svg";

import tsIcon from "assets/ts.svg";

import uploadIcon from "assets/upload.svg";

import { SectionCard, SectionCardPreloader } from "components/SectionCard";

import { useDelayedQuery } from "hooks/useDelayedQuery";

import React from "react";

import { Link } from "react-router-dom";

import { useGetSectionsQuery } from "services/sectionApi";

export const Sections = () => {

const sections = useDelayedQuery(null, useGetSectionsQuery);

return (

<div className="my-4" data-testid="sections">

<h1 className="text-center text-4xl font-bold text-red-600">

Improve your Typing with Open Source Code

</h1>

<h2 className="my-3 text-center text-3xl font-semibold text-red-500">

Choose one section of frontend development

</h2>

<div className="my-8 w-full flex flex-wrap justify-center gap-y-4 gap-x-8">

{sections

? sections.map((section) => (

<SectionCard

key={section.\_id}

\_id={section.\_id}

route={section.route}

name={section.name}

description={section.description}

iconStyle={getIconStyle(section.name)}

/>

))

: [...Array(4)].map((v, i) => <SectionCardPreloader key={i} />)}

</div>

</div>

);

};

export type IconStyleType = {

src: string;

color: string;

};

const getIconStyle = (id: string): IconStyleType => {

switch (id) {

case "JavaScript":

return {

src: jsIcon,

color: "bg-yellow-400",

};

case "TypeScript":

return {

src: tsIcon,

color: "bg-blue-600",

};

case "CSS":

return {

src: cssIcon,

color: "bg-green-500",

};

case "React and Redux":

return {

src: reactIcon,

color: "bg-lightBlue",

};

default:

return {

src: uploadIcon,

color: "bg-gray-400",

};

}

};

Додаток Е. Реалізація структури Text.tsx

import React, { useState } from 'react';

import { removeSpaces } from 'utils/removeSpaces';

export const Text = () => {

const [value, setValue] = useState('');

const handleChange = (e: any) => {

console.log(e.target.value);

setValue(e.target.value);

};

const handleTab = (e: any) => {

if (e.key === 'Tab') {

e.preventDefault();

}

};

const save = () => {

console.log(removeSpaces(value));

console.log(JSON.stringify(value).split('\\n').length);

};

return (

<div>

<textarea

tabIndex={10}

onKeyDown={handleTab}

onChange={handleChange}

value={value}

cols={100}

rows={10}

></textarea>

<button onClick={save}>Console</button>

</div>

);

};

Додаток Є. Реалізація структури user.service.tsx

import { Injectable } from '@nestjs/common';

import { InjectModel } from '@nestjs/mongoose';

import { Model } from 'mongoose';

import { CreateUserDto } from './dto/create-user.dto';

import { User, UserDocument } from './schemas/user.schema';

@Injectable()

export class UsersService {

constructor(

@InjectModel(User.name)

private readonly userModel: Model<UserDocument>,

) {}

async create(createUserDto: CreateUserDto): Promise<User> {

const createdUser = await this.userModel.create(createUserDto);

return createdUser;

}

async findAll(): Promise<User[]> {

return this.userModel.find().exec();

}

async findOne(id: string): Promise<User> {

return this.userModel.findById(id).populate('attempts').exec();

}

async getProfile(reqUser: User & { \_id: string }): Promise<any> {

const user = await this.userModel

.findById(reqUser.\_id)

.populate('attempts');

let totalTime = 0;

const profile = {

totalTime: '',

totalLessons: user.attempts.length,

topSpeed: 0,

averageSpeed: 0,

typingChart: [] as { lessonNumber: number; wpm: number }[],

};

user.attempts.forEach((attempt, index) => {

totalTime += attempt.time;

profile.topSpeed = Math.max(profile.topSpeed, attempt.wpm);

profile.averageSpeed += Math.round(attempt.wpm / user.attempts.length);

profile.typingChart.push({ lessonNumber: index + 1, wpm: attempt.wpm });

});

profile.totalTime = [

(totalTime / 60 / 60).toFixed(0),

((totalTime / 60) % 60).toFixed(0),

(totalTime % 60).toFixed(0),

]

.join(':')

.replace(/\b(\d)\b/g, '0$1');

return profile;

}

}