МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
Фізико-математичний факультет
Кафедра інформатики

«Допущено до захисту»
В.о. завідувача кафедри
_____ _____.
(підпис) (прізвище, ініціали)
«___» _____ 2025 р.

Реєстраційний № _____
«___» _____ 2025 р.

РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ПЕРСОНАЛІЗОВАНОГО ПЛАНУВАННЯ ТА МОНІТОРИНГУ ТРЕНУВАНЬ

Кваліфікаційна робота студента групи І-21
ступінь вищої освіти «бакалавр»
спеціальності
014.09 Середня освіта (Інформатика)
Ступака Дмитра Ростиславовича
Керівник: Степанюк Олександр Миколайович

Оцінка:
Національна шкала _____
Шкала ECTS ____ Кількість балів _____
Голова ЕК: _____
(підпис) (прізвище та ініціали)
Члени ЕК _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)
_____ _____
(підпис) (прізвище та ініціали)

Кривий Ріг – 2025

ЗАПЕВНЕННЯ

Я, Ступак Дмитро Ростиславович, розумію і підтримую політику Криворізького державного державного педагогічного університету з академічної доброчесності. Запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Я не надавав і не одержував недозволену допомогу допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають покликання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Криворізького державного педагогічного університету ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

01.06.2025

ЗМІСТ

ВСТУП

У сучасному світі фізична активність відіграє ключову роль у підтриманні здоров’я, покращенні якості життя та підвищенні загального добробуту. Зростаюча популярність фітнес-тренувань свідчить про прагнення людей до активного способу життя, однак ефективне планування тренувань і моніторинг прогресу залишаються викликом для багатьох, особливо для новачків. Відсутність структурованого підходу до організації тренувального процесу може знижувати мотивацію та ускладнювати досягнення поставлених цілей.

Актуальність теми дослідження зумовлена потребою в розробці зручного та багатофункціонального веб-додатку, який би спрощував процес планування, відстеження та аналізу фітнес-тренувань. Такий додаток має забезпечувати інтуїтивний інтерфейс, гнучкість у налаштуванні тренувальних програм і можливість моніторингу результатів, що сприятиме підвищенню ефективності занять і підтримці мотивації користувачів.

Мета роботи – створення веб-платформи для персонального планування та моніторингу фітнес-тренувань, яка буде зручною, функціональною та адаптованою до потреб користувачів із різним рівнем фізичної підготовки.

Об’єкт дослідження – процес організації, виконання та аналізу фітнес-тренувань.

Предмет дослідження – веб-платформа для планування та моніторингу фітнес-тренувань, її функціональні можливості, інтерфейс і технології реалізації.

Основні завдання роботи:

– провести аналіз існуючих рішень, визначивши їх сильні та слабкі сторони;

– обґрунтувати вибір технологічного стеку для розробки платформи;

– сформулювати функціональні та нефункціональні вимоги до веб-додатку;

– реалізувати веб-платформу відповідно до визначених вимог;

– виконати тестування та оцінити ефективність розробленого рішення.

Розроблена веб-платформа стане цінним інструментом для користувачів, які прагнуть систематизувати свої тренування та досягати фітнес-цілей. Вона допоможе організувати тренувальний процес, відстежувати прогрес і аналізувати результати, що позитивно вплине на мотивацію та результативність занять.

Перелік основних функцій та можливостей веб-платформи:

– Реєстрація та авторизація користувачів для забезпечення персоналізованого доступу.

– Планування тренувань:

- Вибір типу тренувань (силові, кардіо, стретчинг тощо).
- Створення індивідуальних тренувальних програм.
- Додавання вправ із зазначенням кількості підходів, повторень, ваги та тривалості.

– Моніторинг прогресу через статистику та графіки виконаних тренувань.

Вимоги до інтерфейсу та юзабіліті:

– Простий і привабливий дизайн, що сприяє комфортному користуванню.

– Зручна навігація з логічною структурою меню.

– Інформативні підказки та повідомлення для полегшення взаємодії.

– Використання графіків і візуальних елементів для наочної презентації даних.

– Висока швидкість завантаження сторінок і відгук інтерфейсу.

Ці вимоги забезпечать створення ефективної та зручної веб-платформи, яка відповідатиме потребам користувачів із різними цілями та рівнями підготовки, сприяючи їхньому прогресу у фітнесі.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд сучасних веб-платформ для персонального планування та моніторингу тренувань

Сучасні технології значною мірою трансформують підходи до організації фізичних тренувань, надаючи користувачам інструменти для планування, моніторингу та аналізу їхньої спортивної активності. Веб-платформи для персонального планування та моніторингу тренувань стають дедалі популярнішими завдяки зручності, доступності та широкому спектру функціональних можливостей. Ці рішення дозволяють як професійним спортсменам, так і аматорам систематизувати свої заняття, відстежувати прогрес і отримувати персоналізовані рекомендації. У цьому розділі здійснено огляд сучасних веб-платформ, які є лідерами у сфері фітнес-технологій, з акцентом на їхні ключові особливості, переваги та обмеження. Аналіз базується на доступній інформації про популярні рішення, що відповідають потребам користувачів у плануванні та моніторингу тренувань.

Однією з найвідоміших платформ для моніторингу фізичної активності та харчування є MyFitnessPal (рис. 1.1), розроблена компанією Under Armour [1]. Ця платформа інтегрує функції відстеження калорій, планування тренувань і аналізу дієти. MyFitnessPal має базу даних із понад 6 мільйонами продуктів, що дозволяє користувачам точно вести облік спожитих калорій і макронутрієнтів. Для тренувань платформа пропонує можливість створення індивідуальних планів, які включають кардіо, силові вправи та заняття з власною вагою. Користувачі можуть синхронізувати платформу з фітнес-трекерами, такими як Fitbit або Apple Watch, для автоматичного збору даних про фізичну активність.

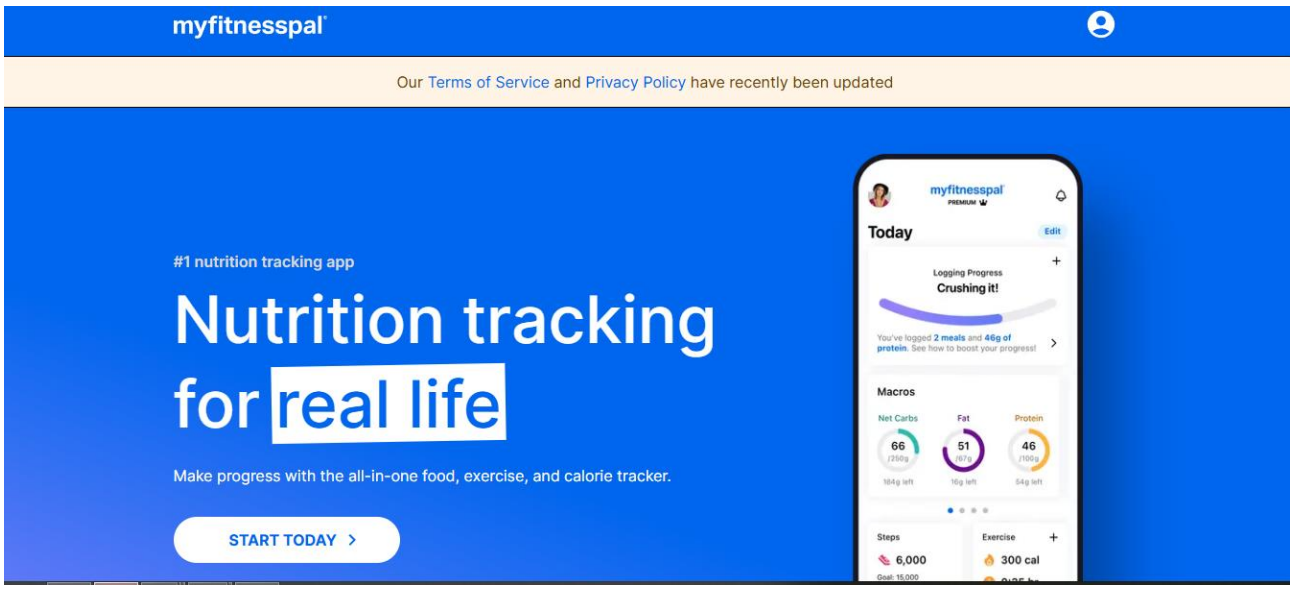

Рис. 1.1 – MyFitnessPal [1]

Переваги MyFitnessPal включають інтуїтивний інтерфейс, широку базу даних продуктів і можливість інтеграції з іншими пристроями. Однак безкоштовна версія має обмежений функціонал, а повний доступ до аналітики та персоналізованих планів тренувань вимагає платної підписки. Крім того, платформа більше орієнтована на контроль харчування, ніж на детальне планування складних тренувальних програм, що може бути недоліком для професійних спортсменів.

Nike Training Club (NTC) — це веб-платформа та мобільний додаток, який пропонує широкий вибір тренувань для різних рівнів підготовки (рис. 1.2) [2]. Платформа включає бібліотеку з більш ніж 200 тренувань, розроблених професійними тренерами, що охоплюють йогу, силові вправи, кардіо та розтяжку. Кожне тренування супроводжується покроковими інструкціями, фото- та відеоматеріалами, що полегшує виконання вправ. NTC дозволяє створювати персоналізовані плани тренувань на основі цілей користувача, таких як втрата ваги, набір м’язової маси чи покращення витривалості.

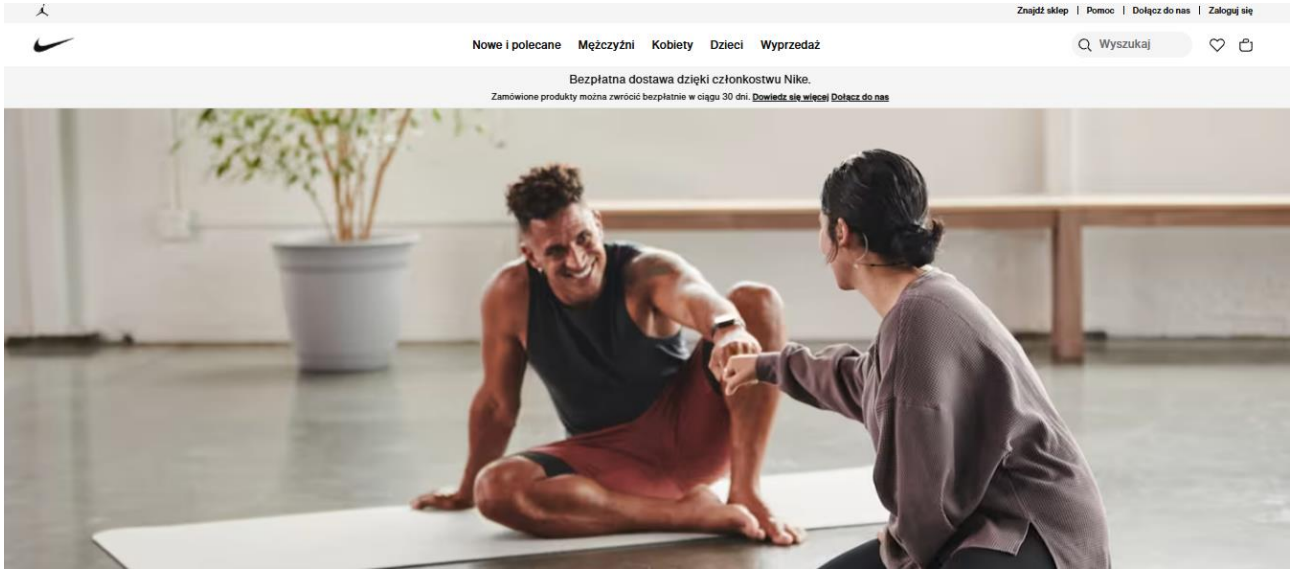

Рис. 1.2 – Nike Training Club (NTC) [2]

Особливістю Nike Training Club є адаптивність: платформа пропонує тренування як для занять вдома без обладнання, так і для тренажерних залів. Безкоштовний доступ до більшості функцій робить NTC привабливим для широкої аудиторії. Проте преміум-версія, яка включає додаткові програми та консультації тренерів, доступна лише за передплатою. Недоліком є обмежена інтеграція з зовнішніми пристроями для моніторингу показників здоров’я, таких як частота серцевих скорочень, що може бути важливим для користувачів, які прагнуть детального аналізу.

Fitbit (рис. 1.3), відомий насамперед завдяки своїм фітнес-трекерам, також пропонує веб-платформу та додаток для планування та моніторингу тренувань [3]. Платформа дозволяє відстежувати фізичну активність, включаючи кількість кроків, витрачені калорії, дистанцію та якість сну. Fitbit підтримує створення тренувальних планів, які базуються на даних, зібраних із носимих пристроїв. Користувачі можуть отримувати рекомендації щодо вправ, таких як ходьба, біг чи силові тренування, залежно від їхньої фізичної форми та цілей.

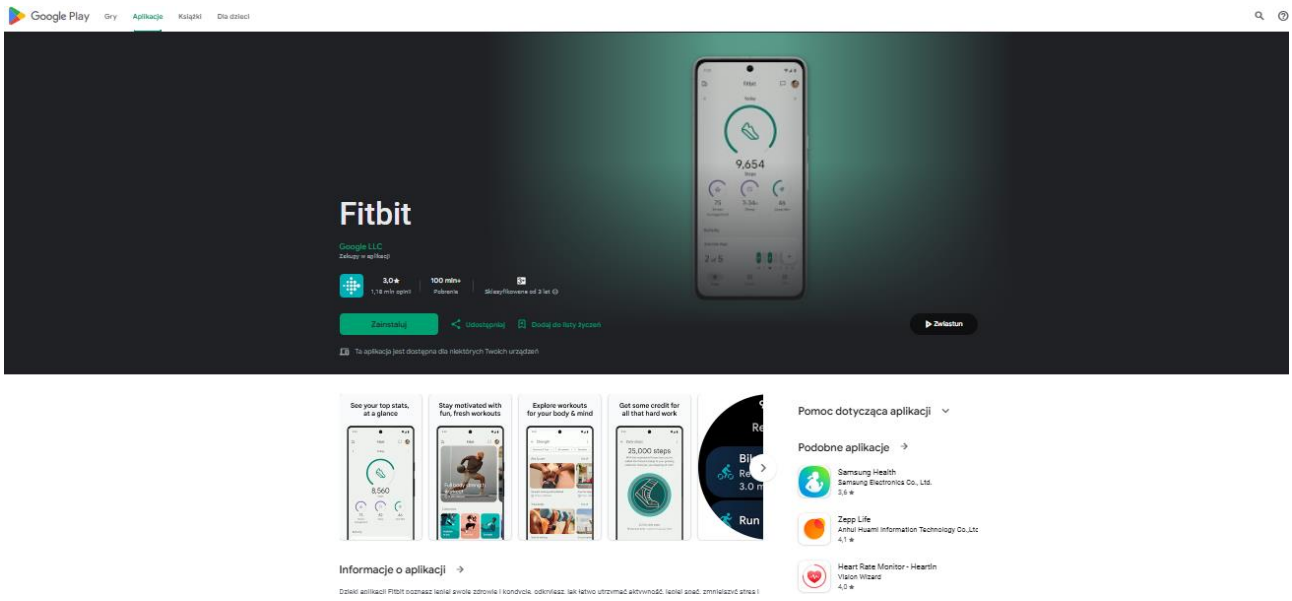

Рис.1.3 – Fitbit [3]

Сильною стороною Fitbit є точність збору даних завдяки інтеграції з власними пристроями. Платформа також пропонує соціальні функції, такі як змагання з друзями, що підвищує мотивацію. Однак без фітнес-трекера Fitbit можливості платформи значно обмежені, оскільки вона значною мірою залежить від апаратного забезпечення. Крім того, преміум-функції, такі як детальна аналітика та персоналізовані плани, доступні лише за додаткову плату, що може відштовхувати деяких користувачів.

Strava (рис. 1.4) — це платформа, орієнтована переважно на любителів бігу, велоспорту та інших видів кардіотренувань [4]. Вона дозволяє користувачам відстежувати свої тренування за допомогою GPS, аналізувати маршрути, швидкість, дистанцію та витрачені калорії. Strava має розвинену соціальну складову, що дає змогу ділитися результатами, брати участь у викликах і порівнювати свої досягнення з іншими спортсменами. Платформа також підтримує інтеграцію з фітнес-пристроями, такими як Garmin і Polar.

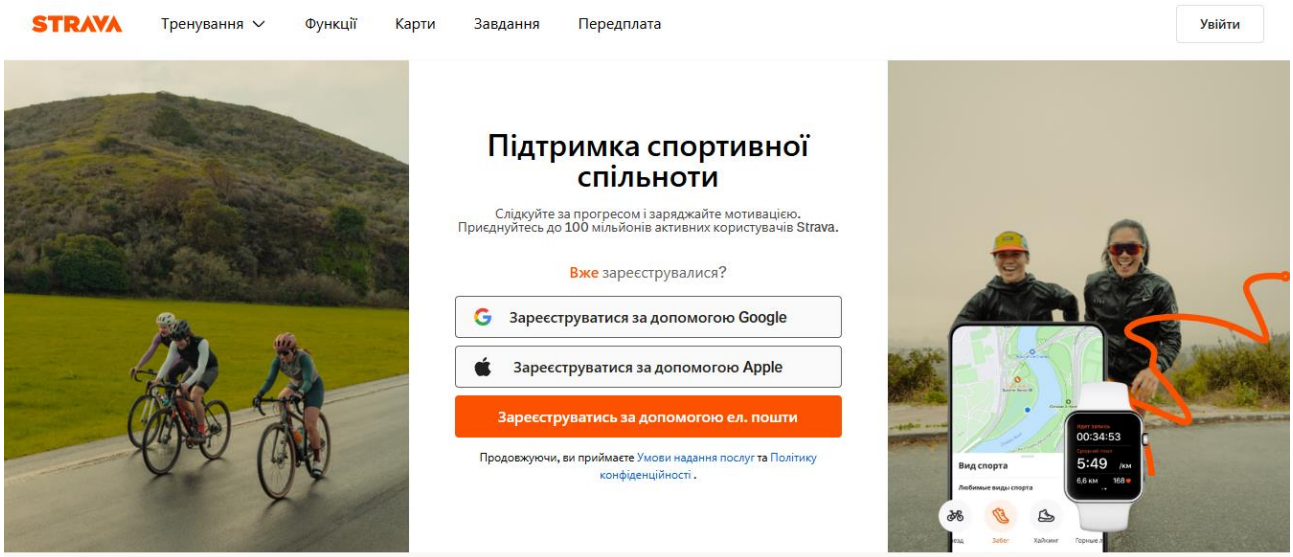

Рис. 1.4 – Strava [4]

Перевагою Strava є її фокус на спільноті та мотиваційні елементи, такі як рейтинги та сегменти маршрутів. Проте платформа менш підходить для силових тренувань або занять у тренажерному залі, оскільки її функціонал зосереджений на активностях на відкритому повітрі. Безкоштовна версія має базові можливості, тоді як розширені функції, такі як аналіз продуктивності та планування тренувань, доступні лише в преміум-підписці.

8fit (рис. 1.5) — це платформа, яка поєднує планування тренувань і рекомендації щодо харчування [5]. Вона пропонує короткі, але інтенсивні тренування (від 5 до 20 хвилин), які підходять для зайнятих людей. Платформа створює персоналізовані плани на основі фізичних даних користувача, його цілей і наявного обладнання. 8fit також включає поради від дієтологів, що допомагає користувачам досягати комплексних результатів у фітнесі та здоров'ї.

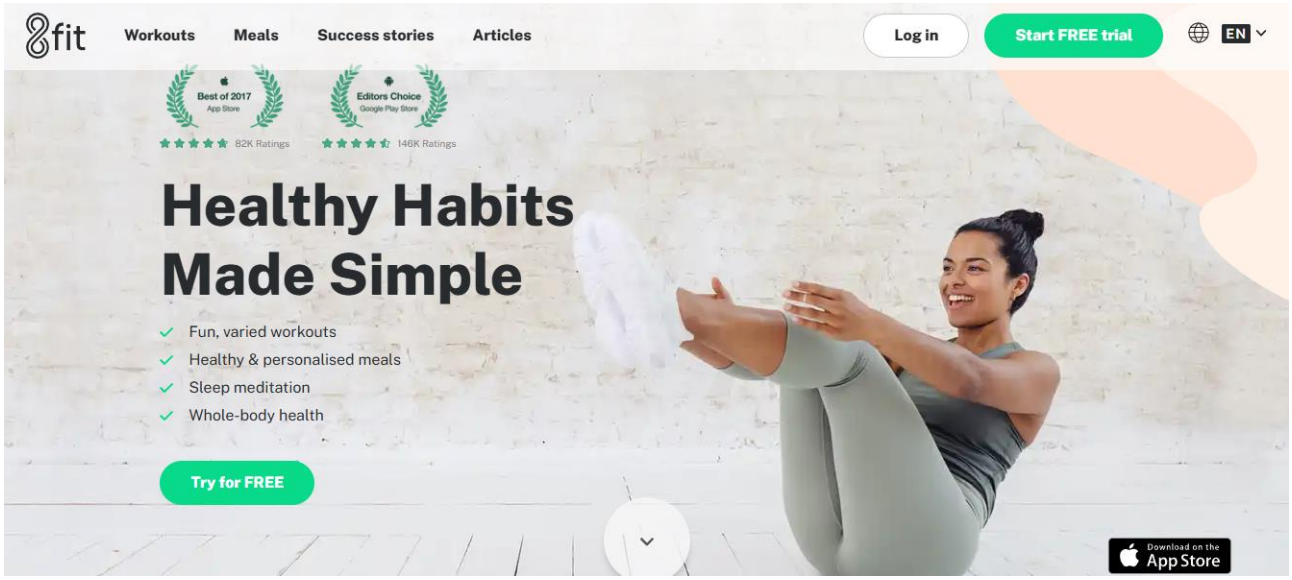

Рис. 1.5 - 8fit [5]

Сильною стороною 8fit є її універсальність і простота використання, що робить платформу ідеальною для початківців. Однак обмежений вибір тренувань і менша глибина аналітики порівняно з іншими платформами можуть не задовольнити досвідчених спортсменів. Крім того, повний доступ до всіх функцій вимагає підписки, що може бути недоліком для користувачів, які шукають безкоштовні рішення.

Сучасні веб-платформи для планування та моніторингу тренувань демонструють кілька ключових тенденцій. По-перше, більшість рішень прагнуть до персоналізації, використовуючи дані користувачів для створення індивідуальних планів. По-друге, інтеграція з носимими пристроями та штучним інтелектом стає стандартом, що дозволяє підвищити точність моніторингу та якість рекомендацій. По-третє, соціальні функції, такі як спільноти та змагання, відіграють важливу роль у підвищенні мотивації користувачів.

Проте існують і виклики. Багато платформ пропонують обмежений функціонал у безкоштовних версіях, що змушує користувачів купувати підписки. Крім того, не всі платформи однаково ефективні для різних типів тренувань: деякі краще підходять для кардіо, інші — для силових вправ. Також важливо враховувати питання безпеки даних, оскільки платформи збирають конфіденційну інформацію про здоров’я та фізичну активність користувачів.

Огляд сучасних веб-платформ для персонального планування та моніторингу тренувань показує, що ринок пропонує різноманітні рішення, які відповідають потребам різних груп користувачів. MyFitnessPal вирізняється своєю універсальністю та фокусом на харчування, Nike Training Club приваблює широким вибором тренувань і доступністю, Fitbit забезпечує точний моніторинг завдяки інтеграції з пристроями, Strava мотивує любителів кардіо через соціальну взаємодію, а 8fit пропонує зручні рішення для початківців. Кожна платформа має свої сильні сторони, але також стикається з обмеженнями, такими як платний доступ до преміум-функцій або вузька спеціалізація.

Розробка нової веб-платформи потребує врахування цих тенденцій і викликів. Для забезпечення конкурентоспроможності необхідно поєднати персоналізацію, інтеграцію з носимими пристроями, інтуїтивний інтерфейс і доступність для широкої аудиторії. У наступних розділах буде розглянуто технічні аспекти реалізації такого рішення на основі кодів проєкту, що дозволить оцінити практичну можливість створення платформи з урахуванням сучасних стандартів і потреб користувачів.

1.2 Обґрунтування вибору технологій для розробки веб-платформи

Розробка веб-платформи для персонального планування та моніторингу тренувань вимагає ретельного вибору технологій, які забезпечать високу продуктивність, зручність для користувачів, масштабованість та безпеку системи. Обґрунтування вибору технологічного стеку базується на аналізі вимог до проєкту, зокрема необхідності створення інтерактивного інтерфейсу, обробки даних у реальному часі, інтеграції з мобільними пристроями, а також забезпечення легкості підтримки та розширення функціоналу в майбутньому. У цьому розділі розглянуто ключові аспекти вибору технологій для фронтенду, бекенду, бази даних та інших компонентів системи, враховуючи сучасні тенденції у веб-розробці та специфіку предметної області.

Перед вибором технологій було визначено основні вимоги до веб-платформи, що наведені в таблиці 1.1.

Таблиця 1.1

Вимоги до технологічного стеку

№	Вимога	Опис вимоги
1	Інтерактивність і зручність інтерфейсу	платформа повинна мати адаптивний і сучасний інтерфейс, який забезпечує комфортне використання як на комп'ютерах, так і на мобільних пристроях [6]
2	Продуктивність	швидке завантаження сторінок і обробка запитів користувачів, включаючи відображення графіків прогресу та рекомендацій щодо тренувань [7]
3	Масштабованість	можливість додавання нових функцій, таких як інтеграція зі сторонніми сервісами (наприклад, фітнес-трекери) або підтримка великої кількості користувачів [8]
4	Безпека	захист персональних даних користувачів, зокрема інформації про здоров'я та тренувальні плани, відповідно до стандартів GDPR [9]
5	Простота розробки та підтримки	використання технологій із широкою спільнотою розробників і доступною документацією для полегшення подальшого розвитку платформи [10]

Для реалізації клієнтської частини платформи обрано React.js у поєднанні з бібліотекою компонентів Material-UI та мовою стилізації CSS-in-JS (через Styled-Components). Обґрунтування цього вибору наступне:

– React.js є однією з найпопулярніших бібліотек для створення інтерактивних інтерфейсів завдяки компонентному підходу, який дозволяє створювати модульні та повторно використовувані елементи [11]. Це особливо важливо для платформи, де різні розділи (планування тренувань, моніторинг прогресу, профіль користувача) мають схожі елементи інтерфейсу. Крім того, React забезпечує високу швидкість рендерингу завдяки віртуальному DOM, що критично для відображення динамічних даних, таких як графіки тренувального прогресу.

– Material-UI пропонує готові компоненти, які відповідають принципам Material Design, забезпечуючи сучасний вигляд і адаптивність інтерфейсу [12].

Це зменшує час на розробку дизайну та гарантує однаковий вигляд платформи на різних пристроях.

– Styled-Components дозволяє писати CSS безпосередньо в JavaScript, що полегшує підтримку стилів і їх інтеграцію з компонентами React [13]. Це також сприяє модульності, оскільки стилі прив’язані до конкретних компонентів, що зменшує ризик конфліктів.

Такий набір технологій забезпечує швидке створення адаптивного та інтуїтивно зрозумілого інтерфейсу, що відповідає потребам користувачів, які очікують зручного доступу до тренувальних планів і статистики.

Для серверної частини платформи обрано Node.js із фреймворком Express.js. Цей вибір обґрунтований наступними факторами:

– Node.js базується на JavaScript, що дозволяє використовувати єдину мову програмування для фронтенду і бекенду [14]. Це спрощує розробку, зменшує час на навчання команди та полегшує обмін кодом між клієнтською та серверною частинами.

– Express.js є легким і гнучким фреймворком, який ідеально підходить для створення RESTful API [15]. Платформа потребує API для обробки запитів на створення тренувальних планів, збереження даних про тренування та отримання статистики. Express забезпечує швидке налаштування маршрутів і обробку HTTP-запитів, що відповідає вимогам проєкту.

– Висока продуктивність Node.js, завдяки асинхронній моделі обробки запитів, дозволяє ефективно працювати з великою кількістю одночасних підключень, що важливо для масштабованої платформи [16].

Крім того, для автентифікації користувачів обрано JSON Web Tokens (JWT) [17]. JWT забезпечує безпечну передачу даних між клієнтом і сервером, що є критично важливим для захисту персональних даних користувачів. Інтеграція JWT з Express.js дозволяє реалізувати надійну систему авторизації, включаючи ролі користувачів (наприклад, звичайний користувач або тренер).

Для зберігання даних обрано реляційну базу даних PostgreSQL у поєднанні з ORM Sequelize [18]. Обґрунтування вибору:

– PostgreSQL є потужною реляційною базою даних із відкритим кодом, яка підтримує складні запити та забезпечує високу надійність [19]. У контексті платформи вона використовується для зберігання інформації про користувачів, їхні тренувальні плани, історію тренувань і прогрес. Реляційна модель ідеально підходить для структурованих даних, таких як таблиці вправ, розкладів і статистики.

– Sequelize спрощує взаємодію з PostgreSQL, дозволяючи розробникам писати запити на JavaScript замість SQL [20]. Це підвищує продуктивність розробки та полегшує підтримку коду. Крім того, Sequelize підтримує міграції, що спрощує оновлення структури бази даних при додаванні нових функцій.

Для зберігання великих обсягів даних, таких як зображення вправ або відеоінструкції, розглядається інтеграція з хмарним сховищем, наприклад, Amazon S3, що забезпечить масштабованість і швидкий доступ до мультимедійного контенту.

Для забезпечення повноцінної роботи платформи використано низку додаткових технологій:

– Docker для контейнеризації застосунку, що полегшує розгортання платформи на різних серверах і забезпечує однакове середовище для розробки, тестування та продакшену [21].

– Git і GitHub для контролю версій і командної роботи над кодом. GitHub також використовується для автоматизації процесів CI/CD через GitHub Actions, що дозволяє швидко тестувати та розгортати оновлення [22].

– Chart.js для створення інтерактивних графіків, які відображають прогрес користувача (наприклад, зміни ваги, кількість виконаних вправ) [23]. Ця бібліотека інтегрується з React і забезпечує візуально привабливе представлення даних.

– WebSocket (через бібліотеку Socket.IO) для реалізації функцій реального часу, таких як сповіщення про завершення тренування або чат із тренером [24].

Перед остаточним вибором технологій було розглянуто альтернативи:

– Для фронтенду розглядалися Vue.js і Angular. Vue.js є легшим за Angular, але має меншу екосистему порівняно з React. Angular, своєю чергою, є надто громіздким для проєкту середнього масштабу, що могло б ускладнити розробку.

– Для бекенду розглядалися Django (Python) і Ruby on Rails. Django є потужним фреймворком, але потребує знання Python, що могло б ускладнити роботу команди, яка орієнтована на JavaScript. Ruby on Rails, хоча й продуктивний, має меншу популярність і підтримку в сучасних проєктах.

– Для бази даних альтернативою була MongoDB. Однак неструктурована природа NoSQL бази даних менш підходить для проєкту з чітко визначеною структурою даних, де важливі зв’язки між таблицями.

Вибір технологічного стеку для розробки веб-платформи для персонального планування та моніторингу тренувань ґрунтується на потребах проєкту, сучасних тенденціях у веб-розробці та можливостях команди. React.js, Node.js із Express.js, PostgreSQL та додаткові інструменти, такі як Docker і Chart.js, забезпечують оптимальний баланс між продуктивністю, масштабованістю та зручністю розробки. Цей стек дозволяє створити надійну, безпечну та зручну платформу, яка відповідає очікуванням користувачів і може бути легко розширена в майбутньому. Використання JavaScript як основної мови для фронтенду та бекенду сприяє єдності коду та спрощує підтримку, що є важливим для довгострокового успіху проєкту.

Висновки до розділу 1

Аналіз сучасних веб-платформ для персонального планування та моніторингу тренувань демонструє, що провідні рішення (MyFitnessPal, Nike Training Club, Fitbit, Strava, 8fit) пропонують широкий спектр інструментів для різних категорій користувачів. Кожна платформа має унікальні особливості: MyFitnessPal вирізняється детальним відстеженням харчування, Nike Training Club - різноманітністю тренувань, Fitbit забезпечує точний моніторинг через

носимі пристрої, Strava мотивує соціальною взаємодією, а 8fit пропонує рішення для початківців.

Виявлено спільні тенденції, що формують сучасний простір фітнес-платформ: персоналізація, інтеграція з носимими пристроями, використання штучного інтелекту для аналізу даних та соціальні функції для підвищення мотивації. Однак існуючі обмеження, такі як платний доступ до преміум-функцій, вузька спеціалізація або недостатня глибина аналітики, створюють можливості для нових конкурентоспроможних рішень.

Обґрунтування технологічного стеку підтверджує оптимальність використання React.js для створення інтерактивного фронтенду, Node.js з Express.js для ефективного бекенду та PostgreSQL для надійного зберігання даних. Використання JavaScript як основної мови для обох частин системи забезпечує єдність коду, полегшує командну роботу та зменшує витрати на підтримку.

Успішна веб-платформа повинна поєднувати персоналізацію, інтуїтивний інтерфейс, інтеграцію з носимими пристроями та надійний захист даних відповідно до міжнародних стандартів. Ключовим викликом є досягнення балансу між доступністю для широкої аудиторії та глибиною функціоналу для професійних користувачів, що визначатиме конкурентоспроможність та успіх майбутнього рішення на динамічному ринку фітнес-технологій.

РОЗДІЛ 2. РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ПЕРСОНАЛЬНОГО ПЛАНУВАННЯ ТА МОНІТОРИНГУ ТРЕНУВАНЬ

2.1 Розробка серверної частини веб-платформи

Розробка серверної частини веб-платформи для персонального планування та моніторингу тренувань є ключовим етапом створення системи, що забезпечує стабільну роботу, безпечне зберігання даних користувачів та ефективну взаємодію з клієнтською частиною. Серверна частина відповідає за обробку запитів, управління базою даних, забезпечення аутентифікації та авторизації, а також надання API для взаємодії з фронтендом. У цьому розділі детально розглянуто архітектуру серверної частини, основні технології, що використовувалися, та ключові аспекти реалізації функціоналу з наведенням фрагментів коду для ілюстрації.

Таблиця 2.1

Компоненти серверної частини

№	Компоненти	Опис компонентів
1	Веб-сервер	Для обробки HTTP-запитів використано фреймворк Node.js з бібліотекою Express.js, яка дозволяє швидко налаштувати маршрутизацію та обробку запитів
2	База даних	Для зберігання даних користувачів, планів тренувань, прогресу та інших сутностей використано реляційну базу даних PostgreSQL, яка забезпечує надійність, швидкодію та підтримку складних запитів
3	Система аутентифікації	Для захисту даних користувачів реалізовано механізм аутентифікації на основі JSON Web Tokens (JWT), що дозволяє безпечно ідентифікувати користувачів

Для реалізації серверної частини веб-платформи було обрано архітектуру REST API, яка забезпечує чітке розділення між сервером і клієнтом, а також дозволяє масштабувати систему в майбутньому. REST (Representational State Transfer) є стандартом для створення веб-сервісів, що базуються на HTTP-

запитах (GET, POST, PUT, DELETE), що забезпечує простоту інтеграції та універсальність. Основні компоненти серверної частини показані в таблиці 2.1.

Логіка обробки даних, наприклад, створення планів тренувань або збереження прогресу, реалізується у вигляді окремих модулів (сервісів), що сприяє підтримці чистого коду та легкості тестування.

Діаграма компонентів (рис. 2.1) серверної частини веб-платформи ілюструє багатошарову архітектуру, що включає веб-сервер (Node.js з Express.js) для обробки HTTP-запитів, сервісний шар для реалізації бізнес-логіки, шар доступу до реляційної бази даних PostgreSQL для зберігання даних, а також модуль аутентифікації на основі JWT для забезпечення безпеки. Компоненти взаємодіють через чітко визначені інтерфейси, що сприяє модульності та масштабованості системи.

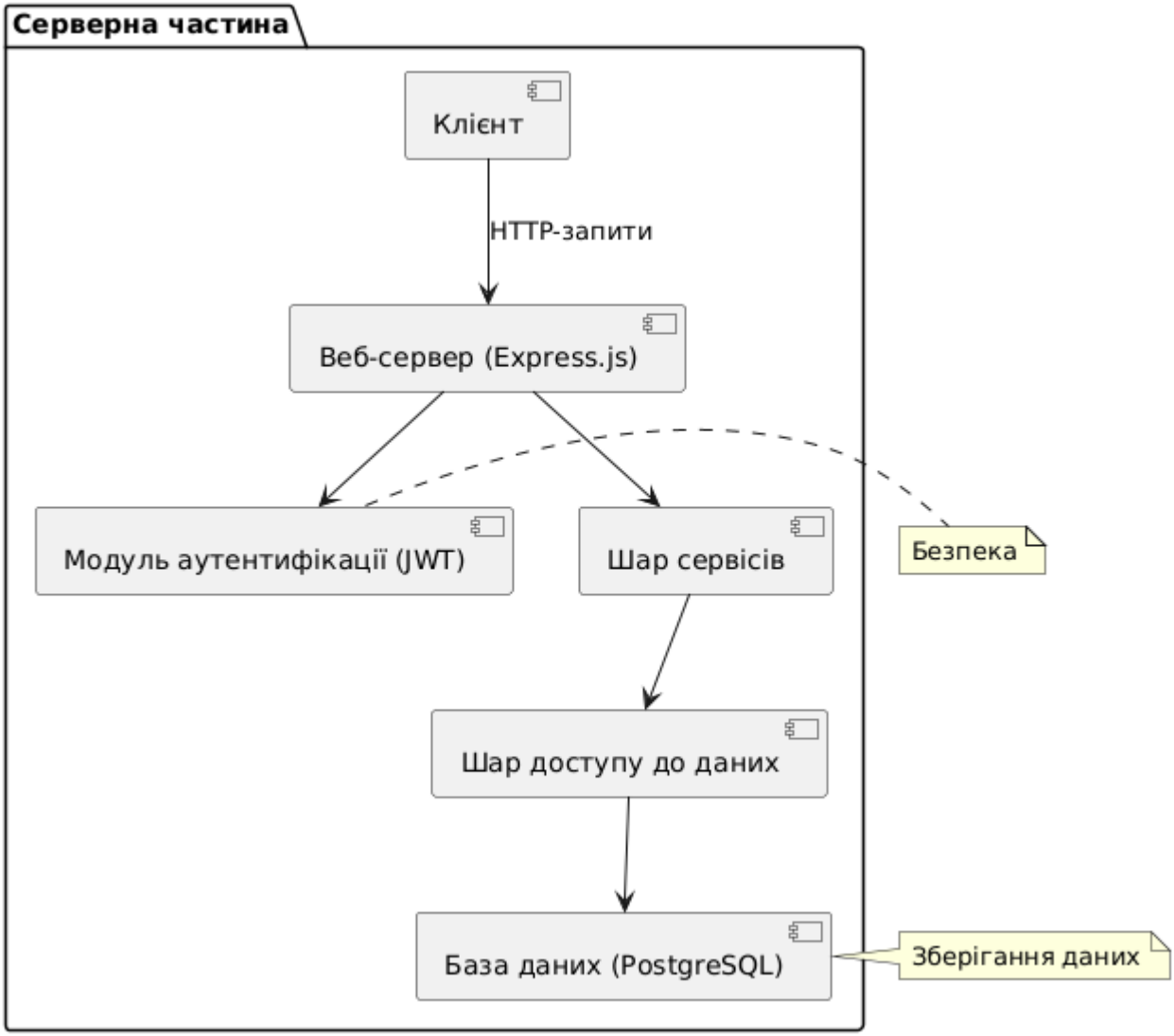

Рис. 2.1 – Діаграма компонентів серверної частини

Серверна частина побудована за принципом багатошарової архітектури, що включає шари маршрутизації, сервісів, моделей даних та доступу до бази даних. Це дозволяє чітко розділити обов’язки між компонентами та полегшує підтримку коду.

Для реалізації серверної частини було використано сучасний стек технологій, який забезпечує високу продуктивність і зручність розробки (таблиця 2.2).

Таблиця 2.2

Використані технології

№	Технології	Опис технологій
1	Node.js	Асинхронне середовище виконання JavaScript, яке ідеально підходить для створення швидких і масштабованих серверів
2	Express.js	Легкий фреймворк для створення RESTful API, який спрощує налаштування маршрутів і обробку запитів
3	PostgreSQL	Реляційна база даних з відкритим кодом, яка підтримує складні запити та забезпечує високу надійність
4	Sequelize	ORM (Object-Relational Mapping) для роботи з PostgreSQL, що дозволяє взаємодіяти з базою даних за допомогою об’єктів JavaScript, спрощуючи написання запитів
5	JWT	Технологія для реалізації безпечної аутентифікації та авторизації
6	Bcrypt	Бібліотека для хешування паролів, що забезпечує захист конфіденційних даних користувачів

Цей набір технологій дозволяє створити надійну та гнучку серверну частину, яка відповідає вимогам безпеки, масштабованості та зручності використання.

На початковому етапі розробки було створено базову структуру серверної частини. Основний файл server.js відповідає за ініціалізацію сервера, підключення до бази даних та налаштування маршрутів. Нижче наведено фрагмент коду, який демонструє базову конфігурацію сервера з використанням Express.js:

```
const express = require('express');
const cors = require('cors');
const sequelize = require('./config/database');
const authRoutes = require('./routes/auth');
const workoutRoutes = require('./routes/workouts');

const app = express();

// Налаштування middleware
app.use(cors());
app.use(express.json());

// Підключення маршрутів
```

```
app.use('/api/auth', authRoutes);
app.use('/api/workouts', workoutRoutes);

// Синхронізація з базою даних
sequelize.sync({ force: false })
  .THEN(() => {
    console.log('Database connected successfully');
    app.listen(5000, () => {
      console.log('Server is running on port 5000');
    });
  })
  .catch((error) => {
    console.error('Unable to connect to the database:', error);
  });
```

У цьому коді:

– express.json() забезпечує парсинг вхідних JSON-даних.

– cors дозволяє обробляти запити з різних доменів, що необхідно для взаємодії з фронтендом.

– sequelize.sync() синхронізує моделі з базою даних, створюючи необхідні таблиці.

– Сервер запускається на порту 5000.

Безпека є критично важливим аспектом веб-платформи, оскільки система обробляє персональні дані користувачів, такі як плани тренувань та прогрес. Для реалізації аутентифікації використано JWT, який дозволяє створювати токени для ідентифікації користувачів. Процес аутентифікації включає реєстрацію, вхід у систему та перевірку токенів для захищених маршрутів.

Фрагмент коду нижче демонструє реалізацію маршруту для реєстрації користувача:

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { User } = require('../models');

router.post('/register', async (req, res) => {
  try {
    const { username, email, password } = req.body;

    // Перевірка, чи існує користувач
    const existingUser = await User.findOne({ where: { email } });
    if (existingUser) {
```

```
      return res.status(400).json({ message: 'User already exists'
});
    }

    // Хешування пароля
    const hashedPassword = await bcrypt.hash(password, 10);

    // Створення нового користувача
    const user = await User.create({
      username,
      email,
      password: hashedPassword,
    });

    // Генерація JWT
    const token = jwt.sign({ id: user.id },
process.env.JWT_SECRET, {
      expiresIn: '1h',
    });

    res.status(201).json({ token, user: { id: user.id, username,
email } });
  } catch (error) {
    res.status(500).json({ message: 'Error registering user',
error });
  }
});
```

У цьому коді:

– bcrypt.hash використовується для безпечного хешування пароля.

– jwt.sign генерує токен, який містить ідентифікатор користувача та має обмежений час дії (1 година).

– Дані користувача зберігаються в базі даних за допомогою моделі User, створеної через Sequelize.

Для захисту маршрутів, наприклад, доступу до планів тренувань, реалізовано middleware для перевірки токена:

```
const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'Access token required'
});
  }
```

```
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) {
      return res.status(403).json({ message: 'Invalid token' });
    }
    req.user = user;
    next();
  });
};

module.exports = authenticateToken;
```

Цей middleware перевіряє наявність і валідність токена, дозволяючи доступ до захищених ресурсів лише авторизованим користувачам.

Одним із ключових функціональних модулів серверної частини є управління планами тренувань. Користувачі можуть створювати, редагувати, видаляти та переглядати плани, які включають вправи, кількість підходів, повторень та інші параметри. Для цього створено окремий маршрут /api/workouts, який обробляє запити, пов’язані з тренуваннями.

Фрагмент коду для створення нового плану тренувань:

```
const { Workout } = require('../models');
const authenticateToken = require('../middleware/auth');

router.post('/', authenticateToken, async (req, res) => {
  try {
    const { title, exercises, date } = req.body;
    const userId = req.user.id;

    const workout = await Workout.create({
      title,
      exercises,
      date,
      userId,
    });

    res.status(201).json({ message: 'Workout created
successfully', workout });
  } catch (error) {
    res.status(500).json({ message: 'Error creating workout',
error });
  }
});
```

У цьому коді:

– authenticateToken забезпечує, що лише авторизовані користувачі можуть створювати плани.

– Дані тренування зберігаються в таблиці Workouts, пов’язаній з користувачем через userId.

– Поле exercises зберігається у форматі JSON, що дозволяє гнучко зберігати список вправ.

Для моніторингу прогресу користувачів реалізовано функціонал збереження виконаних тренувань та їх результатів (наприклад, вага, кількість повторень). Ці дані дозволяють користувачам відстежувати свої досягнення та аналізувати ефективність тренувань. Нижче наведено приклад маршруту для збереження результатів тренування:

```
const { Progress } = require('../models');
const authenticateToken = require('../middleware/auth');

router.post('/progress', authenticateToken, async (req, res) => {
  try {
    const { workoutId, results } = req.body;
    const userId = req.user.id;

    const progress = await Progress.create({
      workoutId,
      userId,
      results,
      date: new Date(),
    });

    res.status(201).json({ message: 'Progress saved successfully',
progress });
  } catch (error) {
    res.status(500).json({ message: 'Error saving progress', error
});
  }
});
```

Цей код зберігає результати тренування в таблиці Progress, пов’язуючи їх із конкретним тренуванням (workoutId) та користувачем (userId).

Для забезпечення високої продуктивності та безпеки серверної частини було реалізовано заходи, представлені в таблиці 2.3.

Таблиця 2.3

Заходи безпеки

№	Заходи	Опис заходів
1	Кешування запитів	Використано Redis для кешування часто запитуваних даних, наприклад, списків тренувань
2	Обмеження запитів	Впроваджено middleware express-rate-limit для захисту від DDoS-атак
3	Валідація даних	Усі вхідні дані перевіряються за допомогою бібліотеки Joi, щоб уникнути SQL-ін’єкцій та інших вразливостей
4	Логування	Реалізовано логування запитів і помилок за допомогою бібліотеки winston для полегшення діагностики проблем

Розробка серверної частини веб-платформи для персонального планування та моніторингу тренувань дозволила створити надійну та функціональну систему, яка забезпечує обробку запитів, безпечне зберігання даних і зручну взаємодію з клієнтською частиною. Використання сучасних технологій, таких як Node.js, Express.js, PostgreSQL і JWT, забезпечило високу продуктивність, безпеку та гнучкість системи. Наведені фрагменти коду ілюструють ключові аспекти реалізації, включаючи налаштування сервера, аутентифікацію, управління планами тренувань і моніторинг прогресу. У подальшій розробці планується додавання нових функцій, таких як інтеграція з носимими пристроями та аналітика тренувань, що ще більше розширить можливості платформи.

2.2 Розробка фронтенд-частини та інтерфейсу користувача

Розробка фронтенд-частини веб-платформи для персонального планування та моніторингу тренувань є ключовим етапом створення зручного, інтуїтивно зрозумілого та функціонального продукту, що забезпечує взаємодію користувача з системою. Фронтенд-розробка охоплює створення клієнтської частини веб-додатку, яка відповідає за візуалізацію даних, інтерактивність і адаптивність інтерфейсу. У цьому розділі розглянуто основні аспекти розробки

інтерфейсу користувача, включаючи технологічний стек, принципи дизайну, опис екранних форм із фрагментами коду, а також підходи до забезпечення адаптивності та оптимізації продуктивності.

Для реалізації фронтенд-частини веб-платформи використано сучасний стек технологій, що забезпечує високу продуктивність, масштабованість і зручність розробки. Основні технології показані в таблиці 2.4.

Таблиця 2.4

Технології створення клієнтської частини

№	Технології	Опис технологій
1	HTML5	для створення структури веб-сторінок, що забезпечує семантичну розмітку та доступність
2	CSS3	для стилізації елементів інтерфейсу, включаючи адаптивні макети за допомогою технологій Flexbox і CSS Grid
3	JavaScript (ES6+)	для реалізації інтерактивних елементів і логіки клієнтської сторони
4	React.js	бібліотека для створення компонентного інтерфейсу, що дозволяє будувати модульні та повторно використовувані компоненти
5	Redux	для управління станом програми, що забезпечує передбачуваність і централізоване зберігання даних
6	Axios	для виконання асинхронних HTTP-запитів до серверної частини
7	Figma	для прототипування дизайну та співпраці з дизайнерами

Додатково використано інструменти для оптимізації розробки: Webpack для збирання проєкту, ESLint для забезпечення якості коду, Git та GitHub для контролю версій і командної роботи. Такий набір технологій відповідає сучасним стандартам фронтенд-розробки та забезпечує ефективну реалізацію складних інтерфейсів.

Інтерфейс користувача розроблено з урахуванням принципів UI/UX-дизайну, щоб забезпечити зручність, інтуїтивність і естетичну привабливість. Основні принципи, застосовані під час розробки, описані в таблиці 2.5.

Таблиця 2.5

Принципи UI/UX-дизайну

№	Принципи	Опис принципів
1	Простота та ясність	Інтерфейс мінімізує когнітивне навантаження, пропонуючи чіткі навігаційні елементи та зрозумілі форми введення даних
2	Консистентність	Використано єдину колірну палітру, типографіку та стилі компонентів, що відповідає брендбуку платформи
3	Адаптивність	Інтерфейс оптимізований для різних пристроїв (десктоп, планшет, смартфон) за допомогою медіа-запитів і адаптивних макетів
4	Доступність (Accessibility)	Дотримано стандартів WCAG 2.1, включаючи семантичну розмітку, підтримку клавіатурної навігації та достатній контраст елементів
5	Інтерактивність	Використано анімації та динамічні елементи для підвищення залученості користувача без шкоди для продуктивності

Дизайн інтерфейсу спочатку був створений у Figma, де розроблено прототипи основних екранних форм. Це дозволило протестувати користувацький досвід ще на етапі планування та внести корективи до початку кодування.

Фронтенд-частина платформи включає кілька ключових екранних форм, які забезпечують основний функціонал: авторизацію, створення тренувального плану, моніторинг прогресу та перегляд статистики. Нижче наведено опис кожної форми з відповідними фрагментами коду.

Форма авторизації (рис. 2.2, рис. 2.3) є початковою точкою взаємодії користувача з платформою. Вона містить поля для введення електронної пошти та пароля, а також кнопки для входу та переходу до реєстрації. Форма має валідацію введених даних і відображає повідомлення про помилки.

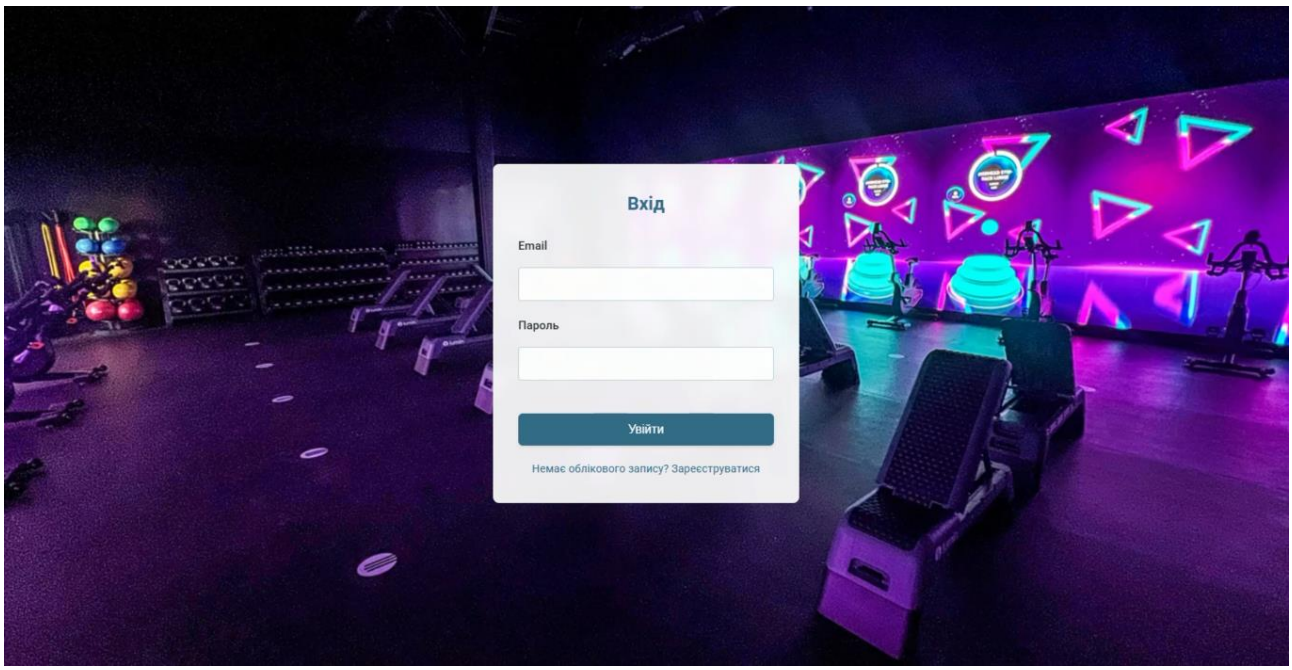

Рис. 2.2 – Форма авторизації

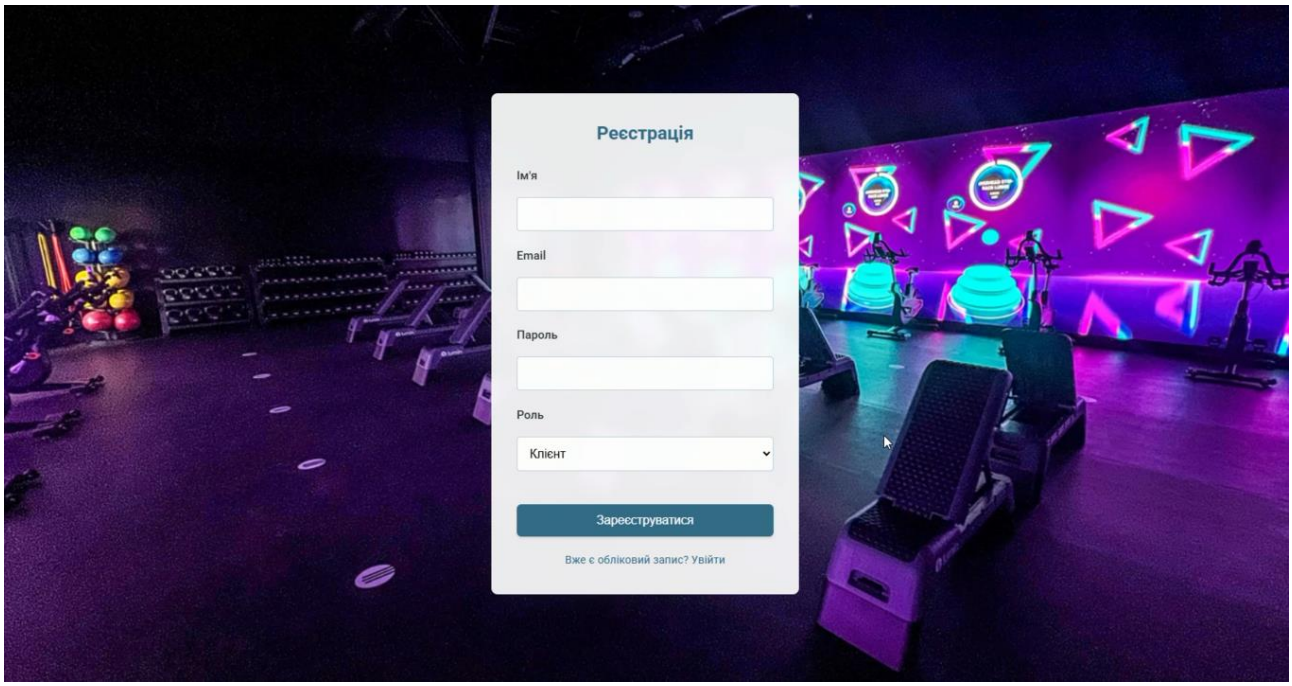

Рис. 2.3 – Форма реєстрації

Фрагмент коду (React-компонент LoginForm.js):

```
import React, { useState } from 'react';
import './LoginForm.css';

const LoginForm = () => {
  const [email, setEmail] = useState('');
```

```
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!email || !password) {
      setError('Будь ласка, заповніть усі поля');
      return;
    }
    // Логіка авторизації (запит до API)
    console.log('Авторизація:', { email, password });
  };

  return (
    <div className="login-container">
      <h2>Вхід</h2>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="email">Електронна пошта</label>
          <input
            type="email"
            id="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="password">Пароль</label>
          <input
            type="password"
            id="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>
        {error && <p className="error">{error}</p>}
        <button type="submit">Увійти</button>
      </form>
      <p>
        Немає облікового запису? <a
href="/register">Зареєструватися</a>
      </p>
    </div>
  );
};

export default LoginForm;
```

Стилі (LoginForm.css):

```
.login-container {
  max-width: 400px;
  margin: 50px auto;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-group {
  margin-bottom: 15px;
}

label {
  display: block;
  margin-bottom: 5px;
}

input {
  width: 100%;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}

.error {
  color: red;
  font-size: 14px;
}
```

Форма має мінімалістичний дизайн із чіткими мітками та полями введення. Використано адаптивний макет, що забезпечує коректне відображення на мобільних пристроях. Валідація виконується на клієнтській стороні, а при натисканні кнопки "Увійти" дані надсилаються до серверної частини через Axios.

Форма створення тренувального плану (рис. 2.4) дозволяє користувачу визначити тип тренування, тривалість, інтенсивність і дні тижня. Вона включає випадаючі списки, поля введення та кнопку збереження.

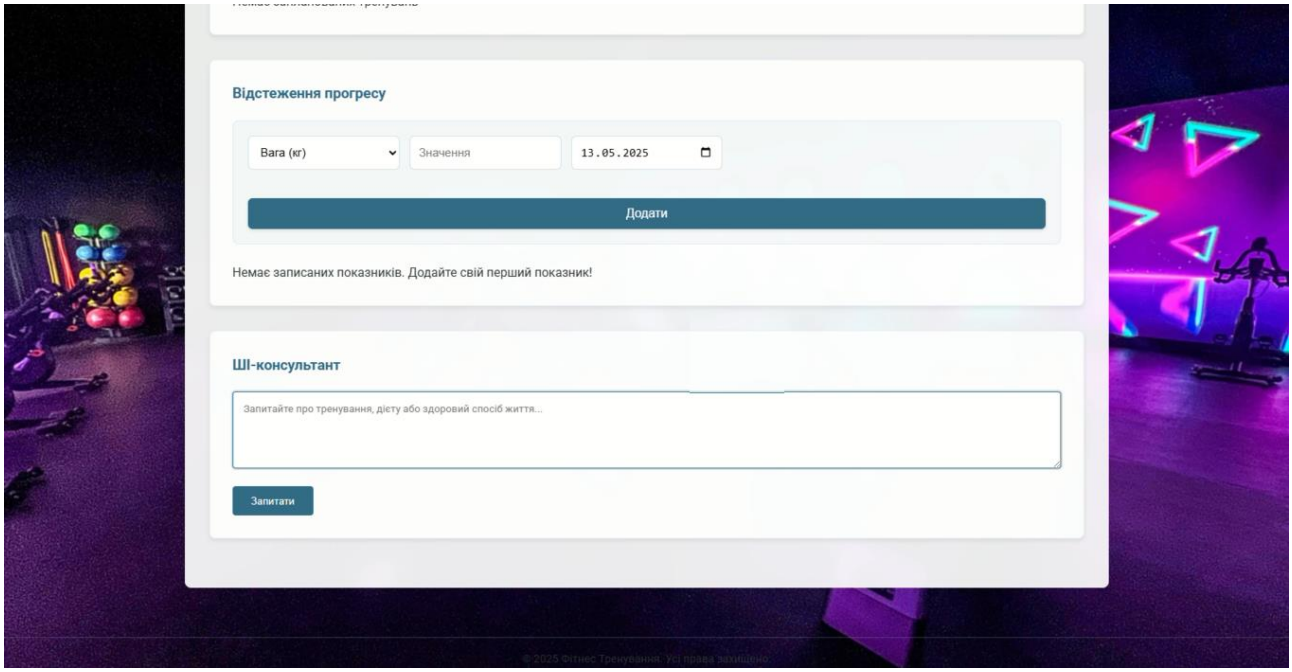

Рис. 2.4 – Форма створення тренувального плану

Фрагмент коду (React-компонент TrainingPlanForm.js):

```
import React, { useState } from 'react';
import './TrainingPlanForm.css';

const TrainingPlanForm = () => {
  const [plan, setPlan] = useState({
    type: '',
    duration: '',
    intensity: '',
    days: [],
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setPlan({ ...plan, [name]: value });
  };

  const handleDaysChange = (e) => {
    const { value, checked } = e.target;
    setPlan((prev) => ({
      ...prev,
      days: checked
```

```
        ? [...prev.days, value]
        : prev.days.filter((day) => day !== value),
    }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    // Логіка збереження плану
    console.log('Новий план:', plan);
  };

  return (
    <div className="plan-container">
      <h2>Створити тренувальний план</h2>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="type">Тип тренування</label>
          <select name="type" id="type" onChange={handleChange}
required>
            <option value="">Оберіть тип</option>
            <option value="cardio">Кардіо</option>
            <option value="strength">Силові</option>
            <option value="yoga">Йога</option>
          </select>
        </div>
        <div className="form-group">
          <label htmlFor="duration">Тривалість (хв)</label>
          <input
            type="number"
            name="duration"
            id="duration"
            onChange={handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="intensity">Інтенсивність</label>
          <select name="intensity" id="intensity"
onChange={handleChange} required>
            <option value="">Оберіть інтенсивність</option>
            <option value="low">Низька</option>
            <option value="medium">Середня</option>
            <option value="high">Висока</option>
          </select>
        </div>
        <div className="form-group">
          <label>Дні тижня</label>
          {['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Нд'].map((day) =>
(
            <label key={day}>
              <input
                type="checkbox"
                value={day}
```

```
                  onChange={handleDaysChange}
                />
                {day}
              </label>
            ))}
          </div>
          <button type="submit">Зберегти план</button>
        </form>
      </div>
    );
};

export default TrainingPlanForm;
```

Форма дозволяє гнучко налаштувати тренувальний план. Випадаючі списки забезпечують швидкий вибір параметрів, а чекбокси дозволяють обрати кілька днів тижня. Дані зберігаються в стані компонента за допомогою хука useState, що забезпечує реактивність інтерфейсу.

Екран моніторингу прогресу (рис. 2.5) відображає статистику тренувань у вигляді таблиці та графіків. Користувач може переглядати виконані тренування, їх тривалість і калорії.

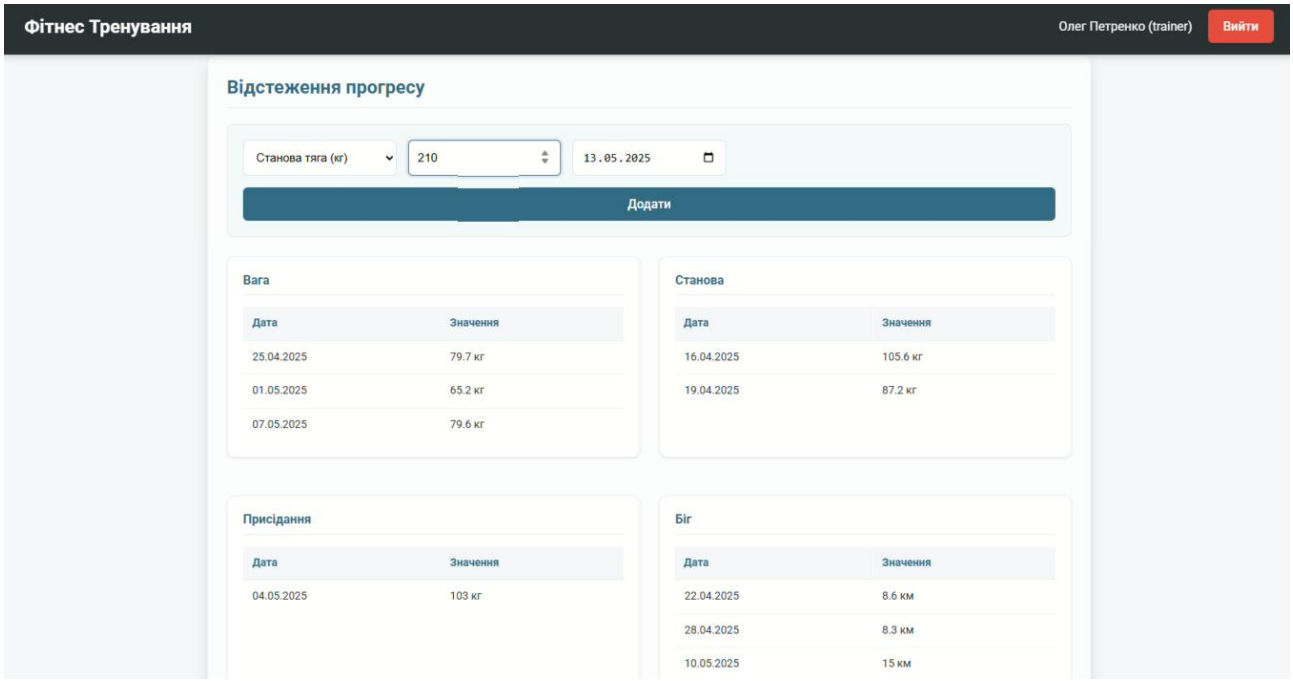

Рис. 2.5 – Екран моніторингу прогресу

Фрагмент коду (React-компонент ProgressTracker.js):

```
import React from 'react';
import './ProgressTracker.css';

const ProgressTracker = ({ workouts }) => {
  return (
    <div className="progress-container">
      <h2>Прогрес тренувань</h2>
      <table>
        <thead>
          <tr>
            <th>Дата</th>
            <th>Тип</th>
            <th>Тривалість (хв)</th>
            <th>Калорії</th>
          </tr>
        </thead>
        <tbody>
          {workouts.map((workout, index) => (
            <tr key={index}>
              <td>{workout.date}</td>
              <td>{workout.type}</td>
              <td>{workout.duration}</td>
              <td>{workout.calories}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default ProgressTracker;
```

Стилі (ProgressTracker.css):

```
.progress-container {
  max-width: 800px;
  margin: 50px auto;
}

table {
  width: 100%;
  border-collapse: collapse;
}

th, td {
  border: 1px solid #ccc;
  padding: 10px;
  text-align: left;
}
```

```
th {
  background-color: #f4f4f4;
}

@media (max-width: 600px) {
  table {
    font-size: 14px;
  }
}
```

Екран відображає дані в табличному форматі, що полегшує аналіз прогресу. Для покращення користувацького досвіду планується інтеграція бібліотеки Chart.js для візуалізації даних у вигляді графіків.

Для забезпечення коректного відображення на різних пристроях використано адаптивний дизайн. Медіа-запити в CSS дозволяють змінювати стилі залежно від розміру екрана:

```
@media (max-width: 768px) {
  .login-container, .plan-container, .progress-container {
    padding: 15px;
    margin: 20px;
  }

  input, select, button {
    font-size: 16px;
  }
}
```

Оптимізація продуктивності досягнута за рахунок:

– Лінивої загрузки (lazy loading) зображень і компонентів.
– Мінімізації CSS і JavaScript за допомогою Webpack.
– Кешування запитів до API для зменшення навантаження на сервер.

Фронтенд-частина пройшла тестування на різних рівнях:

– Юніт-тести за допомогою Jest для перевірки логіки компонентів.
– Інтеграційні тести для перевірки взаємодії з API.
– Кросбраузерне тестування у Chrome, Firefox і Safari.
– Тестування доступності за допомогою інструменту Lighthouse.

Розробка фронтенд-частини веб-платформи для персонального планування та моніторингу тренувань дозволила створити зручний і

функціональний інтерфейс, що відповідає сучасним стандартам веб-розробки. Використання React.js і Redux забезпечило модульність і передбачуваність, а адаптивний дизайн і оптимізація підвищили доступність і продуктивність. Описані екранні форми (авторизація, створення плану, моніторинг прогресу) демонструють гнучкість і орієнтованість на користувача. Подальший розвиток передбачає інтеграцію додаткових функцій, таких як push-повідомлення та інтерактивні графіки, для підвищення залученості користувачів.

Цей підхід до розробки фронтенду може бути застосований до інших проєктів, де важливі зручність, інтерактивність і адаптивність інтерфейсу.

Висновки до розділу 2

Розробка веб-платформи для персонального планування та моніторингу тренувань є комплексним процесом, що охоплює створення серверної та клієнтської частин системи з акцентом на стабільність роботи, безпеку даних і зручність використання.

Серверна частина базується на сучасних технологіях Node.js, Express.js, PostgreSQL і JWT, що дозволило створити надійну, масштабовану та безпечну систему. Багатошарова архітектура включає шари маршрутизації, сервісів, моделей даних і доступу до бази даних, забезпечуючи чітке розподілення обов'язків між компонентами та полегшуючи подальшу підтримку системи. Використання ORM Sequelize спростило взаємодію з базою даних, дозволяючи розробникам зосередитися на бізнес-логіці замість написання складних SQL-запитів.

Особливу увагу приділено безпеці: механізми аутентифікації на основі JWT, хешування паролів через Bcrypt, валідація даних за допомогою Joi та обмеження запитів через express-rate-limit мінімізували ризики несанкціонованого доступу та атак. Кешування через Redis і логування за допомогою Winston підвищили продуктивність і полегшили діагностику

проблем. Функціонал управління планами тренувань і моніторингу прогресу забезпечив гнучкість для користувачів.

Фронтенд-частина зосереджена на створенні інтуїтивного та адаптивного інтерфейсу відповідно до сучасних стандартів UI/UX-дизайну. Використання React.js і Redux дозволило побудувати модульну клієнтську частину з незалежними та повторно використовуваними компонентами. Адаптивний дизайн через CSS3, Flexbox, CSS Grid і медіа-запити гарантує коректне відображення на різних пристроях. Дотримання принципів доступності зробило платформу зручною для широкого кола користувачів, включаючи тих, хто використовує допоміжні технології.

Інтеграція Axios для асинхронних запитів до API забезпечила швидку взаємодію з сервером, а оптимізація через ліниву загрузку та кешування зменшила час завантаження. Тестування, включаючи юніт-тести, інтеграційні тести та перевірку доступності через Lighthouse, підтвердило високу якість системи.

Створена платформа поєднала технічну досконалість з орієнтацією на потреби користувача, досягнувши високої продуктивності та масштабованості. Це відкриває перспективи для подальшого розвитку, зокрема інтеграції з носимими пристроями та додавання аналітичних інструментів, демонструючи потенціал сучасних веб-технологій у створенні інноваційних рішень.

ВИСНОВКИ

Проведений аналіз існуючих рішень у сфері планування та моніторингу фітнес-тренувань виявив, що багато платформ мають обмеження у функціональності або недостатню адаптивність до потреб користувачів. Наприклад, деякі додатки не забезпечують гнучкої персоналізації тренувальних планів або мають складний інтерфейс, що ускладнює відстеження прогресу. Цей аналіз дозволив визначити ключові вимоги до нової веб-платформи, зокрема зручність, інтуїтивність та широкий функціонал.

Обґрунтування вибору технологічного стеку для розробки веб-платформи ґрунтувалося на детальному вивченні сучасних інструментів, їхніх переваг і обмежень у контексті проєктних завдань. Для фронтенду обрано технології, що забезпечують високу швидкість рендерингу та модульність інтерфейсу, сприяючи зручності розробки та підтримки. Бекенд-технології підібрано з урахуванням їхньої масштабованості та надійності, щоб гарантувати стабільну роботу платформи навіть за високого навантаження.

Формулювання функціональних і нефункціональних вимог базувалося на аналізі потреб користувачів і визначенні необхідного функціоналу. Зокрема, для ефективного моніторингу прогресу передбачено можливості збереження результатів тренувань, створення графіків для аналізу досягнень і налаштування сповіщень для підтримки мотивації користувачів.

Розробка веб-платформи відповідно до встановлених вимог включала етапи проєктування, програмування та тестування. Кожен етап ретельно планувався та виконувався з урахуванням поставлених цілей, що дозволило створити функціональний і зручний продукт, орієнтований на користувача.

Після завершення розробки проведено тестування та оцінку платформи, що включало перевірку роботи на різних пристроях, виявлення й усунення помилок, а також збір відгуків від потенційних користувачів. Цей етап підтвердив відповідність платформи встановленим вимогам, її надійність і готовність до використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MyFitnessPal URL: https://www.myfitnesspal.com/en
2. Nike Training Club URL: https://www.nike.com/pl/aplikacja-ntc
3. Fitbit URL: https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile&hl=pl
4. Strava URL: https://www.strava.com/
5. 8fit URL: https://8fit.com/
6. Zuen Cen, Yuxin Zhao. Enhancing User Engagement through Adaptive Interfaces: A Study on Real-time Personalization in Web Applications. 2024
7. Популярні технології для тестування продуктивності URL: https://training.qatestlab.com/blog/technical-articles/popular-technologies-for-performance-testing/
8. Архітектура вебзастосунків 2024: Ультимативний гайд для розробників URL: https://robotdreams.cc/uk/blog/567-arhitektura-vebzastosunkiv
9. Нові правила захисту персональних даних: як відповідати стандартам ЄС? URL: https://yur-gazeta.com/dumka-eksperta/novi-pravila-zahistu-personalnih-danih-yak-vidpovidati-standartam-es-.html
10. A Guide to Web Development Frameworks URL: https://builtin.com/articles/web-development-frameworks
11. React URL: https://react.dev/
12. Material UI URL: http://mui.com/material-ui/
13. CSS for the <Component> Age URL: https://styled-components.com/
14. Які зараз перспективи у Node js URL: https://foxminded.ua/node-js-perspektyvy/
15. Express URL: https://expressjs.com/
16. Детальний огляд та розбір Node.js URL: https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs
17. JSON Web Tokens URL: https://jwt.io/

18. Як Sequelize допомагає в роботі з базами даних у Node.js URL: https://foxminded.ua/sequelize-shcho-tse/

19. Розробка ECOMMERCE проектів Technologies POSTGRESQL URL: https://brander.ua/technologies/postgresql

20. Using Sequelize ORM to manage relationships in a PostgreSQL database URL: https://medium.com/statuscode/using-sequelize-orm-to-manage-relationships-in-a-postgresql-database-4fb3f78dfa5b

21. Досвід контейнеризації Node.js застосунків з Docker URL: https://devzone.org.ua/post/dosvid-konteyneryzatsiyi-nodejs-zastosunkiv-z-docker

22. Що таке GitHub і навіщо він потрібен розробнику URL: https://foxminded.ua/shho-take-github-i-navishho-vin-potriben-rozrobniku/

23. Chart.js URL: https://www.w3schools.com/ai/ai_chartjs.asp

24. Socket-IO для початківців URL: https://javascript.org.ua/socket-io-dlya-pochatkivcziv-%F0%9F%94%8C/

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

server.js

```
const express = require('express');
const cors = require('cors');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { GoogleGenerativeAI } = require('@google/generative-ai');
const sqlite3 = require('sqlite3').verbose();
const { open } = require('sqlite');
        const path = require('path');
        const dotenv = require('dotenv');
// Завантаження змінних середовища
        dotenv.config();
// Ініціалізація додатку
        const app = express();
        const PORT = process.env.PORT || 5000;
        const JWT_SECRET = process.env.JWT_SECRET || 'your_jwt_secret_key';
        const GEMINI_API_KEY = process.env.GEMINI_API_KEY ||
'your_gemini_api_key';
// Middleware
        app.use(cors());
        app.use(express.json());
// Логування запитів для відлагодження
        app.use((req, res, next) => {
        console.log(`${req.method} ${req.path}`);
        next();
        });
// Ініціалізація Gemini AI
        let geminiModel = null;
        let geminiAvailable = false;
        try {
        const genAI = new GoogleGenerativeAI(GEMINI_API_KEY);
        // Використовуємо модель gemini-1.5-flash замість gemini-pro
        geminiModel = genAI.getGenerativeModel({
        model: "gemini-1.5-flash",
        generationConfig: {
        temperature: 0.7,
        topK: 40,
        topP: 0.95,
        maxOutputTokens: 800,
        }
        });
        console.log("Модель Gemini успішно ініціалізована");
        } catch (err) {
        console.error('Помилка ініціалізації Gemini AI:', err);
        }
// Ініціалізація SQLite бази даних
        let db;
// Функція для ініціалізації бази даних
        async function initializeDatabase() {
        // Відкриваємо з'єднання з базою даних
        db = await open({
        filename: path.join(__dirname, 'fitness.db'),
        driver: sqlite3.Database
        });
        // Створюємо таблиці
        await db.exec(`
        CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        name TEXT NOT NULL,
        email TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        role TEXT NOT NULL DEFAULT 'client',
        trainer_id INTEGER,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (trainer_id) REFERENCES users (id)
        );
        CREATE TABLE IF NOT EXISTS events (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        date TIMESTAMP NOT NULL,
        type TEXT NOT NULL,
        client_id INTEGER NOT NULL,
        trainer_id INTEGER,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (client_id) REFERENCES users (id),
        FOREIGN KEY (trainer_id) REFERENCES users (id)
        );
        CREATE TABLE IF NOT EXISTS metrics (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        type TEXT NOT NULL,
        value REAL NOT NULL,
        date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users (id)
        );
        CREATE TABLE IF NOT EXISTS exercises (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        description TEXT,
        type TEXT NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        );
        `);
        // Створюємо адміністратора за замовчуванням, якщо він не існує
        const adminExists = await db.get('SELECT id FROM users WHERE email = ?',
['admin@example.com']);
        if (!adminExists) {
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash('admin123', salt);
        await db.run(
        'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
        ['Administrator', 'admin@example.com', hashedPassword, 'admin']
        );
        console.log('Адміністратор за замовчуванням створений');
        }
        // Додаємо тестові вправи, якщо таблиця порожня
        const exercisesCount = await db.get('SELECT COUNT(*) as count FROM
exercises');
        if (exercisesCount.count === 0) {
        const testExercises = [
        { title: 'Жим штанги лежачи', description: 'Базова вправа для грудних
м\'язів', type: 'силові' },
        { title: 'Присідання зі штангою', description: 'Базова вправа для ніг',
type: 'силові' },
        { title: 'Станова тяга', description: 'Базова вправа для спини та ніг',
type: 'силові' },
        { title: 'Біг на біговій доріжці', description: 'Кардіо тренування для
витривалості', type: 'кардіо' },
        { title: 'Їзда на велотренажері', description: 'Кардіо тренування з
меншим навантаженням на суглоби', type: 'кардіо' },
        { title: 'Інтервальне тренування (HIIT)', description: 'Чергування
```

```
високої та низької інтенсивності', type: 'похудіння' },
        { title: 'Бурпі', description: 'Комплексна вправа для всього тіла',
type: 'похудіння' },
        { title: 'Йога', description: 'Комплекс вправ для гнучкості та балансу',
type: 'інші' },
        { title: 'Пілатес', description: 'Вправи для зміцнення корпусу', type:
'інші' },
        { title: 'Розтяжка', description: 'Комплекс вправ для покращення
гнучкості', type: 'інші' },
        { title: 'Функціональне тренування', description: 'Вправи, що імітують
повсякденні рухи', type: 'інші' }
        ];
        const stmt = await db.prepare('INSERT INTO exercises (title,
description, type) VALUES (?, ?, ?)');
        for (const exercise of testExercises) {
        await stmt.run(exercise.title, exercise.description, exercise.type);
        }
        await stmt.finalize();
        console.log('Тестові вправи додані до бази даних');
        }
        console.log('База даних ініціалізована');
        }
// Функція для перевірки з'єднання з Gemini API
        async function testGeminiConnection() {
        if (!GEMINI_API_KEY || GEMINI_API_KEY === 'your_gemini_api_key') {
        console.warn('ПОПЕРЕДЖЕННЯ: GEMINI_API_KEY не налаштовано. Функція ШІ-
консультанта буде недоступна.');
        return false;
        }
        if (!geminiModel) {
        console.warn('ПОПЕРЕДЖЕННЯ: Модель Gemini не ініціалізована. Функція ШІ-
консультанта буде недоступна.');
        return false;
        }
        try {
        // Тестовий запит із використанням нового форматування
        const prompt = "Скажи 'Я працюю' українською мовою";
        const result = await geminiModel.generateContent({
        contents: [{
        role: "user",
        parts: [{ text: prompt }]
        }]
        });
        if (result && result.response) {
        const text = result.response.text();
        console.log('Перевірка з\'єднання з Gemini API: успішно');
        console.log(`Тестова відповідь: ${text}`);
        return true;
        } else {
        throw new Error('Отримано порожню відповідь від API');
        }
        } catch (err) {
        console.error('Помилка підключення до Gemini API:', err);
        console.warn('Функція ШІ-консультанта буде недоступна.');
        return false;
        }
        }
// Middleware для перевірки токена
        const authMiddleware = async (req, res, next) => {
        try {
        const token = req.headers.authorization?.split(' ')[1];
        if (!token) {
        return res.status(401).json({ message: 'Немає токену авторизації' });
        }
```

```
        const decoded = jwt.verify(token, JWT_SECRET);
        const user = await db.get('SELECT * FROM users WHERE id = ?',
[decoded.userId]);
        if (!user) {
        return res.status(404).json({ message: 'Користувача не знайдено' });
        }
        req.user = {
        _id: user.id,
        name: user.name,
        email: user.email,
        role: user.role,
        trainer_id: user.trainer_id
        };
        next();
        } catch (err) {
        console.error('Помилка авторизації:', err);
        return res.status(401).json({ message: 'Недійсний токен' });
        }
        };
// Middleware для перевірки ролі
        const roleMiddleware = (roles) => {
        return (req, res, next) => {
        if (!roles.includes(req.user.role)) {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        next();
        };
        };
// Маршрути аутентифікації
        app.post('/api/auth/register', async (req, res) => {
        try {
        const { name, email, password, role } = req.body;
        // Перевірка чи існує користувач
        const existingUser = await db.get('SELECT id FROM users WHERE email =
?', [email]);
        if (existingUser) {
        return res.status(400).json({ message: 'Користувач з таким email вже
існує' });
        }
        // Хешування пароля
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash(password, salt);
        // Створення нового користувача
        const result = await db.run(
        'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
        [name, email, hashedPassword, role || 'client']
        );
        const userId = result.lastID;
        // Створення JWT токена
        const token = jwt.sign({ userId }, JWT_SECRET, { expiresIn: '7d' });
        const newUser = await db.get('SELECT * FROM users WHERE id = ?',
[userId]);
        res.status(201).json({
        token,
        user: {
        _id: newUser.id,
        name: newUser.name,
        email: newUser.email,
        role: newUser.role,
        trainer_id: newUser.trainer_id
        }
        });
        } catch (err) {
        console.error('Помилка реєстрації:', err);
```

```
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.post('/api/auth/login', async (req, res) => {
        try {
        const { email, password } = req.body;
        // Пошук користувача
        const user = await db.get('SELECT * FROM users WHERE email = ?',
[email]);
        if (!user) {
        return res.status(400).json({ message: 'Невірний email або пароль' });
        }
        // Перевірка пароля
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
        return res.status(400).json({ message: 'Невірний email або пароль' });
        }
        // Створення JWT токена
        const token = jwt.sign({ userId: user.id }, JWT_SECRET, { expiresIn:
'7d' });
        res.status(200).json({
        token,
        user: {
        _id: user.id,
        name: user.name,
        email: user.email,
        role: user.role,
        trainer_id: user.trainer_id
        }
        });
        } catch (err) {
        console.error('Помилка входу:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
// Маршрути користувачів
        app.get('/api/users/me', authMiddleware, (req, res) => {
        res.json({
        _id: req.user._id,
        name: req.user.name,
        email: req.user.email,
        role: req.user.role,
        trainer_id: req.user.trainer_id
        });
        });
        app.get('/api/users', authMiddleware, roleMiddleware(['admin']), async
(req, res) => {
        try {
        const users = await db.all('SELECT id as _id, name, email, role,
trainer_id FROM users');
        res.json(users);
        } catch (err) {
        console.error('Помилка отримання користувачів:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.get('/api/users/trainers', authMiddleware, async (req, res) => {
        try {
        const trainers = await db.all('SELECT id as _id, name, email FROM users
WHERE role = ?', ['trainer']);
        res.json(trainers);
        } catch (err) {
        console.error('Помилка отримання тренерів:', err);
        res.status(500).json({ message: 'Помилка сервера' });
```

```
        }
        });
        app.get('/api/users/trainer/:trainerId', authMiddleware, async (req,
res) => {
        try {
        const trainer = await db.get('SELECT id as _id, name, email FROM users
WHERE id = ? AND role = ?',
        [req.params.trainerId, 'trainer']);
        if (!trainer) {
        return res.status(404).json({ message: 'Тренера не знайдено' });
        }
        res.json(trainer);
        } catch (err) {
        console.error('Помилка отримання тренера:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.get('/api/users/clients', authMiddleware, roleMiddleware(['trainer',
'admin']), async (req, res) => {
        try {
        let clients;
        if (req.user.role === 'trainer') {
        clients = await db.all('SELECT id as _id, name, email, trainer_id FROM
users WHERE role = ? AND trainer_id = ?',
        ['client', req.user._id]);
        } else {
        clients = await db.all('SELECT id as _id, name, email, trainer_id FROM
users WHERE role = ?', ['client']);
        }
        res.json(clients);
        } catch (err) {
        console.error('Помилка отримання клієнтів:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.delete('/api/users/:id', authMiddleware, roleMiddleware(['admin']),
async (req, res) => {
        try {
        await db.run('DELETE FROM users WHERE id = ?', [req.params.id]);
        res.json({ message: 'Користувача видалено' });
        } catch (err) {
        console.error('Помилка видалення користувача:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
// Маршрут для призначення тренера клієнту
        app.post('/api/users/assign-trainer', authMiddleware, async (req, res)
=> {
        try {
        const { clientId, trainerId } = req.body;
        // Перевірка прав доступу
        if (req.user.role === 'client' && req.user._id !== parseInt(clientId)) {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        if (req.user.role !== 'admin' && req.user.role !== 'client') {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        // Перевіряємо, що тренер існує
        const trainer = await db.get('SELECT id FROM users WHERE id = ? AND role
= ?', [trainerId, 'trainer']);
        if (!trainer) {
        return res.status(404).json({ message: 'Тренера не знайдено' });
        }
        // Призначаємо тренера клієнту
```

```
        await db.run('UPDATE users SET trainer_id = ? WHERE id = ?', [trainerId,
clientId || req.user._id]);
        res.json({ message: 'Тренера успішно призначено' });
        } catch (err) {
        console.error('Помилка призначення тренера:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
// Маршрути подій (тренування)
        app.post('/api/events', authMiddleware, async (req, res) => {
        try {
        const { title, date, type, clientId } = req.body;
        // Переконуємося, що дата в правильному форматі
        console.log(`Отримана дата події: ${date}`);
        // Якщо дата прийшла як рядок ISO, використовуємо її як є
        // Якщо дата містить час, відкидаємо його
        const formattedDate = date.includes('T') ? date.split('T')[0] : date;
        console.log(`Форматована дата події: ${formattedDate}`);
        // Визначення client_id і trainer_id
        let client_id, trainer_id;
        if (req.user.role === 'trainer') {
        // Якщо запит від тренера
        client_id = clientId; // ID клієнта передається в запиті
        trainer_id = req.user._id; // ID тренера - поточний користувач
        // Перевіряємо, чи дійсно цей клієнт закріплений за цим тренером
        const client = await db.get('SELECT * FROM users WHERE id = ? AND
trainer_id = ?', [client_id, trainer_id]);
        if (!client) {
        return res.status(403).json({ message: 'Клієнт не закріплений за цим
тренером' });
        }
        } else if (req.user.role === 'client') {
        // Якщо запит від клієнта
        client_id = req.user._id; // ID клієнта - поточний користувач
        trainer_id = req.user.trainer_id; // ID тренера береться з профілю
клієнта
        // Перевіряємо, чи має клієнт закріпленого тренера
        if (!trainer_id) {
        return res.status(400).json({ message: 'Спочатку потрібно вибрати
тренера' });
        }
        } else {
        // Якщо запит від адміністратора
        client_id = clientId;
        trainer_id = null; // Адміністратор може не вказувати тренера
        }
        const result = await db.run(
        'INSERT INTO events (title, date, type, client_id, trainer_id) VALUES
(?, ?, ?, ?, ?)',
        [title, formattedDate, type, client_id, trainer_id]
        );
        const event = await db.get('SELECT * FROM events WHERE id = ?',
[result.lastID]);
        res.status(201).json({
        _id: event.id,
        title: event.title,
        date: event.date,  // Повертаємо дату в тому ж форматі, в якому зберегли
        type: event.type,
        clientId: event.client_id,
        trainerId: event.trainer_id
        });
        } catch (err) {
        console.error('Помилка створення події:', err);
        res.status(500).json({ message: 'Помилка сервера' });
```

```
        }
        });
        app.get('/api/events/client/:clientId', authMiddleware, async (req, res)
=> {
        try {
        const clientId = req.params.clientId;
        // Перевірка прав доступу
        if (req.user.role === 'client' && req.user._id !== parseInt(clientId)) {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        if (req.user.role === 'trainer') {
        // Перевіряємо, чи клієнт закріплений за цим тренером
        const client = await db.get('SELECT * FROM users WHERE id = ? AND
trainer_id = ?', [clientId, req.user._id]);
        if (!client) {
        return res.status(403).json({ message: 'Клієнт не закріплений за цим
тренером' });
        }
        }
        const events = await db.all('SELECT * FROM events WHERE client_id = ?',
[clientId]);
        const formattedEvents = events.map(event => ({
        _id: event.id,
        title: event.title,
        date: event.date,
        type: event.type,
        clientId: event.client_id,
        trainerId: event.trainer_id
        }));
        res.json(formattedEvents);
        } catch (err) {
        console.error('Помилка отримання подій:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.get('/api/events/trainer/:trainerId', authMiddleware, async (req,
res) => {
        try {
        const trainerId = req.params.trainerId;
        // Перевірка прав доступу
        if (req.user.role === 'trainer' && req.user._id !== parseInt(trainerId))
{
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        const events = await db.all('SELECT * FROM events WHERE trainer_id = ?',
[trainerId]);
        const formattedEvents = events.map(event => ({
        _id: event.id,
        title: event.title,
        date: event.date,
        type: event.type,
        clientId: event.client_id,
        trainerId: event.trainer_id
        }));
        res.json(formattedEvents);
        } catch (err) {
        console.error('Помилка отримання подій:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
// Маршрути метрик (показники прогресу)
        app.post('/api/metrics', authMiddleware, async (req, res) => {
        try {
        const { type, value, date, userId } = req.body;
```

```
        // Перевірка прав доступу
        if (req.user.role === 'client' && req.user._id !== parseInt(userId)) {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        if (req.user.role === 'trainer') {
        // Перевіряємо, чи клієнт закріплений за цим тренером
        const client = await db.get('SELECT * FROM users WHERE id = ? AND
trainer_id = ?', [userId, req.user._id]);
        if (!client) {
        return res.status(403).json({ message: 'Клієнт не закріплений за цим
тренером' });
        }
        }
        const result = await db.run(
        'INSERT INTO metrics (type, value, date, user_id) VALUES (?, ?, ?, ?)',
        [type, value, date || new Date().toISOString(), userId || req.user._id]
        );
        const metric = await db.get('SELECT * FROM metrics WHERE id = ?',
[result.lastID]);
        res.status(201).json({
        _id: metric.id,
        type: metric.type,
        value: metric.value,
        date: metric.date,
        userId: metric.user_id
        });
        } catch (err) {
        console.error('Помилка створення метрики:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.get('/api/metrics/:userId', authMiddleware, async (req, res) => {
        try {
        const userId = req.params.userId;
        // Перевірка прав доступу
        if (req.user.role === 'client' && req.user._id !== parseInt(userId)) {
        return res.status(403).json({ message: 'Доступ заборонено' });
        }
        if (req.user.role === 'trainer') {
        // Перевіряємо, чи клієнт закріплений за цим тренером
        const client = await db.get('SELECT * FROM users WHERE id = ? AND
trainer_id = ?', [userId, req.user._id]);
        if (!client) {
        return res.status(403).json({ message: 'Клієнт не закріплений за цим
тренером' });
        }
        }
        const metrics = await db.all(
        'SELECT * FROM metrics WHERE user_id = ? ORDER BY date DESC',
        [userId]
        );
        const formattedMetrics = metrics.map(metric => ({
        _id: metric.id,
        type: metric.type,
        value: metric.value,
        date: metric.date,
        userId: metric.user_id
        }));
        res.json(formattedMetrics);
        } catch (err) {
        console.error('Помилка отримання метрик:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
```

```
// Маршрути для каталогу вправ
        app.get('/api/exercises', async (req, res) => {
        try {
        console.log('Запит на отримання всіх вправ');
        const exercises = await db.all('SELECT * FROM exercises');
        console.log(`Знайдено ${exercises.length} вправ`);
        res.json(exercises);
        } catch (err) {
        console.error('Помилка отримання каталогу вправ:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.get('/api/exercises/:type', async (req, res) => {
        try {
        console.log(`Запит на отримання вправ типу: ${req.params.type}`);
        const exercises = await db.all('SELECT * FROM exercises WHERE type = ?',
[req.params.type]);
        console.log(`Знайдено ${exercises.length} вправ типу
${req.params.type}`);
        res.json(exercises);
        } catch (err) {
        console.error('Помилка отримання вправ за типом:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.post('/api/exercises', authMiddleware, roleMiddleware(['trainer',
'admin']), async (req, res) => {
        try {
        const { title, description, type } = req.body;
        const result = await db.run(
        'INSERT INTO exercises (title, description, type) VALUES (?, ?, ?)',
        [title, description, type]
        );
        const exercise = await db.get('SELECT * FROM exercises WHERE id = ?',
[result.lastID]);
        res.status(201).json(exercise);
        } catch (err) {
        console.error('Помилка додавання вправи:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
        app.put('/api/exercises/:id', authMiddleware, roleMiddleware(['trainer',
'admin']), async (req, res) => {
        try {
        const { title, description, type } = req.body;
        const exerciseId = req.params.id;
        // Перевіряємо, чи існує вправа
        const exercise = await db.get('SELECT * FROM exercises WHERE id = ?',
[exerciseId]);
        if (!exercise) {
        return res.status(404).json({ message: 'Вправу не знайдено' });
        }
        await db.run(
        'UPDATE exercises SET title = ?, description = ?, type = ? WHERE id =
?',
        [title, description, type, exerciseId]
        );
        const updatedExercise = await db.get('SELECT * FROM exercises WHERE id =
?', [exerciseId]);
        res.json(updatedExercise);
        } catch (err) {
        console.error('Помилка оновлення вправи:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
```

```
        });
        app.delete('/api/exercises/:id', authMiddleware,
roleMiddleware(['trainer', 'admin']), async (req, res) => {
        try {
        const exerciseId = req.params.id;
        // Перевіряємо, чи існує вправа
        const exercise = await db.get('SELECT * FROM exercises WHERE id = ?',
[exerciseId]);
        if (!exercise) {
        return res.status(404).json({ message: 'Вправу не знайдено' });
        }
        await db.run('DELETE FROM exercises WHERE id = ?', [exerciseId]);
        res.json({ message: 'Вправу успішно видалено' });
        } catch (err) {
        console.error('Помилка видалення вправи:', err);
        res.status(500).json({ message: 'Помилка сервера' });
        }
        });
// Функція для отримання заглушки відповіді AI
        const getFakeAIResponse = (prompt) => {
        // Набір заготовлених відповідей для режиму розробки
        const responses = [
        `Я ваш ШІ-консультант з фітнесу. Щодо вашого запитання про ${prompt}:
рекомендую поєднувати силові тренування з кардіо для досягнення найкращих
результатів. Також не забувайте про правильне харчування та режим відпочинку.`,
        `Для ефективних тренувань, пов'язаних з ${prompt}, важливо дотримуватися
принципу прогресивного навантаження. Збільшуйте інтенсивність поступово, даючи
м'язам час на відновлення. Пийте достатньо води та споживайте білок для кращого
відновлення.`,
        `Щодо вашого запитання про ${prompt}: варто пам'ятати, що регулярність
тренувань важливіша за їх інтенсивність. 3-4 помірні тренування на тиждень
дадуть кращий результат, ніж 1-2 дуже інтенсивні. Слідкуйте за своїм
самопочуттям і не забувайте про розминку.`,
        `У контексті ${prompt} пам'ятайте, що здорове харчування становить
близько 70% успіху у фітнесі. Збалансуйте свій раціон, включивши достатньо
білків, здорових жирів і складних вуглеводів. Мінімізуйте вживання обробленої
їжі та цукру.`
        ];
        // Вибираємо випадкову відповідь
        const randomIndex = Math.floor(Math.random() * responses.length);
        return responses[randomIndex];
        };
// Маршрут для Gemini AI
        app.post('/api/gemini/ask', authMiddleware, async (req, res) => {
        try {
        const { prompt } = req.body;
        if (!prompt || prompt.trim() === '') {
        return res.status(400).json({ message: 'Запит не може бути порожнім' });
        }
        try {
        // Перевірка чи доступний Gemini
        if (!geminiAvailable || !geminiModel) {
        throw new Error('Сервіс Gemini недоступний');
        }
        // Створення запиту до Gemini
        const fullPrompt = `Ти - тренер з фітнесу. Дай корисні поради для
українського користувача: ${prompt}. Дай чітку, корисну й обґрунтовану відповідь
українською мовою.`;
        // Виконання запиту з новим форматом API
        const result = await geminiModel.generateContent({
        contents: [{
        role: "user",
        parts: [{ text: fullPrompt }]
        }]
```

```
        });
        if (result && result.response) {
        const text = result.response.text();
        console.log('Отримано відповідь від Gemini:', text.substring(0, 100) +
'...');
        return res.json({ text });
        } else {
        throw new Error('Отримано порожню відповідь від API');
        }
        } catch (geminiErr) {
        console.error('Помилка запиту до Gemini API:', geminiErr);
        // Для розробки: повертаємо заглушку в режимі розробки
        if (process.env.NODE_ENV === 'development' || !geminiAvailable) {
        const fakeResponse = getFakeAIResponse(prompt);
        console.log('Використовуємо заглушку відповіді для розробки');
        return res.json({
        text: fakeResponse,
        _dev_note: 'Це заглушка для розробки. Налаштуйте справжній API ключ для
Gemini.'
        });
        } else {
        // В продакшені повертаємо помилку
        throw geminiErr;
        }
        }
        } catch (err) {
        console.error('Помилка запиту до Gemini:', err);
        res.status(500).json({ message: 'Помилка взаємодії з AI', error:
err.message });
        }
        });
// Ініціалізація бази даних і запуск сервера
        (async function startServer() {
        try {
        await initializeDatabase();
        // Перевірка з'єднання з Gemini
        geminiAvailable = await testGeminiConnection();
        app.listen(PORT, () => {
        console.log(`Сервер запущено на порту ${PORT}`);
        if (!geminiAvailable) {
        console.log('ШІ-консультант відключено. Перевірте API ключ Gemini.');
        console.log('Для локальної розробки будуть використовуватися заглушки
відповідей AI.');
        }
        });
        } catch (err) {
        console.error('Помилка запуску сервера:', err);
        }
        })();
```

seed.js

```
const sqlite3 = require('sqlite3').verbose();
const { open } = require('sqlite');
        const bcrypt = require('bcrypt');
        const path = require('path');
// Каталог вправ за категоріями
        const exercisesCatalog = {
        силові: [
        { title: 'Жим штанги лежачи', description: 'Базова вправа для грудних
м\'язів' },
        { title: 'Присідання зі штангою', description: 'Базова вправа для ніг'
},
```

```
        { title: 'Станова тяга', description: 'Базова вправа для спини та ніг'
},
        { title: 'Жим гантелей сидячи', description: 'Вправа для плечей' },
        { title: 'Підтягування на перекладині', description: 'Вправа для спини
та біцепсів' },
        { title: 'Віджимання на брусах', description: 'Вправа для грудних
м\'язів та трицепсів' },
        { title: 'Тяга штанги в нахилі', description: 'Вправа для спини' },
        { title: 'Підйом штанги на біцепс', description: 'Ізольована вправа для
біцепсів' }
        ],
        кардіо: [
        { title: 'Біг на біговій доріжці', description: 'Кардіо тренування для
витривалості' },
        { title: 'Їзда на велотренажері', description: 'Кардіо тренування з
меншим навантаженням на суглоби' },
        { title: 'Степпер', description: 'Ефективне кардіо для ніг та сідниць'
},
        { title: 'Еліптичний тренажер', description: 'Кардіо з низьким
навантаженням на суглоби' },
        { title: 'Стрибки зі скакалкою', description: 'Високоінтенсивне кардіо
тренування' },
        { title: 'Інтервальний спринт', description: 'Короткі спринти з
перервами для відпочинку' }
        ],
        похудіння: [
        { title: 'Інтервальне тренування високої інтенсивності (HIIT)',
description: 'Чергування високої та низької інтенсивності' },
        { title: 'Кругове тренування', description: 'Серія вправ з мінімальним
відпочинком' },
        { title: 'Бурпі', description: 'Комплексна вправа для всього тіла' },
        { title: 'Планка', description: 'Статична вправа для корпусу' },
        { title: 'Випади з гантелями', description: 'Вправа для ніг та сідниць'
},
        { title: 'Скручування на прес', description: 'Вправа для м\'язів живота'
}
        ],
        інші: [
        { title: 'Йога', description: 'Комплекс вправ для гнучкості та балансу'
},
        { title: 'Пілатес', description: 'Вправи для зміцнення корпусу' },
        { title: 'Розтяжка', description: 'Комплекс вправ для покращення
гнучкості' },
        { title: 'Функціональне тренування', description: 'Вправи, що імітують
повсякденні рухи' }
        ]
        };
// Функція для створення випадкової дати в межах останніх 3 місяців
        function randomDate(start = new Date(Date.now() - 90 * 24 * 60 * 60 *
1000), end = new Date()) {
        return new Date(start.getTime() + Math.random() * (end.getTime() -
start.getTime()));
        }
// Функція для випадкового вибору елемента з масиву
        function randomItem(array) {
        return array[Math.floor(Math.random() * array.length)];
        }
// Функція для генерації випадкових метрик прогресу
        function generateMetrics(userId, count = 10) {
        const metrics = [];
        const types = ['вага', 'станова', 'присідання', 'біг'];
        const today = new Date();
        let date = new Date(today);
        date.setDate(today.getDate() - count * 3); // Починаємо з count*3 днів
```

```
тому
        for (let i = 0; i < count; i++) {
        const type = randomItem(types);
        let value;
        // Генеруємо реалістичні значення для різних типів метрик
        switch (type) {
        case 'вага':
        value = Math.floor(60 + Math.random() * 40); // 60-100 кг
        break;
        case 'станова':
        value = Math.floor(80 + Math.random() * 120); // 80-200 кг
        break;
        case 'присідання':
        value = Math.floor(60 + Math.random() * 100); // 60-160 кг
        break;
        case 'біг':
        value = Math.floor(2 + Math.random() * 8); // 2-10 км
        break;
        default:
        value = Math.floor(1 + Math.random() * 100);
        }
        // Додаємо тренд для прогресу (поступове покращення)
        const trendFactor = i / count; // від 0 до 1
        if (type === 'вага') {
        // Для ваги - зменшення
        value = value - (5 * trendFactor);
        } else {
        // Для інших - збільшення
        value = value + (10 * trendFactor);
        }
        // Додаємо невелику випадковість
        value = value * (0.95 + Math.random() * 0.1);
        date.setDate(date.getDate() + 3); // Кожна метрика з інтервалом у 3 дні
        metrics.push({
        userId,
        type,
        value: Math.round(value * 10) / 10, // Округлюємо до 1 десяткового місця
        date: new Date(date).toISOString().split('T')[0]
        });
        }
        return metrics;
        }
// Функція для генерації випадкових подій календаря
        function generateEvents(clientId, trainerId, count = 15) {
        const events = [];
        const types = Object.keys(exercisesCatalog);
        // Майбутні події (тренування)
        const today = new Date();
        for (let i = 0; i < count; i++) {
        const type = randomItem(types);
        const exercise = randomItem(exercisesCatalog[type]);
        // Генеруємо дату: деякі в минулому, більшість у майбутньому
        let eventDate = new Date(today);
        if (i < count / 3) {
        // 1/3 подій у минулому (останні 15 днів)
        eventDate.setDate(today.getDate() - Math.floor(Math.random() * 15));
        } else {
        // 2/3 подій у майбутньому (наступні 30 днів)
        eventDate.setDate(today.getDate() + Math.floor(Math.random() * 30) + 1);
        }
        events.push({
        title: exercise.title,
        type,
        date: eventDate.toISOString().split('T')[0],
```

```
        clientId,
        trainerId
        });
        }
        return events;
        }
// Головна функція заповнення бази даних
        async function seedDatabase() {
        console.log("Запуск скрипта заповнення бази даних...");
        // Підключення до бази даних
        const db = await open({
        filename: path.join(__dirname, 'fitness.db'),
        driver: sqlite3.Database
        });
        try {
        // Очищення таблиць перед заповненням
        await db.exec('DELETE FROM metrics');
        await db.exec('DELETE FROM events');
        await db.exec('DELETE FROM exercises');
        await db.exec('DELETE FROM users');
        // Скидання автоінкременту
        await db.exec('DELETE FROM sqlite_sequence WHERE name="users"');
        await db.exec('DELETE FROM sqlite_sequence WHERE name="events"');
        await db.exec('DELETE FROM sqlite_sequence WHERE name="metrics"');
        await db.exec('DELETE FROM sqlite_sequence WHERE name="exercises"');
        console.log('Таблиці очищено');
        // Хешування пароля (однаковий для всіх користувачів для спрощення)
        const salt = await bcrypt.genSalt(10);
        const hashedPassword = await bcrypt.hash('password123', salt);
        // Додавання адміністратора
        await db.run(
        'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
        ['Адміністратор', 'admin@example.com', hashedPassword, 'admin']
        );
        console.log('Адміністратор створений');
        // Додавання тренерів
        const trainers = [
        { name: 'Олег Петренко', email: 'oleg@example.com', specialization:
'Силові тренування' },
        { name: 'Марія Коваленко', email: 'maria@example.com', specialization:
'Кардіо та схуднення' },
        { name: 'Ігор Сидоренко', email: 'igor@example.com', specialization:
'Функціональні тренування' }
        ];
        for (const trainer of trainers) {
        const result = await db.run(
        'INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)',
        [trainer.name, trainer.email, hashedPassword, 'trainer']
        );
        trainer.id = result.lastID;
        }
        console.log('Тренери створені');
        // Додавання клієнтів
        const clients = [
        { name: 'Анна Петрова', email: 'anna@example.com', goal: 'схуднення' },
        { name: 'Василь Іванов', email: 'vasyl@example.com', goal: 'набір маси'
},
        { name: 'Ольга Сидорчук', email: 'olga@example.com', goal: 'тонус
м\'язів' },
        { name: 'Максим Ткаченко', email: 'maxim@example.com', goal: 'схуднення'
},
        { name: 'Юлія Шевченко', email: 'yulia@example.com', goal: 'кардіо' },
        { name: 'Артем Бондаренко', email: 'artem@example.com', goal: 'сила' },
        { name: 'Наталія Мельник', email: 'natalia@example.com', goal:
```

```
'гнучкість' },
        { name: 'Денис Кравченко', email: 'denis@example.com', goal: 'схуднення'
},
        { name: 'Тетяна Іваненко', email: 'tetiana@example.com', goal: 'сила' },
        { name: 'Микола Бойко', email: 'mykola@example.com', goal:
'витривалість' }
        ];
        for (const client of clients) {
        // Призначаємо випадкового тренера
        const trainer = randomItem(trainers);
        const result = await db.run(
        'INSERT INTO users (name, email, password, role, trainer_id) VALUES (?,
?, ?, ?, ?)',
        [client.name, client.email, hashedPassword, 'client', trainer.id]
        );
        client.id = result.lastID;
        client.trainerId = trainer.id;
        }
        console.log('Клієнти створені');
        // Додавання подій (тренувань) для кожного клієнта
        for (const client of clients) {
        const events = generateEvents(client.id, client.trainerId);
        for (const event of events) {
        await db.run(
        'INSERT INTO events (title, date, type, client_id, trainer_id) VALUES
(?, ?, ?, ?, ?)',
        [event.title, event.date, event.type, event.clientId, event.trainerId]
        );
        }
        }
        console.log('Події календаря створені');
        // Додавання метрик прогресу для кожного клієнта
        for (const client of clients) {
        const metrics = generateMetrics(client.id);
        for (const metric of metrics) {
        await db.run(
        'INSERT INTO metrics (user_id, type, value, date) VALUES (?, ?, ?, ?)',
        [metric.userId, metric.type, metric.value, metric.date]
        );
        }
        }
        console.log('Метрики прогресу створені');
        // Заповнення каталогу вправ
        for (const [type, exercises] of Object.entries(exercisesCatalog)) {
        for (const exercise of exercises) {
        await db.run(
        'INSERT INTO exercises (title, description, type) VALUES (?, ?, ?)',
        [exercise.title, exercise.description, type]
        );
        }
        }
        console.log('Каталог вправ створено');
        // Виведення інформації про створені записи
        const userCount = await db.get('SELECT COUNT(*) as count FROM users');
        const eventsCount = await db.get('SELECT COUNT(*) as count FROM
events');
        const metricsCount = await db.get('SELECT COUNT(*) as count FROM
metrics');
        const exercisesCount = await db.get('SELECT COUNT(*) as count FROM
exercises');
        console.log(`\nБаза даних успішно заповнена:`);
        console.log(`- Користувачів: ${userCount.count}`);
        console.log(`- Подій календаря: ${eventsCount.count}`);
        console.log(`- Метрик прогресу: ${metricsCount.count}`);
```

```
        console.log(`- Вправ у каталозі: ${exercisesCount.count}`);
        console.log('\nДані для входу (для всіх користувачів):');
        console.log('Пароль: password123');
        console.log('Адмін: admin@example.com');
        console.log('Тренери:');
        trainers.forEach(trainer => console.log(`- ${trainer.name}:
${trainer.email}`));
        console.log('Клієнти:');
        clients.forEach(client => console.log(`- ${client.name}:
${client.email}`));
        console.log("\nСкрипт заповнення бази даних успішно завершено!");
        } catch (error) {
        console.error('Помилка заповнення бази даних:', error);
        } finally {
        await db.close();
        }
        }
// Запускаємо функцію заповнення
        seedDatabase().then(() => {
        console.log('Усі операції завершено');
        }).catch(err => {
        console.error('Помилка виконання скрипта:', err);
        });
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './components/App';
import { AuthProvider } from './context/AuthContext';
        const root = ReactDOM.createRoot(document.getElementById('root'));
        root.render(
    <React.StrictMode>
        <AuthProvider>
        <App />
        </AuthProvider>
        </React.StrictMode>
);
```

AuthContext.js

```
// src/context/AuthContext.js
import React, { createContext, useState, useEffect, useContext } from 'react';
import axios from 'axios';
        const API_URL = 'http://localhost:5000/api'; // Додаємо базовий URL
        const AuthContext = createContext();
export const useAuth = () => useContext(AuthContext);
export const AuthProvider = ({ children }) => {
        const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);
// Перевірка автентифікації при завантаженні
useEffect(() => {
        const checkAuth = async () => {
        const token = localStorage.getItem('token');
            if (token) {
        try {
        console.log("Перевірка автентифікації з токеном");
                    const res = await axios.get(`${API_URL}/users/me`, {
headers: { Authorization: `Bearer ${token}` }
        });
```

```
        console.log("Користувача автентифіковано:", res.data);
setUser(res.data);
                } catch (err) {
        console.error("Помилка автентифікації:", err);
                    localStorage.removeItem('token');
                }
                        }
setLoading(false);
        };
checkAuth();
    }, []);
            // Функція входу
            const login = async (email, password) => {
        try {
        console.log("Вхід користувача:", email);
            const res = await axios.post(`${API_URL}/auth/login`, { email,
password });
        console.log("Успішний вхід:", res.data);
            localStorage.setItem('token', res.data.token);
setUser(res.data.user);
            return res.data.user;
        } catch (err) {
        console.error("Помилка входу:", err);
            throw err;
        }
                };
                // Функція реєстрації
                const register = async (userData) => {
        try {
        console.log("Реєстрація користувача:", userData.email);
            const res = await axios.post(`${API_URL}/auth/register`, userData);
        console.log("Успішна реєстрація:", res.data);
            localStorage.setItem('token', res.data.token);
setUser(res.data.user);
            return res.data.user;
        } catch (err) {
        console.error("Помилка реєстрації:", err);
            throw err;
        }
                };
                // Функція виходу
                const logout = () => {
        console.log("Вихід користувача");
        localStorage.removeItem('token');
setUser(null);
    };
            // Функція для API запитів з токеном
            const authFetch = async (endpoint, options = {}) => {
        try {
        const token = localStorage.getItem('token');
            const url = `${API_URL}${endpoint}`; // Додаємо базовий URL
        console.log(`API запит: ${options.method || 'GET'} ${url}`);
        const config = {
        ...options,
headers: {
        ...options.headers,
Authorization: `Bearer ${token}`
        }
        };
        const response = await axios(url, config);
            console.log(`API відповідь: ${url}`, response.status);
        return response;
        } catch (err) {
        console.error(`API помилка: ${endpoint}`, err.response || err.message);
```

```
        throw err;
        }
                };
                return (
        <AuthContext.Provider value={{
        user,
        login,
        register,
        logout,
        loading,
        authFetch
}}>
        {children}
        </AuthContext.Provider>
        );
        };
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-
dom';
import { useAuth } from '../context/AuthContext';
import Auth from './Auth';
import Dashboard from './Dashboard';
import ExerciseCatalogManager from './ExerciseCatalogManager';
        const App = () => {
        const { user, loading } = useAuth();
    if (loading) {
        return <div className="loading">Завантаження...</div>;
        }
        return (
        <Router>
            <div className="app">
                <Routes>
                    <Route path="/auth" element={user ? <Navigate
to="/dashboard" /> : <Auth />} />
                    <Route
path="/dashboard/*"
element={user ? <Dashboard /> : <Navigate to="/auth" />}
        />
                    <Route
path="/exercise-catalog"
element={
user && (user.role === 'trainer' || user.role === 'admin')
        ? <ExerciseCatalogManager />
        : <Navigate to="/dashboard" />
        }
        />
                    <Route path="*" element={<Navigate to={user ? "/dashboard" :
"/auth"} />} />
                </Routes>
            </div>
        </Router>
        );
        };
export default App;
```

Auth.js

```
import React, { useState } from 'react';
import { useAuth } from '../context/AuthContext';
        const Auth = () => {
        const [isLogin, setIsLogin] = useState(true);
    const [formData, setFormData] = useState({
    name: '',
            email: '',
            password: '',
            role: 'client'
});
        const [error, setError] = useState('');
    const { login, register } = useAuth();
    const handleChange = (e) => {
setFormData({ ...formData, [e.target.name]: e.target.value });
        };
        const handleSubmit = async (e) => {
        e.preventDefault();
setError('');
        try {
                if (isLogin) {
await login(formData.email, formData.password);
            } else {
                    if (!formData.name) {
        return setError("Ім'я обов'язкове");
                }
await register(formData);
            }
                    } catch (err) {
setError(err.response?.data?.message || 'Помилка аутентифікації');
        }
                };
                return (
        <div className="auth-container">
            <div className="auth-box">
<h2>{isLogin ? 'Вхід' : 'Реєстрація'}</h2>
        {error && <div className="error-message">{error}</div>}
                <form onSubmit={handleSubmit}>
        {!isLogin && (
                        <div className="form-group">
                            <label>Ім'я</label>
                            <input
type="text"
name="name"
value={formData.name}
onChange={handleChange}
        />
                        </div>
        )}
                    <div className="form-group">
                        <label>Email</label>
                        <input
type="email"
name="email"
value={formData.email}
onChange={handleChange}
required
                        />
                    </div>
                    <div className="form-group">
                        <label>Пароль</label>
                        <input
type="password"
name="password"
value={formData.password}
```

```
onChange={handleChange}
required
                        />
                    </div>
        {!isLogin && (
                        <div className="form-group">
                            <label>Роль</label>
                            <select
name="role"
value={formData.role}
onChange={handleChange}
        >
                                <option value="client">Клієнт</option>
                                <option value="trainer">Тренер</option>
                            </select>
                        </div>
        )}
                    <button type="submit" className="btn-primary">
        {isLogin ? 'Увійти' : 'Зареєструватися'}
                    </button>
                </form>
                <p onClick={() => setIsLogin(!isLogin)} className="toggle-auth">
        {isLogin
                        ? 'Немає облікового запису? Зареєструватися'
                                : 'Вже є обліковий запис? Увійти'}
                </p>
            </div>
        </div>
        );
        };
export default Auth;
```

Dashboard.js

```
// src/components/Dashboard.js
import React, { useState, useEffect } from 'react';
import { Routes, Route, Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
import GeminiAdvisor from './GeminiAdvisor';
import WorkoutCalendar from './WorkoutCalendar';
// Компонент для відслідковування прогресу
        const ProgressTracker = ({ userId }) => {
        const { authFetch } = useAuth();
    const [metrics, setMetrics] = useState([]);
    const [newMetric, setNewMetric] = useState({ type: 'вага', value: '', date:
new Date().toISOString().split('T')[0] });
        const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
useEffect(() => {
        const getMetrics = async () => {
setLoading(true);
            try {
                    console.log(`Отримання метрик для користувача: ${userId}`);
        const res = await authFetch(`/metrics/${userId}`);
        console.log('Отримані метрики:', res.data);
setMetrics(res.data || []);
setError(null);
            } catch (err) {
        console.error('Помилка завантаження метрик:', err);
setError('Не вдалося завантажити метрики прогресу.');
// Додаємо тестові дані, якщо не вдалося отримати з сервера
setMetrics(generateMockMetrics(userId));
        } finally {
```

```
setLoading(false);
            }
                    };
                    if (userId) {
getMetrics();
        } else {
setLoading(false);
        }
                }, [userId, authFetch]);
        // Функція для генерації тестових метрик
        const generateMockMetrics = (userId) => {
        const mockMetrics = [];
        const today = new Date();
// Типи метрик і початкові значення
        const metricTypes = [
        { type: 'вага', startValue: 85, change: -0.5 }, // Зменшення ваги
        { type: 'станова', startValue: 120, change: 5 }, // Збільшення ваги в
станової тязі
        { type: 'присідання', startValue: 100, change: 2.5 }, // Збільшення ваги
в присіданнях
        { type: 'біг', startValue: 5, change: 0.2 } // Збільшення дистанції бігу
        ];
        // Генеруємо 10 записів для кожного типу метрик
        metricTypes.forEach(metricType => {
    for (let i = 0; i < 10; i++) {
                const date = new Date(today);
        date.setDate(today.getDate() - (10 - i) * 3); // Кожні 3 дні, починаючи
з 30 днів тому
                const value = metricType.startValue + (metricType.change * i);
        mockMetrics.push({
                _id: `mock-${metricType.type}-${i}`,
        userId,
                type: metricType.type,
                value: Math.round(value * 10) / 10, // Округлюємо до 1
десяткового знаку
                date: date.toISOString().split('T')[0]
                });
    }
});
        return mockMetrics;
    };
            const handleAddMetric = async (e) => {
        e.preventDefault();
        if (!newMetric.value) {
alert('Будь ласка, введіть значення');
            return;
                    }
                    try {
                    console.log('Додавання нової метрики:', newMetric);
            const res = await authFetch('/metrics', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data: { ...newMetric, userId }
});
        console.log('Результат додавання метрики:', res.data);
setMetrics([...metrics, res.data]);
setNewMetric({ ...newMetric, value: '' });
        } catch (err) {
        console.error('Помилка додавання метрики:', err);
alert('Не вдалося додати метрику. Спробуйте пізніше.');
        }
                };
                // Фільтрація метрик за типом і сортування за датою
                const getFilteredMetrics = (type) => {
```

```
        return metrics
        .filter(metric => metric.type === type)
            .sort((a, b) => new Date(a.date) - new Date(b.date))
        .slice(-10); // Показуємо останні 10 записів
    };
            if (loading) {
        return <div className="loading">Завантаження метрик...</div>;
        }
        return (
        <div className="progress-tracker card">
            <h3 className="section-title">Відстеження прогресу</h3>
        {error && <div className="alert alert-danger">{error}</div>}
            <form onSubmit={handleAddMetric} className="metric-form">
                <select
className="form-control"
value={newMetric.type}
onChange={e => setNewMetric({ ...newMetric, type: e.target.value })}
        >
                    <option value="вага">Вага (кг)</option>
                    <option value="станова">Станова тяга (кг)</option>
                    <option value="присідання">Присідання (кг)</option>
                    <option value="біг">Біг (км)</option>
                </select>
                <input
type="number"
step="0.1"
className="form-control"
placeholder="Значення"
value={newMetric.value}
onChange={e => setNewMetric({ ...newMetric, value: e.target.value })}
required
                />
                <input
type="date"
className="form-control"
value={newMetric.date}
onChange={e => setNewMetric({ ...newMetric, date: e.target.value })}
required
                />
                <button type="submit" className="btn btn-
primary">Додати</button>
            </form>
            <div className="metrics-charts">
        {metrics.length > 0 ? (
                    <div className="metrics-grid">
        {['вага', 'станова', 'присідання', 'біг'].map(type => {
                            const filteredMetrics = getFilteredMetrics(type);
    if (filteredMetrics.length === 0) return null;
    return (
            <div key={type} className="metric-chart card">
                                    <h4 className="chart-
title">{type.charAt(0).toUpperCase() + type.slice(1)}</h4>
            {/* Тут можна додати візуалізацію з Chart.js або іншу бібліотеку */}
            <table className="table">
                                        <thead>
                                            <tr>
                                                <th>Дата</th>
                                                <th>Значення</th>
                                            </tr>
                                        </thead>
                                        <tbody>
            {filteredMetrics.map(metric => (
                    <tr key={metric._id}>
            <td>{new Date(metric.date).toLocaleDateString('uk-UA')}</td>
```

```
                                                    <td>{metric.value} {type ===
'біг' ? 'км' : 'кг'}</td>
                                                </tr>
                                            ))}
                                        </tbody>
                                    </table>
                                </div>
                            );
})}
                    </div>
        ) : (
<p>Немає записаних показників. Додайте свій перший показник!</p>
        )}
            </div>
        </div>
        );
        };
// Компонент календаря
        const Calendar = ({ userId, isTrainer, clientId }) => {
        const { authFetch } = useAuth();
    const [events, setEvents] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
// Використовуємо clientId, якщо він переданий (для тренера, який дивиться
календар клієнта)
    const targetUserId = clientId || userId;
useEffect(() => {
        const getEvents = async () => {
setLoading(true);
            try {
                    // Визначаємо ендпоїнт залежно від ролі користувача
                    const endpoint = `/events/client/${targetUserId}`;
        console.log("Запит до API (Calendar):", endpoint);
                const res = await authFetch(endpoint);
                console.log("Відповідь API (події):", res.data);
setEvents(res.data || []);
setError(null);
            } catch (err) {
        console.error('Помилка завантаження подій:', err);
                console.error('Деталі помилки:', err.response || err.message);
setError('Не вдалося завантажити розклад тренувань.');
            } finally {
setLoading(false);
            }
                    };
                    if (targetUserId) {
getEvents();
        } else {
                console.log("Не вказано userId для отримання подій");
setLoading(false);
        }
                }, [targetUserId, authFetch]);
        const handleAddEvent = async (eventData) => {
        try {
        console.log("Додавання події (Calendar):", eventData);
// Підготовка даних для відправки на сервер
            const data = {
        ...eventData,
clientId: clientId || userId // Вказуємо ID клієнта
            };
                    const res = await authFetch('/events', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data
```

```
});
        console.log("Результат додавання події:", res.data);
// Додаємо нову подію до списку
setEvents(prevEvents => [...prevEvents, res.data]);
            return res.data;
        } catch (err) {
        console.error('Помилка додавання події:', err);
            console.error('Деталі помилки:', err.response || err.message);
            throw err;
        }
                };
                if (loading) {
        return <div className="loading">Завантаження календаря...</div>;
        }
        if (error) {
        return (
            <div className="error-container">
                <div className="alert alert-danger">{error}</div>
            </div>
        );
        }
        return (
        <div className="calendar-wrapper">
            <WorkoutCalendar
userId={targetUserId}
isTrainer={isTrainer}
events={events}
onEventAdd={handleAddEvent}
        />
        </div>
        );
        };
// Компоненти дашбордів для різних ролей
        const ClientDashboard = () => {
        const { user, authFetch } = useAuth();
    const [trainers, setTrainers] = useState([]);
    const [selectedTrainer, setSelectedTrainer] = useState(null);
    const [assignedTrainer, setAssignedTrainer] = useState(null);
    const [loading, setLoading] = useState(true);
    const [trainerAssigning, setTrainerAssigning] = useState(false);
    const [error, setError] = useState(null);
// Завантаження даних про призначеного тренера
useEffect(() => {
        const getAssignedTrainer = async () => {
        try {
        if (user.trainer_id) {
        const res = await authFetch(`/users/trainer/${user.trainer_id}`);
setAssignedTrainer(res.data);
                }
                        } catch (err) {
        console.error('Помилка завантаження даних тренера:', err);
            }
                    };
getAssignedTrainer();
    }, [user, authFetch]);
// Завантаження списку доступних тренерів, якщо немає призначеного
useEffect(() => {
        const getTrainers = async () => {
setLoading(true);
            try {
                    // Завантажуємо список тренерів тільки якщо немає
призначеного
                    if (!user.trainer_id) {
        const res = await authFetch('/users/trainers');
```

```
setTrainers(res.data || []);
setError(null);
                }
                        } catch (err) {
        console.error('Помилка завантаження тренерів:', err);
setError('Не вдалося завантажити список тренерів.');
            } finally {
setLoading(false);
            }
                    };
getTrainers();
    }, [user, authFetch]);
        // Функція для призначення тренера
        const assignTrainer = async () => {
        if (!selectedTrainer) {
alert('Будь ласка, виберіть тренера');
            return;
                    }
setTrainerAssigning(true);
        try {
await authFetch('/users/assign-trainer', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data: {
        clientId: user._id,
                trainerId: selectedTrainer._id
    }
});
// Оновлюємо дані
setAssignedTrainer(selectedTrainer);
// Оновлюємо user.trainer_id локально
user.trainer_id = selectedTrainer._id;
alert('Тренера успішно призначено! Тепер ви можете працювати з вашим
тренером.');
        } catch (err) {
        console.error('Помилка призначення тренера:', err);
alert('Не вдалося призначити тренера. Спробуйте пізніше.');
        } finally {
setTrainerAssigning(false);
        }
                };
                return (
        <div className="client-dashboard">
            <h2 className="dashboard-title">Кабінет клієнта</h2>
        {error && <div className="alert alert-danger">{error}</div>}
        {!user.trainer_id && !assignedTrainer ? (
// Якщо немає призначеного тренера, показуємо форму вибору
                <div className="trainers-section card">
                    <h3 className="section-title">Виберіть тренера</h3>
        {loading ? (
                        <div className="loading">Завантаження тренерів...</div>
        ) : (
                        <>
                            <div className="trainers-list">
        {trainers.map(trainer => (
        <div
        key={trainer._id}
className={`trainer-card ${selectedTrainer?._id === trainer._id ? 'selected' :
''}`}
onClick={() => setSelectedTrainer(trainer)}
        >
<h4>{trainer.name}</h4>
                                        <p>Email: {trainer.email}</p>
                                    </div>
```

```
        ))}
                            </div>
        {selectedTrainer && (
                                <div className="trainer-selection-footer">
<p>Вибрано тренера: <strong>{selectedTrainer.name}</strong></p>
                                    <button
className="btn btn-primary"
onClick={assignTrainer}
disabled={trainerAssigning}
        >
        {trainerAssigning ? 'Призначення...' : 'Призначити тренера'}
                                    </button>
                                </div>
        )}
                        </>
        )}
                </div>
        ) : (
// Якщо тренер призначений, показуємо робочий дашборд
                <div className="client-workarea">
                    <div className="assigned-trainer card mb-3">
                        <h3 className="section-title">Ваш тренер</h3>
                        <div className="trainer-info">
<h4>{assignedTrainer?.name || `Тренер ID: ${user.trainer_id}`}</h4>
        {assignedTrainer?.email && <p>Email: {assignedTrainer.email}</p>}
                        </div>
                    </div>
                    <div className="dashboard-sections">
                        <Calendar userId={user._id} isTrainer={false} />
                        <ProgressTracker userId={user._id} />
                        <GeminiAdvisor />
                    </div>
                </div>
        )}
        </div>
        );
        };
        const TrainerDashboard = () => {
        const { user, authFetch } = useAuth();
    const [clients, setClients] = useState([]);
    const [selectedClient, setSelectedClient] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
useEffect(() => {
        const getClients = async () => {
setLoading(true);
            try {
                    console.log("Отримання списку клієнтів тренера");
                const res = await authFetch('/users/clients');
                console.log("Отримані клієнти:", res.data);
setClients(res.data || []);
setError(null);
            } catch (err) {
        console.error('Помилка завантаження клієнтів:', err);
setError('Не вдалося завантажити список клієнтів.');
// Додаємо тестові дані, якщо не вдалося отримати з сервера
setClients([
{ _id: 'mock-1', name: 'Анна Петрова', email: 'anna@example.com' },
        { _id: 'mock-2', name: 'Василь Іванов', email: 'vasyl@example.com' },
        { _id: 'mock-3', name: 'Ольга Сидорчук', email: 'olga@example.com' }
        ]);
        } finally {
setLoading(false);
            }
```

```
                    };
getClients();
    }, [authFetch]);
        return (
        <div className="trainer-dashboard">
            <h2 className="dashboard-title">Кабінет тренера</h2>
        {error && <div className="alert alert-danger">{error}</div>}
            <div className="clients-section card mb-3">
                <h3 className="section-title">Ваші клієнти</h3>
        {loading ? (
                    <div className="loading">Завантаження клієнтів...</div>
        ) : (
                    <div className="clients-list">
        {clients.length > 0 ? (
        clients.map(client => (
        <div
        key={client._id}
className={`client-card ${selectedClient?._id === client._id ? 'selected' :
''}`}
onClick={() => setSelectedClient(client)}
        >
<h4>{client.name}</h4>
                                    <p>Email: {client.email}</p>
                                </div>
        ))
        ) : (
<p>У вас поки немає клієнтів.</p>
        )}
                    </div>
        )}
            </div>
        {selectedClient && (
                <div className="client-details">
                    <h3 className="section-title">Деталі клієнта:
{selectedClient.name}</h3>
                    <Calendar userId={user._id} isTrainer={true}
clientId={selectedClient._id} />
                    <ProgressTracker userId={selectedClient._id} />
                </div>
        )}
        </div>
        );
        };
        const AdminDashboard = () => {
        const { authFetch } = useAuth();
    const [users, setUsers] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
useEffect(() => {
        const getUsers = async () => {
setLoading(true);
            try {
                    console.log("Отримання списку всіх користувачів");
                const res = await authFetch('/users');
                console.log("Отримані користувачі:", res.data);
setUsers(res.data || []);
setError(null);
            } catch (err) {
        console.error('Помилка завантаження користувачів:', err);
setError('Не вдалося завантажити список користувачів.');
// Додаємо тестові дані, якщо не вдалося отримати з сервера
setUsers([
{ _id: 'mock-1', name: 'Адміністратор', email: 'admin@example.com', role:
'admin' },
```

```
        { _id: 'mock-2', name: 'Олег Петренко', email: 'oleg@example.com', role:
'trainer' },
        { _id: 'mock-3', name: 'Анна Петрова', email: 'anna@example.com', role:
'client', trainer_id: 'mock-2' },
        { _id: 'mock-4', name: 'Василь Іванов', email: 'vasyl@example.com',
role: 'client', trainer_id: 'mock-2' }
        ]);
        } finally {
setLoading(false);
            }
                    };
getUsers();
    }, [authFetch]);
        const handleDeleteUser = async (userId) => {
        if (window.confirm('Ви впевнені, що хочете видалити цього
користувача?')) {
        try {
        console.log(`Видалення користувача: ${userId}`);
await authFetch(`/users/${userId}`, { method: 'DELETE' });
        console.log('Користувача успішно видалено');
setUsers(users.filter(user => user._id !== userId));
        } catch (err) {
        console.error('Помилка видалення користувача:', err);
alert('Не вдалося видалити користувача. Спробуйте пізніше.');
            }
                    }
                    };
                    return (
        <div className="admin-dashboard">
            <h2 className="dashboard-title">Панель адміністратора</h2>
        {error && <div className="alert alert-danger">{error}</div>}
            <div className="users-list card">
                <h3 className="section-title">Всі користувачі</h3>
        {loading ? (
                    <div className="loading">Завантаження користувачів...</div>
        ) : (
                    <table className="table">
                        <thead>
                            <tr>
                                <th>Ім'я</th>
                                <th>Email</th>
                                <th>Роль</th>
                                <th>Тренер</th>
                                <th>Дії</th>
                            </tr>
                        </thead>
<tbody>
{users.length > 0 ? (
        users.map(user => (
                <tr key={user._id}>
                                        <td>{user.name}</td>
                                        <td>{user.email}</td>
                                        <td>{user.role}</td>
                                        <td>
        {user.trainer_id ? (
                users.find(u => u._id === user.trainer_id)?.name ||
user.trainer_id
                                            ) : (
        '-'
)}
                                        </td>
                                        <td>
                                            <button
        className="btn btn-danger btn-sm"
```

```
    onClick={() => handleDeleteUser(user._id)}
    disabled={user.role === 'admin'} // Не можна видалити адміністратора
            >
            Видалити
            </button>
                                        </td>
                                    </tr>
                                ))
                            ) : (
        <tr>
                                    <td colSpan="5">Немає користувачів</td>
                                </tr>
                            )}
                        </tbody>
                    </table>
        )}
            </div>
        </div>
        );
        };
// Головний компонент дашборду
        const Dashboard = () => {
        const { user, logout } = useAuth();
    const navigate = useNavigate();
    const handleLogout = () => {
logout();
navigate('/auth');
    };
            if (!user) {
        return (
            <div className="loading-container">
                <div className="loading">Завантаження даних користувача...</div>
                <button onClick={() => navigate('/auth')} className="btn btn-
primary">
Повернутися до сторінки входу
                </button>
            </div>
        );
        }
        return (
        <div className="dashboard-container">
            <header className="dashboard-header">
<h1>Фітнес Тренування</h1>
                <nav>
                    <div className="user-info">
<span>{user.name} ({user.role})</span>
                        <button onClick={handleLogout} className="btn btn-
danger">
Вийти
        </button>
                    </div>
                </nav>
            </header>
            <main className="dashboard-content">
        {user.role === 'client' && <ClientDashboard />}
        {user.role === 'trainer' && <TrainerDashboard />}
        {user.role === 'admin' && <AdminDashboard />}
            </main>
            <footer className="dashboard-footer">
                <p>© 2025 Фітнес Тренування. Усі права захищено.</p>
            </footer>
        </div>
        );
```

```
        };
export default Dashboard;
```

ExerciseCatalog.js

```
// src/components/ExerciseCatalog.js
import React, { useState, useEffect } from 'react';
import { useAuth } from '../context/AuthContext';
        const ExerciseCatalog = ({ onSelectExercise }) => {
        const { authFetch } = useAuth();
    const [exercises, setExercises] = useState([]);
    const [activeType, setActiveType] = useState('силові');
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
// Тестові дані для вправ
    const mockExercises = {
силові: [
        { id: 1, title: 'Жим штанги лежачи', description: 'Базова вправа для
грудних м\'язів', type: 'силові' },
        { id: 2, title: 'Присідання зі штангою', description: 'Базова вправа для
ніг', type: 'силові' },
        { id: 3, title: 'Станова тяга', description: 'Базова вправа для спини та
ніг', type: 'силові' },
        { id: 4, title: 'Жим гантелей сидячи', description: 'Вправа для плечей',
type: 'силові' }
        ],
кардіо: [
        { id: 5, title: 'Біг на біговій доріжці', description: 'Кардіо
тренування для витривалості', type: 'кардіо' },
        { id: 6, title: 'Їзда на велотренажері', description: 'Кардіо тренування
з меншим навантаженням на суглоби', type: 'кардіо' },
        { id: 7, title: 'Стрибки зі скакалкою', description: 'Високоінтенсивне
кардіо тренування', type: 'кардіо' }
        ],
похудіння: [
        { id: 8, title: 'Інтервальне тренування (HIIT)', description:
'Чергування високої та низької інтенсивності', type: 'похудіння' },
        { id: 9, title: 'Бурпі', description: 'Комплексна вправа для всього
тіла', type: 'похудіння' },
        { id: 10, title: 'Планка', description: 'Статична вправа для корпусу',
type: 'похудіння' }
        ],
інші: [
        { id: 11, title: 'Йога', description: 'Комплекс вправ для гнучкості та
балансу', type: 'інші' },
        { id: 12, title: 'Пілатес', description: 'Вправи для зміцнення корпусу',
type: 'інші' },
        { id: 13, title: 'Розтяжка', description: 'Комплекс вправ для покращення
гнучкості', type: 'інші' }
        ]
        };
        // Типи вправ
        const exerciseTypes = ['силові', 'кардіо', 'похудіння', 'інші'];
// Завантаження вправ за типом
useEffect(() => {
        const fetchExercises = async () => {
setLoading(true);
            try {
                    console.log(`Завантаження вправ типу: ${activeType}`);
        // Спроба отримати дані з API
        const res = await authFetch(`/exercises/${activeType}`);
        console.log('Отримані вправи з API:', res.data);
                if (res.data && res.data.length > 0) {
```

```
setExercises(res.data);
                } else {
                        console.log('Немає даних з API, використовуємо тестові
дані');
setExercises(mockExercises[activeType] || []);
                }
setError(null);
            } catch (err) {
        console.error('Помилка завантаження вправ:', err);
                console.log('Через помилку API використовуємо тестові дані');
// При помилці використовуємо тестові дані
setExercises(mockExercises[activeType] || []);
setError(null); // Не показуємо помилку користувачу, оскільки у нас є тестові
дані
            } finally {
setLoading(false);
            }
                    };
fetchExercises();
    }, [activeType, authFetch]);
        // Підрахунок кількості вправ за типами
        const [exerciseTypeCounts, setExerciseTypeCounts] = useState({
    силові: mockExercises.силові.length,
            кардіо: mockExercises.кардіо.length,
            похудіння: mockExercises.похудіння.length,
            інші: mockExercises.інші.length
});
        return (
        <div className="exercise-catalog">
            <h3 className="section-title">Каталог вправ</h3>
            <div className="exercise-types">
        {exerciseTypes.map(type => (
        <button
        key={type}
className={`exercise-type-btn ${activeType === type ? 'active' : ''}`}
onClick={() => setActiveType(type)}
        >
        {type.charAt(0).toUpperCase() + type.slice(1)}
                        <span className="exercise-
count">{exerciseTypeCounts[type]}</span>
                    </button>
        ))}
            </div>
        {loading ? (
                <div className="loading">Завантаження вправ...</div>
        ) : error ? (
                <div className="alert alert-danger">{error}</div>
        ) : (
                <div className="exercises-grid">
        {exercises.length > 0 ? (
        exercises.map(exercise => (
        <div
        key={exercise.id}
className="exercise-card"
onClick={() => onSelectExercise && onSelectExercise(exercise)}
        >
<h4>{exercise.title}</h4>
<p>{exercise.description}</p>
                                    <div className="exercise-
type">{exercise.type}</div>
                            </div>
        ))
        ) : (
<p>Немає доступних вправ для цього типу.</p>
```

```
        )}
                </div>
        )}
        </div>
        );
        };
export default ExerciseCatalog;
```

ExerciseCatalogManager.js

```
// src/components/ExerciseCatalogManager.js
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
        const ExerciseCatalogManager = () => {
        const { authFetch } = useAuth();
    const [exercises, setExercises] = useState([]);
    const [activeType, setActiveType] = useState('силові');
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    const [searchTerm, setSearchTerm] = useState('');
    const [formMode, setFormMode] = useState('add'); // 'add' або 'edit'
// Стан для нової/редагованої вправи
    const [exerciseForm, setExerciseForm] = useState({
    id: null,
            title: '',
            description: '',
            type: 'силові'
});
        // Типи вправ
        const exerciseTypes = ['силові', 'кардіо', 'похудіння', 'інші'];
// Завантаження вправ при зміні типу
useEffect(() => {
fetchExercises();
    }, [activeType]);
        // Функція для отримання вправ з API
        const fetchExercises = async () => {
setLoading(true);
        try {
                console.log(`Завантаження вправ типу: ${activeType}`);
        const res = await authFetch(`/exercises/${activeType}`);
setExercises(res.data || []);
setError(null);
        } catch (err) {
        console.error('Помилка завантаження вправ:', err);
setError('Не вдалося завантажити вправи. Спробуйте пізніше.');
// При помилці використовуємо тестові дані
setExercises(getMockExercises());
        } finally {
setLoading(false);
        }
                };
                // Функція для отримання мокованих вправ
                const getMockExercises = () => {
        const mockData = [
        { id: 1, title: 'Жим штанги лежачи', description: 'Базова вправа для
грудних м\'язів', type: 'силові' },
        { id: 2, title: 'Присідання зі штангою', description: 'Базова вправа для
ніг', type: 'силові' },
        { id: 3, title: 'Станова тяга', description: 'Базова вправа для спини та
```

```
ніг', type: 'силові' },
        { id: 4, title: 'Біг на біговій доріжці', description: 'Кардіо
тренування для витривалості', type: 'кардіо' },
        { id: 5, title: 'Їзда на велотренажері', description: 'Кардіо тренування
з меншим навантаженням на суглоби', type: 'кардіо' },
        { id: 6, title: 'Інтервальне тренування (HIIT)', description:
'Чергування високої та низької інтенсивності', type: 'похудіння' },
        { id: 7, title: 'Бурпі', description: 'Комплексна вправа для всього
тіла', type: 'похудіння' },
        { id: 8, title: 'Йога', description: 'Комплекс вправ для гнучкості та
балансу', type: 'інші' },
        { id: 9, title: 'Розтяжка', description: 'Комплекс вправ для покращення
гнучкості', type: 'інші' }
        ];
        return mockData.filter(exercise => exercise.type === activeType);
    };
            // Обробник зміни полів форми
            const handleFormChange = (e) => {
        const { name, value } = e.target;
setExerciseForm({
            ...exerciseForm,
            [name]: value
});
        };
        // Функція для скидання форми
        const resetForm = () => {
setExerciseForm({
    id: null,
            title: '',
            description: '',
            type: activeType
});
setFormMode('add');
    };
            // Функція для підготовки до редагування вправи
            const handleEditClick = (exercise) => {
setExerciseForm({
    id: exercise.id,
            title: exercise.title,
            description: exercise.description,
            type: exercise.type
});
setFormMode('edit');
// Прокручуємо до форми
        window.scrollTo({
    top: 0,
            behavior: 'smooth'
});
        };
        // Функція для збереження вправи (додавання або редагування)
        const handleSubmit = async (e) => {
        e.preventDefault();
        try {
                if (formMode === 'add') {
        // Додавання нової вправи
        const res = await authFetch('/exercises', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data: exerciseForm
});
        console.log('Нова вправа додана:', res.data);
// Оновлюємо список вправ (додаємо тільки якщо тип співпадає з активним)
                if (res.data.type === activeType) {
setExercises([...exercises, res.data]);
```

```
                }
alert('Вправу успішно додано!');
            } else {
                    // Редагування існуючої вправи
                    const res = await authFetch(`/exercises/${exerciseForm.id}`,
{
method: 'PUT',
headers: { 'Content-Type': 'application/json' },
data: exerciseForm
                });
                        console.log('Вправу оновлено:', res.data);
// Оновлюємо список вправ
                if (res.data.type === activeType) {
setExercises(exercises.map(ex =>
        ex.id === exerciseForm.id ? res.data : ex
));
        } else {
// Якщо тип змінився, видаляємо з поточного списку
setExercises(exercises.filter(ex => ex.id !== exerciseForm.id));
        }
alert('Вправу успішно оновлено!');
            }
// Скидаємо форму
resetForm();
        } catch (err) {
        console.error('Помилка збереження вправи:', err);
alert('Не вдалося зберегти вправу. Спробуйте пізніше.');
        }
                };
                // Функція для видалення вправи
                const handleDeleteClick = async (id) => {
        if (!window.confirm('Ви впевнені, що хочете видалити цю вправу?')) {
        return;
        }
        try {
await authFetch(`/exercises/${id}`, {
method: 'DELETE'
        });
        console.log('Вправу видалено:', id);
// Видаляємо вправу зі списку
setExercises(exercises.filter(ex => ex.id !== id));
alert('Вправу успішно видалено!');
        } catch (err) {
        console.error('Помилка видалення вправи:', err);
alert('Не вдалося видалити вправу. Спробуйте пізніше.');
        }
                };
                // Функція для відміни редагування
                const handleCancelEdit = () => {
resetForm();
    };
            // Фільтрація вправ за пошуковим запитом
            const filteredExercises = exercises.filter(exercise =>
        exercise.title.toLowerCase().includes(searchTerm.toLowerCase()) ||
        exercise.description.toLowerCase().includes(searchTerm.toLowerCase())
        );
        return (
        <div className="exercise-catalog-wrapper">
            <div className="catalog-header">
                <div className="catalog-title-container">
<h2>Управління каталогом вправ</h2>
                    <Link to="/dashboard" className="btn-back">
Повернутися до дашборду
        </Link>
```

```
                </div>
                <div className="search-container">
                    <input
type="text"
placeholder="Пошук вправ..."
value={searchTerm}
onChange={(e) => setSearchTerm(e.target.value)}
className="search-input"
        />
                </div>
            </div>
            <div className="exercise-form-container">
<h3>{formMode === 'add' ? 'Додати нову вправу' : 'Редагувати вправу'}</h3>
                <form onSubmit={handleSubmit} className="exercise-form">
                    <div className="form-group">
                        <label htmlFor="title">Назва вправи:</label>
                        <input
type="text"
id="title"
name="title"
value={exerciseForm.title}
onChange={handleFormChange}
className="form-control"
placeholder="Введіть назву вправи"
required
                        />
                    </div>
                    <div className="form-group">
                        <label htmlFor="description">Опис:</label>
                        <textarea
id="description"
name="description"
value={exerciseForm.description}
onChange={handleFormChange}
className="form-control"
placeholder="Опишіть вправу"
rows="3"
required
                        ></textarea>
                    </div>
                    <div className="form-group">
                        <label htmlFor="type">Тип вправи:</label>
                        <select
id="type"
name="type"
value={exerciseForm.type}
onChange={handleFormChange}
className="form-control"
required
                        >
                                {exerciseTypes.map(type => (
        <option key={type} value={type}>
        {type.charAt(0).toUpperCase() + type.slice(1)}
                                </option>
        ))}
                        </select>
                    </div>
                    <div className="form-actions">
                        <button type="submit" className="btn-primary">
        {formMode === 'add' ? 'Додати вправу' : 'Зберегти зміни'}
                        </button>
        {formMode === 'edit' && (
                            <button
type="button"
```

```
className="btn-secondary"
onClick={handleCancelEdit}
        >
Скасувати редагування
                            </button>
        )}
                    </div>
                </form>
            </div>
            <div className="exercises-filter">
<h3>Тип вправи:</h3>
                <div className="filter-buttons">
        {exerciseTypes.map(type => (
        <button
        key={type}
className={`filter-btn ${activeType === type ? 'active' : ''}`}
onClick={() => setActiveType(type)}
        >
        {type.charAt(0).toUpperCase() + type.slice(1)}
                        </button>
        ))}
                </div>
            </div>
            <div className="exercises-list">
<h3>Список вправ ({activeType})</h3>
        {loading ? (
                    <div className="loading-container">
                        <div className="loading-spinner"></div>
<p>Завантаження вправ...</p>
                    </div>
        ) : error ? (
                    <div className="error-message">{error}</div>
        ) : filteredExercises.length === 0 ? (
                    <div className="no-exercises">
<p>Вправи не знайдено. {searchTerm ? 'Спробуйте змінити пошуковий запит.' :
'Додайте нову вправу вище.'}</p>
                    </div>
        ) : (
                    <div className="exercises-table-container">
                        <table className="exercises-table">
                            <thead>
                                <tr>
                                    <th>Назва</th>
                                    <th>Опис</th>
                                    <th>Тип</th>
                                    <th>Дії</th>
                                </tr>
                            </thead>
<tbody>
{filteredExercises.map(exercise => (
        <tr key={exercise.id}>
                                        <td className="exercise-
title">{exercise.title}</td>
                                        <td className="exercise-
description">{exercise.description}</td>
                                        <td className="exercise-type">
                                            <span className="type-
badge">{exercise.type}</span>
                                        </td>
                                        <td className="exercise-actions">
                                            <button
        className="btn-edit"
    onClick={() => handleEditClick(exercise)}
    title="Редагувати"
```

```
            >
                                                    <i className="fas fa-edit"></i>
Ред.
        </button>
                                                <button
        className="btn-delete"
    onClick={() => handleDeleteClick(exercise.id)}
    title="Видалити"
            >
                                                    <i className="fas fa-trash"></i>
Вид.
        </button>
                                        </td>
                                    </tr>
                                ))}
                            </tbody>
                        </table>
                    </div>
        )}
            </div>
        </div>
        );
        };
export default ExerciseCatalogManager;
```

GeminiAdvisor.js

```
// src/components/GeminiAdvisor.js
import React, { useState } from 'react';
import { useAuth } from '../context/AuthContext';
        const GeminiAdvisor = () => {
        const { authFetch } = useAuth();
    const [question, setQuestion] = useState('');
    const [response, setResponse] = useState('');
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);
    const [showResponse, setShowResponse] = useState(false);
    const askGemini = async (e) => {
        e.preventDefault();
        if (!question.trim()) {
        return;
        }
setLoading(true);
setError(null);
setShowResponse(false);
        try {
                console.log('Відправляємо запит до Gemini:', question);
            const res = await authFetch('/gemini/ask', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data: { prompt: question }
});
        console.log('Отримана відповідь від Gemini:', res.data);
            if (res.data && res.data.text) {
setResponse(res.data.text);
setShowResponse(true);
            } else {
                    throw new Error('Отримано порожню відповідь від сервера');
            }
                    } catch (err) {
```

```
        console.error('Помилка запиту до Gemini:', err);
setError('Не вдалося отримати відповідь від ШІ-консультанта. Помилка взаємодії з
AI');
// Для зручності розробки - можемо показувати вміст помилки в консолі
            if (process.env.NODE_ENV === 'development') {
        console.debug('Деталі помилки:', err.response?.data || err.message);
            }
                    } finally {
setLoading(false);
        }
                };
                return (
        <div className="gemini-advisor-card">
            <h3>ШІ-консультант з фітнесу</h3>
            <p className="gemini-subtitle">Поставте питання про тренування,
дієту або здоровий спосіб життя</p>
        {error && (
                <div className="gemini-error">
<p>{error}</p>
                </div>
        )}
            <div className="gemini-form-container">
                <textarea
value={question}
onChange={(e) => setQuestion(e.target.value)}
placeholder="Наприклад: Як скласти програму тренувань для початківця? Які вправи
найкращі для спини?"
rows={3}
disabled={loading}
className="gemini-textarea"
        />
                <button
onClick={askGemini}
className="gemini-ask-btn"
disabled={loading || !question.trim()}
        >
        {loading ? 'Очікування відповіді...' : 'Запитати консультанта'}
                </button>
            </div>
        {showResponse && (
                <div className="gemini-response-container">
                    <div className="gemini-response-header">
<h4>Відповідь консультанта:</h4>
                    </div>
                    <div className="gemini-response-content">
<p>{response}</p>
                    </div>
                </div>
        )}
        </div>
        );
        };
export default GeminiAdvisor;
```

ProgressTracker.js

```
// src/components/ProgressTracker.js
import React, { useState, useEffect } from 'react';
import { useAuth } from '../context/AuthContext';
import {
```

```
        Chart as ChartJS,
        CategoryScale,
        LinearScale,
        PointElement,
        LineElement,
        Title,
        Tooltip,
        Legend,
        Filler
        } from 'chart.js';
import { Line } from 'react-chartjs-2';
// Реєструємо компоненти Chart.js
        ChartJS.register(
        CategoryScale,
        LinearScale,
        PointElement,
        LineElement,
        Title,
        Tooltip,
        Legend,
        Filler
        );
const ProgressTracker = ({ userId }) => {
        const { authFetch } = useAuth();
    const [metrics, setMetrics] = useState([]);
    const [newMetric, setNewMetric] = useState({ type: 'вага', value: '', date:
new Date().toISOString().split('T')[0] });
        const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
useEffect(() => {
        const getMetrics = async () => {
setLoading(true);
            try {
                    console.log(`Отримання метрик для користувача: ${userId}`);
        const res = await authFetch(`/metrics/${userId}`);
        console.log('Отримані метрики:', res.data);
setMetrics(res.data || []);
setError(null);
            } catch (err) {
        console.error('Помилка завантаження метрик:', err);
setError('Не вдалося завантажити метрики прогресу.');
// Додаємо тестові дані, якщо не вдалося отримати з сервера
setMetrics(generateMockMetrics(userId));
        } finally {
setLoading(false);
            }
                    };
                    if (userId) {
getMetrics();
        } else {
setLoading(false);
        }
                }, [userId, authFetch]);
        // Функція для генерації тестових метрик
        const generateMockMetrics = (userId) => {
        const mockMetrics = [];
        const today = new Date();
// Типи метрик і початкові значення
        const metricTypes = [
        { type: 'вага', startValue: 85, change: -0.5 }, // Зменшення ваги
        { type: 'станова', startValue: 120, change: 5 }, // Збільшення ваги в
станової тязі
        { type: 'присідання', startValue: 100, change: 2.5 }, // Збільшення ваги
в присіданнях
```

```
        { type: 'біг', startValue: 5, change: 0.2 } // Збільшення дистанції бігу
        ];
        // Генеруємо 10 записів для кожного типу метрик
        metricTypes.forEach(metricType => {
    for (let i = 0; i < 10; i++) {
                const date = new Date(today);
        date.setDate(today.getDate() - (10 - i) * 3); // Кожні 3 дні, починаючи
з 30 днів тому
                const value = metricType.startValue + (metricType.change * i);
        mockMetrics.push({
                _id: `mock-${metricType.type}-${i}`,
        userId,
                type: metricType.type,
                value: Math.round(value * 10) / 10, // Округлюємо до 1
десяткового знаку
                date: date.toISOString().split('T')[0]
                });
    }
});
        return mockMetrics;
    };
            const handleAddMetric = async (e) => {
        e.preventDefault();
        if (!newMetric.value) {
alert('Будь ласка, введіть значення');
            return;
                    }
                    try {
                    console.log('Додавання нової метрики:', newMetric);
// Виправлення проблеми з датою
            const selectedDate = new Date(newMetric.date);
            const year = selectedDate.getFullYear();
            const month = selectedDate.getMonth();
            const day = selectedDate.getDate();
            const correctedDate = new Date(Date.UTC(year, month, day));
        const correctDateString = correctedDate.toISOString().split('T')[0];
        const metricData = {
        ...newMetric,
date: correctDateString,
userId
            };
                    const res = await authFetch('/metrics', {
    method: 'POST',
            headers: { 'Content-Type': 'application/json' },
    data: metricData
});
        console.log('Результат додавання метрики:', res.data);
setMetrics([...metrics, res.data]);
setNewMetric({ ...newMetric, value: '' });
        } catch (err) {
        console.error('Помилка додавання метрики:', err);
alert('Не вдалося додати метрику. Спробуйте пізніше.');
        }
                };
                // Фільтрація метрик за типом і сортування за датою
                const getFilteredMetrics = (type) => {
        return metrics
        .filter(metric => metric.type === type)
            .sort((a, b) => new Date(a.date) - new Date(b.date));
        };
        // Підготовка даних для графіків
        const prepareChartData = (metricType) => {
        const filteredMetrics = getFilteredMetrics(metricType).slice(-10); //
Останні 10 записів
```

```
        const labels = filteredMetrics.map(metric =>
        new Date(metric.date).toLocaleDateString('uk-UA', { day: 'numeric',
month: 'short' })
        );
        const data = filteredMetrics.map(metric => metric.value);
        // Налаштування кольорів залежно від типу метрики
        let borderColor, backgroundColor;
        switch (metricType) {
        case 'вага':
borderColor = '#e74c3c';
backgroundColor = 'rgba(231, 76, 60, 0.1)';
        break;
        case 'станова':
borderColor = '#3498db';
backgroundColor = 'rgba(52, 152, 219, 0.1)';
        break;
        case 'присідання':
borderColor = '#2ecc71';
backgroundColor = 'rgba(46, 204, 113, 0.1)';
        break;
        case 'біг':
borderColor = '#f39c12';
backgroundColor = 'rgba(243, 156, 18, 0.1)';
        break;
default:
borderColor = '#95a5a6';
backgroundColor = 'rgba(149, 165, 166, 0.1)';
        }
        return {
labels,
datasets: [
        {
label: metricType.charAt(0).toUpperCase() + metricType.slice(1),
data,
borderColor,
backgroundColor,
pointBackgroundColor: borderColor,
pointBorderColor: '#fff',
pointHoverBackgroundColor: '#fff',
pointHoverBorderColor: borderColor,
borderWidth: 2,
tension: 0.3,
fill: true
        }
        ]
        };
        };
        // Налаштування для графіків
        const chartOptions = (metricType) => {
        const unit = metricType === 'біг' ? 'км' : 'кг';
        return {
responsive: true,
maintainAspectRatio: false,
plugins: {
legend: {
position: 'top',
labels: {
font: {
size: 12,
family: "'Roboto', sans-serif"
        }
        }
        },
tooltip: {
```

```
callbacks: {
label: function (context) {
    return `${context.dataset.label}: ${context.raw} ${unit}`;
}
                    },
titleFont: {
size: 13,
family: "'Roboto', sans-serif"
        },
bodyFont: {
size: 12,
family: "'Roboto', sans-serif"
        },
padding: 10,
backgroundColor: 'rgba(42, 49, 50, 0.8)',
titleColor: '#fff',
bodyColor: '#fff',
borderColor: 'rgba(255,255,255,0.2)',
borderWidth: 1,
displayColors: false
        }
        },
scales: {
y: {
title: {
display: true,
text: unit,
font: {
size: 12,
family: "'Roboto', sans-serif"
        }
        },
beginAtZero: false,
grid: {
color: 'rgba(144, 175, 197, 0.1)',
borderDash: [5, 5]
        },
ticks: {
font: {
size: 11,
family: "'Roboto', sans-serif"
        },
precision: 1
        }
        },
x: {
grid: {
display: false
        },
ticks: {
font: {
size: 11,
family: "'Roboto', sans-serif"
        }
        }
        },
elements: {
point: {
radius: 4,
hoverRadius: 6
        },
line: {
tension: 0.3
```

```
        }
        },
interaction: {
mode: 'nearest',
intersect: false
        },
animation: {
duration: 1000
        }
        };
        };
        if (loading) {
        return <div className="loading">Завантаження метрик...</div>;
        }
        return (
        <div className="progress-tracker card">
            <h3 className="section-title">Відстеження прогресу</h3>
        {error && <div className="alert alert-danger">{error}</div>}
            <form onSubmit={handleAddMetric} className="metric-form">
                <select
className="form-control"
value={newMetric.type}
onChange={e => setNewMetric({ ...newMetric, type: e.target.value })}
        >
                    <option value="вага">Вага (кг)</option>
                    <option value="станова">Станова тяга (кг)</option>
                    <option value="присідання">Присідання (кг)</option>
                    <option value="біг">Біг (км)</option>
                </select>
                <input
type="number"
step="0.1"
className="form-control"
placeholder="Значення"
value={newMetric.value}
onChange={e => setNewMetric({ ...newMetric, value: e.target.value })}
required
                />
                <input
type="date"
className="form-control"
value={newMetric.date}
onChange={e => setNewMetric({ ...newMetric, date: e.target.value })}
required
                />
                <button type="submit" className="btn btn-
primary">Додати</button>
            </form>
            <div className="metrics-charts">
        {metrics.length > 0 ? (
                    <div className="metrics-grid">
        {['вага', 'станова', 'присідання', 'біг'].map(type => {
                        const filteredMetrics = getFilteredMetrics(type);
    if (filteredMetrics.length === 0) return null;
                        const chartData = prepareChartData(type);
    return (
            <div key={type} className="metric-chart card">
                                    <h4 className="chart-
title">{type.charAt(0).toUpperCase() + type.slice(1)}</h4>
            {/* Графік для візуалізації прогресу */}
            <div className="chart-container">
                                        <Line data={chartData}
options={chartOptions(type)} />
                                    </div>
```

```
            {/* Таблиця з даними */}
            <div className="table-container">
                                    <table className="table">
                                        <thead>
                                            <tr>
                                                <th>Дата</th>
                                                <th>Значення</th>
                                            </tr>
                                        </thead>
                                        <tbody>
            {filteredMetrics.slice(-5).map(metric => (
                    <tr key={metric._id}>
            <td>{new Date(metric.date).toLocaleDateString('uk-UA')}</td>
                                                    <td>{metric.value} {type
=== 'біг' ? 'км' : 'кг'}</td>
                                                </tr>
                                            ))}
                                        </tbody>
                                    </table>
                                </div>
                            </div>
                        );
})}
                    </div>
        ) : (
<p>Немає записаних показників. Додайте свій перший показник!</p>
        )}
            </div>
        </div>
        );
        };
export default ProgressTracker;
```

WorkoutCalendar.js

```
// src/components/WorkoutCalendar.js
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
import DatePicker, { registerLocale } from 'react-datepicker';
import { uk } from 'date-fns/locale/uk'; // Імпортуємо українську локаль
import "react-datepicker/dist/react-datepicker.css";
// Реєструємо українську локаль
registerLocale('uk', uk);
const WorkoutCalendar = ({ userId, isTrainer, events = [], onEventAdd }) => {
        const { authFetch } = useAuth();
    const [newEvent, setNewEvent] = useState({
    title: '',
            type: 'силові',
            date: new Date() // Тепер використовуємо об'єкт Date замість рядка
});
        const [exerciseOptions, setExerciseOptions] = useState([]);
    const [loading, setLoading] = useState(false);
    const [selectedEvent, setSelectedEvent] = useState(null);
// Завантажуємо список вправ при зміні типу тренування
useEffect(() => {
        const fetchExercises = async () => {
setLoading(true);
            try {
                    const res = await authFetch(`/exercises/${newEvent.type}`);
```

```
setExerciseOptions(res.data || []);
            } catch (err) {
        console.error('Помилка завантаження вправ:', err);
// Якщо помилка, використовуємо моковані дані
setExerciseOptions(getMockExercises(newEvent.type));
        } finally {
setLoading(false);
            }
                    };
fetchExercises();
    }, [newEvent.type, authFetch]);
        // Функція для отримання мокованих вправ
        const getMockExercises = (type) => {
        const mockData = {
силові: [
        { id: 1, title: 'Жим штанги лежачи', description: 'Базова вправа для
грудних м\'язів', type: 'силові' },
        { id: 2, title: 'Присідання зі штангою', description: 'Базова вправа для
ніг', type: 'силові' },
        { id: 3, title: 'Станова тяга', description: 'Базова вправа для спини та
ніг', type: 'силові' }
        ],
кардіо: [
        { id: 4, title: 'Біг на біговій доріжці', description: 'Кардіо
тренування для витривалості', type: 'кардіо' },
        { id: 5, title: 'Їзда на велотренажері', description: 'Кардіо тренування
з меншим навантаженням на суглоби', type: 'кардіо' }
        ],
похудіння: [
        { id: 6, title: 'Інтервальне тренування (HIIT)', description:
'Чергування високої та низької інтенсивності', type: 'похудіння' },
        { id: 7, title: 'Бурпі', description: 'Комплексна вправа для всього
тіла', type: 'похудіння' }
        ],
інші: [
        { id: 8, title: 'Йога', description: 'Комплекс вправ для гнучкості та
балансу', type: 'інші' },
        { id: 9, title: 'Розтяжка', description: 'Комплекс вправ для покращення
гнучкості', type: 'інші' }
        ]
        };
        return mockData[type] || [];
        };
// Функція для отримання кольору події за типом
function getEventColor(type) {
    switch (type) {
        case 'силові':
            return '#336B87'; // Каменний синій
        case 'кардіо':
            return '#90AFC5'; // Туманний синій
        case 'похудіння':
            return '#763626'; // Осіннє листя
        default:
            return '#2A3132'; // Тінь
    }
}
// Обробник зміни полів форми
    const handleInputChange = (e) => {
        const { name, value } = e.target;
setNewEvent({ ...newEvent, [name]: value });
        };
        // Обробник зміни дати з DatePicker
        const handleDateChange = (date) => {
setNewEvent({
```

```
            ...newEvent,
            date: date
});
        };
        // Обробник подання форми
        const handleSubmit = async (e) => {
        e.preventDefault();
        try {
                // Перетворюємо дату в правильний формат для API
                const year = newEvent.date.getFullYear();
            const month = String(newEvent.date.getMonth() + 1).padStart(2, '0');
            const day = String(newEvent.date.getDate()).padStart(2, '0');
            const formattedDate = `${year}-${month}-${day}`;
        console.log("Форматована дата для події:", formattedDate);
            const eventData = {
title: newEvent.title,
date: formattedDate,
type: newEvent.type,
clientId: userId
            };
                    console.log("Додавання нової події:", eventData);
// Викликаємо функцію обробника з батьківського компонента
            if (onEventAdd) {
await onEventAdd(eventData);
            }
// Очищаємо форму
setNewEvent({
    title: '',
            type: 'силові',
            date: new Date()
});
        } catch (err) {
        console.error('Помилка додавання події:', err);
        }
                };
                // Групуємо події за місяцями та датами
                const today = new Date();
    const currentYear = today.getFullYear();
    const currentMonth = today.getMonth();
// Створюємо об'єкт для зберігання подій по датах
    const eventsByDate = {};
        events.forEach(event => {
    try {
        // Нормалізуємо дату (YYYY-MM-DD) без часової зони
            const eventDate = event.date;
        let dateStr;
        if (typeof eventDate === 'string') {
            // Якщо дата вже у форматі рядка, перевіряємо чи це ISO формат
            if (eventDate.includes('T')) {
                // Це ISO формат з часом, відрізаємо час
                dateStr = eventDate.split('T')[0];
            } else {
                // Це вже формат YYYY-MM-DD
                dateStr = eventDate;
            }
        } else {
            // Якщо це об'єкт Date, конвертуємо в ISO і відрізаємо час
            dateStr = new Date(eventDate).toISOString().split('T')[0];
        }
        console.log(`Розміщення події "${event.title}" на дату ${dateStr}
(оригінальна дата: ${event.date})`);
        if (!eventsByDate[dateStr]) {
            eventsByDate[dateStr] = [];
        }
```

```
            eventsByDate[dateStr].push(event);
        } catch (err) {
            console.error(`Помилка при обробці події: ${event.title}`, err);
        }
});
            // Для відлагодження
            console.log("Події по датах:", eventsByDate);
// Генеруємо календар на 3 місяці
    const monthsToShow = 3;
            const calendarMonths = [];
            for (let i = 0; i < monthsToShow; i++) {
            const targetMonth = new Date(currentYear, currentMonth + i, 1);
            const monthYear = targetMonth.toLocaleString('uk-UA', { month: 'long',
year: 'numeric' });
            const daysInMonth = new Date(targetMonth.getFullYear(),
targetMonth.getMonth() + 1, 0).getDate();
// Виправляємо розрахунок першого дня тижня
// JavaScript: 0 = неділя, 1 = понеділок, ..., 6 = субота
// У нашому календарі: 1 = понеділок, ..., 7 = неділя
let firstDayOfWeek = new Date(targetMonth.getFullYear(), targetMonth.getMonth(),
1).getDay();
            if (firstDayOfWeek === 0) firstDayOfWeek = 7; // Переводимо неділю (0) в
7
            const days = [];
            // Додаємо порожні клітинки на початку місяця
            for (let j = 1; j < firstDayOfWeek; j++) {
            days.push({ empty: true });
            }
            // Додаємо дні місяця
            for (let day = 1; day <= daysInMonth; day++) {
            // Створюємо дату в тому ж часовому поясі
            const date = new Date(targetMonth.getFullYear(), targetMonth.getMonth(),
day);
            // Форматуємо дату як YYYY-MM-DD
            // Важливо: додаємо нулі перед однозначними місяцями і днями
            const yyyy = date.getFullYear();
                const mm = String(date.getMonth() + 1).padStart(2, '0'); // Місяці в
JS починаються з 0
                const dd = String(day).padStart(2, '0');
                const dateStr = `${yyyy}-${mm}-${dd}`;
            // Отримуємо події для цього дня
            const dayEvents = eventsByDate[dateStr] || [];
            days.push({
    day,
                date: dateStr,
                events: dayEvents,
                isToday: date.toDateString() === today.toDateString()
});
            }
            calendarMonths.push({
    monthYear,
                days
});
            }
            // Функція форматування дати
            const formatDate = (dateStr) => {
            try {
            const date = new Date(dateStr);
                if (isNaN(date.getTime())) return 'Неправильна дата';
            return date.toLocaleDateString('uk-UA', {
    day: 'numeric',
                month: 'long'
});
            } catch (err) {
```

```
        console.error(`Помилка форматування дати: ${dateStr}`, err);
        return 'Помилка дати';
        }
        };
        // Обробник кліку на подію
        const handleEventClick = (event) => {
setSelectedEvent(event);
    };
            return (
        <div className="workout-calendar card">
            <h3 className="section-title">Календар тренувань</h3>
        {isTrainer && (
                <div className="event-add-section">
                    <form onSubmit={handleSubmit} className="form mb-3">
                        <div className="form-row">
                            <div className="form-group">
<label>Тип тренування:</label>
                                <select
className="form-control"
name="type"
value={newEvent.type}
onChange={handleInputChange}
required
                                >
                                    <option value="силові">Силові</option>
                                    <option value="кардіо">Кардіо</option>
                                    <option value="похудіння">Похудіння</option>
                                    <option value="інші">Інші</option>
                                </select>
                            </div>
                            <div className="form-group">
                                <label>Вправа:</label>
                                <div className="exercise-input-container">
                                    <select
className="form-control"
name="title"
value={newEvent.title}
onChange={handleInputChange}
required
                                    >
                                        <option value="">Виберіть
вправу</option>
        {loading ? (
                                            <option
disabled>Завантаження...</option>
        ) : (
        exerciseOptions.map(exercise => (
        <option key={exercise.id} value={exercise.title}>
        {exercise.title}
                                                </option>
        ))
        )}
                                    </select>
                                </div>
                            </div>
                            <div className="form-group">
                                <label>Дата:</label>
        {/* Використовуємо DatePicker з українською локалізацією */}
                                <DatePicker
selected={newEvent.date}
onChange={handleDateChange}
locale="uk"  // Використовуємо українську локаль
dateFormat="dd.MM.yyyy"
className="form-control"
```

```
required
        placeholderText="Виберіть дату"
calendarStartDay={1} // Тиждень починається з понеділка
        />
                            </div>
                        </div>
                        <div className="form-actions">
                            <button type="submit" className="btn btn-primary">
Додати тренування
                            </button>
        {isTrainer && (
                                <Link to="/exercise-catalog" className="btn btn-
secondary">
Управління каталогом
                                </Link>
        )}
                        </div>
                    </form>
                </div>
        )}
            <div className="calendar-container">
        {calendarMonths.map((month, monthIndex) => (
                    <div key={monthIndex} className="calendar-month">
                        <h4 className="month-title">{month.monthYear}</h4>
                        <div className="calendar-grid">
                            <div className="calendar-weekdays">
                                <div>Пн</div>
                                <div>Вт</div>
                                <div>Ср</div>
                                <div>Чт</div>
                                <div>Пт</div>
                                <div>Сб</div>
                                <div>Нд</div>
                            </div>
                            <div className="calendar-days">
        {month.days.map((day, dayIndex) => (
                                    <div
key={dayIndex}
className={`calendar-day ${day.empty ? 'empty' : ''} ${day.isToday ? 'today' :
''}`}
        >
        {!day.empty && (
                                        <>
                                            <div className="day-
number">{day.day}</div>
                                            <div className="day-events">
        {day.events && day.events.length > 0 ? (
<>
{day.events.slice(0, 3).map((event, eventIndex) => (
        <div
    key={eventIndex}
    className="day-event"
    style={{ backgroundColor: getEventColor(event.type) }}
    title={`${event.title} - ${event.type}`}
    onClick={() => handleEventClick(event)}
                                                >
    {event.title.length > 15
            ? `${event.title.slice(0, 15)}...`
                                                    :
event.title}
                                                </div>
                                            ))}
                                            {day.events.length >
3 && (
```

```
                                                                    <div
className="day-more">
        +{day.events.length - 3} ще
        </div>
        )}
                                                                </>
        ) : null}
                                                        </div>
                                                    </>
        )}
                                    </div>
        ))}
                            </div>
                        </div>
                    </div>
        ))}
            </div>
            <div className="upcoming-events">
                <h4 className="mb-2">Найближчі тренування</h4>
        {events.length > 0 ? (
                    <div className="card-grid">
        {events
        .filter(event => {
                                    const eventDate = new Date(event.date);
    return eventDate >= new Date();
})
        .sort((a, b) => new Date(a.date) - new Date(b.date))
        .slice(0, 5)
                                .map((event, index) => (
                                    <div
key={index}
className="card"
style={{ borderLeft: `4px solid ${getEventColor(event.type)}` }}
onClick={() => handleEventClick(event)}
        >
                                        <div className="event-
date"><strong>{formatDate(event.date)}</strong></div>
                                        <div className="event-
title">{event.title}</div>
                                        <div className="exercise-
type">{event.type}</div>
                                    </div>
        ))}
                    </div>
        ) : (
<p>Немає запланованих тренувань</p>
        )}
            </div>
        {/* Модальне вікно з деталями події */}
        {selectedEvent && (
                <div className="event-details-modal">
                    <div className="event-details-content">
                        <div className="event-details-header">
                            <h3>Тренування: {selectedEvent.title}</h3>
                            <button className="close-modal-btn" onClick={() =>
setSelectedEvent(null)}>×</button>
                        </div>
                        <div className="event-details-body">
                            <p><strong>Тип:</strong> {selectedEvent.type}</p>
                            <p><strong>Дата:</strong>
{formatDate(selectedEvent.date)}</p>
                        </div>
                        <div className="event-details-footer">
                            <button className="btn-primary" onClick={() =>
```

```
setSelectedEvent(null)}>OK</button>
                            </div>
                        </div>
                    </div>
            )}
            </div>
            );
            };
export default WorkoutCalendar;
```

index.html

```
<!-- public/index.html -->
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#336B87" />
    <meta name="description"
content="Веб-додаток для фізичних тренувань" />
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&displ
ay=swap" rel="stylesheet">
<title>Фітнес Тренування</title>
    <style>
/* Додаємо стилі одразу для швидшого завантаження фону */
body::before {
    content: "";
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-image: url('https://sportlife.ua/cdn-
cgi/image/width=968,format=webp/https://r2.sportlife.ua/083_A7262_HDR_Edit_kopiy
a_0e57de2325.jpg');
    background-size: cover;
    background-position: center;
    z-index: -1;
}
        .loading {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    width: 100%;
    font-family: 'Roboto', sans-serif;
    font-size: 1.5rem;
    color: #336B87;
}
    </style>
</head>
<body>
<noscript>Вам потрібно ввімкнути JavaScript для запуску цього
додатку.</noscript>
    <div id="root"></div>
</body>
</html>
```

eslint.config.js

```
module.exports = [
        {
        rules: {
        // Add rules here.
        }
        }
        ];
```