МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ФОРМУВАННЯ ПЕРСОНАЛІЗОВАНИХ ПРОГРАМ ТРЕНУВАНЬ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Н.О. Кіров
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Миронова Н.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кірову Нікіті Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Проєктування та розробка вебзастосунку для формування персоналізованих програм тренувань
керівник роботи Миронова Наталя Олексіївна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі: розробка фітнес-вебзастосунку.
2. Перелік літератури: ДСТУ 3008:2015, технічна документація.
3. Додаткові відомості: стек Node.js, MySQL, PHP.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі: аналіз предметної області.
2. Основні теоретичні відомості: проєктування архітектури системи.
3. Розділ у відповідності з темою: створення та впровадження вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація, UML-діаграми, скріншоти.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.12.2025	
2.	Збір вихідних даних.	21.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	11.05.2026	
6.	Підготовка презентації.	15.05.2026	Для захисту
7.	Оформлення та нормоконтроль	11.06.2026	
	кваліфікаційної роботи бакалавра.		
8.	Захист кваліфікаційної роботи.	19.06.2026	

Студент _____ Н.О. Кіров
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.О. Миронова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та розробка вебзастосунку для формування персоналізованих програм тренувань»: 63 с., 13 рис., 5 табл., 15 джерел, 2 додатки.

АВТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ВЕБІНТЕРФЕЙС, ПЕРСОНАЛІЗОВАНА ПРОГРАМА, ФІТНЕС-ПЛАТФОРМА, ФІТНЕС-ТРЕКЕР, EXPRESS, MYSQL, NODE.JS, PHP, REST API.

Об’єкт дослідження – процес автоматизованого формування персоналізованих програм тренувань.

Предмет дослідження – методи, алгоритми та програмні засоби розробки вебзастосунку для підбору фізичних навантажень.

Мета роботи: розробка вебзастосунку для автоматизованого формування персоналізованих тренувань на основі індивідуальних характеристик користувача.

Методи дослідження – системний аналіз, об’єктно-орієнтоване проєктування, методи проєктування реляційних баз даних, тестування програмного забезпечення.

Роботу присвячено розробці вебзастосунку для генерації адаптивних програм тренувань з урахуванням віку, рівня підготовки, стану здоров’я та цілей користувача. Програмну реалізацію виконано за допомогою платформи Node.js (Express), мови PHP та СУБД MySQL. Клієнт-серверну взаємодію організовано через архітектуру REST API, що забезпечує масштабованість системи. Реалізовано модулі безпечної автентифікації та збереження історії тренувань. Практичне значення полягає у створенні готового продукту, що автоматизує підбір вправ і має потенціал для подальшої інтеграції з фітнес-трекерами та алгоритмами штучного інтелекту.

SUMMARY

Bachelor’s qualifying paper "Design and development of a web application for generating personalized training programs": 63 pages, 13 figures, 5 tables, 15 references, 2 supplements.

KEYWORDS: AUTHENTICATION, DATABASE, EXPRESS, FITNESS PLATFORM, FITNESS TRACKER, MYSQL, NODE.JS, PERSONALIZED PROGRAM, PHP, REST API, WEB APPLICATION, WEB INTERFACE.

The object of the study is the process of automated generation of personalized training programs.

The subject of the research is methods, algorithms, and software tools for developing a web application for selecting physical workloads.

The aim of the study is to develop a web application for the automated generation of personalized workouts based on individual user characteristics.

The methods of research are system analysis, object-oriented design, methods of relational database design, and software testing.

The work is devoted to the development of a web application for generating adaptive training programs taking into account the user’s age, fitness level, health condition, and goals. The software implementation was performed using the Node.js platform (Express), the PHP language, and the MySQL DBMS. Client-server interaction is organized through the REST API architecture, which ensures system scalability. Modules for secure authentication and training history storage have been implemented. The practical significance lies in the creation of a finished product that automates the selection of exercises and has the potential for further integration with fitness trackers and artificial intelligence algorithms.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та вибір технологій 10
1.1 Характеристика предметної області фітнес-застосунків та персоналізації тренувань 10
1.2 Аналіз існуючих вебзастосунків для підбору програм тренувань..... 11
1.3 Формування функціональних та нефункціональних вимог до вебзастосунку 13
1.4 Обґрунтування вибору технологічного стеку 15
1.5 Постановка задачі та визначення основних сценаріїв роботи користувачів 16
2 Побудова структури вебзастосунку та бази даних 18
2.1 Опис архітектурної організації вебзастосунку 18
2.2 Розробка логічної структури бази даних 19
2.3 Характеристика сутностей, атрибутів та зв’язків бази даних 21
2.4 Формування алгоритмів бізнес-логіки підбору вправ 24
2.5 Формування структури інтерфейсу користувача та навігації вебзастосунку 28
3 Створення та впровадження вебзастосунку 32
3.1 Розробка серверної частини вебзастосунку з використанням Node.js та Express 32
3.2 Створення та налаштування реляційної бази даних MySQL 35
3.3 Впровадження функціоналу обробки даних користувача та генерації тренувань..... 38

3.4 Розробка клієнтської частини вебзастосунку засобами HTML, CSS, JavaScript та PHP 39
3.5 Забезпечення захисту даних та перевірка стабільності роботи вебзастосунку 41
Висновки 45
Перелік посилань 48
Додаток А Фрагменти вихідного коду програмного продукту 50
Додаток Б SQL-скрипт ініціалізації бази даних trainsmart_db 62

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та зростання інтересу суспільства до здорового способу життя, процеси планування фізичної активності потребують нових, доступних та ефективних цифрових рішень. Підтримання належної фізичної форми є пріоритетними завданнями для багатьох людей [1]. Проте ефективне та безпечне виконання фізичних вправ неможливе без індивідуального підходу, що враховує ключові параметри: вік, рівень поточної підготовки, наявність обмежень та конкретні цілі (схуднення, набір м’язової маси, підтримка форми). Звернення до професійних фітнес-тренерів часто вимагає значних фінансових та часових витрат, що обмежує доступність персоналізованого підходу для широкого кола осіб.

Аналіз існуючих на ринку програмних рішень, мобільних та вебзастосунків показує, що значна частина з них пропонує користувачам переважно стандартизовані, шаблонні програми тренувань, які не здатні повною мірою адаптуватися під унікальні потреби конкретної людини. Інші ж системи є комерційно закритими або перевантаженими надлишковим функціоналом, що ускладнює їх повсякденне використання. Це підтверджує необхідність у створенні доступного, інтелектуального вебзастосунку, який би автоматизував процес індивідуального підбору фізичних навантажень та об’єднував необхідні функції в єдиній платформі.

Метою роботи є теоретичне обґрунтування та практична розробка вебзастосунку для формування персоналізованих програм тренувань, який забезпечує автоматизований підбір фізичних навантажень на основі індивідуальних характеристик користувача та зручну взаємодію через сучасний вебінтерфейс. Для досягнення поставленої мети необхідно розв’язати низку завдань, що включають аналіз предметної області, розробку алгоритму формування адаптивних рекомендацій, проєктування архітектури системи та

реляційної бази даних, а також безпосередню програмну реалізацію та тестування функціональних модулів.

Об’єктом дослідження є процес автоматизованого формування персоналізованих програм фізичних тренувань. Предметом дослідження виступають методи, алгоритми та програмні засоби розробки вебзастосунку для індивідуального підбору фізичних навантажень. Напрямок досліджень ґрунтується на застосуванні системного підходу та об’єктно-орієнтованого проєктування [2] для створення цілісної та модульної архітектури системи.

Обґрунтуванням основних проєктних рішень є застосування клієнт-серверної архітектури [4] з розділенням рівнів відображення та бізнес-логіки. Інтерфейс (frontend) реалізовано на HTML5, CSS3 [11] та JavaScript [10] з елементами SPA, тоді як PHP [9] використовується для шаблонізації сторінок. Серверна частина (backend) працює на базі Node.js [7] із фреймворком Express [8] як REST API [4] сервер. Для структурованого зберігання даних застосовано СУБД MySQL [6]. Асинхронна взаємодія через REST API забезпечує швидкодію та масштабованість системи. У застосунку реалізовано алгоритм підбору вправ (з урахуванням здоров’я), клієнтський захист маршрутів, модулі безпечної автентифікації, управління профілем та збереження історії тренувань.

Практичне значення одержаних результатів полягає у створенні готового до експлуатації програмного продукту, який підвищує доступність фітнес-планування та автоматизує рутинні процеси. Запропонована система може бути впроваджена у фітнес-центрах, тренажерних залах або використовуватися індивідуально. Подальший розвиток дослідження передбачає інтеграцію з мобільними застосунками, фітнес-трекерами та впровадження елементів штучного інтелекту для точного прогнозування результатів тренувань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Характеристика предметної області фітнес-застосунків та персоналізації тренувань

У сучасному світі спостерігається стрімке зростання інтересу до здорового способу життя та підтримки фізичної форми [1]. Водночас темп життя сучасної людини часто обмежує можливості для регулярних консультацій із професійними тренерами чи лікарями у сфері спортивної медицини. Це призвело до активної цифровізації фітнес-індустрії та появи великої кількості програмних продуктів, спрямованих на підтримку фізичної активності користувачів.

Предметною областю даного дослідження є процеси автоматизованого формування та управління персоналізованими програмами фізичних тренувань. На сьогоднішній день ринок фітнес-застосунків пропонує широкий спектр рішень: від простих трекерів активності, які фіксують кількість кроків чи витрачені калорії до складних екосистем, що поєднують віртуальний коучинг, соціальну взаємодію та збір біометричних даних.

Однак ключовою проблемою більшості масових фітнес-продуктів є їхня орієнтація на усередненого споживача. Значна частина таких вебзастосунків та мобільних програм пропонує шаблонні рішення, які не здатні повною мірою адаптуватися під унікальні характеристики конкретної людини. Такий підхід має суттєві недоліки, оскільки ефективність та безпечність фізичних навантажень критично залежать від низки індивідуальних факторів.

Для розробки ефективного програмного продукту у цій сфері необхідно враховувати наступні базові параметри персоналізації:

- вік та антропометричні дані користувача;
- поточний рівень фізичної підготовки, що охоплює категорії від початківців до професіоналів;

- наявність медичних протипоказань або обмежень щодо здоров’я, зокрема травм суглобів, проблем із серцево-судинною системою чи захворювань хребта;
- головна мета тренувань, серед яких зниження ваги, набір м’язової маси або підтримка загального тонусу організму.

Таким чином, розробка спеціалізованого вебзастосунку, який використовує алгоритмічний підхід для генерації адаптивних програм тренувань з урахуванням перелічених вище параметрів, є актуальним завданням. Використання клієнт-серверної архітектури дозволить забезпечити гнучкість системи, швидку обробку даних та можливість постійного оновлення бази вправ без необхідності встановлення додаткового програмного забезпечення на пристрої кінцевого користувача.

1.2 Аналіз існуючих вебзастосунків для підбору програм тренувань

Ринок програмного забезпечення у сфері фітнесу та спорту є надзвичайно насиченим. Сучасні інформаційні системи для підтримки фізичної активності можна умовно поділити на кілька категорій: трекери активності, цифрові щоденники харчування, бібліотеки готових відео-тренувань та інтелектуальні генератори тренувальних програм. Для визначення конкурентних переваг розроблюваного вебзастосунку було проведено аналіз найбільш популярних існуючих рішень у ніші автоматизованого планування тренувань.

Одним із лідерів глобального ринку є застосунок Nike Training Club (NTC). Основною перевагою цієї системи є наявність великої бази якісного відеоконтенту та детальних інструкцій від професійних тренерів. Проте система має суттєві недоліки щодо персоналізації:

- тренування здебільшого є шаблонними комплексними програмами, розрахованими на усередненого користувача з базовим рівнем здоров’я;

- відсутня гнучка система фільтрації вправ на основі конкретних медичних обмежень організму;
- недостатня адаптація алгоритмів під вікові особливості користувачів старших вікових груп.

Іншим популярним рішенням є система Fitbod, яка використовує алгоритмічний підхід для створення тренувань на основі доступного спортивного інвентарю та історії попередньої активності користувача. Незважаючи на високий рівень автоматизації, цей застосунок має певні функціональні обмеження:

- фокус спрямовано переважно на силові показники та м’язову гіпертрофію, що не завжди відповідає цілям початківців або осіб, які прагнуть підтримувати форму;
- відсутність деталізованого обліку специфічних травм (наприклад, система не завжди здатна коректно підібрати заміну базовим вправам при поєднанні кардіологічних протипоказань та травм колінних суглобів);
- комерційна закритість екосистеми та висока вартість платної підписки, що значно знижує доступність продукту для широкої аудиторії.

Платформа Freeletics пропонує високоінтенсивні тренування переважно з власною вагою та використовує елементи штучного інтелекту для виконання ролі цифрового тренера. Головним недоліком цієї системи є високий поріг входження: запропоновані програми часто є занадто виснажливими для нетренованих осіб. Крім того, алгоритм не передбачає м’якої адаптації навантажень для користувачів із захворюваннями опорно-рухового апарату або серцево-судинної системи, що може призвести до травмування.

Аналіз існуючих програмних рішень дозволяє зробити висновок, що на сучасному ринку бракує доступного, інтуїтивно зрозумілого вебзастосунку, який би ставив на перше місце безпеку та фізіологічні особливості кінцевого користувача. Більшість комерційних продуктів орієнтовані на досягнення швидкого спортивного результату, нехтуючи детальним збором анамнезу,

зокрема наявністю травм суглобів, болю у спині, проблем із тиском чи дихальною системою.

Отже, розробка власного вебзастосунку, який інтегрує багатокритеріальний алгоритм фільтрації фізичних вправ на основі віку, поточного рівня підготовки, цілей та чітко визначених медичних протипоказань, дозволить зайняти вільну нішу. Такий підхід дасть змогу запропонувати користувачам безпечний і високоперсоналізований інструмент для планування фізичної активності, позбавлений недоліків існуючих комерційних аналогів.

1.3 Формування функціональних та нефункціональних вимог до вебзастосунку

Процес розробки програмного забезпечення вимагає чіткого визначення вимог до майбутнього продукту, які поділяються на функціональні та нефункціональні [2]. Функціональні вимоги описують специфічну поведінку системи та її безпосередні завдання. Для розроблюваного вебзастосунку головним функціональним завданням є забезпечення зручної взаємодії між користувачем та базою даних тренувань. До базових функціональних вимог належать реєстрація та безпечна автентифікація користувачів, створення і редагування персонального профілю, а також введення даних, віку, цілей та наявних медичних обмежень. Крім того, система повинна автоматично генерувати тренувальну програму на основі введених параметрів, надавати доступ до бібліотеки вправ із детальним описом та візуальним супроводом, а також зберігати історію тренувань для відстеження прогресу.

Нефункціональні вимоги у свою чергу визначають атрибути якості системи та стандарти її роботи. Проєктований вебзастосунок має бути інтуїтивно зрозумілим та оптимізованим для роботи на десктопних комп'ютерах, забезпечуючи кросбраузерну сумісність і стабільне відображення графічних елементів на сучасних моніторах. Надзвичайно важливим аспектом є продуктивність, що передбачає швидку генерацію тренувальних планів та

безперебійний доступ до бази даних. Увага приділяється безпеці, яка гарантує надійний захист конфіденційної інформації користувачів за допомогою безпечної автентифікації та захисту маршрутів. Використання асинхронної взаємодії через архітектуру програмного інтерфейсу забезпечить високу швидкодію та масштабованість системи у майбутньому.

Для візуального та наочного представлення логіки взаємодії користувачів із системою створюється діаграма прецедентів [3]. На рисунку 1.1 зображено основні сценарії роботи застосунку. Головним актором на цій діаграмі виступає авторизований користувач, який безпосередньо ініціює ключові процеси: заповнення медичної анкети, отримання персоналізованої програми, перегляд техніки виконання вправ та управління налаштуваннями власного облікового запису. Така деталізація вимог та сценаріїв дозволяє сформувати чіткий план для подальшого вибору технологічного стеку та переходу до етапу архітектурного проєктування.

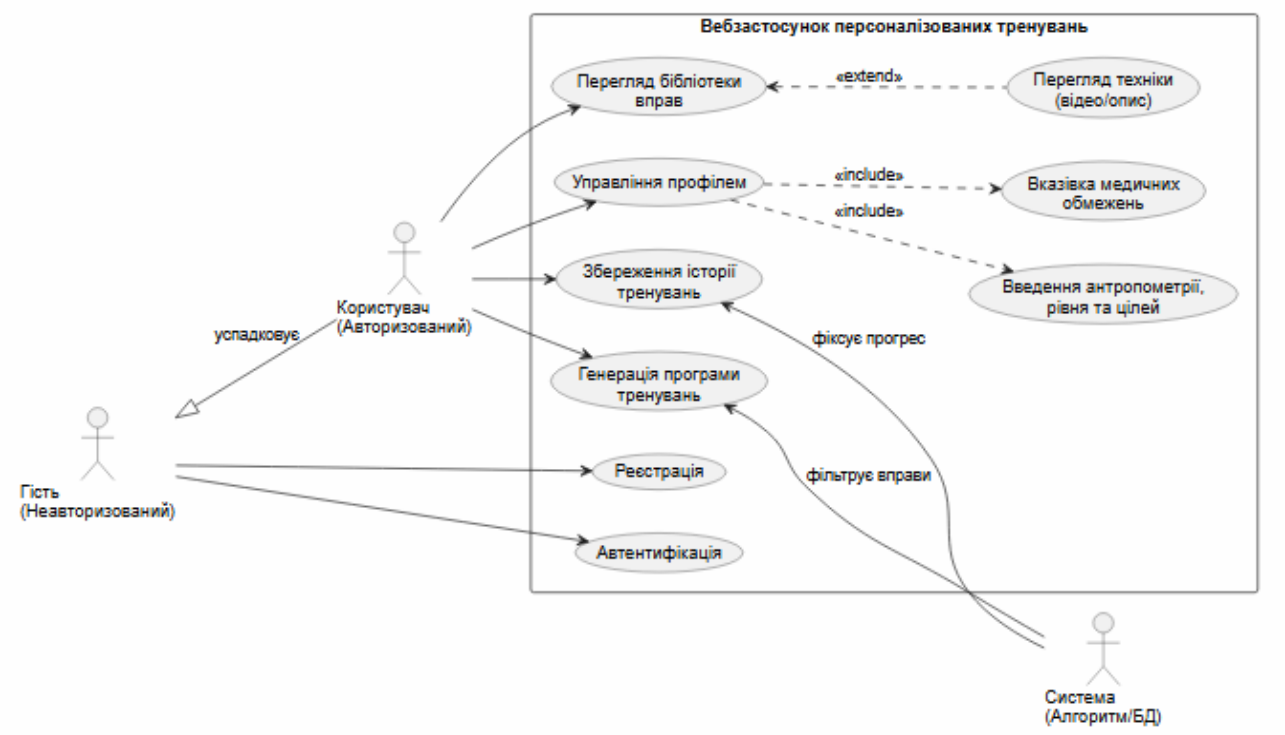


Рисунок 1.1 – Діаграма прецедентів вебзастосунку

1.4 Обґрунтування вибору технологічного стеку

Успішна реалізація будь-якого програмного продукту значною мірою залежить від правильного вибору технологічного стеку. Для забезпечення високої продуктивності, масштабованості та зручності підтримки вебзастосунку було обрано клієнт-серверну архітектуру [4] з використанням сучасних інструментів веброзробки. Розділення рівнів відображення та бізнес-логіки дозволяє незалежно оновлювати різні модулі системи та полегшує подальше масштабування проєкту.

Візуальна частина застосунку побудована з використанням базових вебтехнологій, серед яких мова гіпертекстової розмітки HTML5, каскадні таблиці стилів CSS3 [11] та мова програмування JavaScript [10]. Такий набір дозволяє створити зручний користувацький інтерфейс, оптимізований для роботи на десктопних пристроях. Використання цих технологій гарантує кросбраузерну сумісність та коректне відображення графічних елементів на сучасних моніторах і ноутбуках. Для генерації динамічних сторінок та шаблонізації інтерфейсу додатково залучено мову препроцесора PHP [9]. Використання PHP на рівні відображення забезпечує зручну інтеграцію серверних даних у розмітку сторінки та прискорює первинне завантаження контенту для користувача.

Серверна логіка застосунку реалізована на базі середовища виконання Node.js [7]. Головною перевагою цієї платформи є асинхронна та неблокуюча модель обробки запитів, яка ідеально підходить для створення швидких мережевих додатків. Для спрощення маршрутизації використовується мінімалістичний вебфреймворк Express [8]. Завдяки Express вдалося легко налаштувати архітектуру програмного інтерфейсу REST API [4], що дозволяє клієнтській частині ефективно обмінюватися даними із сервером у зручному текстовому форматі.

Для структурованого зберігання інформації обрано систему управління реляційними базами даних MySQL [5, 6]. Вибір саме реляційної моделі

обґрунтований чіткою структурою предметної області, де сутності, такі як користувачі, тренувальні програми, вправи та медичні обмеження, мають суворі зв’язки між собою. Система MySQL забезпечує високу надійність збереження даних, швидке виконання складних фільтраційних запитів та підтримує механізми транзакцій, що є важливим для безпечної роботи з персональними обліковими записами.

Поєднання наведених технологій створює надійний фундамент для розробки. Платформа Node.js та фреймворк Express гарантують високу швидкість обробки логіки підбору тренувань, система MySQL забезпечує цілісність складної бази даних вправ, а зв’язка HTML, CSS, JavaScript та PHP дозволяє реалізувати надійний інструмент для кінцевого споживача фітнес-продукту.

1.5 Постановка задачі та визначення основних сценаріїв роботи користувачів

На основі проведеного аналізу предметної області та виявлених недоліків існуючих рішень формулюється головна мета кваліфікаційної роботи. Метою дослідження є проєктування та розробка вебзастосунку для автоматизованого формування персоналізованих програм тренувань, який базується на індивідуальних фізичних та медичних показниках користувача. Для досягнення цієї мети необхідно вирішити комплекс завдань, що охоплюють усі ключові етапи життєвого циклу розробки програмного забезпечення.

Основними завданнями розробки є створення реляційної бази даних для структурованого зберігання інформації про користувачів, вправи та медичні протипоказання; побудова серверної частини на базі Node.js та Express для реалізації алгоритмів бізнес-логіки; розробка зручного кросбраузерного інтерфейсу користувача, оптимізованого для десктопних моніторів, із застосуванням HTML, CSS, JavaScript та PHP. Ключовим технічним завданням

виступає реалізація програмного алгоритму, який здатний фільтрувати базу вправ та генерувати безпечний тренувальний план, суворо враховуючи вікові обмеження, поточний рівень підготовки та специфічні захворювання клієнта.

Для забезпечення коректної роботи алгоритмів та зручності використання системи необхідно чітко визначити основні сценарії взаємодії кінцевого користувача з вебзастосунком. Життєвий цикл роботи із платформою починається з етапу реєстрації та подальшої безпечної автентифікації, що дозволяє створити унікальний обліковий запис для захищеного зберігання персональних даних. Після успішної авторизації користувач отримує доступ до панелі управління профілем, де обов'язковим кроком є заповнення детальної анкети. У цій анкеті фіксуються антропометричні параметри, рівень фізичної активності, головна мета занять та наявні медичні обмеження.

Наступним сценарієм є генерація тренувальної програми. Система обробляє введені користувачем дані, звертається до серверної бази даних через архітектуру REST API та формує індивідуальний перелік безпечних вправ, автоматично відсіюючи ті навантаження, які можуть загострити вказані травми чи проблеми зі здоров'ям. Отримавши готову програму, користувач має змогу детально переглядати інформацію про кожну призначену вправу, включаючи текстовий опис та візуальну демонстрацію правильної техніки виконання. Фінальним етапом базового сценарію є збереження згенерованого тренування в історії профілю, що дозволяє відстежувати регулярність активності та контролювати загальний спортивний прогрес.

Визначена постановка задачі та детально описані сценарії роботи користувачів створюють надійне підґрунтя для переходу до наступного етапу дослідження, а саме до проєктування логічної структури реляційної бази даних та архітектурної організації вебзастосунку.

2 ПОБУДОВА СТРУКТУРИ ВЕБЗАСТОСУНКУ ТА БАЗИ ДАНИХ

2.1 Опис архітектурної організації вебзастосунку

Архітектура програмного забезпечення є основою будь-якої інформаційної системи, оскільки вона визначає структуру, взаємодію компонентів та принципи обміну даними. Для розроблюваного вебзастосунку з генерації персоналізованих програм тренувань було обрано трирівневу клієнт-серверну архітектуру [4]. Такий підхід забезпечує чітке розділення відповідальності між рівнем представлення даних, рівнем бізнес-логіки та рівнем зберігання інформації, що значно підвищує надійність, безпеку та масштабованість розробленої системи.

Рівень представлення реалізується у веббраузері на десктопному пристрої користувача. Цей компонент відповідає за візуалізацію інтерфейсу, збір вхідних даних, зокрема антропометричних показників, інформації про травми та медичні обмеження, а також відображення згенерованих тренувальних програм. Клієнтський рівень побудовано з використанням базових вебтехнологій розмітки та стилізації, мови JavaScript для забезпечення інтерактивності, а також препроцесора PHP для динамічної шаблонізації сторінок.

Рівень бізнес-логіки, який становить серверну частину застосунку, є ядром системи. Він функціонує на базі середовища Node.js із використанням вебфреймворку Express [7, 8]. Цей вузол відповідає за обробку запитів від клієнта, управління сесіями та автентифікацію користувачів, валідацію вхідних даних та виконання ключового алгоритму фільтрації вправ на основі індивідуальних медичних протипоказань. Взаємодія між клієнтською та серверною частинами здійснюється через архітектурний стиль REST API [4] за допомогою мережевих протоколів передачі даних.

Рівень зберігання даних представлений системою управління реляційними базами даних MySQL [6]. Він забезпечує структуроване та безпечне збереження профілів користувачів, деталізованого довідника фізичних вправ, довідника

медичних обмежень та персональної історії тренувань. Серверний застосунок взаємодіє з базою даних для виконання операцій пошуку, запису та оновлення інформації.

Для наочного відображення логічної структури системи та взаємодії її основних програмних модулів розроблено діаграму компонентів [3], яку наведено на рисунку 2.1. Фізичне розташування вузлів системи та шляхи обміну даними між ними проілюстровано за допомогою діаграми розгортання [3] на рисунку 2.2.



Рисунок 2.1 – Діаграма компонентів вебзастосунку

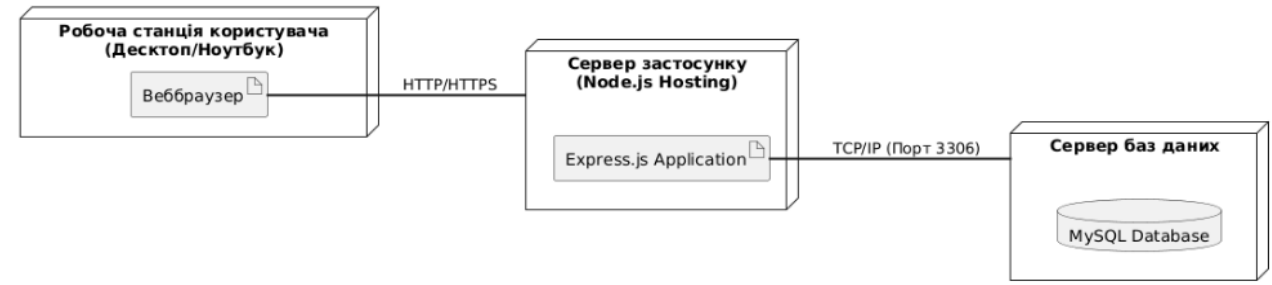


Рисунок 2.2 – Діаграма розгортання вебзастосунку

2.2 Розробка логічної структури бази даних

Основою для функціонування інформаційного вебзастосунку є правильно спроєктована база даних, яка забезпечує швидкий доступ до інформації та гарантує її цілісність. Для розроблюваної системи персоналізованих тренувань було обрано реляційну модель даних [5], реалізовану за допомогою системи управління базами даних MySQL [6]. Процес проєктування логічної структури

складався з визначення основних сутностей предметної області, встановлення зв’язків між ними та проведення нормалізації [5] з метою усунення надмірності та дублювання інформації.

Головною сутністю системи є таблиця користувачів, яка зберігає облікові дані для безпечної автентифікації, зокрема електронну пошту та захешований пароль. Персональні показники винесені в окрему пов’язану таблицю профілів. У цій таблиці зберігаються ім’я користувача, його вік, поточний рівень підготовки, обрана мета тренувань та перелік наявних обмежень по здоров’ю. Такий підхід дозволяє розмежувати авторизаційні дані від фізіологічних метрик, які серверний алгоритм використовує для генерації безпечних тренувальних планів.

Основним масивом інформації в системі є каталог фізичних вправ. Структура відповідної таблиці містить детальний опис кожної вправи, цільову групу м’язів, рівень складності, цільове призначення та рекомендовану кількість підходів і повторень. Важливою особливістю цієї сутності є наявність полів для збереження мультимедійного супроводу, а саме посилання на відео, а також атрибута для фіксації медичних протипоказань. Завдяки цим полям система отримує можливість динамічно відсіювати небезпечні навантаження за допомогою структурованих запитів до бази даних.

Для фіксації згенерованих планів розроблено таблицю тренувальних програм, яка пов’язана з конкретним користувачем та містить назву і дату створення програми. Деталізація кожного тренування зберігається у спеціальній проміжній таблиці складу програми. Ця сутність об’єднує загальну програму з конкретними вправами з каталогу, додатково фіксуючи індивідуальну кількість підходів та повторень для кожного окремого випадку. Це забезпечує гнучкість системи та дозволяє зберігати хронологію призначених навантажень.

Для наочної демонстрації взаємозв’язків між описаними таблицями розроблено діаграму «Сутність-Зв’язок» [3] , яку наведено на рисунку 2.3. На ній відображено первинні та зовнішні ключі, атрибути кожної таблиці, а також типи кардинальності зв’язків між усіма елементами логічної структури бази даних.

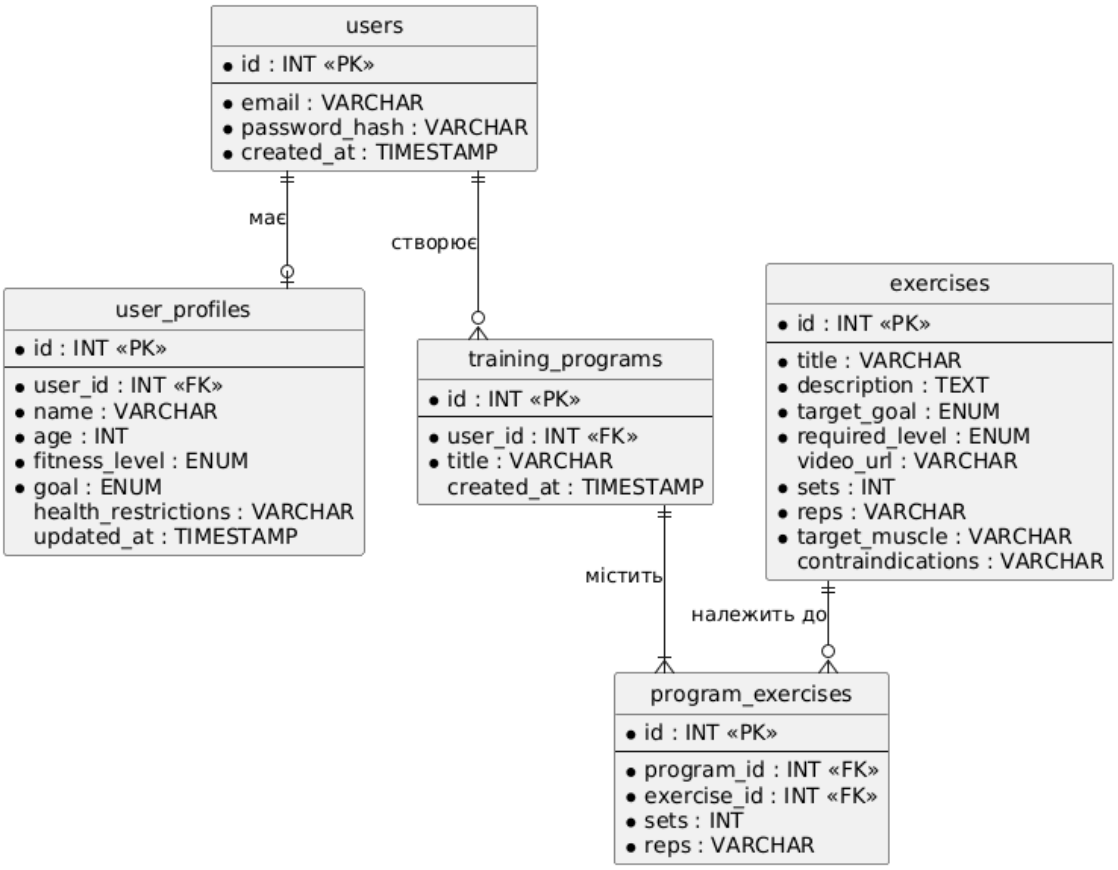


Рисунок 2.3 – Логічна структура реляційної бази даних

2.3 Характеристика сутностей, атрибутів та зв’язків бази даних

Для детального розуміння логічної структури розробленої бази даних необхідно розглянути кожну сутність окремо. Основні таблиці системи містять набори атрибутів, які забезпечують виконання закладеної бізнес-логіки вебзастосунку.

Таблиця користувачів призначена для збереження базових облікових даних, необхідних для безпечної реєстрації та авторизації клієнтів у системі. Ця сутність має зв’язок один до одного з таблицею профілів, оскільки кожному обліковому запису відповідає лише одна персональна анкета. Також користувачі пов’язані відношенням один до багатьох із таблицею збережених програм, адже один клієнт може згенерувати та зберегти необмежену кількість планів

тренувань протягом усього часу користування застосунком. Її структуру наведено у таблиці 2.1.

Таблиця 2.1 – Опис атрибутів таблиці користувачів

Назва поля	Тип даних	Опис атрибута
id	INT	Унікальний ідентифікатор користувача, первинний ключ
email	VARCHAR	Електронна пошта для входу в систему, унікальне значення
password_hash	VARCHAR	Захешований пароль для гарантування безпеки даних
created_at	TIMESTAMP	Час та дата реєстрації облікового запису

Таблиця профілів містить розширену персональну інформацію про фізичні показники та стан здоров’я клієнта. Дані з цієї сутності є важливими для роботи алгоритму індивідуального підбору вправ. Як було зазначено вище, профілі мають жорсткий зв’язок один до одного з користувачами. Головним елементом цього зв’язку виступає зовнішній ключ, який гарантує, що фізичні метрики надійно прив’язані до конкретного власника і не можуть існувати окремо від нього. Опис полів представлено у таблиці 2.2.

Таблиця 2.2 – Опис атрибутів таблиці профілів

Назва поля	Тип даних	Опис атрибута
id	INT	Унікальний ідентифікатор профілю, первинний ключ
user_id	INT	Зовнішній ключ для зв’язку з таблицею користувачів
name	VARCHAR	Ім’я користувача для персоналізації інтерфейсу
age	INT	Вік користувача у роках
fitness_level	ENUM	Поточний рівень фізичної підготовки
goal	ENUM	Головна мета тренувального процесу
health_restrictions	VARCHAR	Перелік наявних медичних протипоказань та обмежень
updated_at	TIMESTAMP	Час останнього оновлення даних анкети

Каталог вправ виступає головним інформаційним довідником фізичних навантажень. Ця сутність містить не лише текстові та мультимедійні описові характеристики, але й критерії фільтрації за медичними протипоказаннями.

Довідник вправ має зв’язок один до багатьох із проміжною таблицею складу програми. Це означає, що одна й та сама вправа з каталогу може бути призначена різним користувачам та входити до складу багатьох різних тренувальних планів без дублювання базового опису. Структуру сутності відображено у таблиці 2.3.

Таблиця 2.3 – Опис атрибутів таблиці вправ

Назва поля	Тип даних	Опис атрибута
id	INT	Унікальний ідентифікатор вправи, первинний ключ
title	VARCHAR	Назва фізичної вправи
description	TEXT	Детальний текстовий опис техніки виконання
target_goal	ENUM	Цільове призначення вправи
required_level	ENUM	Мінімальний рівень підготовки для безпечного виконання
video_url	VARCHAR	Посилання на відео з демонстрацією правильної техніки
sets	INT	Базова рекомендована кількість підходів
reps	VARCHAR	Базова рекомендована кількість повторень
target_muscle	VARCHAR	Цільова група м’язів, на яку спрямовано навантаження
contraindications	VARCHAR	Ключові слова медичних обмежень для відсіювання вправи

Таблиця збережених програм фіксує факт генерації персонального тренувального плану та формує історію активності. Вона виступає центральним вузлом для групування навантажень. Ця сутність має зв’язок багато до одного з користувачами, оскільки безліч програм можуть належати одному клієнту. З іншого боку, тренувальні програми пов’язані відношенням один до багатьох із таблицею складу програми, адже кожен згенерований план складається з цілого переліку окремих вправ. Деталі наведено у таблиці 2.4.

Таблиця 2.4 – Опис атрибутів таблиці збережених програм

Назва поля	Тип даних	Опис атрибута
id	INT	Унікальний ідентифікатор програми, первинний ключ
user_id	INT	Зовнішній ключ для зв’язку з власником програми

title	VARCHAR	Назва збереженого тренувального плану
created_at	TIMESTAMP	Дата та час автоматичної генерації плану

Таблиця складу програми є проміжною сутністю, яка вирішує проблему зв’язку багато до багатьох між каталогом вправ та збереженими програмами. Вона поєднує їх, утворюючи два незалежні зв’язки багато до одного. Таким чином, кожен запис у цій таблиці чітко вказує, яка саме вправа належить до якої програми, дозволяючи при цьому фіксувати індивідуальні параметри виконання, зокрема кількість підходів та повторень для конкретного тренування. Характеристику полів показано у таблиці 2.5.

Таблиця 2.5 – Опис атрибутів таблиці складу програми

Назва поля	Тип даних	Опис атрибута
id	INT	Унікальний ідентифікатор запису, первинний ключ
program_id	INT	Зовнішній ключ для зв’язку з таблицею програм
exercise_id	INT	Зовнішній ключ для зв’язку з довідником вправ
sets	INT	Індивідуально призначена кількість підходів
reps	VARCHAR	Індивідуально призначена кількість повторень

Усі перелічені сутності об’єднані системою реляційних зв’язків. Використання зовнішніх ключів із правилами каскадного видалення [5, 6] гарантує відсутність ізольованих записів. Зокрема, при видаленні облікового запису користувача автоматично знищується його профіль, історія згенерованих програм та детальний склад кожної з них, що підтримує базу даних в актуальному та оптимізованому стані.

2.4 Формування алгоритмів бізнес-логіки підбору вправ

Основним завданням розроблюваного вебзастосунку є забезпечення користувачів персоналізованими тренувальними планами. Виконання цього

завдання покладається на програмне ядро системи, яке реалізує алгоритми бізнес-логіки. Бізнес-логіка виступає сполучною ланкою між клієнтським інтерфейсом та базою даних, забезпечуючи коректну обробку вхідних запитів, виконання математичних та логічних операцій, а також формування відповіді у зручному для відображення форматі.

Ключовим алгоритмом системи є процес генерації індивідуальної програми тренувань. Робота цього алгоритму базується на багатокритеріальній фільтрації масиву даних. Ініціалізація процесу відбувається після того, як авторизований користувач надсилає відповідний запит через інтерфейс застосунку. Серверна частина системи першочергово звертається до таблиці профілів користувачів для отримання актуальних фізіологічних показників клієнта. Основними критеріями для подальшої вибірки виступають поточний рівень підготовки, обрана мета тренувального процесу та наявні медичні обмеження.

Після отримання персональних даних алгоритм формує структурований запит до довідника фізичних вправ. Програмний модуль виконує суворе зіставлення атрибутів. Зокрема мета клієнта порівнюється з цільовим призначенням кожної вправи, а рівень підготовки визначає допустиму складність виконання. Навантаження, які виходять за межі поточних можливостей користувача або не відповідають його спортивним цілям, відсіюються.

Найважливішим етапом алгоритмічної фільтрації є перевірка медичних протипоказань. Система аналізує текстові ідентифікатори хвороб та травм з анкети користувача і зіставляє їх із відповідним атрибутом обмежень кожної вправи в каталозі. У разі виявлення збігу вправа автоматично виключається з потенційного списку. Такий підхід до фільтрації гарантує високий рівень безпеки згенерованого плану та унеможливлює призначення навантажень, здатних загострити існуючі проблеми зі здоров’ям.

З відфільтрованого масиву безпечних вправ алгоритм за заданою пріоритетністю обирає необхідну кількість позицій для формування повноцінного заняття, охоплюючи різні цільові групи м’язів. Наступним кроком є розрахунок індивідуальної кількості підходів та повторень. Цей параметр

базується на еталонних рекомендаціях з довідника вправ, проте програмний модуль може коригувати його відповідно до обраної мети, збільшуючи інтенсивність для схуднення або об’єм навантаження для набору м’язової маси.

Для наочного представлення описаного алгоритму та розуміння послідовності переходів між станами системи розроблено діаграму діяльності [3]. На рисунку 2.4 відображено детальний процес взаємодії серверної логіки з даними під час автоматичного формування тренувальної програми.

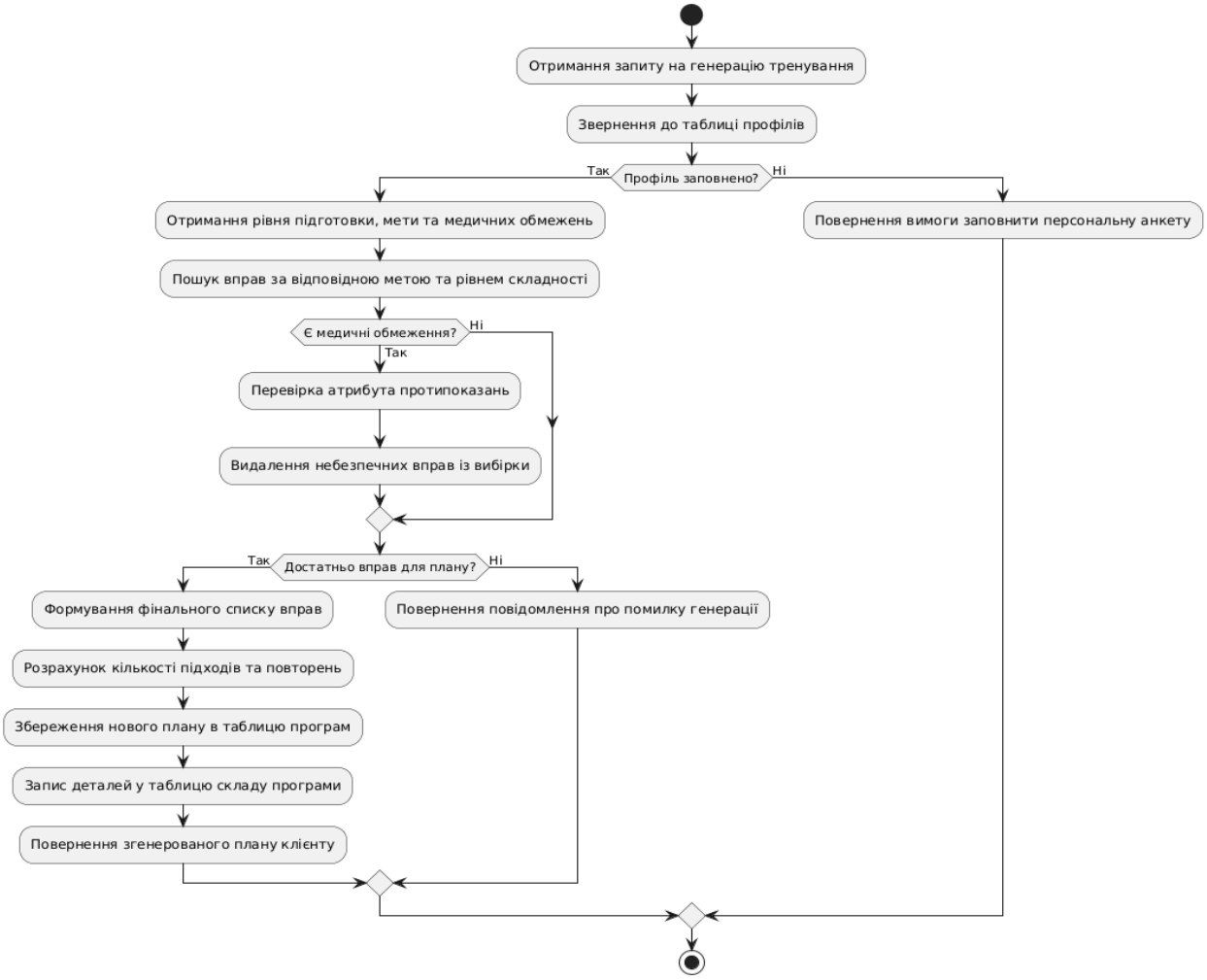


Рисунок 2.4 – Діаграма діяльності алгоритму генерації тренувальної програми

Окрім внутрішньої логіки роботи алгоритмів, важливим аспектом проєктування є визначення порядку взаємодії між різними програмними компонентами у часі. Для цього використовується діаграма послідовності, яка деталізує етапи обміну повідомленнями між клієнтським інтерфейсом, контролерами серверної частини та системою управління базами даних.

Сценарій генерації програми охоплює не лише базовий алгоритм підбору, але й процедури перевірки прав доступу та надійної фіксації результатів у системі збереження даних.

Процес розпочинається з ініціалізації запиту від веббраузера користувача. Запит обов’язково містить захищений маркер авторизації, який серверна частина перевіряє для підтвердження особи клієнта. Після успішної перевірки серверний контролер ініціює серію послідовних транзакцій до бази даних. Спочатку виконується операція читання персональних показників. Отримавши ці дані, серверне ядро запускає описаний раніше алгоритм фільтрації. Після успішного формування готового переліку вправ сервер ініціює операцію запису. Створюється новий рядок у таблиці збережених програм, після чого сервер отримує унікальний ідентифікатор щойно створеного плану. Використовуючи цей ідентифікатор, виконується пакетне додавання інформації до проміжної таблиці складу програми, пов’язуючи кожну призначену вправу з новим тренуванням.

Лише після успішного підтвердження всіх транзакцій базою даних сервер формує остаточну відповідь у форматі структурованого текстового об’єкта та надсилає її клієнтському інтерфейсу для подальшої візуалізації. Взаємодію компонентів під час цього процесу проілюстровано діаграмою послідовності на рисунку 2.5.

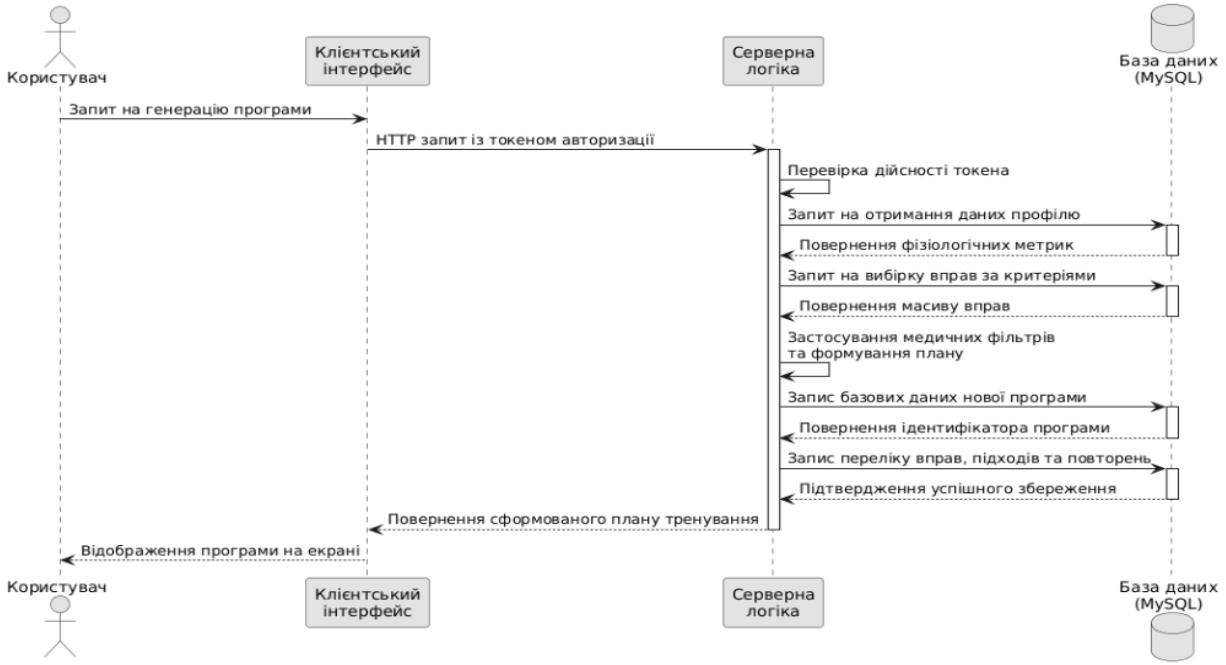


Рисунок 2.5 – Діаграма послідовності процесу створення та збереження тренування

2.5 Формування структури інтерфейсу користувача та навігації вебзастосунку

Розробка зручного та інтуїтивно зрозумілого інтерфейсу користувача є важливим етапом проєктування вебзастосунку. Візуальна частина системи виступає єдиним каналом взаємодії між кінцевим споживачем та складними серверними алгоритмами. Для забезпечення високого рівня зручності використання було проведено етап прототипування, який дозволив заздалегідь продумати логіку розміщення інформаційних блоків та навігаційних елементів на екранах десктопних моніторів. Головною метою цього етапу стало створення ергономічної структури сторінок, де користувач може швидко знаходити потрібні функції без додаткового когнітивного навантаження.

Процес візуального проєктування базувався на принципах кросбраузерної сумісності та оптимізації простору під широкі екрани стаціонарних комп’ютерів і ноутбуків. На етапі створення каркасних моделей, відомих як вайрфрейми, було

визначено основні зони уваги. Верхня частина інтерфейсу традиційно відводиться під глобальну навігаційну панель, яка забезпечує швидкий перехід між панеллю управління, каталогом вправ, особистим профілем та історією тренувань. Центральна область екрана виділяється для відображення ключового контенту, зокрема форм введення даних, згенерованих планів та відеоматеріалів.

Першим екраном, з яким взаємодіє новий клієнт, є форма формування персонального профілю. Цей інтерфейс спроєктовано максимально лаконічно, але інформативно. Візуальна структура сторінки анкети містить згруповані поля для введення віку, вибору рівня підготовки та головної мети. Окремий акцент зроблено на блоці вибору медичних обмежень, який реалізовано у вигляді списку з можливістю множинного вибору. Це гарантує безпомилкове зчитування системою протипоказань для подальшої коректної роботи алгоритму медичної фільтрації. Додатковим інтерфейсом є сторінка історії тренувань, яка виконує функцію цифрового щоденника. На ній у хронологічному порядку відображаються всі раніше збережені програми. Користувач має змогу розгорнути кожен запис для перегляду деталей минулих занять. Макет відповідної сторінки профілю представлено на рисунку 2.6.

Центральним елементом вебзастосунку виступає головна панель управління, звідки ініціюється процес підбору тренувань. Після відпрацювання серверної логіки інтерфейс миттєво оновлюється, демонструючи згенеровану програму. Візуальне представлення плану тренування реалізовано у вигляді структурованого списку карток. Кожна картка містить назву вправи, індивідуально розраховану кількість підходів та повторень, а також кнопку швидкого переходу до перегляду техніки виконання. Такий підхід дозволяє користувачеві швидко оцінити загальний обсяг майбутнього навантаження.

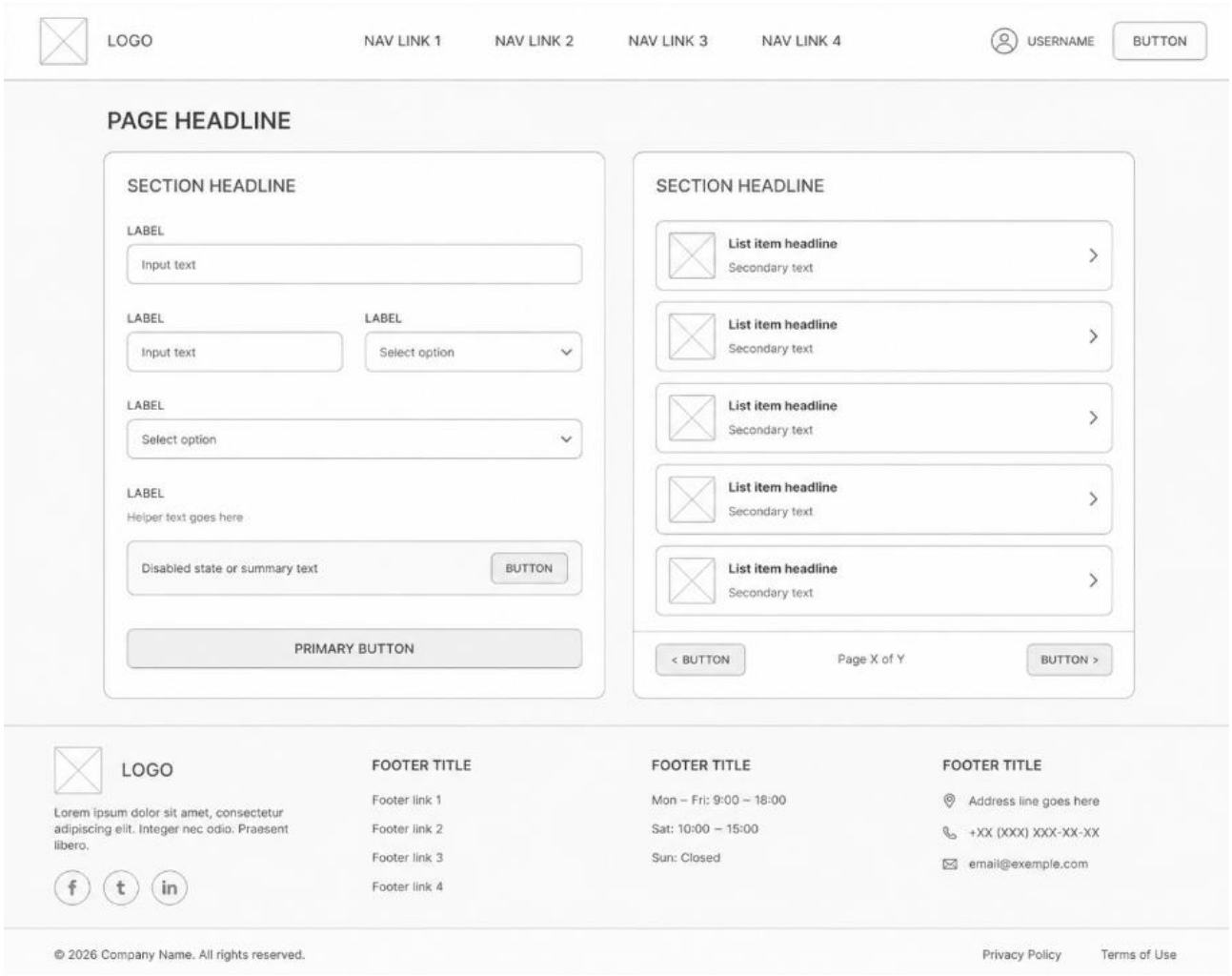


Рисунок 2.6 – Каркасна модель сторінки профілю користувача та історії тренувань

Для детального ознайомлення з технікою виконання кожної призначеної вправи розроблено окремий інтерфейс з можливістю переходу до зовнішніх мультимедійних ресурсів. Замість вбудованого програвача система використовує механізм формування динамічних посилань на пошукові запити у відеохостингу за назвою конкретної вправи. Це дозволяє отримати актуальну добірку відеоінструкцій із техніки виконання без завантаження серверного сховища важким контентом. Під навігаційним блоком розташовується детальний текстовий опис алгоритму рухів та цільова група м’язів. Відсутність зайвих графічних елементів на цій сторінці забезпечує фокусування уваги на правильному алгоритмі тренування. Макет екрана зі згенерованою програмою та елементами взаємодії наведено на рисунку 2.7.

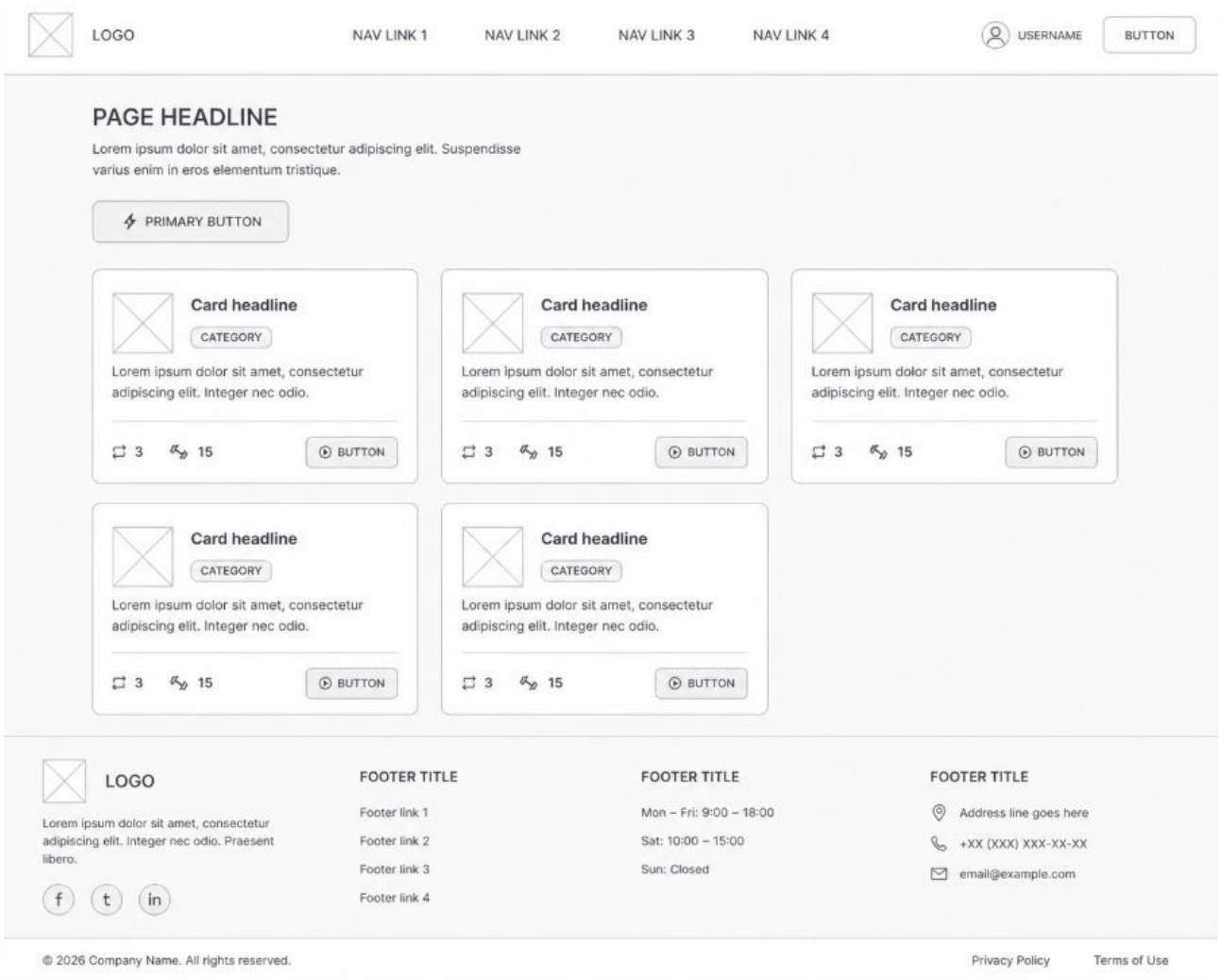


Рисунок 2.7 – Каркасна модель сторінки згенерованого тренувального плану

3 СТВОРЕННЯ ТА ВПРОВАДЖЕННЯ ВЕБЗАСТОСУНКУ

3.1 Розробка серверної частини вебзастосунку з використанням Node.js та Express

Процес реалізації програмного продукту розпочинається з розробки серверної частини, яка виступає фундаментом для всієї бізнес-логіки системи та забезпечує взаємодію з базою даних. Серверна частина вебзастосунку для формування персоналізованих тренувань побудована на базі платформи Node.js, яка забезпечує високу швидкість обробки мережевих запитів завдяки подійній моделі та асинхронному вводу-виводу. Для структурування серверного коду та побудови архітектури програмного інтерфейсу застосовано фреймворк Express.

Розробка серверного модуля ініціюється з налаштування головного файлу входу, який відповідає за ініціалізацію вебсервера, підключення необхідних бібліотек та визначення конфігурації проміжних обробників. На початковому етапі виконується встановлення залежностей через пакетний менеджер, що дозволяє автоматизувати управління зовнішніми модулями. Основними компонентами конфігурації сервера є модулі для підтримки крос-доменних запитів та парсингу вхідних даних у форматі JSON. Це необхідно для забезпечення коректного обміну інформацією між клієнтським інтерфейсом та сервером.

Організація вихідного коду базується на принципі модульності, що дозволяє ізолювати окремі функції системи. Уся структура серверної частини розділена на маршрути та контролери. Маршрути визначають шлях запиту від клієнта до конкретної функції сервера, а контролери містять безпосередню логіку обробки цих запитів. Такий підхід спрощує процес тестування та модернізації програмного продукту. Опис структури каталогів серверної частини представлено на рисунку 3.1.

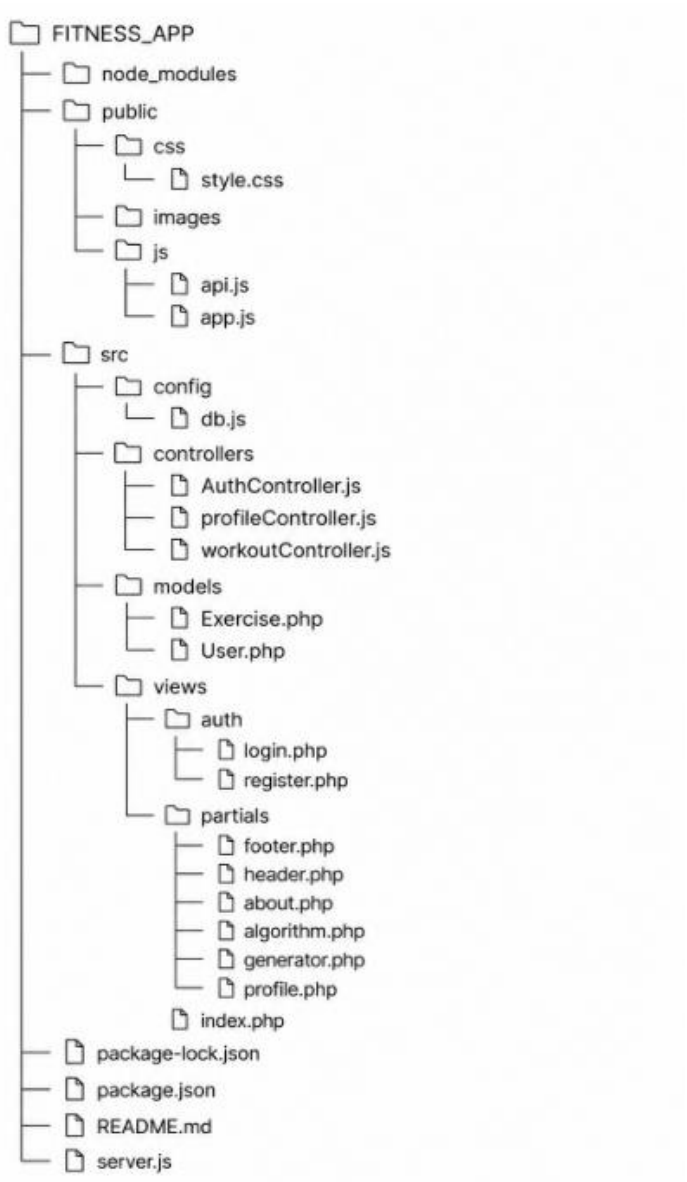


Рисунок 3.1 – Структура каталогів проєкту у середовищі розробки

Важливим аспектом розробки серверної частини є забезпечення безпечного з’єднання з базою даних. Для цього створено окремий конфігураційний модуль, який використовує пул з’єднань. Використання пулу з’єднань дозволяє оптимізувати навантаження на систему, оскільки сервер не створює нове підключення для кожного запиту, а використовує вже існуючі вільні ресурси. Параметри підключення, такі як хост, користувач та назва бази даних, винесені в окремі налаштування для зручності розгортання системи на різних серверах.

У межах реалізації серверної логіки розроблено систему маршрутизації, яка охоплює всі функціональні можливості застосунку. Зокрема, виділено групу маршрутів для автентифікації користувачів та окрему групу для роботи з програмами тренувань. Кожен маршрут пов’язаний із відповідним методом у контролері, який виконує валідацію вхідних даних та взаємодіє з моделями даних. Повний текст програмного коду головного серверного файлу та опис маршрутів наведено у додатку А.

Для обробки вхідних даних від користувача, таких як ім’я, вік та фізичні показники, на сервері реалізовано механізми перевірки типів та значень. Це запобігає виникненню помилок під час виконання SQL-запитів та захищає базу даних від некоректної інформації. Усі відповіді сервера формуються у стандартному структурованому форматі, що містить статус виконання операції та безпосередні дані або повідомлення про помилку. Приклад процесу ініціалізації сервера та підключення маршрутів відображено на рисунку 3.2.

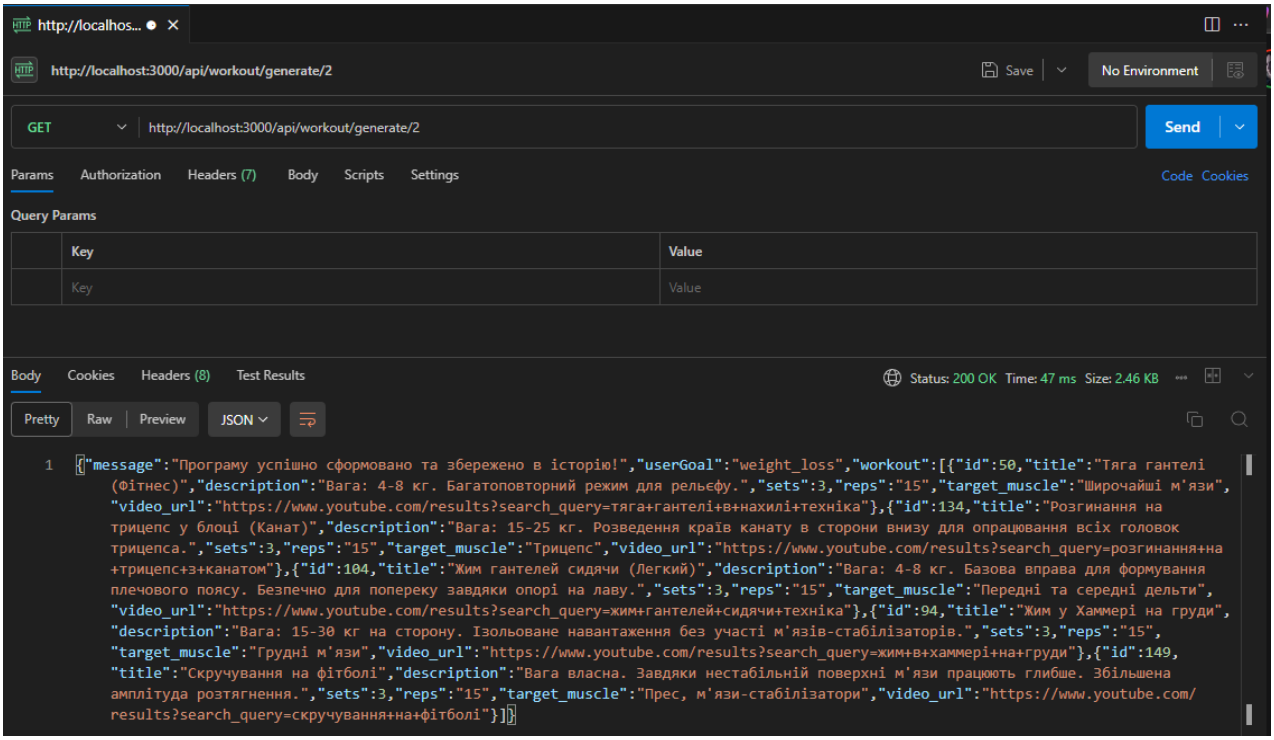


Рисунок 3.2 – Скриншот тестування програмного інтерфейсу через Postman

Реалізована серверна частина забезпечує функціональність вебзастосунку, включаючи реєстрацію користувачів, управління профілем та запуск складних алгоритмів підбору вправ. Використання Node.js та Express дозволило створити гнучкий програмний інтерфейс, який легко інтегрується з будь-яким типом клієнтського застосунку та демонструє високу стабільність під час одночасної роботи кількох користувачів. Перевірка працездатності розроблених серверних модулів здійснюється через зовнішні інструменти тестування мережевих запитів, що дозволяє виявити та усунути помилки до моменту повної інтеграції з візуальною частиною проєкту.

3.2 Створення та налаштування реляційної бази даних MySQL

Етап розробки та конфігурації бази даних є важливим для забезпечення стабільного функціонування вебзастосунку та гарантування цілісності інформації. Для структурованого зберігання даних обрано систему управління реляційними базами даних MySQL [6]. Вибір даної СУБД обґрунтований її високою продуктивністю при виконанні складних вибірок, підтримкою транзакцій та надійністю механізмів захисту персональної інформації. Повний текст SQL-запитів для ініціалізації таблиць та налаштування зв’язків представлено у додатку Б.

Процес створення бази даних розпочато з формування логічної структури, яка охоплює всі сутності предметної області: користувачів, їхні профілі, каталог вправ та історію тренувальних планів. Для забезпечення зв’язків між таблицями використано механізм зовнішніх ключів. Це дозволяє побудувати цілісну екосистему даних, де зміна або видалення інформації в одній таблиці автоматично враховується в пов’язаних структурах завдяки правилам каскадного оновлення та видалення.

Реалізовано таблицю для автентифікації користувачів. У цій структурі зберігаються лише важливі дані для входу в систему, а саме унікальна адреса

електронної пошти та захешований пароль. Використання хеш-функцій на рівні серверної логіки перед записом у базу даних забезпечує захист від несанкціонованого доступу. Кожен запис отримує унікальний числовий ідентифікатор, який виступає первинним ключем і використовується для ідентифікації клієнта в інших модулях системи.

Для деталізації персональних характеристик розроблено таблицю профілів. Ця сутність має жорсткий зв’язок один до одного з таблицею користувачів. Окрім антропометричних даних, таких як ім’я та вік, у таблиці фіксуються рівень фізичної підготовки та головна мета занять. Для оптимізації зберігання та запобігання введенню некоректних даних для полів цілей та рівнів використано перелічуваний тип даних ENUM. Це обмежує можливі значення лише тими варіантами, які підтримуються алгоритмом генерації. Окремий атрибут виділено для збереження текстових кодів медичних обмежень, що є основою для подальшої фільтрації безпечних вправ.

Інформаційним ядром системи виступає каталог фізичних вправ. Кожен запис у цій таблиці містить текстовий опис техніки виконання, цільову групу м’язів та рекомендовані параметри навантаження. Для забезпечення мультимедійного супроводу передбачено збереження посилань на зовнішні відеоресурси. Особливістю даної таблиці є наявність поля протипоказань. У ньому зберігаються ідентифікатори травм та захворювань, при яких виконання конкретної вправи є недопустимим.

Для реалізації функціоналу збереження прогресу розроблено систему з двох пов’язаних таблиць: заголовків тренувальних програм та складу цих програм. Оскільки між програмами та вправами існує зв’язок багато до багатьох, застосовано проміжну таблицю. Це дозволяє для кожної конкретної програми зберігати індивідуальні налаштування кількості підходів та повторень, не дублюючи при цьому основну інформацію про вправу. Така архітектура забезпечує гнучкість системи та можливість ведення детальної історії активності користувача. Загальну логічну схему взаємозв’язків між усіма сутностями бази даних відображено у вигляді ER-діаграми на рисунку 3.3.

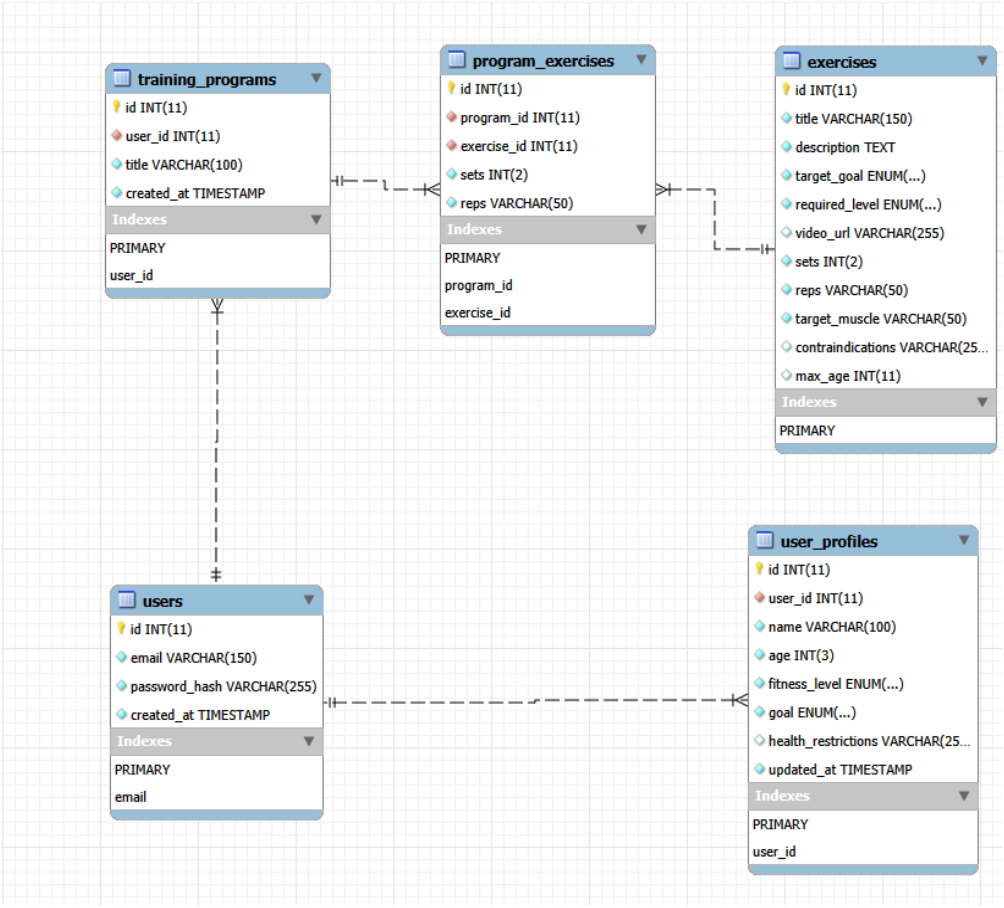


Рисунок 3.3 – ER-діаграма структури бази даних

Налаштування бази даних виконано з використанням сучасного кодування символів utf8mb4. Це гарантує коректне відображення кириличних символів у текстових описах та підтримку спеціальних знаків. Використання пулу з’єднань на рівні серверної конфігурації дозволяє системі стабільно працювати при одночасних запитах від багатьох клієнтів, мінімізуючи витрати ресурсів на встановлення нових підключень до бази даних. Програмна реалізація модуля конфігурації та налаштування пулу з’єднань із базою даних на стороні сервера представлена у додатку А.

3.3 Впровадження функціоналу обробки даних користувача та генерації тренувань

Реалізація функціональних можливостей вебзастосунку базується на створенні механізмів обробки вхідних даних та інтелектуальних алгоритмів підбору вправ. Основна логіка системи зосереджена в контролерах серверної частини, які забезпечують життєвий цикл інформації від моменту реєстрації користувача до отримання ним готової тренувальної програми. Програмна реалізація описуваних методів та алгоритмів представлена у додатку А.

Процес обробки даних розпочинається з модуля автентифікації. Під час реєстрації система виконує комплексне очищення та перевірку вхідних параметрів. Особлива увага приділяється безпеці збереження облікових даних, що реалізується шляхом хешування паролів [13] перед їх записом у базу даних. Такий підхід унеможливлює витік конфіденційної інформації у відкритому вигляді. Програмна логіка модуля реєстрації побудована на використанні транзакцій, що гарантує цілісність даних: запис у таблицю профілів створюється лише за умови успішного створення системного акаунта користувача.

Важливим етапом є впровадження функціоналу управління профілем. Система забезпечує динамічне оновлення антропометричних даних та фізичних показників клієнта. Обробка інформації про стан здоров’я реалізована через збір кодів медичних обмежень, які зберігаються у форматі текстового рядка. Це дозволяє додавати нові типи протипоказань без необхідності зміни структури таблиць бази даних. Кожне оновлення профілю супроводжується серверною валідацією, яка обмежує введення некоректних значень віку або порожніх імен.

Центральним компонентом розробленого програмного продукту є алгоритм генерації персоналізованих тренувальних програм. Робота цього модуля базується на багатоетапній фільтрації бази вправ за декількома критеріями одночасно. На першому етапі алгоритм аналізує головну мету користувача та рівень його фізичної підготовки, відсікаючи вправи, які не відповідають заданій складності або цільовому призначенню.

Важливою частиною алгоритму є механізм медичної фільтрації. Система порівнює масив обмежень зі списку профілю користувача із полем протипоказань у таблиці вправ. Якщо для певної вправи знайдено збіг хоча б за одним медичним параметром, вона повністю виключається із процесу підбору. Такий підхід гарантує формування безпечного плану навантажень, що мінімізує ризик травмування під час занять. Після завершення фільтрації система виконує випадковий вибір необхідної кількості вправ із переліку доступних, що забезпечує різноманітність тренувального процесу при повторних генераціях.

Завершальним етапом роботи даного функціоналу є автоматичне збереження сформованої програми в історію активності. Це реалізується через створення запису у таблиці програм із подальшим заповненням проміжної таблиці складу тренування, де для кожної вправи фіксуються індивідуальні параметри підходів та повторень. Програмна архітектура контролера тренувань дозволяє також перевіряти наявність актуальних програм, сформованих раніше, що зменшує навантаження на базу даних при частому зверненні до інтерфейсу генератора.

3.4 Розробка клієнтської частини вебзастосунку засобами HTML, CSS, JavaScript та PHP

Клієнтська частина системи є основним інструментом взаємодії користувача з розробленими алгоритмами формування тренувань. Реалізація візуального інтерфейсу базується на використанні сучасних стандартів вебверстки та мов програмування, що забезпечують динамічність і високу швидкість відгуку сторінок. Архітектура фронтенд-частини побудована на принципі розділення структури контенту, його візуального оформлення та програмної логіки поведінки елементів.

Для створення структури сторінок та організації шаблонів на стороні клієнта застосовано мову PHP [9]. Це дозволяє реалізувати модульну структуру

вебзастосунку, де спільні елементи інтерфейсу, такі як верхня панель навігації та підвал сайту, винесені в окремі файли та підключаються до основних представлень. Такий підхід забезпечує єдність дизайну на всіх сторінках системи та спрощує внесення змін у навігаційне меню. Приділено увагу створенню форм введення даних, які забезпечують передачу параметрів користувача до серверної частини через асинхронні запити.

Візуальне оформлення вебзастосунку реалізовано за допомогою каскадних таблиць стилів стандарту CSS3 [11]. Дизайн системи виконано у темній кольоровій гамі з використанням акцентного неоново-помаранчевого кольору для активних елементів, кнопок та полів фокусу. Це створює сучасний вигляд інтерфейсу, притаманний спортивним та технологічним застосункам. Для забезпечення гнучкості та адаптивності макетів використано технологію Flexbox [11], що гарантує коректне відображення інформаційних карток із вправами та навігаційних панелей на екранах із різною роздільною здатністю.

Динамічна взаємодія з користувачем забезпечується мовою програмування JavaScript [10]. Клієнтська логіка розділена на два основні рівні. Перший рівень реалізовано у вигляді сервісного модуля взаємодії з програмним інтерфейсом сервера. Цей модуль інкапсулює складність виконання мережевих операцій через Fetch API [10] та обробляє відповіді у форматі JSON. Другий рівень логіки відповідає за безпосередню маніпуляцію елементами об’єктної моделі документа, обробку подій натискання кнопок та валідацію форм перед відправкою даних.

Важливим аспектом клієнтської розробки є реалізація механізму захисту приватних сторінок. На стороні фронтенду впроваджено спеціалізований перевіряльник, який аналізує наявність ідентифікатора сесії у локальному сховищі браузера перед завантаженням вмісту особистого кабінету або генератора тренувань. У разі відсутності авторизації система автоматично перенаправляє користувача на сторінку входу. Такий підхід забезпечує додатковий рівень безпеки та покращує користувацький досвід.

Особлива увага приділена інтерактивним елементам форми профілю. Реалізовано функціонал динамічного перемикання між режимом перегляду та режимом редагування медичних обмежень. Це дозволяє користувачеві бачити поточний стан здоров’я у текстовому вигляді та відкривати список чекбоксів лише за потреби внесення змін. Опис програмної реалізації цих елементів та приклад роботи клієнтських скриптів наведено у додатку А.

Застосований комплекс технологій дозволив створити інтуїтивно зрозумілий та функціональний інтерфейс. Використання асинхронного обміну даними забезпечує миттєве оновлення інформації про згенеровані тренування без необхідності повного перезавантаження сторінок, що підвищує ефективність взаємодії користувача із системою. Візуальне представлення розробленого інтерфейсу сторінки генератора тренувань відображено на рисунку 3.4.

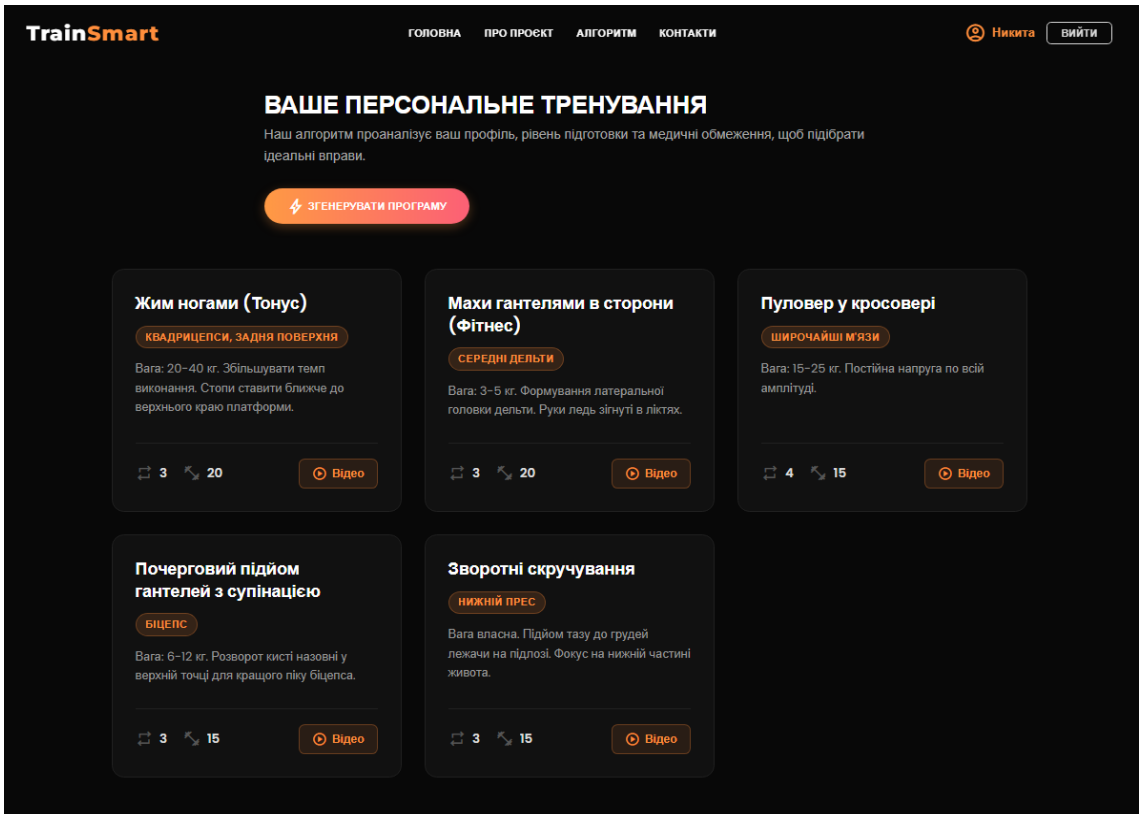


Рисунок 3.4 – Скриншот сторінки згенерованого тренування з переліком вправ

3.5 Забезпечення захисту даних та перевірка стабільності роботи вебзастосунку

Фінальним етапом розробки вебзастосунку є впровадження комплексних заходів із захисту персональної інформації та проведення серії тестів для підтвердження стабільності роботи системи за різних умов експлуатації. Оскільки програмний продукт оперує персональними даними користувачів та інформацією про стан їхнього здоров’я, питання безпеки та відмовостійкості є пріоритетними для гарантування надійності системи.

Захист облікових даних реалізовано на рівні серверної логіки із застосуванням сучасних методів криптографічного перетворення. Для збереження паролів використовується алгоритм хешування bcrypt із застосуванням солі [13], що унеможливлює відновлення оригінального тексту пароля навіть у разі отримання несанкціонованого доступу до бази даних. Процес перевірки автентичності під час входу в систему базується на порівнянні отриманого від клієнта хешу із записом у таблиці користувачів, що забезпечує високий рівень захисту від атак методом перебору. Програмна реалізація цих механізмів у контролері автентифікації наведена у додатку А.

Для захисту від найбільш розповсюджених вразливостей вебзастосунків, зокрема SQL-ін’єкцій, у системі застосовано метод параметризованих запитів [12]. Взаємодія з базою даних MySQL здійснюється виключно через підготовлені вирази, де вхідні дані від користувача передаються як окремі параметри, а не як частина тексту запиту. Це гарантує, що будь-який шкідливий код, переданий через форми введення, буде розцінюватися системою як звичайний текст і не зможе вплинути на структуру бази даних. Приклади використання параметризованих запитів у контролерах системи представлені у додатку А.

Стабільність роботи вебзастосунку забезпечується через багаторівневу систему обробки виняткових ситуацій. Серверна частина містить централізовані блоки перехоплення помилок, які запобігають аварійному завершенню роботи скриптів при некоректних запитах або збоях у мережевому з’єднанні. Для кожного типу помилки передбачено повернення відповідного HTTP-статусу та структурованого повідомлення у форматі JSON, що дозволяє клієнтській частині інформувати користувача про проблему без порушення цілісності інтерфейсу.

Перевірка загальної працездатності серверних маршрутів та успішної генерації даних була раніше підтверджена інструментарієм Postman [14] та відображена на рисунку 3.2. Тому на фінальному етапі тестування головна увага приділялася перевірці захисних механізмів та валідації. Зокрема, проведено тести на спробу реєстрації з уже існуючою електронною адресою та введення завідомо некоректних значень віку. Результати тестування підтвердили, що серверна логіка успішно відхиляє такі запити, повертаючи помилку із кодом 400 та відповідним текстовим попередженням. Візуальне відображення роботи механізму перехоплення помилок на стороні клієнта наведено на рисунку 3.5.

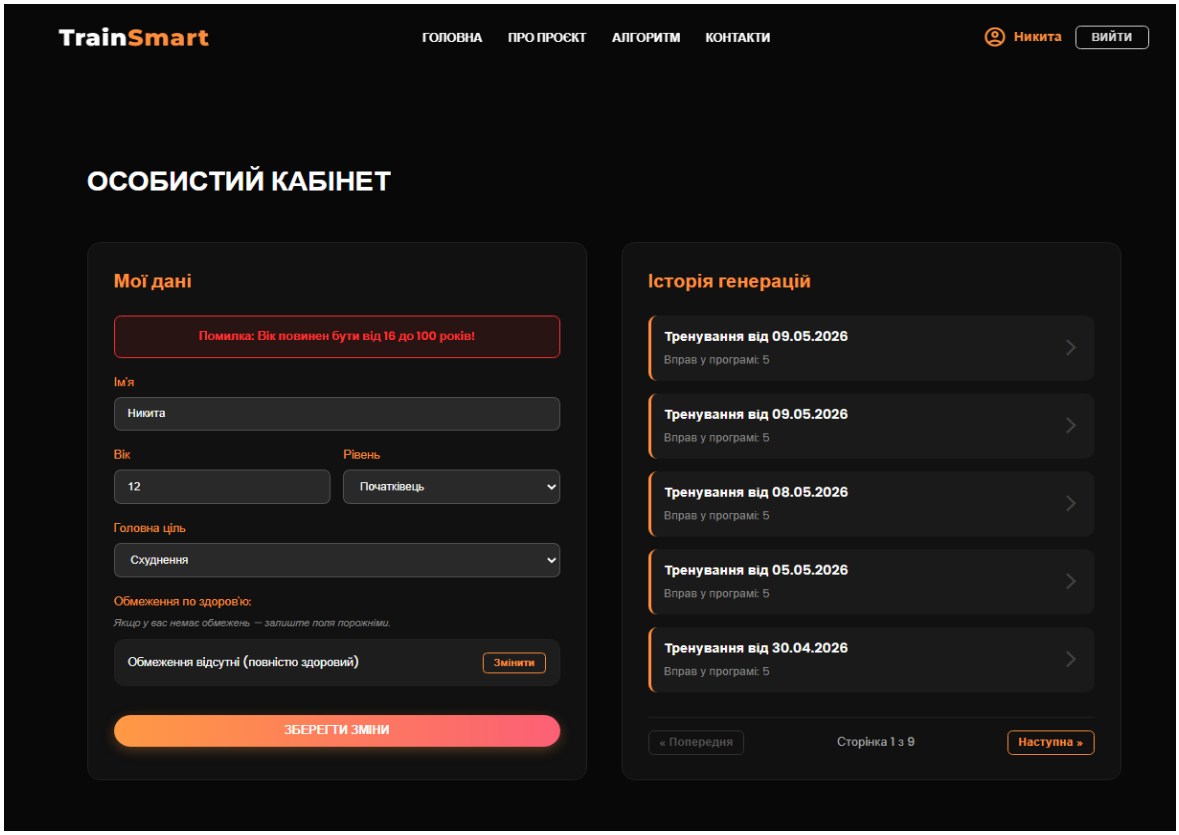


Рисунок 3.5 – Скриншот клієнтського інтерфейсу із демонстрацією обробки помилки валідації

Додатково перевірено роботу захисних механізмів маршрутизації на стороні фронтенду. Система успішно блокує доступ до сторінок профілю та генератора для неавторизованих відвідувачів, автоматично перенаправляючи їх на форму входу. Впроваджені заходи гарантують безпечне використання

вебзастосунку студентами та викладачами, забезпечують цілісність збережених тренувальних програм та підтверджують готовність системи до експлуатації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи вирішено актуальне науково-практичне завдання, яке полягає у проєктуванні та розробці сучасного вебзастосунку для формування персоналізованих програм тренувань з урахуванням індивідуальних медичних обмежень користувачів. Під час виконання дослідження досягнуто всіх поставлених цілей та реалізовано повний цикл створення програмного продукту від етапу збору початкових вимог до фінального тестування розроблених алгоритмів.

На початковому етапі роботи проведено детальний системний аналіз предметної області та здійснено огляд існуючих програмних рішень на ринку фітнес-послуг. За результатами дослідження встановлено, що переважна більшість комерційних застосунків пропонує стандартизовані плани тренувань і не враховує специфічні протипоказання, пов’язані зі станом здоров’я клієнтів. З огляду на виявлені недоліки сформовано технічні та функціональні вимоги до нової інформаційної системи, головною інноваційною особливістю якої стала інтелектуальна багатокритеріальна фільтрація фізичних навантажень.

Відповідно до визначених вимог спроєктовано архітектуру програмного продукту на базі клієнт-серверної моделі. Спроектовано комплекс діаграм, які детально описують прецеденти використання, логічну структуру та поведінку компонентів системи. Для цілісного зберігання інформації побудовано реляційну базу даних під управлінням системи MySQL. Створена структура бази даних забезпечує зберігання облікових записів, детальних антропометричних профілів, розширеного каталогу вправ із класифікацією медичних протипоказань та повної історії згенерованих тренувальних планів. Використання механізмів зовнішніх ключів гарантує високий рівень зв’язності та надійності даних.

Практичну реалізацію серверної частини вебзастосунку виконано з використанням середовища виконання Node.js та програмного каркасу Express.

Головним здобутком розробки є алгоритм генерації адаптивних програм тренувань. Даний програмний модуль автоматично аналізує вік, рівень фізичної підготовки, головну мету користувача та перелік його медичних обмежень. Завдяки застосуванню адаптивних запитів до бази даних алгоритм повністю виключає з результатів ті вправи, які можуть нашкодити здоров'ю клієнта. Після етапу фільтрації система формує безпечний та максимально ефективний план занять, оптимізований під індивідуальні потреби.

Клієнтську частину системи розроблено з використанням комплексу вебтехнологій, що включає мови гіпертекстової розмітки, каскадні таблиці стилів, сучасний стандарт JavaScript та препроцесор гіпертексту PHP. Створено інтуїтивно зрозумілий адаптивний інтерфейс користувача, який забезпечує зручну взаємодію на пристроях із різною роздільною здатністю екрана. Обмін даними між клієнтом та сервером реалізовано на основі архітектурного стилю програмного інтерфейсу за допомогою асинхронних мережевих запитів. Це дозволило реалізувати оновлення інформації на сторінках вебзастосунку без їхнього повного перезавантаження, що підвищило загальну продуктивність програмного продукту.

Особливу увагу під час розробки приділено питанням інформаційної безпеки та стабільності роботи системи. Упроваджено надійні механізми криптографічного захисту паролів шляхом їхнього алгоритмічного хешування перед збереженням. Для захисту бази даних від несанкціонованого доступу та атак впровадження SQL-коду застосовано метод параметризованих запитів. Проведене тестування функціональних модулів за допомогою спеціалізованого програмного забезпечення підтвердило відмовостійкість розробленого вебзастосунку та коректність роботи захисних механізмів валідації вхідних даних.

Таким чином, створений програмний продукт повністю відповідає технічному завданню та сучасним стандартам веброзробки. Практична цінність одержаних результатів полягає у можливості впровадження розробленої системи у щоденний тренувальний процес для автоматизації роботи фітнес-інструкторів

або для безпечного самостійного використання особами, які бажають покращити фізичну форму з урахуванням наявних обмежень здоров’я.

Перспективи подальшого розвитку проєкту включають перенесення клієнтської логіки у формат повноцінного мобільного додатка для операційних систем смартфонів. Додатково планується інтеграція вебзастосунку із портативними пристроями для зчитування пульсу та впровадження алгоритмів штучного інтелекту для більш точної адаптації фізичних навантажень на основі глибокого аналізу динаміки прогресу користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Глобальні рекомендації щодо фізичної активності для здоров'я. Всесвітня організація охорони здоров'я. URL : https://www.who.int/publications/i/item/9789240015128 (дата звернення : 05.03.2026).

2. Sommerville I. Software Engineering. 10th ed. London : Pearson, 2015. 810 p.

3. Fowler M. UML Distilled : A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.

4. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : Ph.D. dissertation. Irvine : University of California, 2000. 162 p.

5. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2019. P. 44–48, 143–144, 236–242.

6. MySQL 8.0 Reference Manual. Oracle Corporation. URL : https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення : 15.04.2026).

7. Node.js Documentation. OpenJS Foundation. URL : https://nodejs.org/en/docs/ (дата звернення : 20.04.2026).

8. Express - Node.js web application framework. OpenJS Foundation. URL : https://expressjs.com/ (дата звернення : 25.04.2026).

9. PHP : Hypertext Preprocessor Manual. The PHP Group. URL : https://www.php.net/manual/en/ (дата звернення : 28.04.2026).

10. Flanagan D. JavaScript : The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.

11. MDN Web Docs : HTML & CSS. Mozilla. URL : https://developer.mozilla.org/en-US/ (дата звернення : 30.04.2026).

12. OWASP Top 10:2021. The Open Web Application Security Project. URL : https://owasp.org/Top10/ (дата звернення : 1.05.2026).

13. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Hoboken : Pearson, 2020. P. 502–508.

14. Postman Learning Center: API Testing. Postman. URL : https://learning.postman.com/docs/writing-scripts/test-scripts/ (дата звернення : 03.05.2026).

ДОДАТОК А

Фрагменти вихідного коду програмного продукту

А.1 – Головний файл ініціалізації сервера та маршрутизації (server.js)

```
      const express = require('express');
      const cors = require('cors');

      require('dotenv').config();

      const authController = require('./src/controllers/AuthController');
      const workoutController =
require('./src/controllers/workoutController');
      const profileController =
require('./src/controllers/profileController');

      const app = express();

      app.use(cors());
      app.use(express.json());

      app.use((req, res, next) => {
        console.log(`[${new Date().toLocaleString()}] ${req.method} запит
на ${req.url}`);
        next();
      });

      app.use('/uploads', express.static('public/uploads'));

      app.post('/api/auth/register', authController.register);
      app.post('/api/auth/login', authController.login);

      app.get('/api/workout/generate/:userId',
workoutController.generateProgram);

      app.get('/api/workout/latest/:userId',
workoutController.getLatestWorkout);
      app.get('/api/workout/details/:workoutId',
workoutController.getWorkoutDetails);

      app.get('/api/profile/:userId', profileController.getProfile);
      app.put('/api/profile/:userId', profileController.updateProfile);
      app.get('/api/profile/:userId/history',
profileController.getHistory);

      app.use((req, res) => {
```

```
        res.status(404).json({ error: 'Запитуваний ресурс не знайдено' });
      });

      app.use((err, req, res, next) => {
        console.error('Помилка сервера:', err.stack);
        res.status(500).json({ error: 'Сталася внутрішня помилка сервера'
});
      });

      const PORT = 3000;
      app.listen(PORT, () => {
        console.log(`Сервер TrainSmart успішно запущений на порту
${PORT}`);
        console.log(`Очікування API-запитів...`);
});
```

А.2 – Модуль конфігурації бази даних (db.js)

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'trainsmart_db',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
});

module.exports = pool;
```

А.3 – Контролер алгоритму генерації персональних тренувань (workoutController.js)

```
const db = require('../config/db');

exports.generateProgram = async (req, res) => {
  const userId = req.params.userId;

  try {
    const [profiles] = await db.execute(
      'SELECT goal, fitness_level, health_restrictions FROM user_profiles
WHERE user_id = ?',
      [userId]
```

```
        );

        if (profiles.length === 0) {
          return res.status(404).json({ error: 'Профіль користувача не
знайдено. Будь ласка, заповніть дані.' });
        }

        const userProfile = profiles[0];
        const userRestriction = userProfile.health_restrictions;

        const userAge = parseInt(userProfile.age);

        let query = `
          SELECT id, title, description, sets, reps, target_muscle, video_url
          FROM exercises
          WHERE target_goal = ?
          AND required_level = ?
          AND max_age >= ?
        `;

        const queryParams = [userProfile.goal, userProfile.fitness_level,
userAge];

        if (userRestriction && userRestriction !== 'none' &&
userRestriction.trim() !== '') {
          const restrictionsArray = userRestriction.split(',');

          for (let restriction of restrictionsArray) {
            query += ` AND (contraindications IS NULL OR contraindications NOT
LIKE ?)`;
            queryParams.push(`%${restriction.trim()}%`);
          }
        }

        let exerciseCount = 5;

        if (userProfile.fitness_level === 'beginner') {
          exerciseCount -= 1;
        } else if (userProfile.fitness_level === 'advanced') {
          exerciseCount += 1;
        }

        if (userProfile.goal === 'weight_loss') {
          exerciseCount += 1;
        } else if (userProfile.goal === 'muscle_gain') {
          exerciseCount -= 1;
        }

        if (userAge >= 50) {
          exerciseCount -= 1;
        } else if (userAge >= 18 && userAge <= 35) {
          exerciseCount += 1;
        }

        exerciseCount = Math.max(4, Math.min(exerciseCount, 7));
```

```
          query += ` ORDER BY RAND() LIMIT ${exerciseCount}`;

          const [exercises] = await db.execute(query, queryParams);

          if (exercises.length === 0) {
            return res.status(404).json({ error: 'На жаль, не знайдено вправ для
ваших параметрів.' });
          }

          // ========================================
          // ЗБЕРЕЖЕННЯ ІСТОРІЇ ТРЕНУВАНЬ
          // ========================================
          const [progResult] = await db.execute(
            'INSERT INTO training_programs (user_id) VALUES (?)',
            [userId]
          );
          const programId = progResult.insertId;

          for (const ex of exercises) {
            await db.execute(
              'INSERT INTO program_exercises (program_id, exercise_id, sets,
reps) VALUES (?, ?, ?, ?)',
              [programId, ex.id, ex.sets, ex.reps]
            );
          }

          res.status(200).json({
              message: 'Програму успішно сформовано та збережено в історію!',
              userGoal: userProfile.goal,
              workout: exercises
          });

        } catch (error) {
          console.error('Помилка в алгоритмі генерації:', error);
          res.status(500).json({ error: 'Сталася внутрішня помилка сервера при
підборі вправ' });
        }
      };

      // ========================================
      // ОТРИМАННЯ ОСТАННЬОГО ТРЕНУВАННЯ
      // ========================================
      exports.getLatestWorkout = async (req, res) => {
        const userId = req.params.userId;

        try {
          const [programs] = await db.execute(
            `SELECT id, created_at
             FROM training_programs
             WHERE user_id = ? AND created_at >= NOW() - INTERVAL 12 HOUR
             ORDER BY created_at DESC
             LIMIT 1`,
            [userId]
          );

          if (programs.length === 0) {
```

```
            return res.status(200).json({ workout: null, message: 'Немає
активних тренувань.' });
          }

          const programId = programs[0].id;

          const [exercises] = await db.execute(
            `SELECT e.id, e.title, e.description, pe.sets, pe.reps,
e.target_muscle, e.video_url
             FROM program_exercises pe
             JOIN exercises e ON pe.exercise_id = e.id
             WHERE pe.program_id = ?`,
            [programId]
          );

          res.status(200).json({ workout: exercises });

        } catch (error) {
          console.error('Помилка отримання останнього тренування:', error);
          res.status(500).json({ error: 'Помилка сервера при завантаженні
тренування' });
        }
      };

      // ==========================================
      // ОТРИМАННЯ ДЕТАЛЕЙ БУДЬ-ЯКОГО ТРЕНУВАННЯ З ІСТОРІЇ
      // ==========================================
      exports.getWorkoutDetails = async (req, res) => {
        const workoutId = req.params.workoutId;

        try {
          const [exercises] = await db.execute(
            `SELECT e.id, e.title, e.description, pe.sets, pe.reps,
e.target_muscle, e.video_url
             FROM program_exercises pe
             JOIN exercises e ON pe.exercise_id = e.id
             WHERE pe.program_id = ?`,
            [workoutId]
          );

          if (exercises.length === 0) {
            return res.status(404).json({ error: 'Тренування не знайдено або
воно порожнє.' });
          }

          res.status(200).json({ workout: exercises });

        } catch (error) {
          console.error('Помилка отримання деталей тренування:', error);
          res.status(500).json({ error: 'Помилка сервера при завантаженні
історії' });
        }
      };
```

А.4 – Контролер автентифікації користувачів (AuthController.js)

```
const db = require('../config/db');
const bcrypt = require('bcryptjs');

exports.register = async (req, res) => {
  const { name, age, goal, fitness_level, health_restrictions, email,
password } = req.body;
  const connection = await db.getConnection();

  if (!name || /\s/.test(name.trim())) {
    return res.status(400).json({ error: "Ім'я повинно складатися з одного
слова без пробілів" });
  }

  try {
    await connection.beginTransaction();

    const [existingUser] = await connection.execute(
      'SELECT id FROM users WHERE email = ?',
      [email]
    );

    if (existingUser.length > 0) {
      return res.status(400).json({ error: 'Користувач з такою поштою вже
зареєстрований' });
    }

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const [userResult] = await connection.execute(
      'INSERT INTO users (email, password_hash) VALUES (?, ?)',
      [email, hashedPassword]
    );
    const userId = userResult.insertId;

    await connection.execute(
      `INSERT INTO user_profiles (user_id, name, age, goal, fitness_level,
health_restrictions)
       VALUES (?, ?, ?, ?, ?, ?)`,
      [userId, name, age, goal, fitness_level, health_restrictions ||
null]
    );

    await connection.commit();
    res.status(201).json({ message: 'Реєстрація успішна' });

  } catch (error) {
    await connection.rollback();
    console.error('Помилка реєстрації:', error);
    res.status(500).json({ error: 'Сталася внутрішня помилка сервера' });
  } finally {
    connection.release();
```

```
  }
};

exports.login = async (req, res) => {
  const { email, password } = req.body;

  try {
    const [users] = await db.execute(`
        SELECT u.id, u.password_hash, p.name
        FROM users u
        LEFT JOIN user_profiles p ON u.id = p.user_id
        WHERE u.email = ?
    `, [email]);

    if (users.length === 0) {
      return res.status(401).json({ error: 'Користувача з таким email не
знайдено' });
    }

    const user = users[0];

    const isMatch = await bcrypt.compare(password, user.password_hash);

    if (!isMatch) {
      return res.status(401).json({ error: 'Невірний пароль' });
    }

    res.status(200).json({
      message: 'Вхід успішний',
      userId: user.id,
      name: user.name
    });

  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Помилка сервера' });
  }
};
```

А.5 – Сервісний модуль клієнтської взаємодії з API (api.js)

```
const BASE_URL = 'http://localhost:3000/api';

export const api = {
    async _request(url, options) {
        try {
            options = options || {};

            const token = localStorage.getItem('trainsmart_token');

            if (token) {
```

```
                    options.headers = {
                        ...options.headers,
                        'Authorization': `Bearer ${token}`
                    };
                }

                const response = await fetch(url, options);
                const result = await response.json();
                return { ok: response.ok, result };
            } catch (error) {
                console.error('Network Error:', error);
                return { ok: false, result: { error: 'Не вдалося встановити
зв'язок із сервером' } };
            }
        },

        // ==========================================
        // АВТЕНТИФІКАЦІЯ
        // ==========================================
        async register(userData) {
            return await this._request(`${BASE_URL}/auth/register`, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(userData)
            });
        },

        async login(userData) {
            return await this._request(`${BASE_URL}/auth/login`, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(userData)
            });
        },

        // ==========================================
        // ГЕНЕРАТОР ТРЕНУВАНЬ
        // ==========================================
        async generateWorkout(userId) {
            return await
this._request(`${BASE_URL}/workout/generate/${userId}`, {
                method: 'GET',
                headers: { 'Content-Type': 'application/json' }
            });
        },

        // ==========================================
        // ОСОБИСТИЙ КАБІНЕТ
        // ==========================================

        async getProfile(userId) {
            return await this._request(`${BASE_URL}/profile/${userId}`, {
                method: 'GET',
                headers: { 'Content-Type': 'application/json' }
            });
        },
```

```
        async getHistory(userId) {
            return await
this._request(`${BASE_URL}/profile/${userId}/history`, {
                method: 'GET',
                headers: { 'Content-Type': 'application/json' }
            });
        },

        async updateProfile(userId, profileData) {
            return await this._request(`${BASE_URL}/profile/${userId}`, {
                method: 'PUT',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(profileData)
            });
        },

        async getLatestWorkout(userId) {
            return await this._request(`${BASE_URL}/workout/latest/${userId}`,
{
                method: 'GET',
                headers: { 'Content-Type': 'application/json' }
            });
        },

        async getWorkoutDetails(workoutId) {
            return await
this._request(`${BASE_URL}/workout/details/${workoutId}`, {
                method: 'GET',
                headers: { 'Content-Type': 'application/json' }
            });
        }
    };
```

А.6 – Фрагмент головного клієнтського скрипта інтерфейсу (app.js)

```
import { api } from './api.js';

document.addEventListener('DOMContentLoaded', () => {

    // ========================================
    // ЗАХИСТ МАРШРУТІВ
    // ========================================
    (function () {
        const path = window.location.pathname;
        const userId = localStorage.getItem('trainsmart_user_id');

        const privatePages = [
            '/fitness_app/views/profile.php',
            '/fitness_app/views/generator.php'
        ];
```

```
        if (privatePages.includes(path) && !userId) {
            window.location.href = '/fitness_app/views/auth/login.php';
        }
    })();

    // ========================================
    // ОБРОБКА ВІДПРАВКИ ФОРМИ ПРОФІЛЮ ТА ВАЛІДАЦІЯ
    // ========================================
    const profileForm = document.getElementById('profileForm');

    if (profileForm) {
        profileForm.addEventListener('submit', async (e) => {
            e.preventDefault();

            const msg = document.getElementById('profileMessage');
            const newName = document.getElementById('prof_name').value.trim();
            const profAge = parseInt(document.getElementById('prof_age').value);
            const userId = localStorage.getItem('trainsmart_user_id');

            if (/\s/.test(newName)) {
                msg.textContent = "Помилка: Ім'я повинно складатися лише з
одного слова!";
                msg.className = "custom-alert error";
                msg.style.display = 'block';
                return;
            }

            if (isNaN(profAge) || profAge < 16 || profAge > 100) {
                msg.textContent = "Помилка: Вік повинен бути від 16 до 100
років!";
                msg.className = "custom-alert error";
                msg.style.display = 'block';
                return;
            }

            const selectedCbs = Array.from(document.querySelectorAll('.health-
cb:checked'));
            const healthRestrictionsValue = selectedCbs.length > 0
                ? selectedCbs.map(cb => cb.value).join(',')
                : 'none';

            const updateData = {
                name: newName,
                age: profAge,
                fitness_level: document.getElementById('prof_level').value,
                goal: document.getElementById('prof_goal').value,
                health_restrictions: healthRestrictionsValue
            };

            try {
                const { ok, result } = await api.updateProfile(userId,
updateData);

                if (ok) {
                    msg.textContent = "Дані успішно збережено!";
```

```
                    msg.className = "custom-alert success";
                    msg.style.display = 'block';

                    localStorage.setItem('trainsmart_user_name', newName);

                    const displaySpan =
document.getElementById('current_health_text');
                    if (displaySpan) {
                        displaySpan.textContent = selectedCbs.length > 0
                            ? selectedCbs.map(cb => cb.getAttribute('data-
label')).join(', ')
                            : "Обмеження відсутні (повністю здоровий)";
                    }
                } else {
                    msg.textContent = result.error || "Помилка при збереженні!";
                    msg.className = "custom-alert error";
                    msg.style.display = 'block';
                }
            } catch (error) {
                console.error('Update error:', error);
            }

            setTimeout(() => {
                msg.style.display = 'none';
                msg.className = "custom-alert";
            }, 3000);
        });
    }
});
```

А.7 – Контролер управління профілем користувача (profileController.js)

```
const db = require('../config/db');

exports.getProfile = async (req, res) => {
    try {
        const [rows] = await db.execute(
            'SELECT u.email, p.name, p.age, p.fitness_level, p.goal,
p.health_restrictions FROM users u JOIN user_profiles p ON u.id = p.user_id
WHERE u.id = ?',
            [req.params.userId]
        );
        if (rows.length === 0) return res.status(404).json({ error:
'Профіль не знайдено' });
        res.status(200).json(rows[0]);
    } catch (error) {
        res.status(500).json({ error: 'Помилка отримання профілю' });
    }
};
```

```
    exports.updateProfile = async (req, res) => {
        const { name, age, fitness_level, goal, health_restrictions } =
req.body;
        const userId = req.params.userId;
        const parsedAge = parseInt(age);
        if (isNaN(parsedAge) || parsedAge < 16 || parsedAge > 100) {
            return res.status(400).json({ error: "Вказано некоректний вік
(дозволено від 16 до 100 років)." });
        }

        try {
            await db.execute(
                'UPDATE user_profiles SET name = ?, age = ?, fitness_level =
?, goal = ?, health_restrictions = ? WHERE user_id = ?',
                [name, age, fitness_level, goal, health_restrictions || null,
req.params.userId]
            );
            res.status(200).json({ message: 'Дані успішно оновлено' });
        } catch (error) {
            res.status(500).json({ error: 'Помилка оновлення даних' });
        }
    };

    exports.getHistory = async (req, res) => {
        try {
            const [history] = await db.execute(
                `SELECT tp.id, tp.created_at, COUNT(pe.id) as ex_count
                 FROM training_programs tp
                 LEFT JOIN program_exercises pe ON tp.id = pe.program_id
                 WHERE tp.user_id = ?
                 GROUP BY tp.id
                 ORDER BY tp.created_at DESC`,
                [req.params.userId]
            );
            res.status(200).json(history);
        } catch (error) {
            res.status(500).json({ error: 'Помилка отримання історії' });
                                    }};
```

ДОДАТОК Б

SQL-скрипт ініціалізації бази даних trainsmart_db

Б.1 – SQL-запит ініціалізації бази даних та таблиці облікових записів користувачів

```
CREATE DATABASE IF NOT EXISTS `trainsmart_db`
CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

USE `trainsmart_db`;

CREATE TABLE `users` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(150) NOT NULL UNIQUE,
  `password_hash` VARCHAR(255) NOT NULL,
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Б.2 – SQL-запит створення таблиці персональних профілів користувачів

```
CREATE TABLE `user_profiles` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `user_id` INT(11) NOT NULL,
  `name` VARCHAR(100) NOT NULL,
  `age` INT(3) NOT NULL,
  `fitness_level` ENUM('beginner', 'intermediate', 'advanced') NOT NULL
DEFAULT 'beginner',
  `goal` ENUM('weight_loss', 'muscle_gain', 'maintenance') NOT NULL,
  `health_restrictions` VARCHAR(255) DEFAULT NULL,
  `updated_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Б.3 – SQL-запит створення довідника фізичних вправ

```
CREATE TABLE `exercises` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(150) NOT NULL,
  `description` TEXT NOT NULL,
  `target_goal` ENUM('weight_loss', 'muscle_gain', 'maintenance') NOT NULL,
  `required_level` ENUM('beginner', 'intermediate', 'advanced') NOT NULL,
  `video_url` VARCHAR(255) DEFAULT NULL,
  `sets` INT(2) NOT NULL,
  `reps` VARCHAR(50) NOT NULL,
  `target_muscle` VARCHAR(50) NOT NULL,
  `contraindications` VARCHAR(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Б.4 – SQL-запит створення таблиці заголовків тренувальних програм

```
CREATE TABLE `training_programs` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `user_id` INT(11) NOT NULL,
  `title` VARCHAR(100) NOT NULL DEFAULT 'Персональна програма',
  `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`user_id`) REFERENCES `users`(`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Б.5 – SQL-запит створення таблиці складу тренувальних програм

```
CREATE TABLE `program_exercises` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `program_id` INT(11) NOT NULL,
  `exercise_id` INT(11) NOT NULL,
  `sets` INT(2) NOT NULL,
  `reps` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`program_id`) REFERENCES `training_programs`(`id`) ON DELETE
CASCADE,
  FOREIGN KEY (`exercise_id`) REFERENCES `exercises`(`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```