Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ НАВАНТАЖЕННЯ ТА ПРОГРЕСУ У ТРЕНУВАННЯХ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-3
Збітнєв Д.С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Збітнєву Даниїлу Станіславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення застосунку для відстеження навантаження та прогресу у тренуваннях

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 23 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література з питань розроблення веборієнтованих інформаційних систем, мікросервісної архітектури, технологій моніторингу тренувального процесу та аналізу спортивних показників; матеріали наукових конференцій; ресурси мережі Інтернет; офіційна документація React, TypeScript, ASP.NET Core, Entity Framework Core, SQL Server, Docker, SignalR, gRPC та OpenTelemetry; документація щодо використання великих мовних моделей (LLM) для реалізації функцій інтелектуального персонального тренера.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних вебзастосунків для відстеження тренувань та фізичної активності.
2. Дослідження методів збору, зберігання та обробки даних про тренування користувачів.
3. Проєктування архітектури та структури вебзастосунку.
4. Розроблення клієнтської та серверної частин застосунку.
5. Реалізація функцій аналізу тренувань та інтеграція інтелектуального помічника на основі LLM.
Тестування та оцінка працездатності розробленого програмного продукту.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Схема архітектури вебзастосунку, діаграма бази даних, інтерфейси користувача, діаграма взаємодії компонентів системи, результати тестування програмного продукту.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	15.09.25-30.09.25	
3	Аналіз літератури з проблеми, що вирішується	01.10.25-31.10.25	
4	Аналіз існуючих підходів до моніторингу ефективності тренувань	01.11.25-30.11.25	
5	Формування кроків вирішення проблеми	01.12.25-31.12.25	
6	Програмна реалізація	01.01.26-01.04.26	
7	Аналіз отриманих результатів	02.04.26-01.05.26	
8	Оформлення пояснювальної записки	02.05.26-01.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ ____доц. Тітова О. В.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 72 с., 35 рис., 1 дод., 32 джерела.

ASP.NET CORE WEB API, REACT, TYPESCRIPT, SQL SERVER, DOCKER, REST API, МІКРОСЕРВІСИ, SIGNALR, GRPC, ENTITY FRAMEWORK CORE, ВЕБЗАСТОСУНОК, ТРЕКІНГ ТРЕНУВАНЬ, ПЕРСОНАЛЬНИЙ ТРЕНЕР, ЧАТ-БОТ.

Об’єктом роботи є процес моніторингу та аналізу фізичної активності користувача під час тренувань.

Метою роботи є підвищення ефективності тренувального процесу шляхом розробки програмного застосунку для збору, аналізу та оцінювання показників фізичної активності користувача.

Під час роботи використано методи веброзробки, проєктування мікросервісних систем та технології зберігання й обробки даних.

Практична цінність вебзастосунку полягає в автоматизації ведення тренувальної статистики та аналізу результатів користувача.

Розроблено вебзастосунок, що забезпечує керування тренуваннями, відстеження прогресу та взаємодію з персональним тренером.

Область застосування: фітнес, спортивні клуби, персональні тренування, самостійний контроль фізичної активності.

ASP.NET CORE WEB API, REACT, TYPESCRIPT, SQL SERVER, DOCKER, REST API, MICROSERVICES, SIGNALR, GRPC, ENTITY FRAMEWORK CORE, WEB APPLICATION, TRAINING TRACKING, PERSONAL TRAINER, CHATBOT.

The subject of this study is the process of monitoring and analysing a user’s physical activity during training sessions.

The aim of the study is to improve the effectiveness of the training process by developing a software application to collect, analyse and evaluate a user’s physical activity metrics.

The work utilised web development methods, microservices architecture, and data storage and processing technologies.

The practical value of the web application lies in automating workout tracking and performance analysis.

A web application has been developed that enables workout management, progress tracking, and interaction with a personal trainer.

Applications: fitness, sports clubs, personal training, and individual physical activity monitoring.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface, програмний інтерфейс застосунку

ASP.NET Core Web API – фреймворк для створення вебсервісів та REST API на платформі .NET

CRUD – Create, Read, Update, Delete, базові операції роботи з даними

Docker – платформа для контейнеризації та розгортання застосунків

BFF – Backend for Frontend, архітектурний підхід для створення окремого серверного шару взаємодії між фронтендом і мікросервісами

gRPC – високопродуктивний фреймворк віддаленого виклику процедур

HTTP – HyperText Transfer Protocol, протокол передачі гіпертексту

Identity – система автентифікації та авторизації користувачів у ASP.NET Core

JSON – JavaScript Object Notation, текстовий формат обміну даними

OData – Open Data Protocol, протокол для роботи з даними через REST API

OpenTelemetry – набір інструментів для моніторингу та трасування застосунків

React – JavaScript-бібліотека для створення користувацьких інтерфейсів

REST API – архітектурний стиль взаємодії між компонентами вебсистем

SignalR – бібліотека для забезпечення обміну даними в реальному часі

SQL Server – система керування реляційними базами даних

UI – User Interface, користувацький інтерфейс

XUnit – фреймворк для модульного тестування застосунків на платформі .NET

YARP – Yet Another Reverse Proxy, бібліотека для реалізації проксі-сервера у .NET

ВСТУП

Інформаційні технології активно застосовуються у сфері спорту та фітнесу для автоматизації процесів збору, зберігання та аналізу даних про фізичну активність користувачів. Одним із важливих напрямів є розробка систем трекінгу тренувань, які дозволяють фіксувати результати занять, контролювати фізичні навантаження та відстежувати прогрес користувача.

Основним завданням систем трекінгу тренувань є накопичення та обробка інформації про фізичну активність з метою подальшого аналізу отриманих результатів. Такі системи дають можливість вести облік тренувань, встановлювати спортивні цілі, формувати статистику та візуалізувати показники прогресу. Сучасні рішення також використовують технології аналітики даних та інтелектуальних помічників для надання персоналізованих рекомендацій користувачам.

Системи моніторингу фізичної активності знаходять широке застосування у спортивних клубах, фітнес-центрах та серед користувачів, які самостійно контролюють власний тренувальний процес. Використання таких систем сприяє підвищенню ефективності тренувань, покращенню контролю за досягненням поставлених цілей та спрощенню аналізу фізичних показників.

Актуальність роботи полягає у необхідності розробки сучасного вебзастосунку для автоматизації трекінгу тренувань, аналізу фізичної активності та керування тренувальними даними користувача. Використання мікросервісної архітектури, засобів аналітики та AI-чатбота персонального тренера дозволяє створити масштабовану систему, яка забезпечує персоналізовану взаємодію з користувачем та підвищує ефективність тренувального процесу.

1 АНАЛІЗ ОСНОВНИХ ПІДХОДІВ ДО МОНІТОРИНГУ ЕФЕКТИВНОСТІ ТРЕНУВАНЬ ТА ПОРІВНЯННЯ ІСНУЮЧИХ ЗАСТОСУНКІВ

1.1 Огляд застосунків для моніторингу ефективності тренувань

У сучасному світі спостерігається стрімке зростання інтересу до здорового способу життя, фізичної активності та контролю власного фізичного стану. Це пов’язано як із популяризацією спорту, так і з підвищенням обізнаності людей щодо важливості регулярних тренувань для підтримки здоров’я. У зв’язку з цим значного розвитку набули інформаційні технології, спрямовані на підтримку та аналіз фізичної активності користувачів [1, 2].

Одним із ключових напрямків у цій сфері є створення мобільних та вебзастосунків для моніторингу ефективності тренувань. Такі застосунки дозволяють не лише фіксувати факт виконання фізичних вправ, але й аналізувати їх результати, оцінювати прогрес користувача, а також формувати рекомендації щодо подальших тренувань.

На сьогоднішній день існує велика кількість застосунків, які реалізують різні підходи до моніторингу фізичної активності. Деякі з них орієнтовані на відстеження базових параметрів тренувань, інші – на створення індивідуальних тренувальних програм, а також на комплексний аналіз стану користувача.

У цьому підрозділі буде детально розглянуто кілька популярних застосунків, призначених для моніторингу ефективності тренувань і відстеження фізичної активності користувачів. Окрім загального огляду, буде проведено аналіз їхніх функціональних можливостей, визначено ключові переваги та недоліки кожного рішення, а також оцінено, наскільки вони відповідають сучасним вимогам користувачів у сфері фітнесу, здорового способу життя та самоконтролю фізичної форми. Також буде звернено увагу на зручність інтерфейсу, точність відстеження даних і наявність додаткових функцій, таких як аналітика прогресу та персоналізовані рекомендації.

1.1.1 Strava

Strava є одним із найбільш відомих застосунків для відстеження фізичної активності, який набув широкого поширення серед користувачів, що займаються бігом, велоспортом та іншими видами аеробних тренувань.

Основною функціональною можливістю застосунку є використання технології GPS для відстеження маршруту користувача, а також визначення таких параметрів, як дистанція, швидкість руху, темп та тривалість тренування (рис. 1.1).

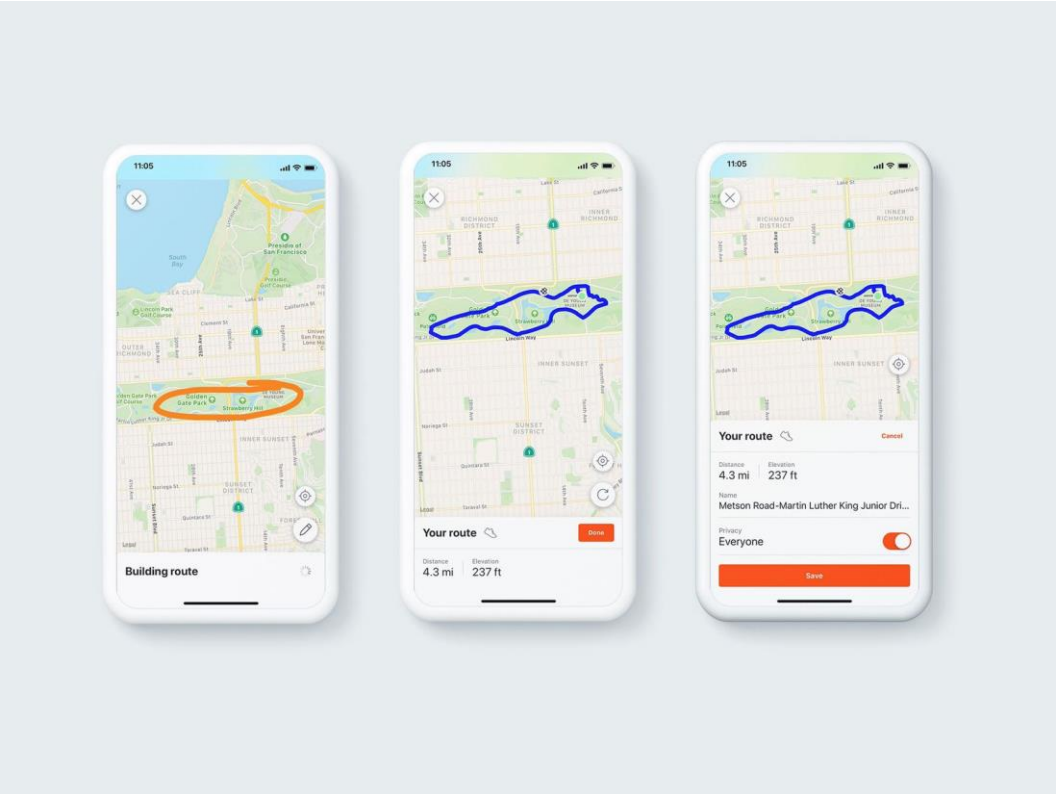

Рисунок 1.1 – Відображення маршруту тренування у застосунку Strava

Завдяки цьому користувач може отримати детальну інформацію про виконану фізичну активність.

Крім того, Strava надає можливість аналізувати історію тренувань, що дозволяє оцінити динаміку змін фізичної форми користувача (рис. 1.2).

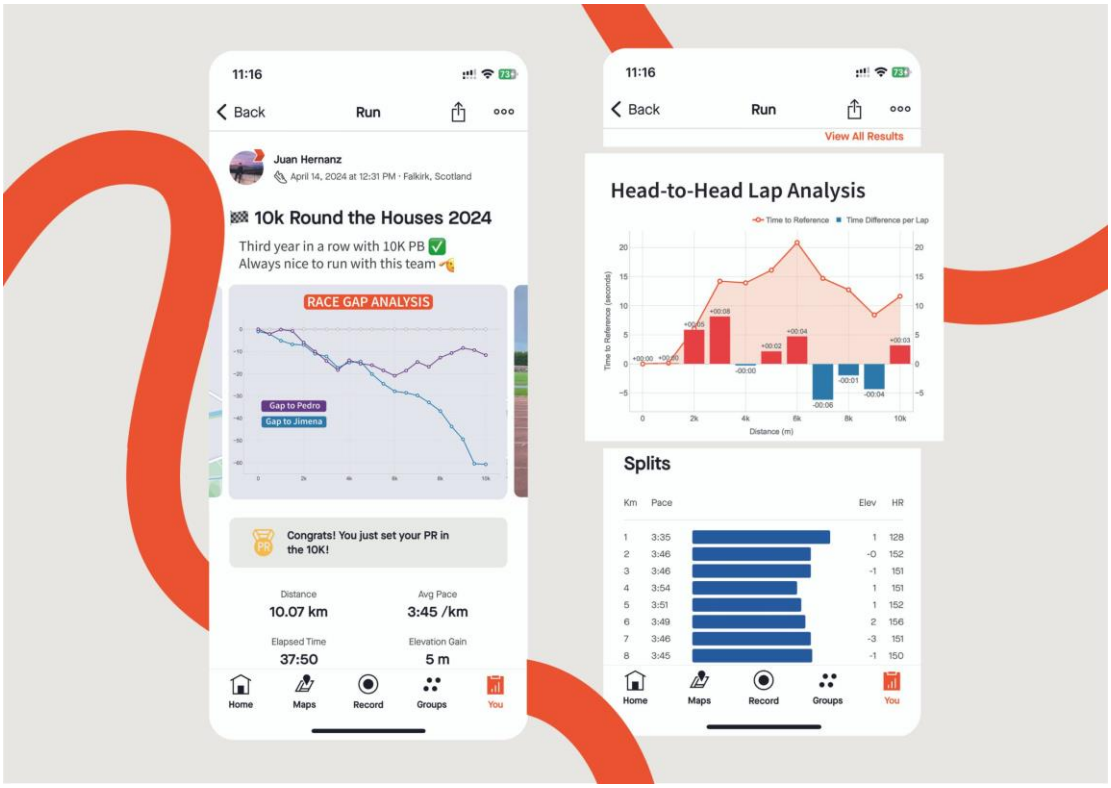

Рисунок 1.2 – Статистичні показники тренування у Strava

Важливою особливістю застосунку є наявність соціальної складової, яка дає змогу користувачам взаємодіяти між собою, порівнювати результати та брати участь у змаганнях.

До основних переваг застосунку можна віднести:

– високий рівень деталізації даних;

– зручний інтерфейс користувача;

– наявність мотиваційних механізмів;

– підтримку інтеграції з різними пристроями.

Водночас застосунок має і певні недоліки:

– частина функціоналу доступна лише за платною підпискою;

– обмежена універсальність щодо різних типів тренувань;

– відсутність глибокого аналізу ефективності тренувань як комплексного показника.

Таким чином, Strava є ефективним інструментом для відстеження фізичної активності, проте не забезпечує повноцінного аналізу ефективності тренувального процесу.

1.1.2 MyFitnessPal

MyFitnessPal є популярним застосунком, який орієнтований на комплексний контроль способу життя користувача, поєднуючи відстеження фізичної активності та харчування.

Основною функцією застосунку є ведення харчового щоденника, що дозволяє користувачам контролювати споживання калорій, білків, жирів та вуглеводів (рис. 1.3).

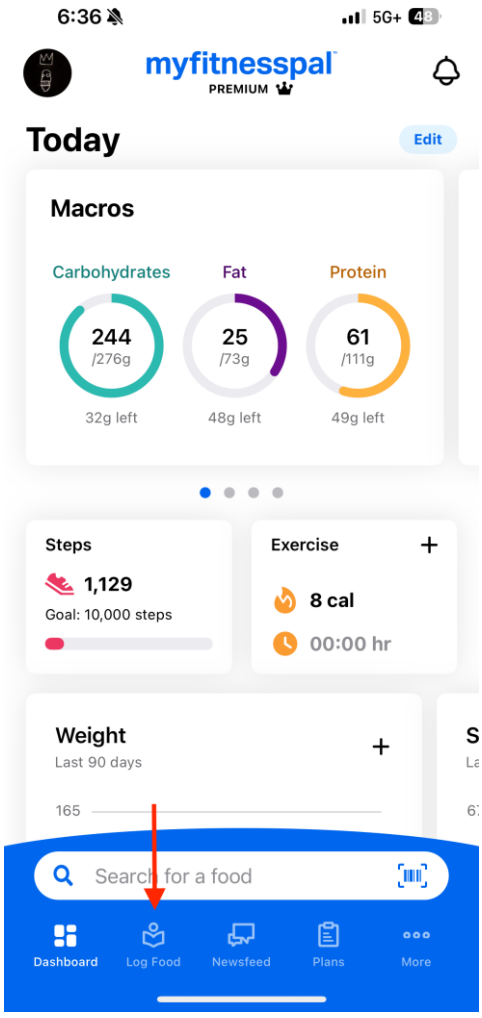

Рисунок 1.3 – Інтерфейс харчового щоденника у MyFitnessPal

Крім того, застосунок надає можливість фіксувати фізичну активність, що дозволяє оцінити загальний баланс енергії.

MyFitnessPal має велику базу даних продуктів харчування, що значно спрощує процес введення інформації (рис. 1.4).

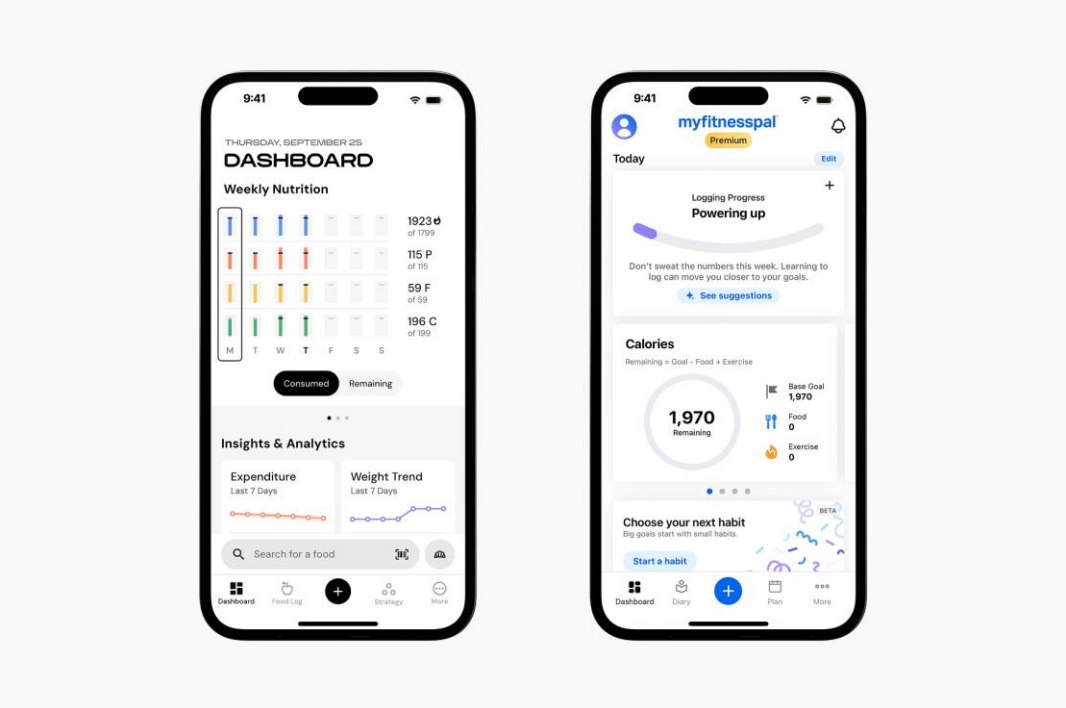

Рисунок 1.4 – Відображення макронутрієнтів у MyFitnessPal

Також застосунок підтримує інтеграцію з іншими сервісами та фітнес-пристроями, що дозволяє автоматизувати збір даних.

До переваг застосунку належать:

– широкий функціонал для контролю харчування;

– зручний інтерфейс;

– можливість інтеграції з іншими сервісами.

Серед недоліків варто відзначити:

– недостатній акцент на аналізі тренувань;

– відсутність глибокої оцінки ефективності фізичних навантажень;

– залежність від ручного введення даних.

Отже, MyFitnessPal більше орієнтований на контроль харчування, ніж на детальний аналіз ефективності тренувань. Основний акцент у застосунку

зроблено на підрахунку калорій, веденні харчового щоденника та контролі балансу поживних речовин, тоді як функції, пов’язані з тренуваннями, мають допоміжний характер і не забезпечують глибокої аналітики фізичної активності користувача.

1.1.3 Nike Training Club

Nike Training Club є застосунком, який орієнтований на надання користувачам готових тренувальних програм та інструкцій щодо виконання фізичних вправ.

Застосунок містить велику кількість тренувань різної складності, що дозволяє користувачам обирати програми відповідно до свого рівня підготовки (рис. 1.5).

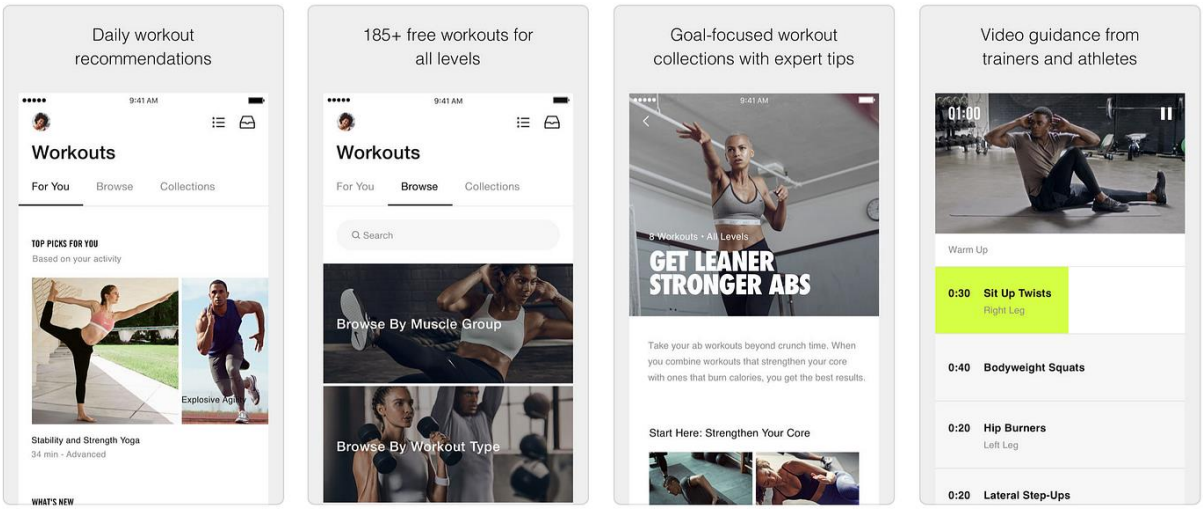

Рисунок 1.5 – Перелік тренувальних програм у Nike Training Club

Крім того, користувач отримує доступ до відеоінструкцій, які допомагають правильно виконувати вправи (рис. 1.6).

До переваг застосунку можна віднести:

– доступність великої кількості тренувальних програм;

– зручність використання;

– можливість персоналізації тренувань.

Водночас недоліками є:

– відсутність детального аналізу результатів;

– обмежені можливості моніторингу прогресу;

– відсутність комплексної оцінки ефективності тренувань.

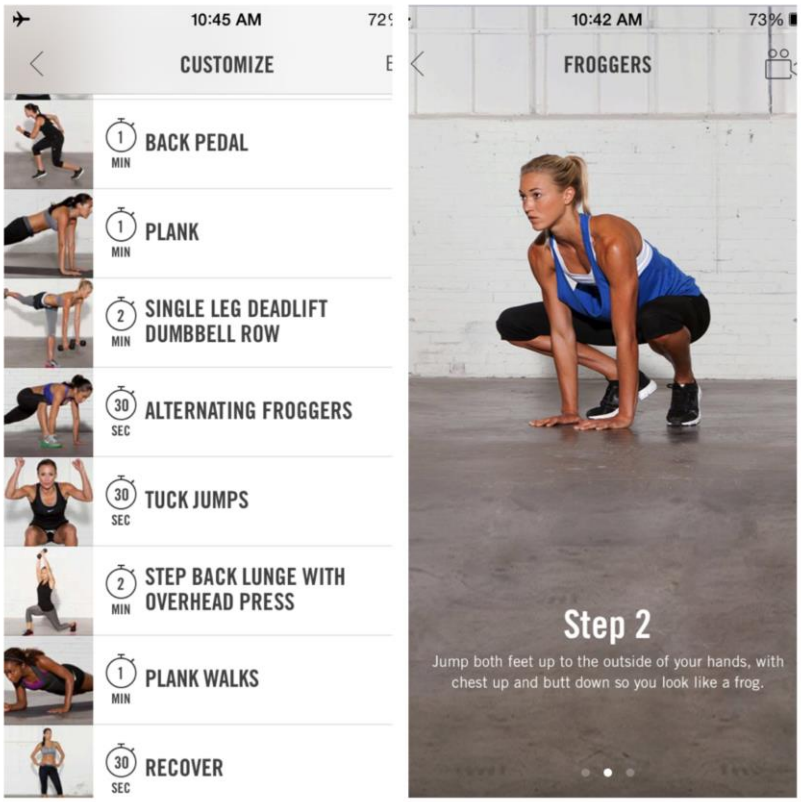

Рисунок 1.6 – Приклад виконання вправи у Nike Training Club

Таким чином, цей застосунок є досить зручним і практичним інструментом для виконання тренувань, оскільки забезпечує простий та зрозумілий процес користування. Водночас він не надає достатнього рівня аналітики та детального відстеження результатів, що могло б значно покращити ефективність тренувального процесу та допомогти користувачам глибше аналізувати свій прогрес.

1.2 Огляд основних підходів до моніторингу ефективності тренувань

Моніторинг ефективності тренувань є важливою складовою процесу фізичної підготовки, оскільки дозволяє оцінити результати виконаної роботи, визначити динаміку змін фізичного стану користувача та своєчасно коригувати тренувальний процес. У загальному випадку під ефективністю тренувань розуміють ступінь досягнення поставлених цілей, таких як підвищення витривалості, збільшення сили, зниження маси тіла або покращення загального фізичного стану.

З розвитком інформаційних технологій з’явилася можливість автоматизувати процес збору, обробки та аналізу даних, пов’язаних із фізичною активністю користувача. Сучасні застосунки використовують різні підходи до моніторингу ефективності тренувань, які відрізняються рівнем складності, точністю та обсягом оброблюваної інформації.

Основними підходами, що застосовуються у сучасних системах, є: відстеження базових показників, використання носимих пристроїв та застосування аналітичних методів обробки даних. Кожен із цих підходів має свої особливості, переваги та обмеження, що впливають на доцільність їх використання в конкретних умовах

1.2.1 Відстеження базових показників

Найпростішим та найбільш поширеним підходом до моніторингу ефективності тренувань є відстеження базових показників фізичної активності. До таких показників належать тривалість тренування, кількість виконаних вправ або повторень, пройдена дистанція, швидкість руху, а також орієнтовна кількість спалених калорій.

Цей підхід є основою для більшості початкових рівнів фітнес-застосунків, оскільки він не потребує складних обчислень або використання додаткового

обладнання. Дані можуть вводитися користувачем вручну або автоматично фіксуватися за допомогою базових сенсорів мобільного пристрою, таких як акселерометр або GPS.

Основною перевагою такого підходу є його простота та доступність. Користувач може легко зрозуміти отримані показники та відслідковувати власний прогрес без необхідності спеціальних знань. Крім того, реалізація даного підходу не потребує значних ресурсів, що робить його привабливим для розробників.

Однак, незважаючи на свою простоту, цей підхід має ряд суттєвих обмежень. По-перше, він не враховує індивідуальні особливості користувача, такі як рівень фізичної підготовки, вік, стан здоров’я або мета тренувань. По-друге, відстеження лише кількісних показників не дозволяє оцінити якість виконання вправ, що є важливим фактором для досягнення бажаних результатів.

Крім того, даний підхід не забезпечує можливості глибокого аналізу ефективності тренувального процесу. Наприклад, однакова кількість виконаних вправ може мати різний вплив на різних користувачів залежно від інтенсивності, техніки виконання та інших факторів. Таким чином, відстеження базових показників є доцільним для початкового рівня аналізу, але недостатнім для комплексної оцінки ефективності тренувань.

1.2.2 Використання носимих пристроїв

З розвитком технологій значного поширення набули носимі пристрої, такі як фітнес-браслети, смарт-годинники та інші сенсорні гаджети, які дозволяють здійснювати автоматичний збір даних про фізичну активність користувача. Використання таких пристроїв є більш сучасним підходом до моніторингу ефективності тренувань.

Носимі пристрої оснащені різноманітними датчиками, які дозволяють фіксувати широкий спектр показників, зокрема частоту серцевих скорочень, рівень фізичної активності, кількість кроків, витрати енергії, якість сну та інші параметри. Отримані дані передаються до мобільного застосунку, де вони можуть бути оброблені та проаналізовані.

Основною перевагою цього підходу є висока точність та об’єктивність отриманих даних. Автоматичний збір інформації зменшує вплив людського фактора та дозволяє отримувати більш достовірні результати. Крім того, використання носимих пристроїв дозволяє здійснювати безперервний моніторинг фізичного стану користувача, що відкриває можливості для більш глибокого аналізу.

Ще однією важливою перевагою є можливість оцінки інтенсивності тренувань. Наприклад, аналіз частоти серцевих скорочень дозволяє визначити, в якій зоні навантаження перебуває користувач, що є важливим для досягнення певних тренувальних цілей.

Разом з тим, даний підхід має і певні недоліки. По-перше, його реалізація потребує наявності додаткових пристроїв, що може бути фінансово затратним для користувача. По-друге, точність вимірювань може залежати від якості пристрою та умов його використання.

Крім того, незважаючи на великий обсяг зібраних даних, не всі застосунки забезпечують їх глибокий аналіз. У багатьох випадках користувач отримує лише сирі дані без належної інтерпретації, що ускладнює їх практичне використання.

Таким чином, використання пристроїв значно розширює можливості моніторингу фізичної активності, забезпечуючи більш точний і безперервний збір даних про стан користувача. Водночас це вимагає застосування спеціальних засобів для обробки, зберігання та аналізу отриманої інформації, а також ефективних алгоритмів, які дозволяють перетворювати великі обсяги даних у зрозумілі та корисні показники для подальшого використання.

1.2.3 Аналітика на основі даних

Найбільш сучасним та перспективним підходом до моніторингу ефективності тренувань є використання аналітичних методів обробки даних. Даний підхід передбачає не лише збір інформації про фізичну активність користувача, але й її комплексний аналіз з метою отримання корисних висновків.

Основою цього підходу є використання історичних даних про тренування, які накопичуються у процесі використання застосунку. На основі цих даних можуть визначатися тенденції, оцінюватися прогрес користувача та виявлятися закономірності у його фізичній активності.

Однією з ключових можливостей аналітичного підходу є персоналізація. Застосунок може враховувати індивідуальні характеристики користувача та адаптувати рекомендації відповідно до його цілей та поточного стану. Це дозволяє підвищити ефективність тренувального процесу та зробити його більш цілеспрямованим.

Крім того, сучасні аналітичні системи можуть використовувати методи машинного навчання для прогнозування результатів та формування рекомендацій. Наприклад, на основі попередніх тренувань система може визначити оптимальний рівень навантаження або запропонувати зміни у програмі тренувань.

Перевагою даного підходу є можливість отримання комплексної та глибокої оцінки ефективності тренувань. На відміну від попередніх підходів, він дозволяє враховувати не лише кількісні, але й якісні показники, що значно підвищує точність аналізу.

Водночас реалізація аналітичного підходу є більш складною та потребує використання сучасних технологій обробки даних. Крім того, для досягнення високої точності необхідна наявність достатнього обсягу даних, що може бути проблемою на початкових етапах використання застосунку.

Таким чином, аналітика на основі даних є найбільш ефективним підходом до моніторингу ефективності тренувань, проте її реалізація потребує значних ресурсів та продуманого підходу до обробки інформації.

1.3 Порівняльний аналіз застосунків

Проведений аналіз існуючих застосунків для моніторингу фізичної активності показує, що вони використовують різні підходи до реалізації функціональних можливостей, інтерфейсу користувача та методів обробки даних. Кожен із розглянутих застосунків має свої особливості, які визначають його сферу застосування та цільову аудиторію.

Незважаючи на значну кількість доступних рішень, жоден із них не забезпечує повноцінного комплексного підходу до моніторингу ефективності тренувань. Більшість застосунків зосереджуються або на відстеженні базових показників фізичної активності, або на наданні тренувальних програм, або на аналізі окремих параметрів, що обмежує їх можливості у контексті повноцінної оцінки результативності тренувального процесу.

Для більш об’єктивного аналізу існуючих рішень доцільно використовувати ряд критеріїв, які дозволяють оцінити їх ефективність з різних точок зору. Основними критеріями порівняння є функціональність, зручність використання, рівень аналітики та доступність.

Функціональність є одним із ключових критеріїв оцінки застосунку. Вона визначає набір можливостей, які надаються користувачу, зокрема відстеження фізичної активності, формування тренувальних програм, аналіз результатів та інтеграцію з іншими сервісами. Проведений аналіз показує, що більшість застосунків мають обмежений функціонал, орієнтований на виконання конкретних задач, що не дозволяє користувачу отримати комплексний інструмент для управління тренувальним процесом.

Зручність використання також відіграє важливу роль, оскільки від цього залежить ефективність взаємодії користувача із системою. Застосунок повинен мати інтуїтивно зрозумілий інтерфейс, логічну структуру та мінімальну кількість дій для виконання основних операцій. У ході аналізу було встановлено, що хоча більшість сучасних застосунків мають привабливий дизайн, не всі з них забезпечують достатній рівень зручності, особливо при роботі з аналітичними даними.

Рівень аналітики є критично важливим для оцінки ефективності тренувань. Саме аналітичні можливості дозволяють користувачу отримати інформацію про прогрес, визначити сильні та слабкі сторони тренувального процесу, а також сформувати рекомендації щодо його покращення. Однак більшість застосунків обмежуються відображенням базових статистичних даних без їх глибокого аналізу, що значно знижує їх практичну цінність.

Доступність включає як фінансовий аспект, так і технічну доступність застосунку. Багато застосунків використовують модель умовно-безкоштовного доступу, при якій базовий функціонал є безкоштовним, а розширені можливості доступні лише за підпискою. Це обмежує можливості користувачів, які не готові оплачувати додаткові функції, і знижує загальну ефективність використання таких рішень.

На основі проведеного аналізу можна зробити висновок, що більшість існуючих застосунків мають ряд спільних недоліків.

По-перше, вони не забезпечують комплексної оцінки ефективності тренувань. У більшості випадків користувач отримує лише окремі показники, які не дають повного уявлення про результативність тренувального процесу.

По-друге, спостерігається обмежена персоналізація. Застосунки часто не враховують індивідуальні особливості користувача, такі як рівень підготовки, фізичні можливості або цілі тренувань, що знижує ефективність їх використання.

По-третє, значна частина функціоналу є доступною лише у платних версіях застосунків. Це створює додаткові бар’єри для користувачів та обмежує можливість повноцінного використання системи.

Таким чином, проведений порівняльний аналіз показує, що існуючі рішення не повністю відповідають сучасним вимогам до систем моніторингу ефективності тренувань. Це обумовлює необхідність розробки нового застосунку, який буде поєднувати широкий функціонал, зручність використання, високий рівень аналітики та доступність для користувачів.

1.4 Постановка задачі

Таким чином, аналіз сучасних підходів до моніторингу фізичної активності та існуючих програмних рішень показав актуальність розроблення застосунку для комплексного аналізу ефективності тренувального процесу. Прийнято рішення щодо розроблення програмного застосунку моніторингу тренувань, який забезпечуватиме збір, аналіз, візуалізацію даних та формування персоналізованих рекомендацій для користувача.

Об’єктом роботи є процес моніторингу та аналізу фізичної активності користувача під час тренувань.

Метою роботи є підвищення ефективності тренувального процесу шляхом розробки програмного застосунку для збору, аналізу та оцінювання показників фізичної активності користувача.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз літературних джерел та сучасних підходів до моніторингу фізичної активності;

– провести аналіз існуючих мобільних і веб-застосунків для відстеження тренувального процесу;

– визначити основні недоліки сучасних систем моніторингу тренувань;

– сформувати функціональні вимоги до програмного застосунку моніторингу тренувального процесу;

– розробити алгоритми збору, аналізу та інтерпретації даних про тренування;

– реалізувати механізм формування персоналізованих рекомендацій на основі показників користувача;

– забезпечити візуалізацію результатів та динаміки прогресу користувача;

– розробити програмний застосунок із зручним та інтуїтивно зрозумілим інтерфейсом користувача;

– забезпечити можливість адаптації застосунку до різних типів тренувань (силові, кардіо, функціональні тощо).

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ТЕХНІЧНЕ ОБҐРУНТУВАННЯ СИСТЕМИ

2.1 Загальна характеристика проєктованої системи

На основі проведеного аналізу існуючих застосунків для моніторингу фізичної активності було визначено необхідність розробки системи, яка дозволяє не лише фіксувати окремі тренування, але й відстежувати навантаження, аналізувати прогрес користувача та підтримувати формування цілей. Розроблювана система призначена для користувачів, які займаються фізичними тренуваннями та потребують зручного інструмента для контроля власного тренувального процесу.

Основною метою проєктованої системи є забезпечення користувача засобами для створення, збереження та аналізу інформації про тренування. На відміну від простих застосунків, які обмежуються лише записом виконаних вправ або базовою статистикою, дана система орієнтована на комплексний підхід. Вона передбачає роботу з тренувальними даними, цілями, аналітичними показниками, повідомленнями та додатковою підтримкою користувача за допомогою AI-чату.

Система для відстеження тренувального прогресу має веборієнтований характер. Це означає, що користувач взаємодіє з нею через клієнтську частину, яка надає графічний інтерфейс для виконання основних дій. Через інтерфейс користувач може проходити автентифікацію, створювати записи про тренування, переглядати історію активності, працювати з цілями, отримувати аналітичну інформацію та користуватися додатковими сервісами системи [4–8].

З архітектурної точки зору система побудована за мікросервісним підходом. Це дозволяє розділити загальну функціональність на окремі незалежні компоненти, кожен з яких відповідає за власну частину бізнес-логіки. Такий підхід є доцільним для даного проєкту, оскільки система містить декілька різних функціональних напрямів: роботу з тренуваннями, авторизацію

користувачів, аналітику, повідомлення та AI-чат. Розподіл цих функцій між окремими сервісами спрощує підтримку системи, полегшує її подальше розширення та зменшує залежність між різними частинами застосунку.

Основними компонентами системи є клієнтська частина, BFF (Backend for Frontend), сервіс тренувань, сервіс автентифікації, сервіс аналітики, сервіс повідомлень, сервіс AI-чату, кешування та засоби моніторингу. Клієнтська частина відповідає за взаємодію з користувачем і відображення даних. BFF-рівень виконує роль проміжного шлюзу між клієнтом і внутрішніми сервісами. Завдяки цьому клієнтська частина не звертається напряму до кожного сервісу, а використовує єдину вхідну точку для взаємодії із системою.

Сервіс тренувань є одним із центральних компонентів системи. Він відповідає за обробку даних, пов’язаних із тренуваннями користувача, зокрема за створення, редагування, перегляд і збереження інформації про виконані вправи, тренувальні сесії та цілі. Саме цей сервіс формує основну функціональну частину застосунку, оскільки безпосередньо пов’язаний із предметною областю роботи.

Сервіс автентифікації відповідає за реєстрацію користувачів, вхід до системи та перевірку прав доступу. Його наявність є важливою, оскільки тренувальні дані користувача є персональними і повинні бути захищені від несанкціонованого доступу. Для цього в системі передбачено використання механізмів автентифікації та авторизації.

Сервіс аналітики призначений для обробки накопичених тренувальних даних і формування висновків щодо прогресу користувача. За допомогою цього компонента система може відображати динаміку змін, аналізувати виконання поставлених цілей та надавати користувачу більш інформативне уявлення про результати тренувального процесу.

Сервіс повідомлень використовується для організації нагадувань та інформування користувача про важливі події, пов’язані з тренуваннями або цілями. Наприклад, система може використовувати цей компонент для

нагадування про заплановане тренування або необхідність оновлення прогресу за певною ціллю.

Окремим компонентом системи є AI-чат. Його призначення полягає у наданні користувачу додаткової підтримки під час роботи із застосунком. Такий компонент може використовуватися для пояснення окремих функцій системи, допомоги з аналізом тренувальних даних або формування загальних рекомендацій на основі введеної інформації.

Для підвищення продуктивності системи передбачено використання кешування. Це дозволяє зменшити кількість повторних запитів до сервісів і баз даних, а також покращити швидкість відповіді системи для користувача. Крім того, у проєкті передбачено засоби моніторингу, які дають можливість відстежувати стан сервісів, аналізувати помилки, контролювати продуктивність і виявляти проблеми під час роботи системи.

Загальна високорівнева архітектура системи, що ілюструє базову взаємодію її ключових складових, наведена на рис. 2.1. Відповідно до запропонованої схеми, робочий процес розпочинається з клієнтської частини, через яку користувач взаємодіє з візуальним інтерфейсом платформи. Замість прямого звернення до внутрішніх ресурсів, усі клієнтські запити спрямовуються до проміжного інтеграційного рівня — шлюзу Backend for Frontend (BFF). Цей компонент виступає єдиною безпечною точкою входу, виконуючи функції базової маршрутизації, агрегації даних та перевірки авторизації. Після обробки на рівні шлюзу, BFF перенаправляє запити до відповідних ізольованих внутрішніх мікросервісів, кожен з яких відповідає виключно за свій бізнес-домен та працює з власною базою даних. Така організація топології забезпечує суворе логічне розділення відповідальності між компонентами, мінімізує їхню зв’язність (coupling) та створює надійну, масштабовану основу для подальшої реалізації системи.

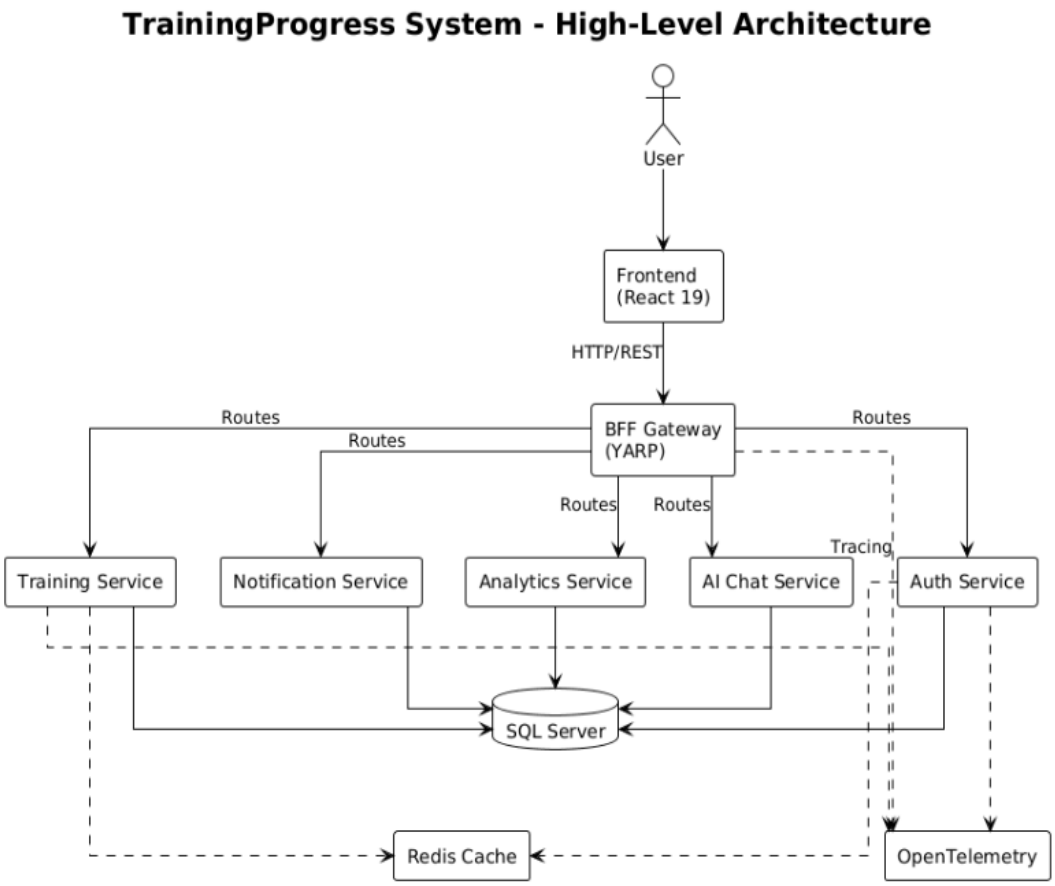

Рисунок 2.1 – Загальна структура системи для відстеження тренувального прогресу

Отже, проєктована система являє собою комплексний вебзастосунок для відстеження навантаження та прогресу у тренуваннях, побудований на основі мікросервісної архітектури. Така структура дозволяє чітко розділити ключові функції між незалежними сервісами, що не лише підвищує загальну відмовостійкість платформи, але й полегшує її подальше масштабування. Завдяки цьому забезпечується захищена робота з персональними даними користувачів, ефективна організація безперервної аналітики тренувального процесу, а також можливість гнучкого розширення функціоналу без впливу на стабільність існуючих компонентів.

2.2 Аналіз функціональних і нефункціональних вимог

Аналіз вимог є важливим етапом проєктування інформаційної системи, оскільки він дозволяє визначити, які функції має виконувати система та яким технічним характеристикам вона повинна відповідати. Для системи відстеження навантаження та прогресу у тренуваннях вимоги формуються з урахуванням потреб користувача, специфіки предметної області та обраної мікросервісної архітектури.

Функціональні вимоги описують основні можливості системи з точки зору користувача. У межах даного проєкту система повинна забезпечувати роботу з обліковим записом користувача, тренуваннями, тренувальними цілями, аналітичними даними, повідомленнями та додатковою підтримкою через AI-чат. Тобто система має не лише зберігати інформацію про фізичну активність, але й допомагати користувачу аналізувати власний прогрес і контролювати тренувальний процес.

Однією з основних функціональних вимог є можливість реєстрації та автентифікації користувача. Користувач повинен мати змогу створити власний обліковий запис, увійти до системи та отримати доступ лише до персональних даних. Це є важливим, оскільки інформація про тренування, фізичне навантаження, цілі та прогрес має індивідуальний характер. Для реалізації цієї вимоги в системі передбачено окремий сервіс автентифікації, який відповідає за перевірку користувача, роботу з токенами доступу та контроль авторизованих запитів.

Наступною функціональною вимогою є створення та збереження тренувань. Користувач повинен мати можливість додавати інформацію про виконані тренування, зокрема зазначати вправи, дату тренування, кількість підходів, повторень, навантаження та інші необхідні параметри. Така можливість дозволяє вести історію тренувального процесу та накопичувати дані для подальшого аналізу.

Система також повинна надавати можливість редагування та перегляду раніше створених тренувань. Це потрібно для того, щоб користувач міг виправляти неточності, оновлювати дані або переглядати попередні результати. Перегляд історії тренувань є важливою частиною системи, оскільки саме на основі збережених записів користувач може оцінювати регулярність занять, зміну навантаження та загальну динаміку прогресу.

Окремою функціональною вимогою є робота з тренувальними цілями. Користувач повинен мати можливість створювати цілі, пов’язані з власним тренувальним процесом. Такими цілями можуть бути, наприклад, збільшення робочої ваги, покращення регулярності тренувань, виконання певної кількості занять за визначений період або досягнення конкретного показника. Система повинна зберігати ці цілі, відстежувати їх виконання та оновлювати інформацію про поточний прогрес.

Важливою вимогою є аналітична обробка тренувальних даних. Система повинна не лише зберігати окремі записи про тренування, але й обробляти накопичену інформацію. Це дозволяє користувачу бачити зміни у власному прогресі, оцінювати виконання поставлених цілей і краще розуміти результати тренувального процесу. Аналітичний компонент системи має забезпечувати формування статистичних показників, відображення динаміки та підготовку узагальненої інформації про активність користувача.

Ще однією функціональною вимогою є можливість отримання повідомлень і нагадувань. Система повинна інформувати користувача про важливі події, пов’язані з тренуваннями або цілями. Наприклад, користувач може отримувати нагадування про заплановане тренування, необхідність оновити прогрес або виконати певну дію в системі. Така функціональність підвищує практичну користь застосунку, оскільки допомагає підтримувати регулярність і дисципліну у тренувальному процесі.

Додатковою функціональною можливістю є AI-чат. Його призначення полягає у наданні користувачу додаткової підтримки під час роботи із системою. Такий компонент може допомагати користувачу краще зрозуміти

функції застосунку, отримати пояснення щодо тренувальних даних або сформувати загальні рекомендації на основі введеної інформації. Наявність AI-чату розширює можливості системи та робить її більш адаптивною до індивідуальних запитів користувача.

Окрім функціональних вимог, важливе значення мають нефункціональні вимоги. Вони визначають не конкретні дії користувача, а якісні характеристики системи. До них належать безпека, продуктивність, надійність, масштабованість, зручність використання, підтримуваність і можливість моніторингу роботи застосунку.

Однією з головних нефункціональних вимог є безпека. Система повинна забезпечувати захист персональних даних користувача та не допускати несанкціонованого доступу до інформації про тренування, цілі та прогрес. Для цього необхідно використовувати механізми автентифікації та авторизації, а доступ до основних функцій системи має надаватися лише після успішної перевірки користувача. Оскільки в системі використовується JWT-автентифікація і BFF може передавати авторизаційний заголовок до внутрішніх сервісів, безпека є частиною як клієнтської, так і серверної взаємодії.

Продуктивність також є важливою вимогою до системи. Застосунок повинен швидко реагувати на дії користувача, зокрема під час перегляду тренувань, аналітики, цілей і повідомлень. Для зменшення навантаження на сервіси та бази даних може використовуватися кешування. У проєкті для цього передбачено Redis, який використовується як частина runtime-інфраструктури системи.

Надійність системи означає, що застосунок повинен стабільно працювати під час виконання основних користувацьких сценаріїв. Оскільки система складається з кількох окремих сервісів, важливо передбачити коректну обробку помилок, перевірку стану сервісів і можливість відстеження проблем у роботі окремих компонентів. Для цього у системі передбачені health checks, які дозволяють контролювати доступність сервісів і використовуються BFF-рівнем для активної перевірки стану внутрішніх компонентів.

Масштабованість є важливою характеристикою системи, оскільки кількість користувачів і обсяг тренувальних даних можуть збільшуватися. Мікросервісна архітектура дозволяє розвивати й масштабувати окремі частини системи незалежно одна від одної. Наприклад, сервіс тренувань, аналітичний сервіс або сервіс повідомлень можуть розширюватися окремо залежно від навантаження та функціональних потреб системи.

Зручність використання визначає, наскільки просто користувач може взаємодіяти із застосунком. Інтерфейс повинен бути логічним, зрозумілим і не перевантаженим зайвими діями. Основні операції, такі як створення тренування, перегляд прогресу, створення цілі або отримання аналітичної інформації, повинні виконуватися з мінімальною кількістю кроків. Це особливо важливо для системи, яка орієнтована на регулярне використання.

Підтримуваність системи означає можливість зручного внесення змін, виправлення помилок і додавання нових функцій. Для цього в системі застосовується розділення на окремі сервіси та рівні. Кожен сервіс має власну відповідальність, а спільні технічні можливості винесені у shared-проєкти. Такий підхід зменшує дублювання коду та спрощує подальший розвиток системи.

Окремою нефункціональною вимогою є спостережуваність роботи системи. Для мікросервісного застосунку важливо мати можливість відстежувати помилки, затримки, стан сервісів і загальну продуктивність. У проєкті для цього передбачено OpenTelemetry [9] та Aspire Dashboard, які використовуються для роботи з логами, метриками й трасуванням. Це дозволяє аналізувати поведінку системи під час виконання запитів і швидше знаходити проблеми у взаємодії між компонентами.

Таким чином, у процесі аналізу вимог було визначено основні функціональні та нефункціональні характеристики системи для відстеження навантаження та прогресу у тренуваннях. Функціональні вимоги охоплюють роботу з користувацьким обліковим записом, тренуваннями, цілями, аналітикою, повідомленнями та AI-чатом. Нефункціональні вимоги визначають

технічну якість системи, зокрема її безпеку, продуктивність, надійність, масштабованість, зручність використання, підтримуваність і спостережуваність. Визначені вимоги є основою для подальшого проєктування архітектури системи та її програмної реалізації.

2.3 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку є важливим етапом проєктування системи, оскільки саме він визначає засоби, за допомогою яких буде реалізовано клієнтську частину, серверну логіку, збереження даних, взаємодію між сервісами, моніторинг і подальше розгортання застосунку. Для системи відстеження навантаження та прогресу у тренуваннях було обрано стек сучасних технологій, який відповідає вимогам мікросервісної архітектури, підтримує масштабованість і дозволяє розділити функціональність системи між окремими компонентами.

Основою серверної частини системи є платформа .NET [10] та фреймворк ASP.NET Core. Їх використання є доцільним для розробки вебзастосунків і мікросервісів, оскільки вони забезпечують стабільну роботу серверної логіки, підтримують побудову API [11], мають вбудовані механізми для роботи з автентифікацією, авторизацією, middleware та dependency injection. Для даного проєкту це є важливим, оскільки система складається з декількох окремих сервісів, кожен з яких має власну відповідальність і повинен бути організований за єдиним принципом.

Серверна частина проєкту має багатопроєктну структуру, що дозволяє розділити застосунок на окремі сервіси та шари. Така організація полегшує підтримку коду, тестування й подальше розширення функціональності. Наприклад, сервіси, пов’язані з тренуваннями, автентифікацією, аналітикою, повідомленнями та AI-чатом, можуть розвиватися окремо, не вимагаючи повної

перебудови всієї системи. Це відповідає загальній логіці мікросервісного підходу, де кожен сервіс відповідає за свою частину бізнес-логіки.

Для роботи з даними у системі використовується Entity Framework Core [12]. Цей інструмент забезпечує зручну взаємодію між серверною частиною та базою даних, дозволяючи працювати з даними через об’єктну модель. Використання EF Core спрощує реалізацію операцій створення, читання, оновлення та видалення даних, а також підтримує міграції бази даних. Це є важливим для системи, яка повинна зберігати інформацію про користувачів, тренування, цілі, аналітичні показники та інші об’єкти предметної області.

Як система управління базами даних використовується SQL Server. Його вибір є обґрунтованим, оскільки система працює зі структурованими даними, які мають чіткі зв’язки між сутностями. Наприклад, користувач може мати декілька тренувань, цілей, досягнень або аналітичних записів. Реляційна база даних дозволяє зберігати такі дані у впорядкованому вигляді, забезпечувати цілісність інформації та виконувати запити до пов’язаних таблиць.

Окрему роль у системі виконує ASP.NET Core Identity. Цей механізм використовується для зберігання та керування даними користувачів, а також для реалізації процесів реєстрації, входу до системи й перевірки доступу. Оскільки тренувальні дані є персональними, система повинна забезпечувати захист облікових записів і не допускати доступу користувача до чужої інформації. Для цього в системі використовується JWT-автентифікація, яка дозволяє перевіряти запити користувача та передавати інформацію про його доступ між компонентами системи.

Для організації взаємодії між клієнтською частиною та внутрішніми сервісами використовується BFF-рівень. BFF виконує роль проміжного шлюзу, через який клієнтська частина звертається до серверної логіки. Такий підхід дозволяє не відкривати всі внутрішні сервіси напряму для браузера, а зосередити маршрутизацію, перевірку доступу та частину оркестрації запитів в одному місці. У проєкті BFF використовує YARP reverse proxy [13] для маршрутизації запитів до відповідних сервісів. Документація проєкту також

зазначає, що браузер переважно входить у систему через BFF, а внутрішня комунікація може виконуватися через HTTP reverse proxy та gRPC-клієнти [14].

Для взаємодії між окремими сервісами використовується gRPC. Ця технологія є доцільною для внутрішньої комунікації в мікросервісній архітектурі, оскільки дозволяє організувати швидкий і типізований обмін даними між сервісами. У межах даного проєкту gRPC може використовуватися для сценаріїв, де BFF координує виконання декількох дій між різними сервісами. Наприклад, при збереженні цілі можуть бути задіяні сервіс тренувань, сервіс аналітики та сервіс повідомлень.

Для підтримки оновлень у режимі реального часу використовується SignalR. Його використання є доцільним у системі, де користувач може отримувати оновлення про стан тренувальних цілей, повідомлення або інші зміни без необхідності постійно оновлювати сторінку. Це покращує інтерактивність застосунку та дозволяє швидше відображати актуальні дані на клієнтській стороні.

Для реалізації клієнтської частини використовується React [15, 16] у поєднанні з TypeScript [17] і Vite. React дозволяє створювати інтерфейс користувача на основі компонентів, що є зручним для побудови сторінок і повторного використання елементів інтерфейсу. TypeScript додає статичну типізацію, що допомагає зменшити кількість помилок під час розробки та зробити код більш зрозумілим. Vite використовується як сучасний інструмент для швидкого збирання клієнтського застосунку, що покращує процес розробки та прискорює запуск проєкту.

Для керування станом на клієнтській стороні використовуються Redux Toolkit і React Redux [18]. Вони дозволяють організувати централізоване збереження стану застосунку, що є важливим для системи з багатьма взаємопов’язаними сторінками. Наприклад, інформація про авторизованого користувача, вибрані тренування, цілі або поточні налаштування інтерфейсу може використовуватися в різних частинах клієнтського застосунку.

Для роботи із серверними запитами використовується TanStack React Query разом з Axios. Axios забезпечує відправлення HTTP-запитів до серверної частини, а React Query спрощує отримання, кешування та оновлення даних. Це є корисним для сторінок, де потрібно часто отримувати інформацію про тренування, цілі, статистику або повідомлення. Завдяки кешуванню даних зменшується кількість повторних запитів і покращується швидкість взаємодії користувача із застосунком.

Для маршрутизації на клієнтській стороні використовується React Router. Він дозволяє організувати переходи між сторінками застосунку, наприклад між сторінкою входу, головною сторінкою, сторінкою тренувань, цілями, аналітикою та AI-чатом. Такий підхід робить застосунок зручним для користувача та дозволяє створити логічну структуру інтерфейсу.

Для оформлення інтерфейсу використовуються Tailwind CSS і Radix UI. Tailwind CSS дозволяє швидко створювати стилі для компонентів без написання великої кількості окремих CSS-файлів. Radix UI надає готові примітиви інтерфейсу, які можна адаптувати до потреб проєкту. Для відображення графіків і візуалізації тренувального прогресу використовується Recharts. Це є важливим саме для даної системи, оскільки аналітика тренувань повинна бути представлена користувачу не лише у текстовому вигляді, але й через зрозумілі візуальні елементи.

Для підвищення продуктивності системи використовується Redis. Він виконує роль кешу та дозволяє зберігати тимчасові або часто використовувані дані. Це допомагає зменшити навантаження на бази даних і внутрішні сервіси. У системі, де користувачі можуть часто переглядати тренування, цілі та аналітику, кешування є важливим технічним засобом для покращення швидкодії.

Для локального запуску та організації runtime-середовища використовується Docker Compose [19, 20]. Це дозволяє описати всі основні компоненти системи в одному середовищі: клієнтську частину, BFF, backend-сервіси, Redis [21] і засоби моніторингу. Такий підхід спрощує запуск проєкту,

оскільки всі необхідні сервіси можуть бути підняті разом із попередньо визначеними налаштуваннями. Крім того, контейнеризація полегшує перенесення системи між різними середовищами.

Для моніторингу роботи системи використовується OpenTelemetry та Aspire Dashboard. OpenTelemetry забезпечує збір логів, метрик і трасування, що є важливим для мікросервісної архітектури. Оскільки запит користувача може проходити через BFF і декілька внутрішніх сервісів, необхідно мати можливість відстежити весь шлях виконання запиту. Aspire Dashboard використовується для локальної візуалізації телеметрії та допомагає аналізувати стан системи під час розробки й тестування.

Також у проєкті передбачено інструменти для тестування. Для backend-частини використовуються NUnit, Moq, Microsoft.NET.Test.Sdk і Coverlet collector. Наявність тестових проєктів для окремих шарів і сервісів дозволяє перевіряти коректність роботи системи та зменшувати ризик помилок під час внесення змін. Це особливо важливо для мікросервісної системи, де зміни в одному компоненті можуть впливати на взаємодію з іншими частинами застосунку.

Отже, обраний технологічний стек відповідає вимогам системи для відстеження навантаження та прогресу у тренуваннях. Backend-частина на основі .NET та ASP.NET Core [22] забезпечує стабільну серверну логіку, EF Core і SQL Server відповідають за роботу зі структурованими даними, а Identity і JWT забезпечують захист користувацьких облікових записів. Frontend-частина на основі React, TypeScript і Vite дозволяє створити зручний і динамічний інтерфейс користувача. YARP, gRPC і SignalR [23] забезпечують взаємодію між компонентами системи, а Redis, Docker, OpenTelemetry та Aspire Dashboard підтримують продуктивність, розгортання й моніторинг. Такий стек є доцільним для розробки масштабованого, підтримуваного та функціонального вебзастосунку.

2.4 Проєктування мікросервісної архітектури системи

Проєктування архітектури системи є одним із ключових етапів розробки, оскільки саме на цьому етапі визначається загальна структура застосунку, розподіл відповідальності між компонентами та принципи їхньої взаємодії. Для системи відстеження навантаження та прогресу у тренуваннях було обрано мікросервісну архітектуру. Такий підхід є доцільним, оскільки система складається з кількох різних функціональних частин, які можуть розвиватися та масштабуватися окремо.

Мікросервісна архітектура передбачає поділ системи на набір незалежних сервісів, кожен з яких відповідає за окремий функціональний напрям. На відміну від монолітної архітектури, де вся логіка застосунку зосереджена в одному програмному блоці, мікросервісний підхід дозволяє розділити систему на автономні компоненти. Це спрощує підтримку коду, полегшує тестування, зменшує залежність між частинами системи та створює можливість для подальшого розширення функціональності.

У межах даного проєкту система складається з клієнтської частини, BFF-рівня, сервісу тренувань, сервісу автентифікації, сервісу аналітики, сервісу повідомлень, сервісу AI-чату, Redis та засобів моніторингу. У документації проєкту як основні runtime-компоненти вказані client, bff, training-service, auth-service, notification-service, analytics-service, ai-chat-service, redis та aspire-dashboard.

Клієнтська частина є візуальною частиною системи, через яку користувач взаємодіє із застосунком. Вона відповідає за відображення сторінок, форм, списків тренувань, аналітичних даних, повідомлень та інших елементів інтерфейсу. Користувач не взаємодіє напряму з кожним внутрішнім сервісом системи. Замість цього всі запити з клієнтської частини проходять через BFF-рівень.

BFF, або Backend for Frontend, є проміжним рівнем між клієнтською частиною та внутрішніми сервісами. Його основне призначення полягає в тому,

щоб надати клієнту єдину вхідну точку для взаємодії із системою. Завдяки цьому фронтенд не повинен знати внутрішню структуру всіх мікросервісів, їхні адреси та особливості маршрутизації. Такий підхід зменшує зв’язаність між клієнтською частиною та серверною архітектурою.

BFF-рівень також виконує роль шлюзу для маршрутизації запитів. У проєкті для цього використовується YARP reverse proxy, який перенаправляє запити до відповідних внутрішніх сервісів. У документації зазначено, що маршрути мають префікси окремих сервісів, наприклад /training-service, /analytics-service, /notification-service, /auth-service та /ai-chat-service. Перед переспрямуванням запиту префікс може видалятися, а сам запит передається до відповідного downstream-сервісу.

Важливою перевагою використання BFF є централізація частини логіки, пов’язаної з безпекою та маршрутизацією. Зокрема, BFF може передавати авторизаційний заголовок до внутрішніх сервісів, якщо користувач уже має токен доступу. Це дозволяє зберігати єдиний підхід до перевірки користувача та не дублювати зайву логіку на клієнтській стороні. У документації проєкту зазначено, що JWT з cookie може перетворюватися BFF-рівнем в Authorization header для downstream-запитів.

Центральним доменним компонентом системи є training-service. Він відповідає за основну предметну область застосунку, тобто за роботу з тренуваннями, вправами, цілями та досягненнями користувача. Саме цей сервіс обробляє запити, пов’язані зі створенням, редагуванням, переглядом і збереженням тренувальних даних. Його можна вважати основним сервісом системи, оскільки більшість функціональності застосунку безпосередньо пов’язана з відстеженням тренувального процесу.

Auth-service відповідає за автентифікацію та авторизацію користувачів. Його основними функціями є реєстрація користувачів, вхід до системи, оновлення токенів доступу та перевірка ідентичності користувача. Наявність окремого сервісу автентифікації є обґрунтованою, оскільки безпека персональних даних користувача є однією з головних вимог до системи.

Завдяки виділенню цієї функціональності в окремий сервіс спрощується контроль доступу та підтримка логіки, пов’язаної з користувацькими обліковими записами.

Analytics-service відповідає за обробку тренувальних даних і формування аналітичної інформації. Його завдання полягає в тому, щоб на основі збережених тренувань і цілей користувача формувати показники прогресу, статистику та узагальнені результати. Такий сервіс дозволяє відокремити безпосереднє збереження тренувальних записів від їхньої подальшої аналітичної обробки. Це робить архітектуру більш гнучкою, оскільки аналітична логіка може розвиватися окремо від основного сервісу тренувань.

Notification-service використовується для роботи з повідомленнями та нагадуваннями. Його функціональність може включати створення нагадувань про тренування, повідомлення про зміну стану цілі або інші важливі події в системі. Виділення повідомлень в окремий сервіс є доцільним, оскільки така логіка не повинна перевантажувати основний сервіс тренувань. Крім того, у майбутньому notification-service може бути розширений, наприклад, для підтримки різних каналів повідомлень.

Ai-chat-service є окремим компонентом, який призначений для реалізації AI-чату. Його основне завдання полягає у наданні користувачу додаткової підтримки під час роботи із системою. Цей сервіс може обробляти запити користувача, пов’язані з поясненням функцій застосунку, інтерпретацією тренувальних даних або загальними рекомендаціями. Виділення AI-чату в окремий сервіс є доцільним, оскільки така функціональність має специфічну логіку і може вимагати окремого масштабування або підключення зовнішніх інструментів.

Окрему роль у системі виконує Redis. Він використовується як інфраструктурний компонент для кешування та підвищення продуктивності системи. Завдяки кешуванню можна зменшити кількість повторних запитів до баз даних і внутрішніх сервісів, що є важливим для сторінок, де користувач часто переглядає тренування, аналітику або цілі. У документації зазначено, що

Redis налаштований у compose-файлі та використовується через shared caching abstractions.

Важливою особливістю архітектури є те, що кожен доменний сервіс володіє власною бізнес-логікою та власною областю даних. У документації проєкту зазначено, що сервіси використовують окремі DbContext та міграції, що вказує на підхід database-per-service. Прикладами є TrainingServiceDbContext, AuthServiceDbContext та AnalyticsServiceDbContext. Такий підхід означає, що різні сервіси не повинні напряму працювати з таблицями один одного. Замість цього інтеграція між ними здійснюється через API або gRPC-взаємодію.

Комунікація між компонентами системи організована на кількох рівнях. Зовнішній користувацький запит спочатку надходить із браузера до BFF. Далі BFF або маршрутизує запит до відповідного сервісу через HTTP reverse proxy, або використовує gRPC-клієнти для виконання складніших сценаріїв. Такий підхід дозволяє поєднати просту маршрутизацію запитів із можливістю внутрішньої оркестрації декількох сервісів. У документації прямо зазначено, що зовнішній вхід від браузера переважно проходить через BFF, а внутрішня взаємодія здійснюється через HTTP reverse proxy та gRPC-клієнти.

Особливо важливими є сценарії, де одна дія користувача потребує виконання кількох операцій у різних сервісах. Наприклад, при збереженні тренувальної цілі система може спочатку зберегти ціль у TrainingService, потім перерахувати прогрес і після цього запланувати нагадування у NotificationService. У документації такий сценарій описаний як приклад workflow: збереження цілі, перерахунок прогресу, створення нагадувань і компенсація у випадку критичної помилки.

Для таких сценаріїв використовується оркестрація на рівні BFF Application layer. Це означає, що BFF не лише переспрямовує прості запити, але й може координувати складніші міжсервісні процеси. Якщо один із критичних кроків не виконується, система може перейти до компенсаційної логіки. Такий

підхід дозволяє підвищити надійність виконання складних операцій без використання спільної бази даних або розподілених транзакцій.

У системі передбачено підтримку health checks, які дозволяють перевіряти стан окремих сервісів і визначати їхню доступність для обробки запитів. BFF використовує ці перевірки для моніторингу downstream-сервісів, що є важливим у мікросервісній архітектурі, оскільки недоступність одного компонента повинна оперативно виявлятися.

Також реалізовано підтримку realtime-оновлень за допомогою SignalR. BFF містить SignalR sync hub, який забезпечує передачу оновлень на клієнтську сторону без необхідності ручного оновлення сторінки, що особливо корисно для сповіщень та відображення змін у реальному часі.

Загальну структуру мікросервісної архітектури системи доцільно подати у вигляді схеми з відображенням ключових компонентів: клієнта, BFF-рівня, внутрішніх сервісів, Redis, баз даних і Aspire Dashboard (рис. 2.2).

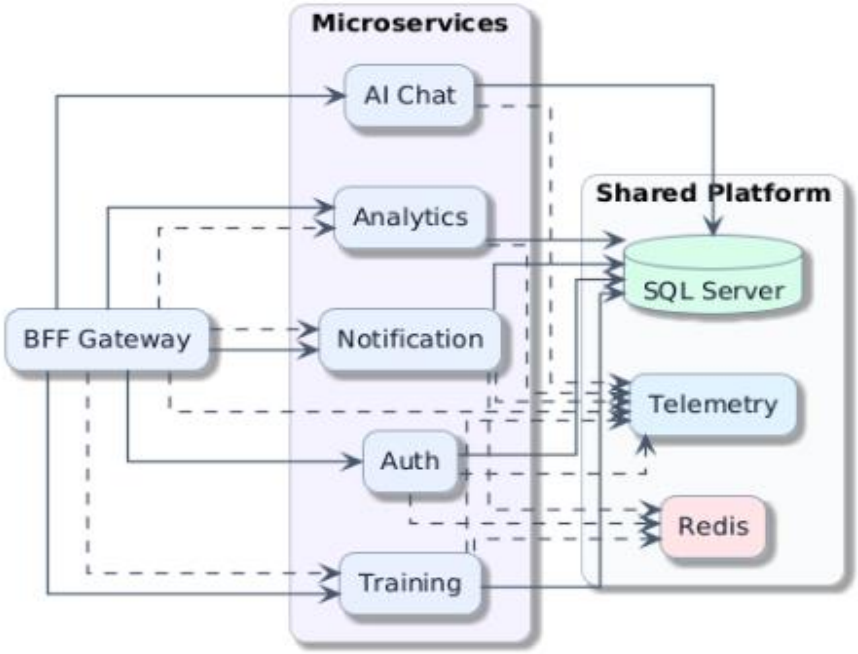

Рисунок 2.2 – Мікросервісна архітектура системи для відстеження тренувального прогресу

Отже, мікросервісна архітектура системи дозволяє розділити функціональність застосунку між окремими сервісами відповідно до їхньої відповідальності. BFF забезпечує єдину вхідну точку для клієнтської частини, training-service відповідає за основну логіку тренувань, auth-service – за автентифікацію, analytics-service – за обробку даних, notification-service – за повідомлення, а ai-chat-service – за AI-підтримку користувача. Така структура підвищує гнучкість системи, полегшує її підтримку та створює основу для подальшого масштабування й розвитку.

2.5 Архітектурні патерни, використані в системі

У процесі проєктування системи для відстеження навантаження та прогресу у тренуваннях було використано декілька архітектурних патернів. Їх застосування дозволяє чітко розподілити відповідальність між компонентами, спростити підтримку коду та забезпечити стабільну взаємодію між мікросервісами.

Одним із базових рішень є багаторівнева архітектура сервісів. Кожен backend-сервіс має поділ на рівні Api, Application, Domain та Infrastructure [24, 25]. Api-рівень відповідає за обробку запитів, Application-рівень — за прикладну логіку, Domain-рівень – за основні сутності та правила предметної області, а Infrastructure-рівень – за роботу з базою даних і зовнішніми залежностями. Такий поділ дозволяє відокремити бізнес-логіку від технічних деталей реалізації.

Важливим патерном у системі є Backend for Frontend. BFF виступає єдиною точкою входу для клієнтської частини та перенаправляє запити до відповідних внутрішніх сервісів. Завдяки цьому frontend не взаємодіє напряму з кожним мікросервісом, а використовує один проміжний рівень. Це зменшує залежність клієнтської частини від внутрішньої структури backend-системи.

Окрему роль відіграє Saga Pattern, який використовується для координації складних міжсервісних операцій. У мікросервісній архітектурі різні сервіси мають власні бази даних, тому не можна просто виконати одну спільну транзакцію для всієї системи. Саме тому використовується saga-підхід: операція розбивається на послідовні кроки, а у випадку помилки запускається компенсаційна логіка.

У даній системі Saga Pattern може використовуватися, наприклад, під час збереження тренувальної цілі. Спочатку ціль зберігається у сервісі тренувань, потім може виконуватися перерахунок прогресу, а після цього створюються нагадування через сервіс повідомлень. Якщо один із критичних кроків не виконується, система може перейти до компенсаційного сценарію. Це дозволяє підтримувати узгодженість між сервісами без використання розподілених транзакцій.

Також у системі використовується Repository Pattern. Репозиторії відповідають за доступ до даних і приховують деталі роботи з Entity Framework Core. Завдяки цьому прикладна логіка не працює напряму з базою даних, а використовує окремі методи репозиторіїв. Такий підхід робить код більш структурованим і зручним для тестування.

Для повторного використання спільної технічної логіки застосовується Shared Kernel. У shared-проєкти винесено загальні контракти, механізми автентифікації, idempotency, OpenTelemetry та saga-компоненти. Це зменшує дублювання коду між сервісами й забезпечує однакову поведінку спільних механізмів у всій системі.

Ще одним важливим рішенням є Idempotency Pattern. Він потрібен для безпечної обробки повторних запитів. Наприклад, якщо користувач випадково або через помилку мережі повторно надсилає запит на створення цілі, система не повинна створювати дублікати. Idempotency дозволяє уникати таких ситуацій і є особливо корисною разом із saga-оркестрацією.

Для захисту користувацьких даних використовується Identity + JWT (JSON Web Token) Authentication Pattern. Identity відповідає за керування

користувачами, а JWT-токени використовуються для перевірки авторизованих запитів. Це дозволяє обмежити доступ до персональних тренувальних даних і забезпечити роботу захищених endpoint-ів.

Отже, використані архітектурні патерни формують основу стабільної та підтримуваної системи. Багаторівнева архітектура впорядковує код, BFF спрощує взаємодію frontend-частини з backend-сервісами, Saga Pattern забезпечує надійне виконання міжсервісних операцій, Repository Pattern відокремлює доступ до даних, Shared Kernel зменшує дублювання, Idempotency захищає від повторного виконання запитів, а Identity + JWT забезпечує безпечну роботу з користувацькими даними.

2.6 Проєктування моделей даних і меж зберігання інформації

Проєктування моделей даних визначає, які сутності зберігаються в системі, як вони пов’язані між собою та який сервіс відповідає за їхню обробку. У системі для відстеження навантаження та прогресу у тренуваннях дані розподілені між окремими сервісами. Такий підхід відповідає мікросервісній архітектурі, де кожен сервіс має власну область відповідальності та власні межі зберігання інформації.

У проєкті використовуються окремі контексти бази даних для різних сервісів. Зокрема, у документації зазначені TrainingServiceDbContext, AuthServiceDbContext та AnalyticsServiceDbContext. Це означає, що сервіси не працюють напряму з однією спільною базою даних, а зберігають власні дані у межах відповідного bounded context. Взаємодія між сервісами відбувається через API або gRPC, а не через пряме звернення до таблиць іншого сервісу.

Першою важливою частиною є схема Shared Identity. Вона пов’язана з автентифікацією та авторизацією користувачів. У цій частині зберігаються дані, необхідні для роботи з обліковими записами, ролями, токенами та доступом до

системи. Для реалізації цього механізму використовується ASP.NET Core Identity разом із JWT-автентифікацією (рис. 2.3).

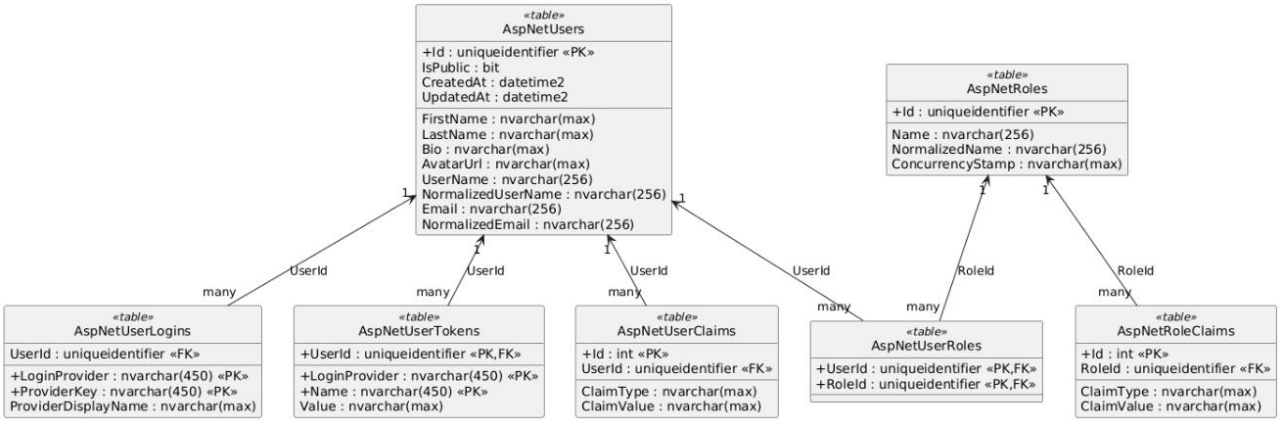

Рисунок 2.3 – Схема бази даних Shared Identity

Другою основною частиною є схема TrainingService, яка відображає структуру даних, безпосередньо пов’язаних із тренувальним процесом користувача. Вона включає основні сутності предметної області, що забезпечують збереження інформації про тренування, постановку та виконання цілей, а також фіксацію досягнень і супутніх показників, необхідних для оцінки фізичної активності в динаміці та формування об’єктивної картини прогресу користувача.

TrainingService виконує ключову роль у системі, оскільки саме в ньому зосереджено основну бізнес-логіку, пов’язану з управлінням тренувальними даними. Сервіс забезпечує централізоване зберігання, обробку та оновлення інформації про тренування, що дозволяє відстежувати зміни фізичних показників і прогрес користувача протягом тривалого часу та в різних часових проміжках. Отримані дані можуть надалі використовуватися іншими компонентами системи для аналітики, побудови статистики, формування персоналізованих рекомендацій та відображення результатів у зручному для користувача інтерфейсі. У результаті TrainingService забезпечує цілісність, узгодженість і актуальність інформації про фізичну активність у межах усієї системи (рис. 2.4).

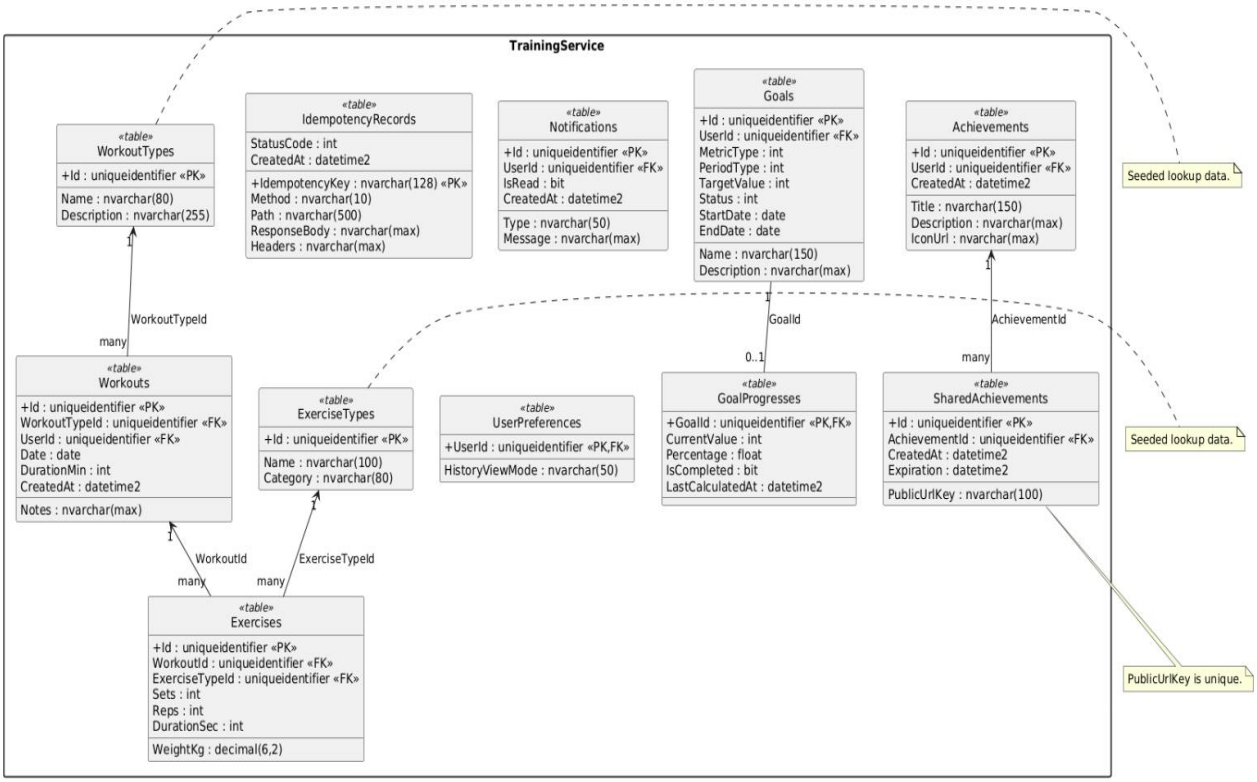

Рисунок 2.4 – Схема бази даних TrainingService

Такий поділ моделей даних дозволяє зберігати чіткі межі відповідальності між сервісами: AuthService відповідає за користувачів і доступ, TrainingService – за тренувальні дані, а AnalyticsService – за їх аналітичну обробку.

Отже, моделі даних у системі спроєктовані відповідно до принципів мікросервісної архітектури. Виокремлення схем Shared Identity та TrainingService дозволяє розмежувати дані авторизації та основні тренувальні дані, що робить систему більш структурованою, гнучкою та зручною для подальшого розширення.

2.7 Проєктування системи моніторингу та аналізу роботи застосунку

Для мікросервісної системи важливо не лише реалізувати основну функціональність, але й мати можливість контролювати її роботу. Оскільки система складається з кількох окремих сервісів, необхідно відстежувати стан

кожного компонента, помилки, затримки запитів і взаємодію між сервісами. Для цього в проєкті передбачено систему моніторингу та аналізу [26].

Основою моніторингу є OpenTelemetry. У проєкті він використовується для збору логів, метрик і трасування запитів. Це дозволяє бачити, як запит проходить через систему, які сервіси були задіяні, скільки часу зайняло виконання операції та де могла виникнути помилка. У документації зазначено, що налаштування OpenTelemetry винесене у shared-компоненти, що забезпечує однакову поведінку моніторингу в різних сервісах.

Зібрані дані передаються через OTLP (OpenTelemetry Protocol) до Aspire Dashboard. У цьому процесі сервіси створюють логи, метрики та traces, після чого OpenTelemetry SDK експортує їх до dashboard, де вони можуть бути переглянуті у зручному вигляді. Aspire Dashboard використовується як локальний інструмент для аналізу роботи системи під час розробки та тестування.

Окреме значення має кореляція запитів між сервісами. У мікросервісній архітектурі одна дія користувача може проходити через BFF, TrainingService, AnalyticsService або NotificationService. Щоб відстежити весь шлях такого запиту, у системі використовуються x-correlation-id та x-idempotency-key. Correlation id дозволяє пов’язати кілька запитів в один користувацький сценарій, а idempotency key допомагає аналізувати повторні запити й уникати дублювання записувальних операцій.

Також у системі використовуються health checks. Вони дозволяють перевіряти, чи доступний конкретний сервіс і чи може він обробляти запити. Це особливо важливо для BFF-рівня, який перенаправляє запити до внутрішніх сервісів. Якщо певний сервіс недоступний, така інформація може бути швидко виявлена через механізми перевірки стану.

Моніторинг у системі охоплює кілька рівнів. На рівні BFF можна аналізувати маршрутизацію запитів, помилки reverse proxy та доступність внутрішніх сервісів. На рівні окремих сервісів можна відстежувати затримки endpoint-ів, помилки, звернення до зовнішніх залежностей і проблеми запуску.

На рівні оркестрації можна аналізувати виконання saga-сценаріїв, критичні помилки та запуск компенсаційної логіки.

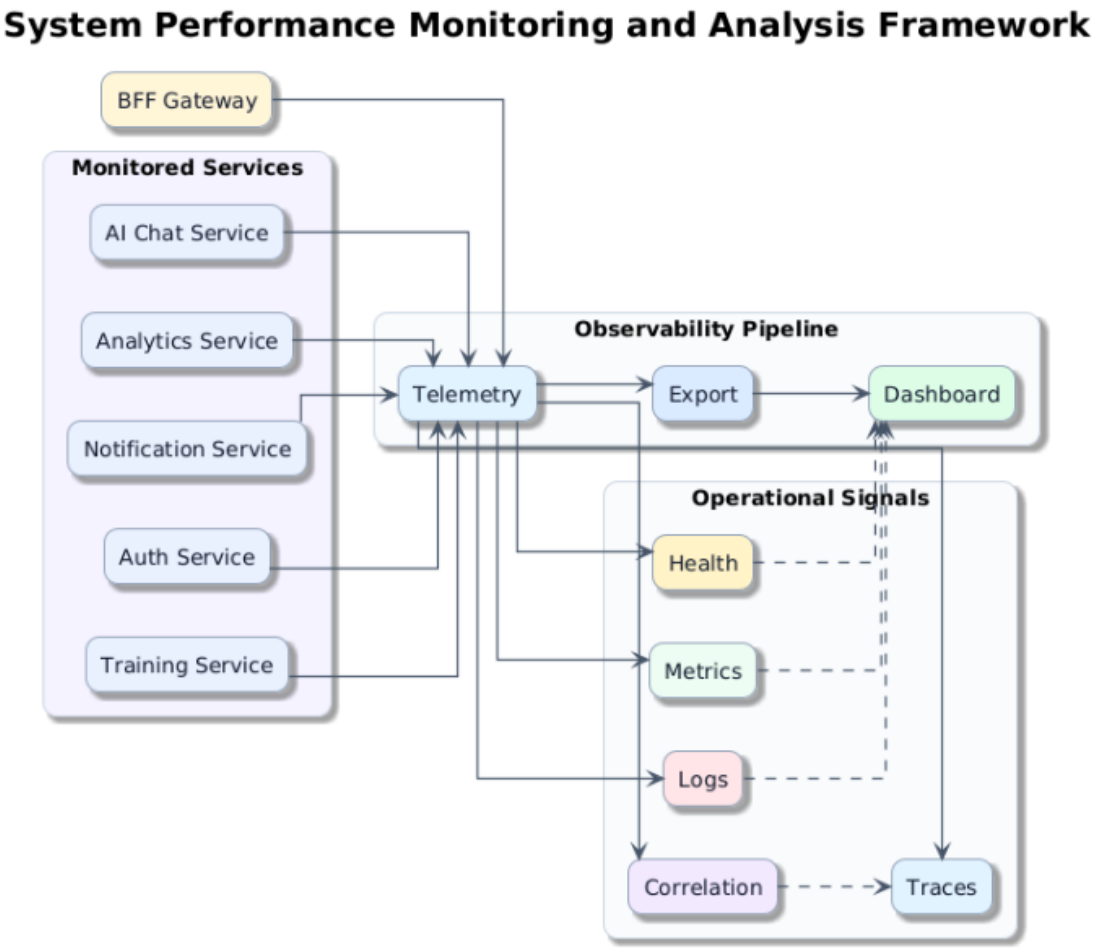

Рисунок 2.5 – Схема моніторингу та аналізу роботи системи

Отже, система моніторингу дозволяє контролювати роботу мікросервісного застосунку, аналізувати помилки та відстежувати виконання запитів між сервісами. Використання OpenTelemetry, Aspire Dashboard, health checks, correlation id та idempotency key робить систему більш прозорою для аналізу, тестування й подальшої підтримки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ ВІДСТЕЖЕННЯ ТРЕНУВАНЬ

3.1 Обґрунтування вибору середовища програмної реалізації

Для реалізації вебзастосунку відстеження навантаження та прогресу у тренуваннях було обрано сучасний стек технологій, який забезпечує високу продуктивність, масштабованість, зручність розробки та підтримку мікросервісної архітектури.

Розробка програмного продукту здійснювалася з використанням середовища Microsoft Visual Studio 2022 для серверної частини та Visual Studio Code для клієнтської частини. Дані середовища розробки є одними з найбільш поширених інструментів для створення сучасних вебзастосунків та забезпечують широкий набір засобів для написання, налагодження та тестування програмного коду. Загальна структура програмного рішення, що включає окремі мікросервіси [27], спільні бібліотеки та тестові проєкти, наведена на рис. 3.1.

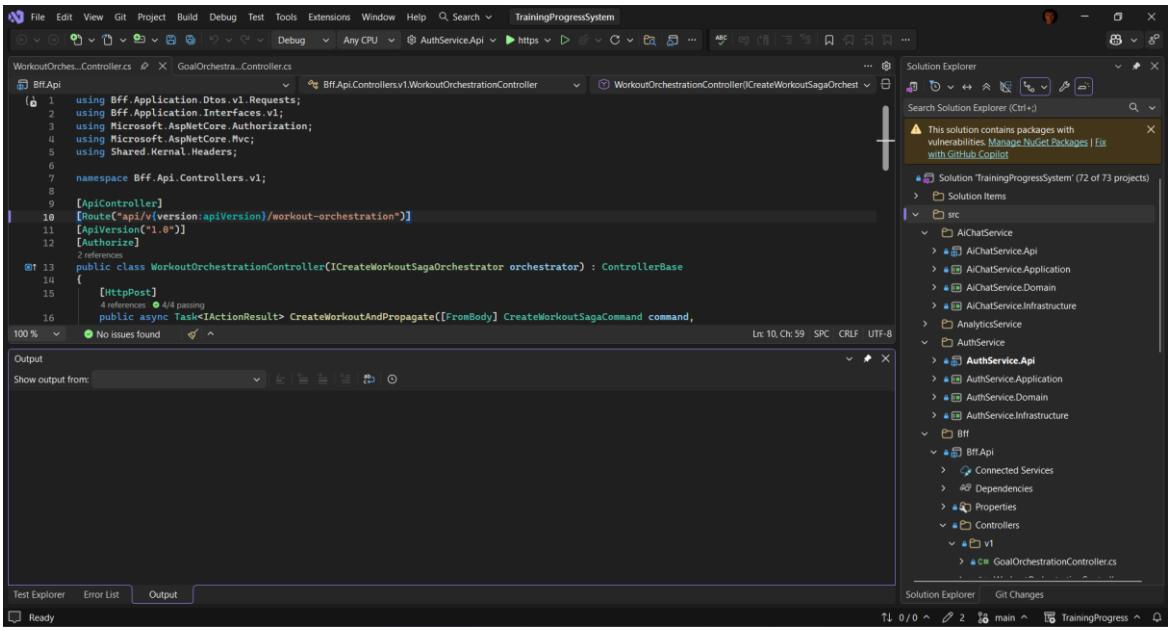

Рисунок 3.1 – Структура програмного рішення в середовищі Visual Studio 2022

Клієнтську частину застосунку реалізовано за допомогою React і TypeScript. React забезпечує створення динамічних користувацьких інтерфейсів на основі компонентного підходу, а TypeScript підвищує надійність коду завдяки статичній типізації.

Серверну частину розроблено на платформі ASP.NET Core Web API, яка забезпечує високу продуктивність і підтримує механізми автентифікації, авторизації та роботи з базами даних. Для зберігання даних використовується Microsoft SQL Server, а доступ до них реалізовано через Entity Framework Core.

Для розгортання застосунку використовується Docker [28], що забезпечує контейнеризацію компонентів системи, спрощує запуск у різних середовищах та полегшує супровід. Приклад розгорнутих контейнерів наведено на рис. 3.2.

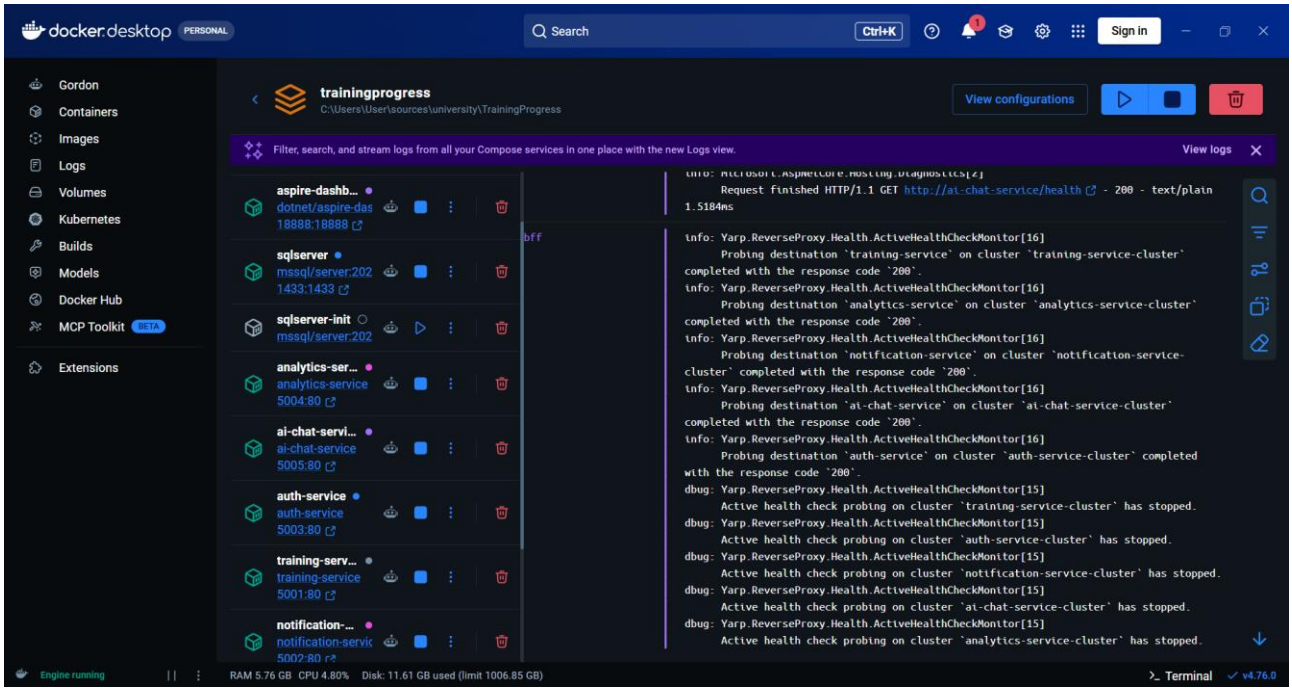

Рисунок 3.2 – Контейнери вебзастосунку в середовищі Docker Desktop

Для підвищення продуктивності системи використовується Redis, який застосовується для кешування даних та зменшення навантаження на базу даних. Крім того, для забезпечення обміну даними в режимі реального часу використовується технологія SignalR, що дозволяє автоматично оновлювати

інформацію в інтерфейсі користувача без необхідності повторного завантаження сторінок.

Моніторинг роботи сервісів та аналіз телеметричних даних реалізовано за допомогою OpenTelemetry у поєднанні з Aspire Dashboard. Використання даних інструментів дозволяє отримувати інформацію про продуктивність сервісів, час виконання запитів, кількість помилок та інші показники функціонування системи. Це значно спрощує процес виявлення та усунення можливих проблем під час експлуатації програмного продукту. Приклад панелі моніторингу наведено на рис. 3.3.

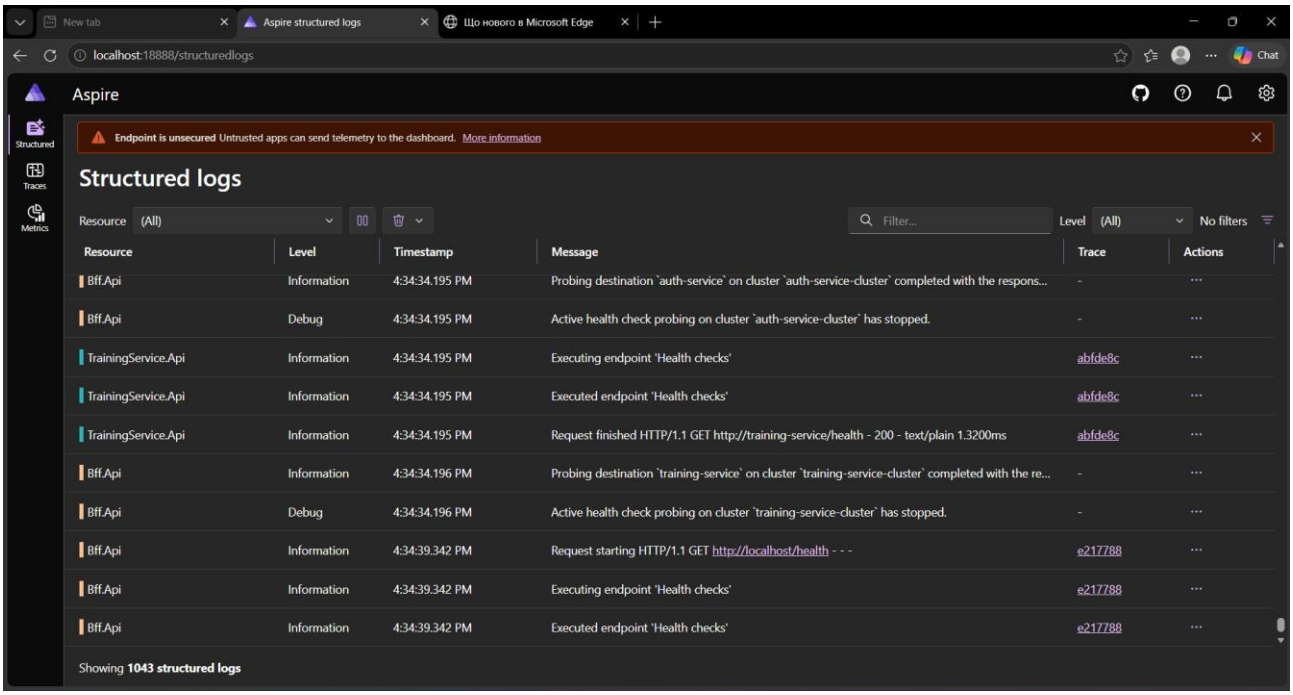

Рисунок 3.3 – Панель моніторингу Aspire Dashboard

Таким чином, обраний набір технологій та інструментів дозволяє створити сучасний вебзастосунок із високою продуктивністю, підтримкою масштабування, можливістю контейнерного розгортання та ефективними засобами моніторингу його роботи. Застосування зазначених технологій забезпечує надійність функціонування системи та створює передумови для її подальшого розвитку й розширення функціональних можливостей.

3.2 Реалізація системи автентифікації та авторизації користувачів

Однією з основних складових вебзастосунку є система автентифікації та авторизації користувачів, яка забезпечує безпечний доступ до функціональних можливостей системи та захист персональних даних користувачів.

Для роботи із застосунком користувач повинен пройти процедуру реєстрації, вказавши адресу електронної пошти та пароль. Інтерфейс сторінки реєстрації наведено на рис. 3.4.

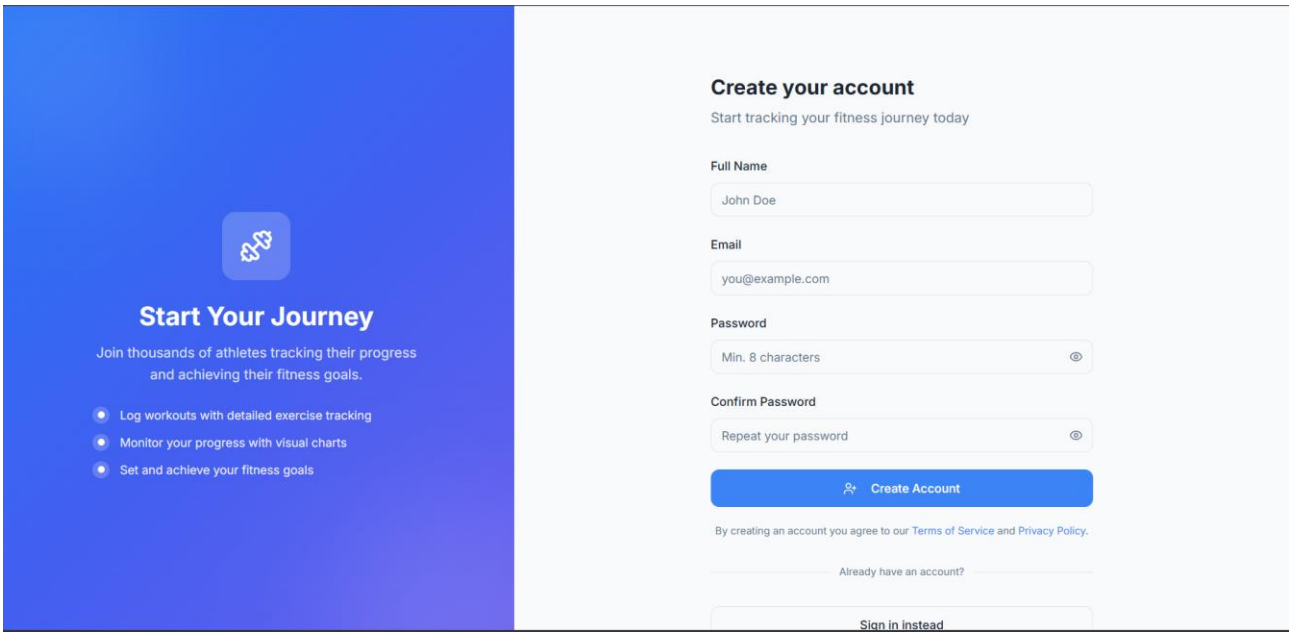

Рисунок 3.4 – Сторінка реєстрації користувача

Після успішного створення облікового запису користувач може виконати вхід до системи за допомогою власних облікових даних. Процес авторизації передбачає введення електронної пошти та пароля, після чого дані передаються на сервер для перевірки їх коректності. У випадку успішної валідації сервер формує JWT-токен доступу, який використовується для подальшої взаємодії користувача із захищеними ресурсами системи. Для зручності користувача та підвищення безпеки доступу реалізовано механізм автоматичного оновлення токена без необхідності повторного введення облікових даних. Для цього використовується Refresh Token, який зберігається у захищеному HTTP-only

cookie. Для виконання входу в систему реалізовано окрему сторінку авторизації, приклад якої наведено на рис. 3.5.

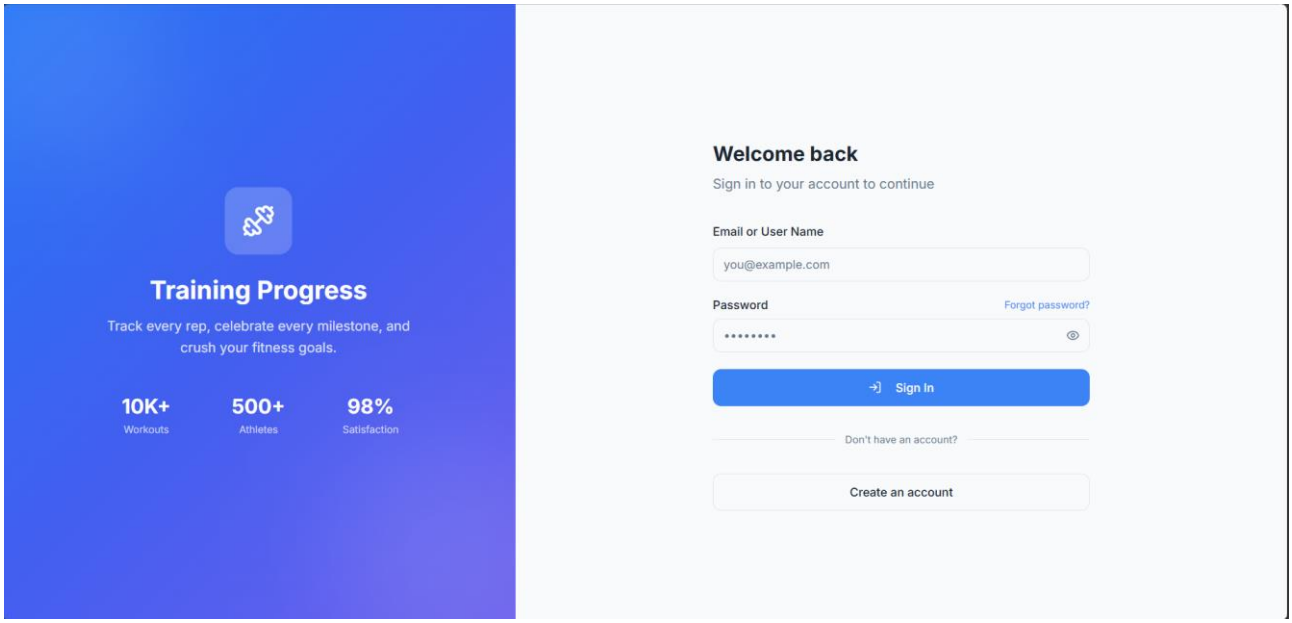

Рисунок 3.5 – Сторінка авторизації користувача

Під час авторизації сервер перевіряє коректність введених даних та формує JWT-токен доступу. Отриманий токен використовується для підтвердження особи користувача під час виконання запитів до захищених ресурсів системи. Для підвищення безпеки також використовується механізм Refresh Token, який зберігається у HTTP-only cookie та дозволяє автоматично оновлювати токен доступу без повторного введення облікових даних.

Для реалізації авторизації використовується механізм розмежування доступу до ресурсів застосунку. Незареєстровані користувачі мають доступ лише до сторінок входу та реєстрації, тоді як основний функціонал системи доступний лише після успішної автентифікації.

Після успішного входу користувач отримує доступ до особистого кабінету та функцій керування тренуваннями, цілями, статистикою та чат-ботом персонального тренера. Приклад головної сторінки після авторизації наведено на рис. 3.6.

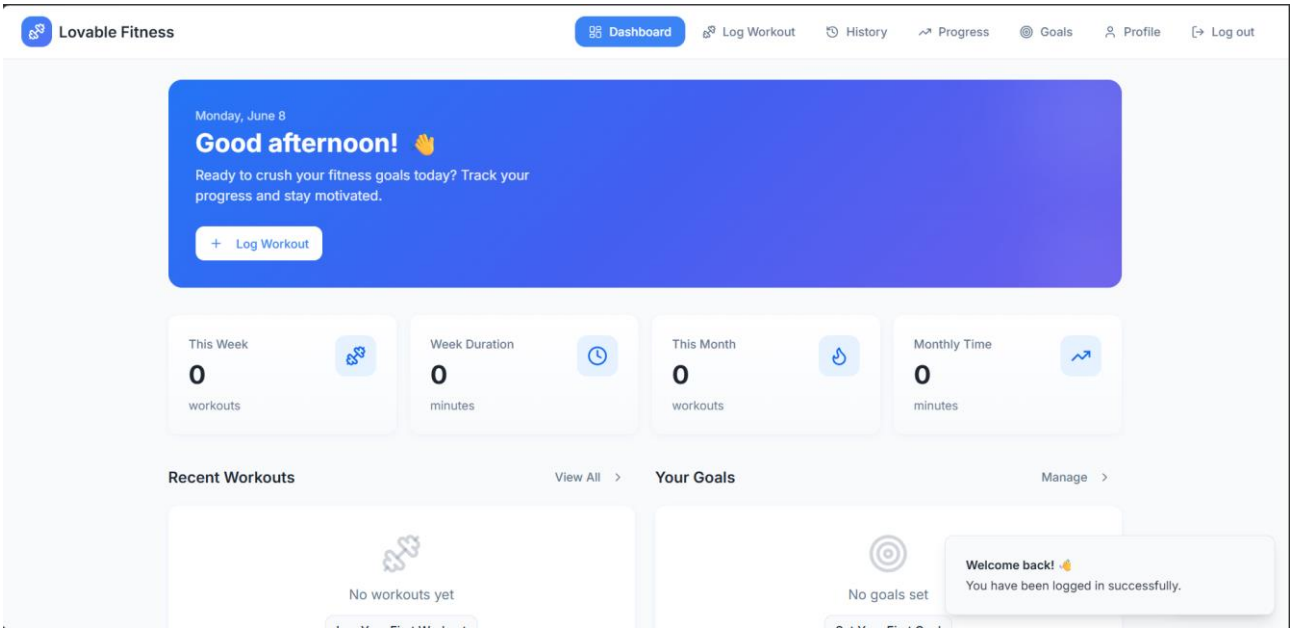

Рисунок 3.6 – Головна сторінка застосунку після входу в систему

Для завершення роботи із системою реалізовано функцію виходу з облікового запису. Після виконання даної операції токени користувача видаляються, а доступ до захищених ресурсів припиняється до повторної авторизації.

Таким чином, реалізована система автентифікації та авторизації забезпечує безпечний доступ до функціональних можливостей вебзастосунку та захист персональних даних користувачів.

3.3 Реалізація підсистеми керування тренуваннями

Однією з основних складових вебзастосунку є підсистема керування тренуваннями, яка забезпечує користувачеві можливість створювати, редагувати, видаляти та переглядати тренувальні сесії, а також детально фіксувати виконані вправи. Даний модуль є ключовим для збору даних, які надалі використовуються в аналітиці та системі відстеження прогресу користувача.

Процес створення тренування реалізовано через окрему форму, у якій користувач вказує основні параметри тренування, зокрема дату проведення, тип

тренування (Strength, Cardio, Flexibility, HIIT, Yoga, Sports, Other), тривалість та додаткові нотатки. Після підтвердження введених даних нове тренування зберігається у базі даних та стає доступним у загальному списку тренувань користувача. Приклад інтерфейсу створення тренування наведено на рис. 3.7.

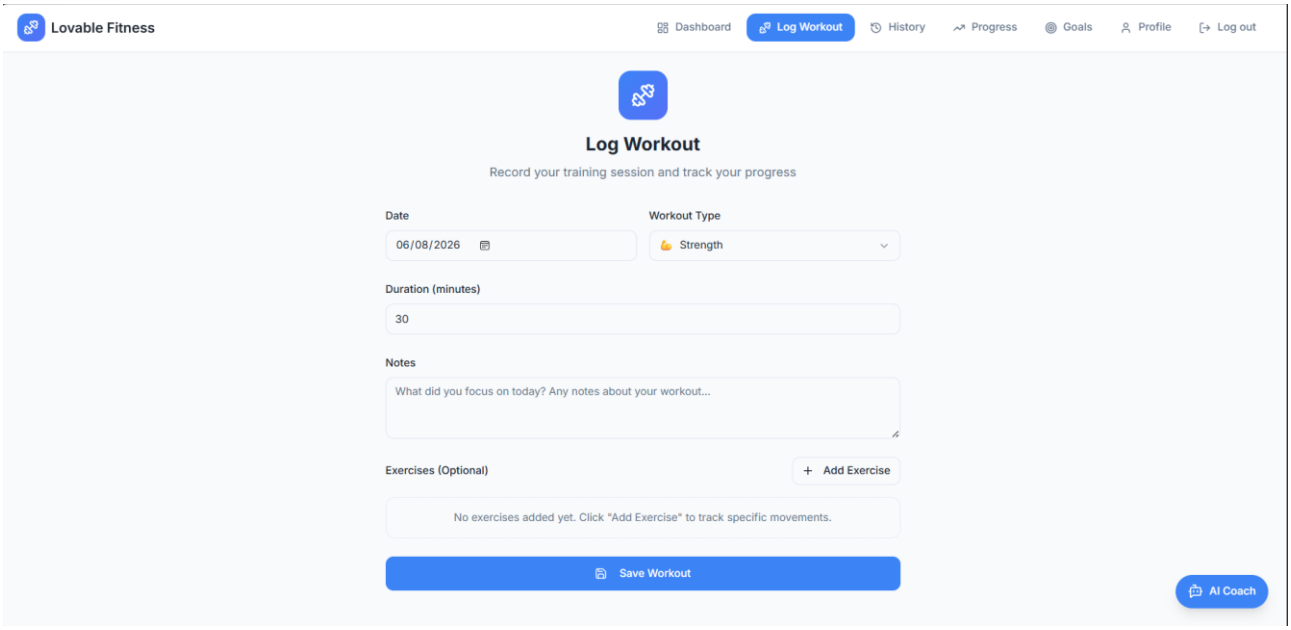

Рисунок 3.7 – Форма створення тренування

Кожне тренування може містити набір вправ, які деталізують фізичне навантаження користувача та дозволяють більш точно відображати виконану тренувальну активність. Для кожної вправи задаються такі параметри, як кількість підходів, повторень, вага та тривалість виконання. Збереження зазначених характеристик дає можливість не лише вести облік виконаних вправ, а й аналізувати динаміку фізичних показників користувача протягом тривалого періоду часу.

Додатково реалізовано категоризацію вправ (Strength, Cardio, Flexibility, Core), що дозволяє структурувати дані та спрощує подальший аналіз тренувального процесу. Використання категорій забезпечує більш зручний пошук вправ, покращує візуальне представлення інформації та створює основу для формування статистичних звітів і аналітичних показників. Крім того, такий

підхід дає можливість користувачу швидко орієнтуватися серед великої кількості записів та ефективніше планувати власний тренувальний процес.

Інтерфейс додавання вправ реалізовано у вигляді окремої форми з полями введення необхідних параметрів та механізмами перевірки коректності введених даних. Приклад інтерфейсу додавання вправ наведено на рис. 3.8.

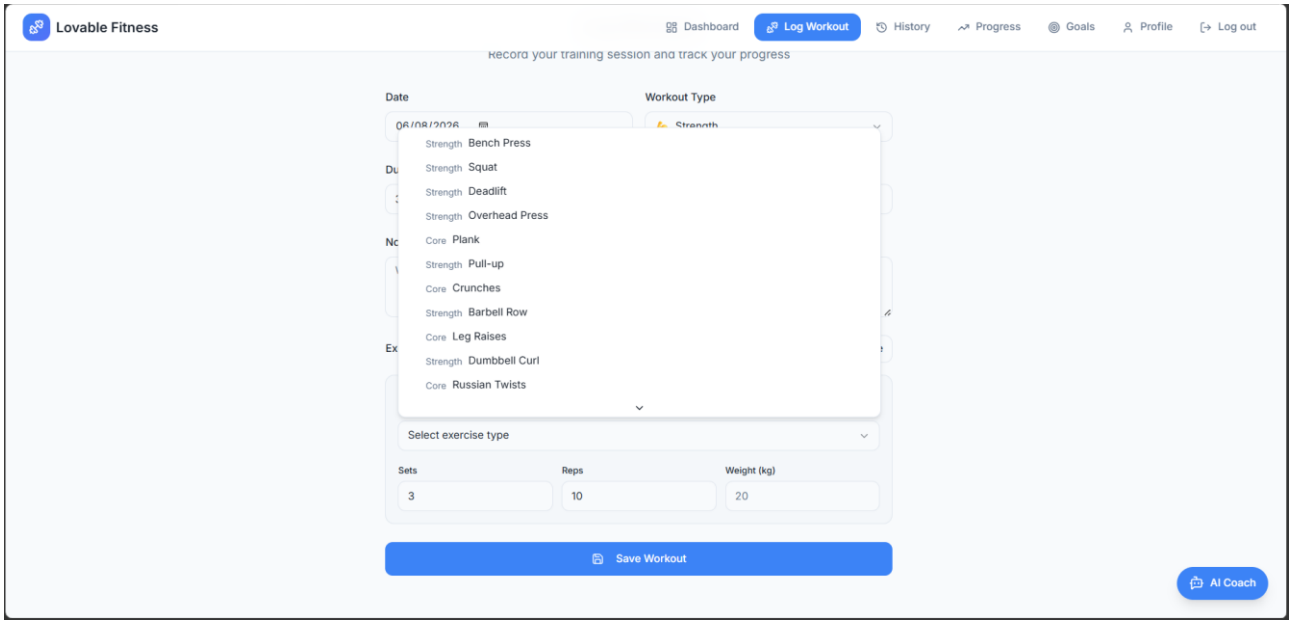

Рисунок 3.8 – Додавання вправ до тренування

Для перегляду історії тренувань реалізовано два режими відображення: списковий та календарний. Списковий режим дозволяє переглядати всі тренування у вигляді впорядкованого списку із сортуванням за датою у спадному порядку. Для кожного тренування відображається основна інформація, що дає змогу швидко оцінити виконану фізичну активність та отримати доступ до детальних відомостей про тренування.

Календарний режим забезпечує візуальне відображення тренувальної активності користувача за обраний період часу. Такий підхід дозволяє більш наочно аналізувати регулярність занять, відстежувати динаміку тренувального процесу та контролювати виконання поставлених цілей. Наявність двох режимів перегляду надає користувачу можливість обирати найбільш зручний спосіб роботи з історією тренувань залежно від власних потреб та вподобань.

Приклад спискового перегляду історії тренувань наведено на рис. 3.9.

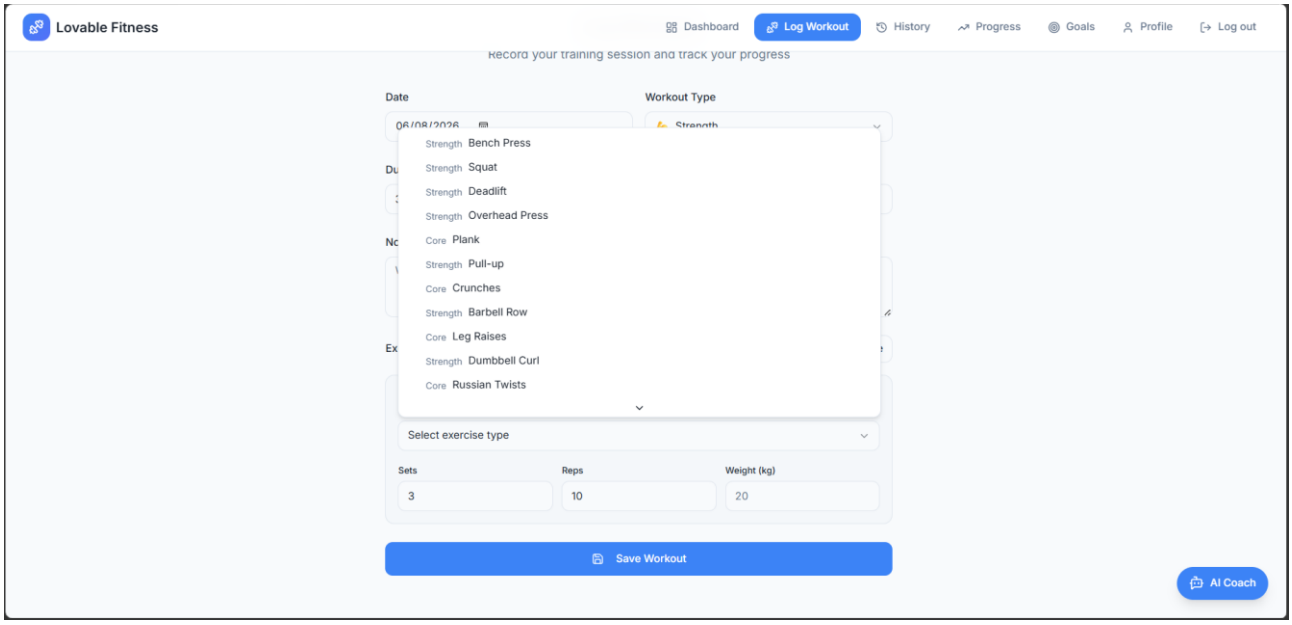

Рисунок 3.9 – Список тренувань користувача

Календарний вигляд дозволяє оцінити регулярність тренувань та їх розподіл у часі, що є зручним для аналізу довгострокової активності користувача. Усі тренування відображаються безпосередньо в календарі відповідно до дати їх проведення, що дозволяє швидко визначити дні з найбільшою або найменшою фізичною активністю. Такий спосіб представлення інформації сприяє більш наочному сприйняттю тренувального процесу та допомагає користувачу контролювати дотримання власного плану занять.

Використання календарного режиму також дає можливість виявляти пропуски у тренуваннях, оцінювати стабільність фізичної активності та аналізувати зміни навантаження протягом тривалого періоду часу. Це особливо корисно під час досягнення довгострокових спортивних цілей та контролю особистого прогресу.

Приклад календарного представлення тренувань наведено на рис. 3.10.

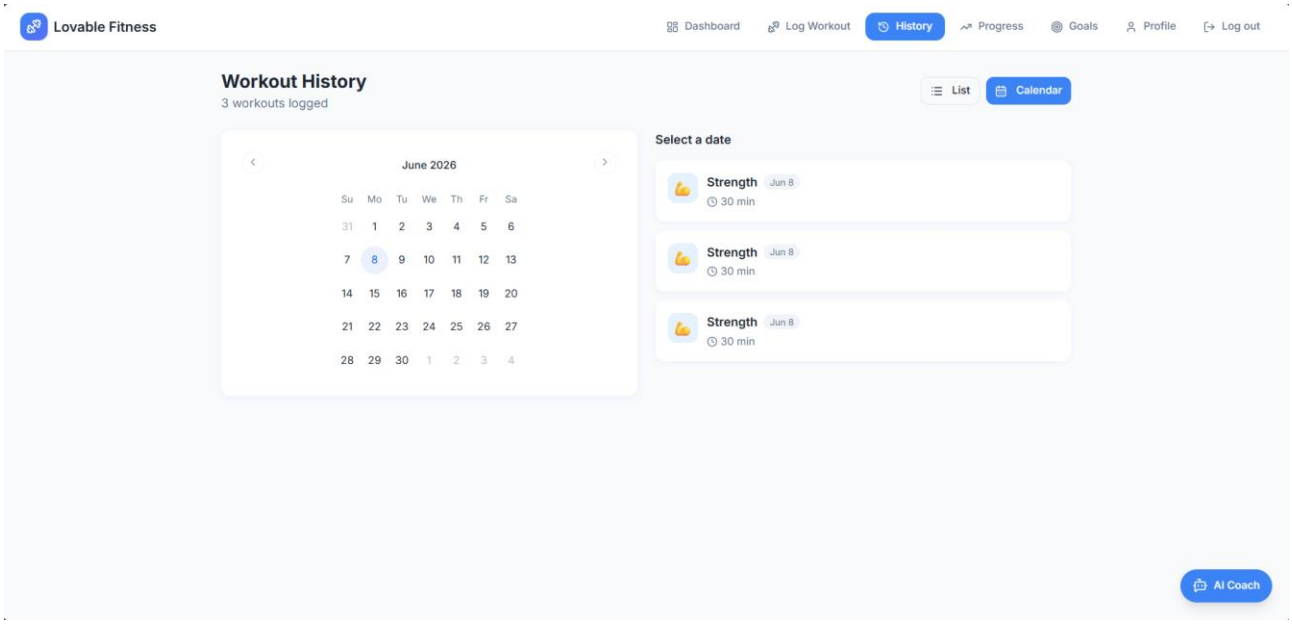

Рисунок 3.10 – Календарний вигляд тренувань

Список тренувань підтримує серверну пагінацію з використанням OData, що дозволяє ефективно працювати з великими обсягами даних. За замовчуванням записи відсортовані за датою у порядку спадання, що забезпечує швидкий доступ до останніх тренувань користувача. Приклад реалізації пагінації наведено на рис. 3.11.

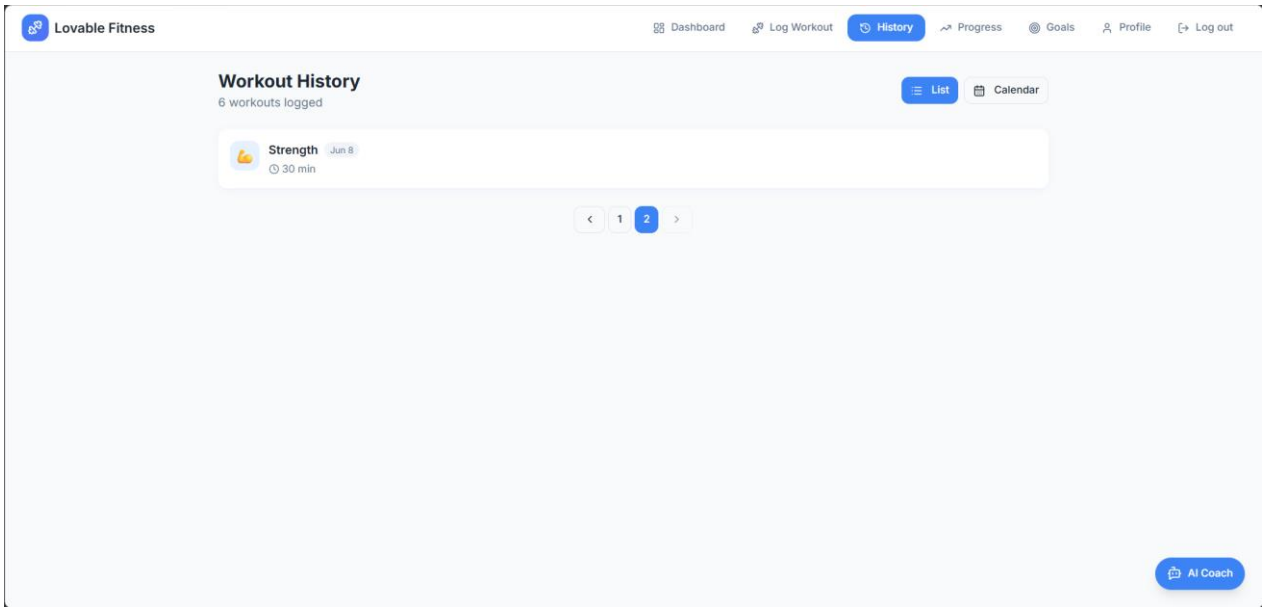

Рисунок 3.11 – Пагінація списку тренувань

Таким чином, реалізована підсистема керування тренуваннями забезпечує повний цикл роботи з тренувальними даними, включаючи їх створення, редагування, видалення та перегляд, а також інтеграцію з аналітичними модулями та системою цілей користувача.

3.4 Реалізація підсистеми цілей та нагадувань

Однією з функціональних складових вебзастосунку є підсистема керування фітнес-цілями, яка дозволяє користувачеві встановлювати тренувальні цілі та відстежувати прогрес їх виконання на основі фактичної фізичної активності.

Створення цілі реалізовано через окрему форму, де користувач задає тип метрики, цільове значення та часовий період. Система підтримує вісім типів метрик і чотири типи періодів: тижневий, місячний, кастомний діапазон дат та ковзне вікно 30 днів. Інтерфейс створення цілі наведено на рис. 3.12.

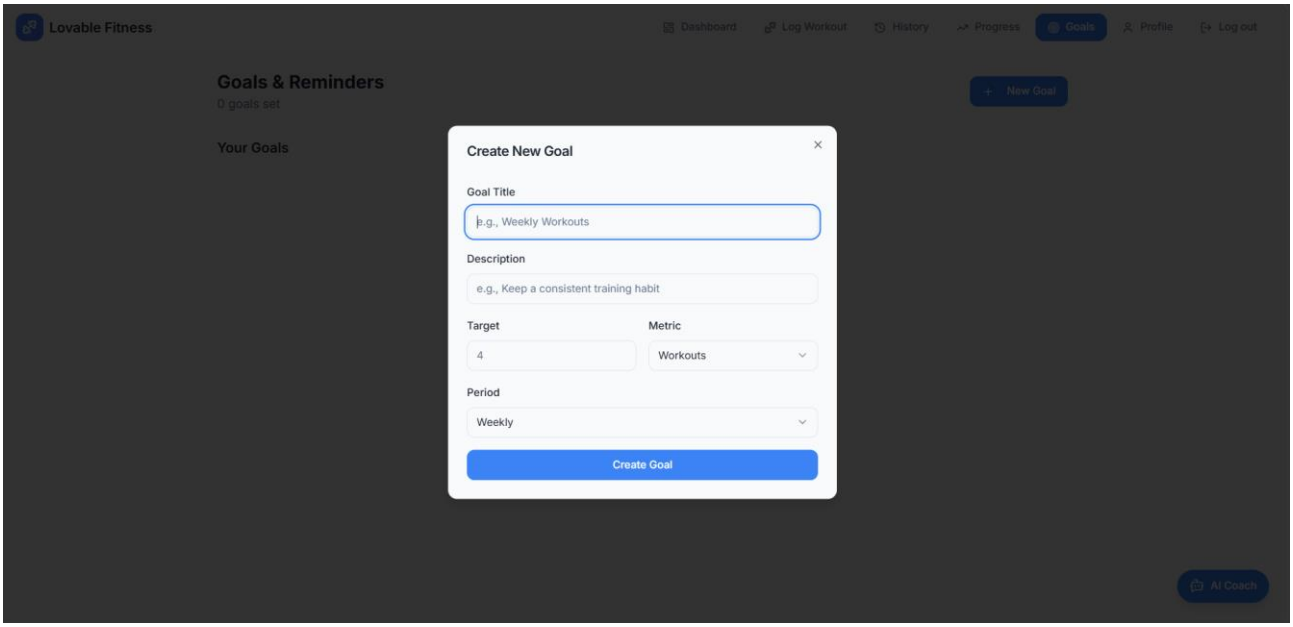

Рисунок 3.12 – Форма створення фітнес-цілі

Після створення цілі система автоматично ініціює процес відстеження її виконання. Поточний прогрес розраховується як відношення фактичного значення до запланованого та відображається у вигляді відсоткового індикатора, що дозволяє користувачеві в режимі реального часу оцінювати ступінь наближення до досягнення поставленої мети. Приклад списку цілей із відображенням прогресу наведено на рис. 3.13.

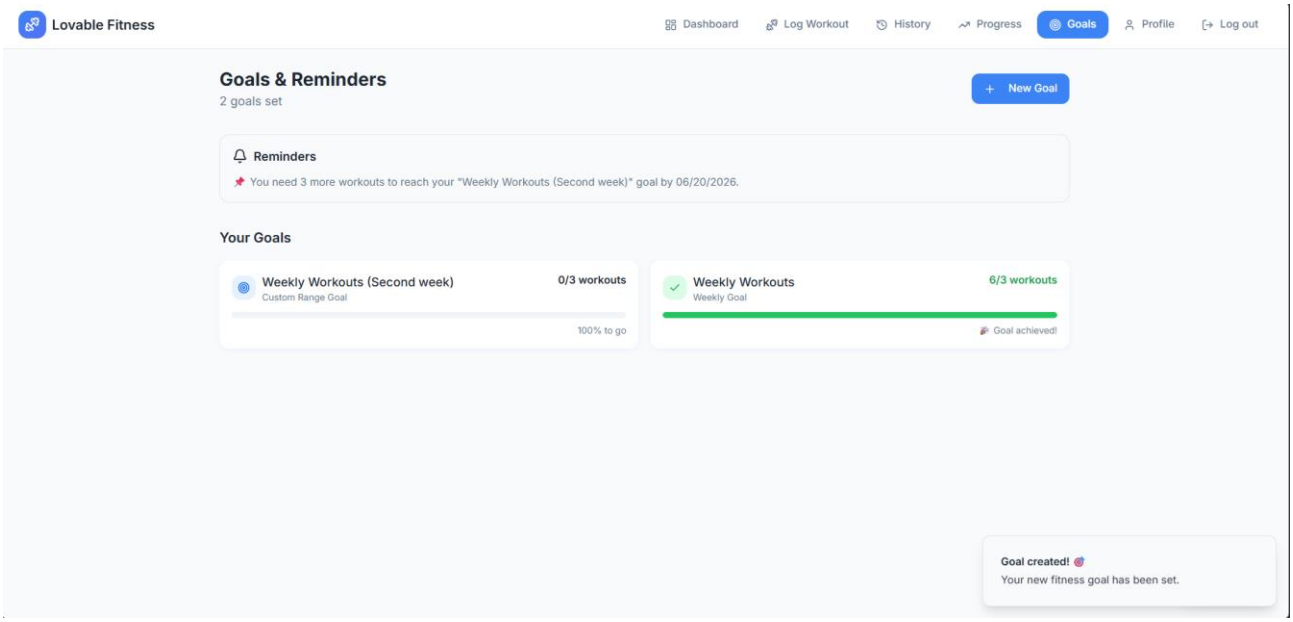

Рисунок 3.13 – Відображення списку цілей та прогресу виконання

Кожна ціль у системі має власний життєвий цикл, який включає стани Active, Completed та Cancelled. Перехід між станами відбувається автоматично при досягненні необхідних умов або вручну за рішенням користувача. Такий підхід дозволяє гнучко керувати поставленими цілями та підтримувати актуальність інформації про поточний стан тренувального процесу.

У разі досягнення цільового значення система автоматично переводить ціль у стан виконаної, що додатково фіксується в аналітичних даних користувача. Це дає можливість відстежувати досягнення, оцінювати ефективність тренувань та аналізувати особистий прогрес протягом тривалого періоду часу. Приклад відображення статусів цілей наведено на рис. 3.14.

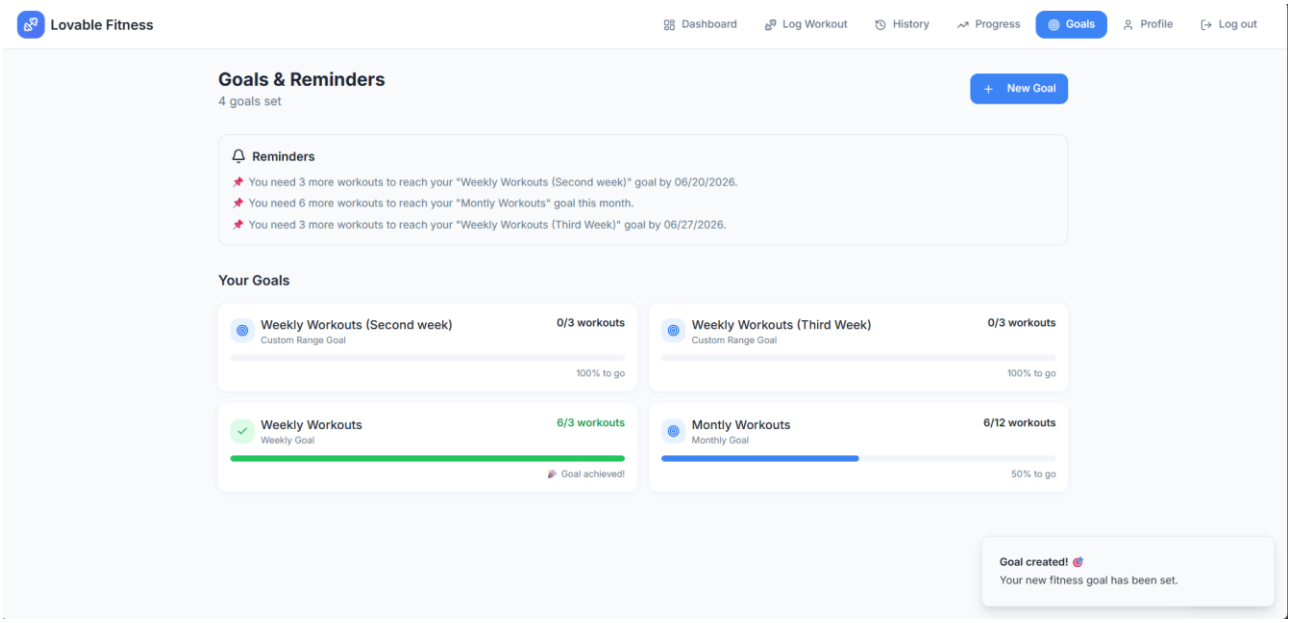

Рисунок 3.14 – Стани життєвого циклу цілей користувача

Важливою складовою підсистеми є система нагадувань, яка формує контекстні повідомлення на основі поточного прогресу користувача. Наприклад, якщо користувач не виконав достатню кількість тренувань для досягнення тижневої цілі, система автоматично формує відповідне нагадування із зазначенням необхідної різниці. Після кожного нового тренування нагадування оновлюються відповідно до актуального стану прогресу. Приклад системного повідомлення наведено на рис. 3.15.

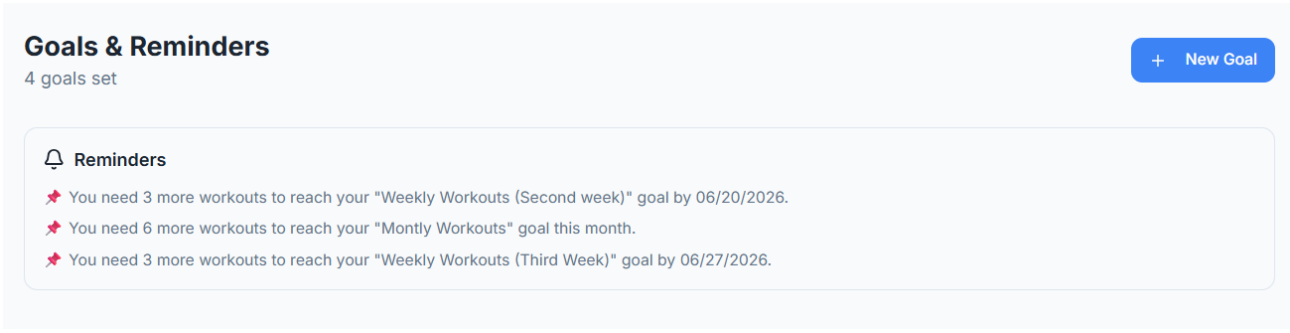

Рисунок 3.15 – Приклад системного нагадування щодо прогресу цілі

Оновлення прогресу цілей відбувається автоматично після кожного доданого або зміненого тренування. Це забезпечує синхронізацію даних між підсистемами та дозволяє користувачеві отримувати актуальну інформацію без

необхідності перезавантаження сторінки або ручного оновлення інтерфейсу. Приклад автоматичного оновлення прогресу наведено на рис. 3.16.

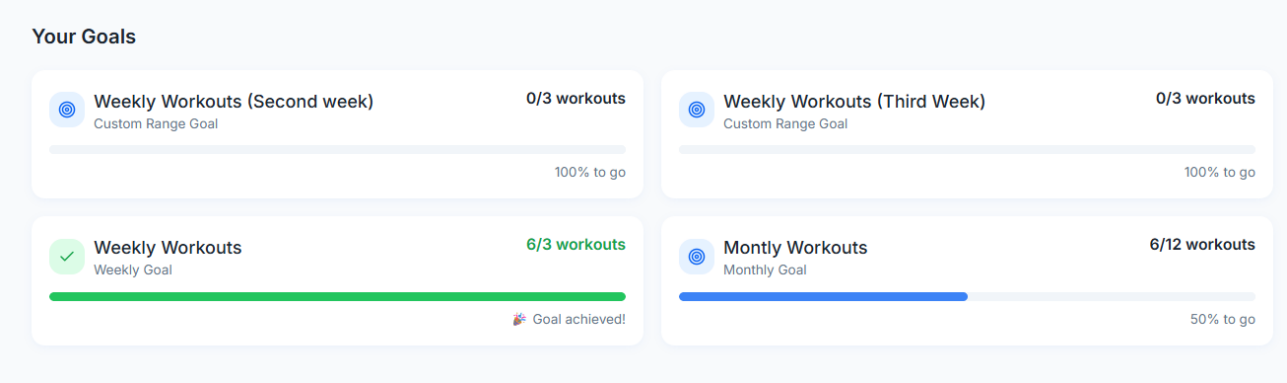

Рисунок 3.16 – Автоматичне оновлення прогресу цілей

Таким чином, підсистема цілей та нагадувань забезпечує користувачеві можливість формувати персональні орієнтири тренувального процесу, відстежувати їх виконання та отримувати автоматичний зворотний зв’язок у вигляді прогресу та системних повідомлень, що підвищує ефективність використання вебзастосунку.

3.5 Реалізація аналітики та моніторингу прогресу

Підсистема аналітики та моніторингу прогресу є важливою складовою вебзастосунку, яка забезпечує користувача узагальненою інформацією про його фізичну активність та дозволяє оцінювати ефективність тренувального процесу на основі статистичних даних. Даний модуль агрегує інформацію з підсистеми тренувань та цілей і формує на її основі набір графіків та показників, що відображають загальну динаміку активності користувача.

Основною сторінкою підсистеми є аналітичний Dashboard, на якому відображаються ключові показники, зокрема кількість тренувань за тиждень та місяць, загальна кількість тренувань та сумарна тривалість тренувальних сесій. Даний підхід дозволяє користувачеві швидко отримати загальне уявлення про

рівень своєї фізичної активності без необхідності перегляду окремих записів. Приклад реалізації Dashboard наведено на рис. 3.17.

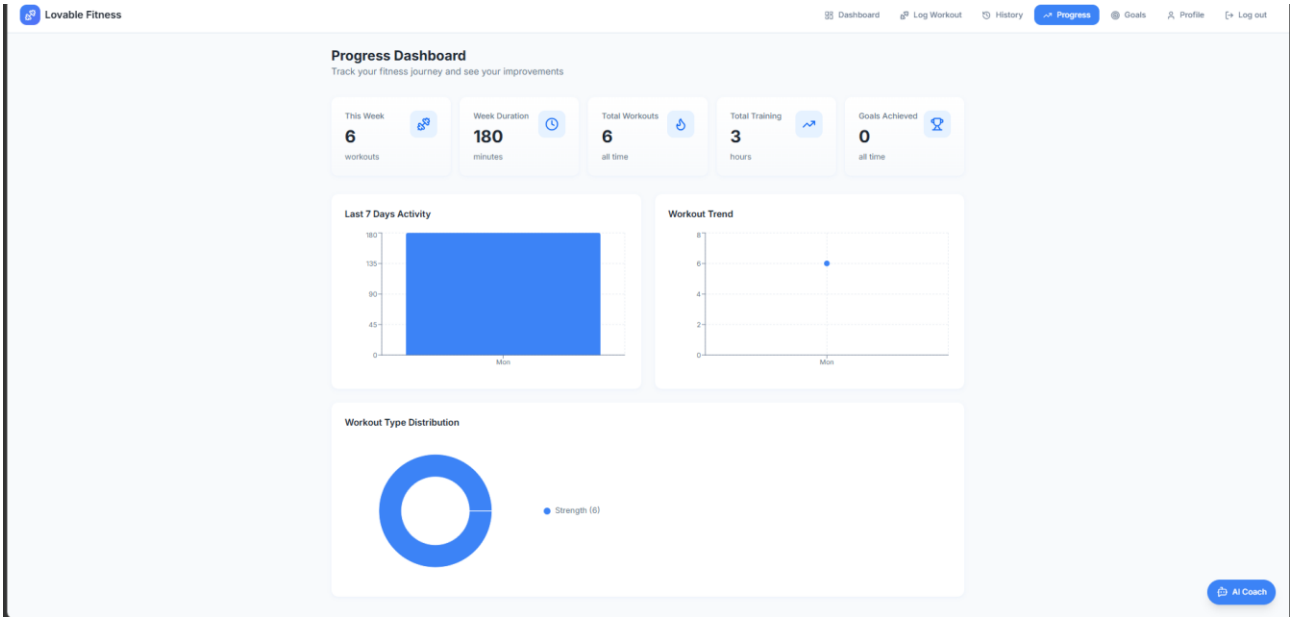

Рисунок 3.17 – Головна сторінка аналітичного Dashboard

Для аналізу статистики в системі реалізовано графічне представлення даних. Стовпчаста діаграма відображає тренувальну активність за останні 7 днів, а кругова діаграма демонструє розподіл типів тренувань. Приклад відповідних графіків наведено на рис. 3.18.

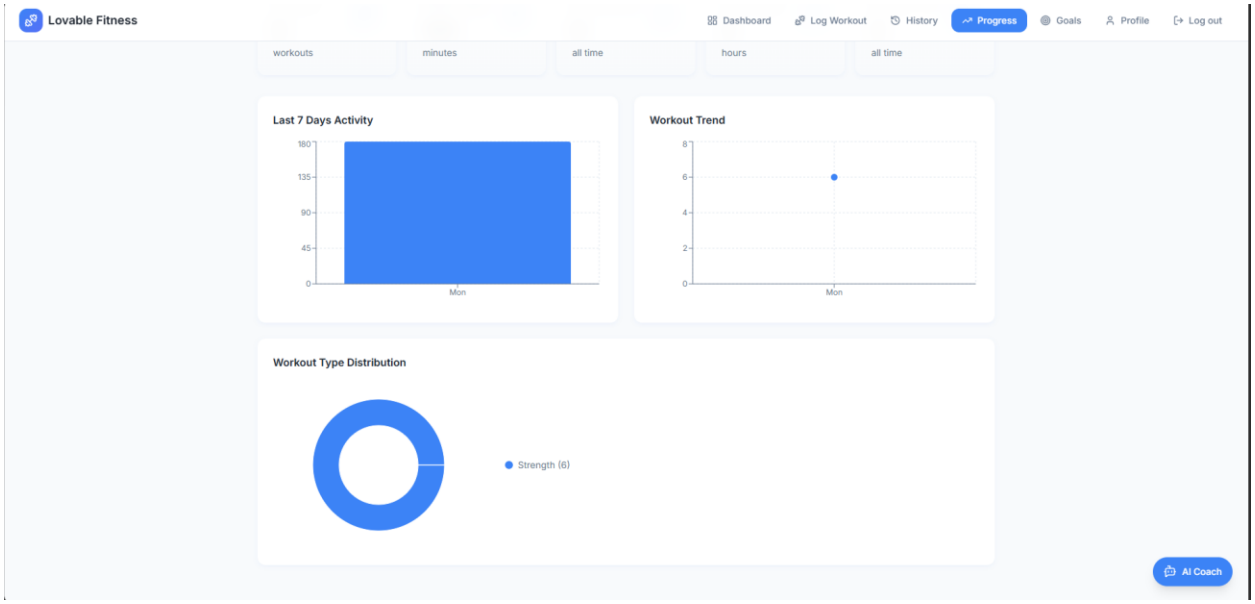

Рисунок 3.18 – Графічне відображення розподілу та тренду тренувань

Також у системі реалізовано лінійний графік, який відображає зміну тривалості тренувань у часі. Це дозволяє користувачеві аналізувати, як змінюється інтенсивність тренувального процесу залежно від періоду, та оцінювати власний прогрес у довгостроковій перспективі. Окремо передбачено графік кількості тренувань за типами, що дозволяє більш детально проаналізувати структуру фізичної активності.

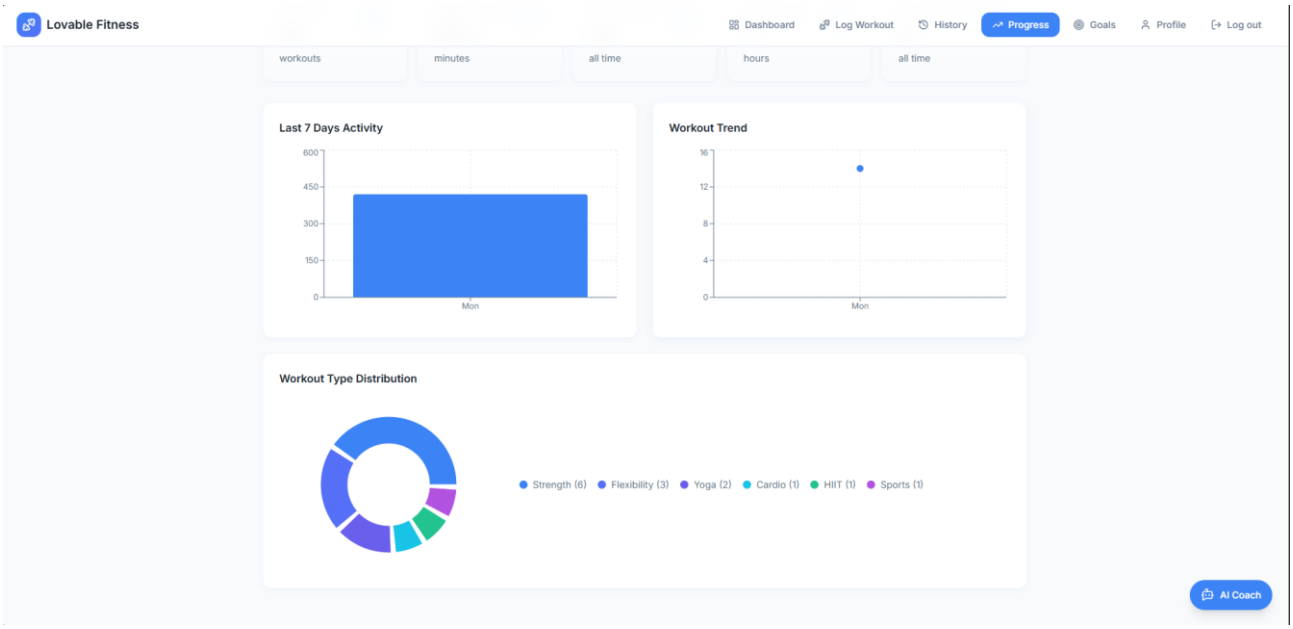

Рисунок 3.19 – Лінійна динаміка тривалості тренувань та розподіл за типами

Таким чином, підсистема аналітики забезпечує користувачеві зручний інструмент для оцінки фізичної активності та аналізу прогресу. Це дозволяє відстежувати результати тренувань і контролювати досягнення поставлених цілей.

3.6 Реалізація чат-бота персонального тренера

Однією з найбільш інноваційних складових вебзастосунку є підсистема AI-чатбота персонального тренера, яка забезпечує користувачеві можливість

отримувати персоналізовані рекомендації щодо тренувального процесу, планування фізичних навантажень та загальних порад щодо підтримки активного способу життя. Даний модуль інтегровано з мовною моделлю Groq LLM [29], що дозволяє формувати контекстно-залежні відповіді у режимі реального часу.

Взаємодія користувача з чат-ботом реалізована у форматі потокового обміну повідомленнями (SSE – Server-Sent Events), що забезпечує поступове відображення відповіді моделі без необхідності очікування повного завершення генерації. Такий підхід покращує користувацький досвід, оскільки створює ефект живого діалогу та зменшує затримки при отриманні відповідей. Інтерфейс чату дозволяє користувачеві вводити запити у довільній формі та отримувати рекомендації щодо тренувань, відновлення та загальної фізичної активності. Приклад інтерфейсу чат-бота наведено на рис. 3.20.

Під час роботи системи зберігається контекст останніх 10 повідомлень, що дозволяє підтримувати логіку розмови та забезпечувати узгодженість відповідей у межах одного діалогу. Крім того, реалізовано механізм збереження історії повідомлень у базі даних, що дозволяє користувачеві переглядати попередні діалоги та повертатися до них у будь-який момент. Загальний обсяг збереженої історії обмежений 50 повідомленнями для оптимізації продуктивності системи.

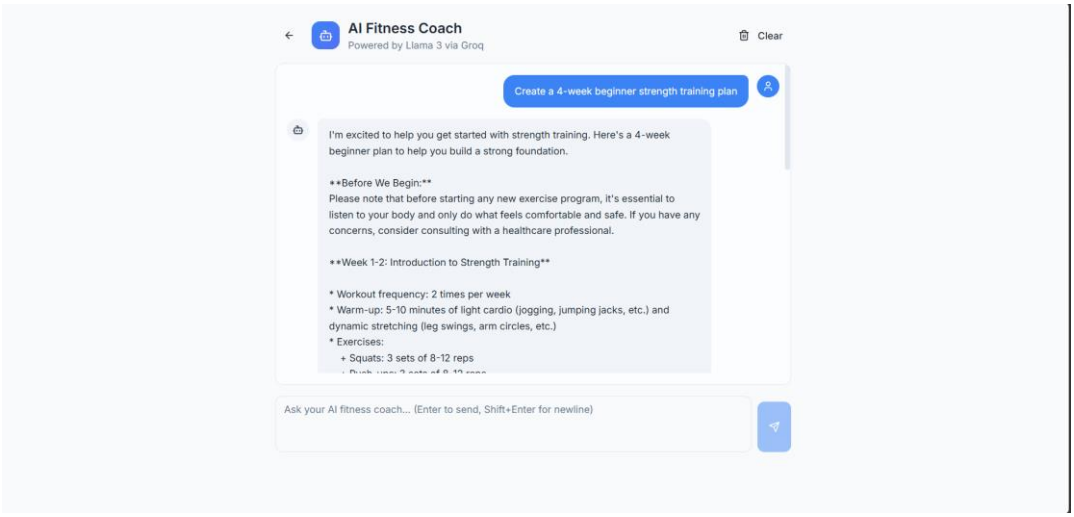

Рисунок 3.20 – Інтерфейс AI-чатбота персонального тренера

Додатково передбачено функцію очищення історії чату, яка дозволяє користувачеві видалити попередні повідомлення та почати новий діалог без збереження попереднього контексту. Також реалізовано набір швидких стартових запитів (suggested prompts), які допомагають користувачеві швидше сформулювати перші звернення до системи та отримати приклади можливих сценаріїв використання.

Таким чином, підсистема AI-чатбота персонального тренера забезпечує інтерактивну взаємодію користувача із системою, розширює функціональні можливості вебзастосунку та підвищує рівень персоналізації рекомендацій, що сприяє більш ефективному досягненню фітнес-цілей.

3.7 Тестування та результати роботи застосунку

Тестування вебзастосунку є важливим етапом перевірки коректності реалізації програмної системи та забезпечення стабільності її роботи. У межах даного проєкту основний акцент зроблено на автоматизованому тестуванні з використанням фреймворку xUnit, який дозволяє реалізувати модульні та інтеграційні тести для всіх ключових компонентів мікросервісної архітектури. Використання автоматизованого тестування дає можливість своєчасно виявляти помилки та контролювати якість програмного забезпечення під час його подальшого розвитку [30].

Unit-тестування застосовувалося для перевірки окремих елементів бізнес-логіки, зокрема сервісів автентифікації, управління тренуваннями, цілей та аналітики. Даний тип тестів дозволяє ізолювати функціональність окремих методів і перевірити їх коректність незалежно від інших частин системи. Завдяки цьому забезпечується надійність реалізації основних сценаріїв роботи застосунку та зменшується ймовірність виникнення помилок під час експлуатації.

Інтеграційне тестування використовувалося для перевірки взаємодії між мікросервісами та коректності обміну даними між ними. Особлива увага приділялася сценаріям створення та оновлення тренувань, оскільки вони впливають на роботу підсистем аналітики, цілей та нагадувань. Це дозволяє перевірити коректність роботи всієї бізнес-логіки в комплексі та підтвердити правильність взаємодії між окремими компонентами системи.

Окремо виконувалася перевірка API-сценаріїв, що забезпечує відповідність HTTP-запитів очікуваним результатам, коректну обробку помилок та правильність повернення кодів відповіді. Такий підхід гарантує стабільну взаємодію між клієнтською та серверною частинами системи та забезпечує коректну роботу користувацького інтерфейсу.

Результати виконання автоматизованих тестів представлені у середовищі розробки Visual Studio у вікні Test Explorer, де всі тести успішно проходять перевірку. Це підтверджує коректність реалізації бізнес-логіки, працездатність основних функціональних можливостей та відсутність критичних помилок у системі. Приклад успішного виконання тестів наведено на рис. 3.21.

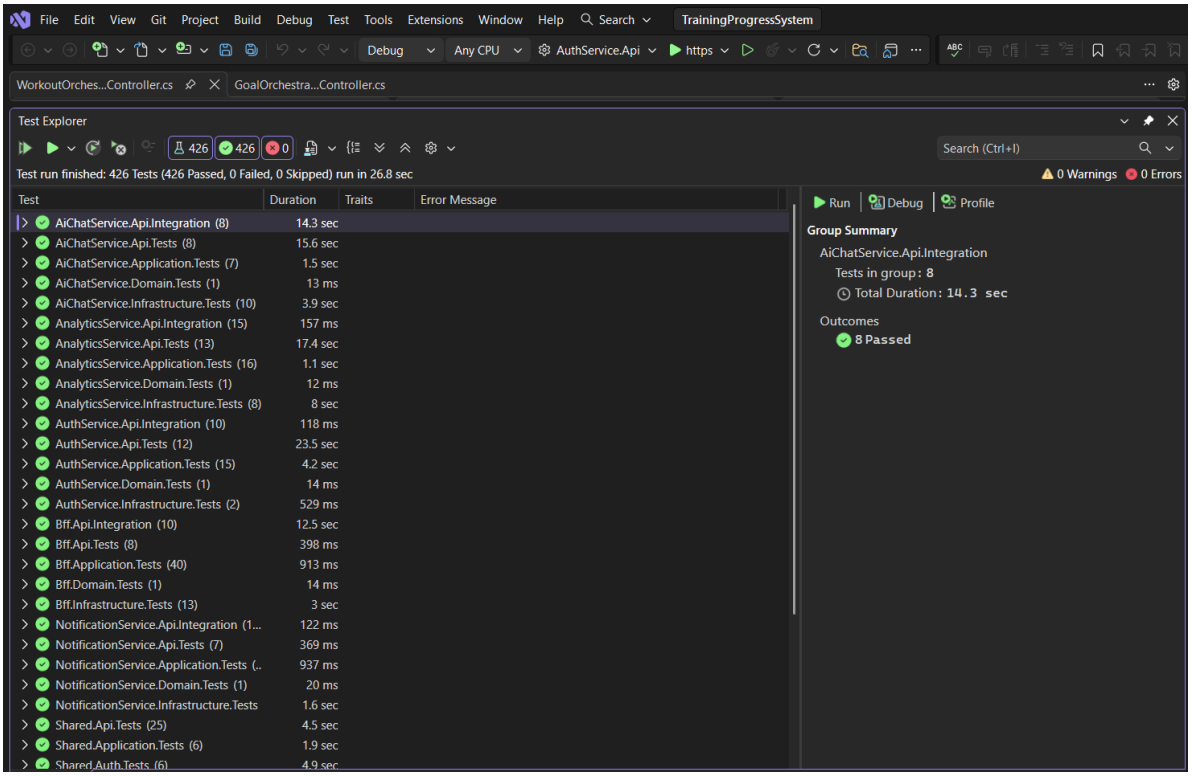

Рисунок 3.21 – Результати виконання unit та integration тестів у Test Explorer

Таким чином, результати тестування підтверджують коректність роботи всіх основних модулів вебзастосунку, стабільність взаємодії між мікросервісами та готовність системи до використання в реальних умовах. Успішне проходження модульних, інтеграційних та API-тестів свідчить про правильність реалізації бізнес-логіки та коректність обміну даними між компонентами системи.

Проведене тестування також підтвердило працездатність механізмів автентифікації та авторизації, управління тренуваннями, формування цілей, аналітики та AI-чатбота персонального тренера. Отримані результати демонструють надійність функціонування програмного забезпечення та його здатність забезпечувати стабільну роботу при виконанні основних користувацьких сценаріїв. Це дозволяє зробити висновок про відповідність розробленої системи поставленим вимогам та можливість її подальшого практичного використання.

ВИСНОВКИ

Таким чином, у ході виконання кваліфікаційної роботи було досліджено сучасні підходи до розробки вебзастосунків для відстеження фізичної активності та вирішено такі завдання:

− проаналізовано сучасні застосунки для трекінгу тренувань, визначено їхні переваги та недоліки;

− досліджено підходи до побудови вебзастосунків і мікросервісної архітектури та обрано відповідні технології;

− спроєктовано структуру вебзастосунку та взаємодію його компонентів;

− розроблено вебзастосунок для відстеження тренувань і аналізу фізичної активності користувача;

− реалізовано чат-бота як персонального тренера для взаємодії та рекомендацій;

− проведено модульне й ручне тестування програмного забезпечення.

У межах кваліфікаційної роботи створено систему керування тренувальними даними на основі сучасних вебтехнологій, що забезпечує зручний аналіз прогресу користувача та зберігання інформації про тренування.

Результатом є вебзастосунок для автоматизації трекінгу тренувань, аналізу фізичних показників і керування тренувальними даними, який може бути основою для подальшого розвитку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Liu, Y., & Avello, M. (2021). Status of the research in fitness apps: A bibliometric analysis. Telematics and Informatics, 57, 101506.

2. The intention to use fitness and physical activity apps: A systematic review. (2020). Sustainability, 12(16), 6641.

3. Düking, P., Achtzehn, S., Holmberg, H.-C., & Sperlich, B. (2018). Integrated framework of load monitoring by smartphone applications, wearables and point-of-care testing. Sensors, 18(5), 1632.

4. Elder, A., Guillen, G., Isip, R., Zepeda, R., & Lewis, Z. (2023). A deeper look into exercise intensity tracking through mobile applications: A brief report. Technologies, 11(3), 66.

5. Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

6. Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley.

7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.

8. Sommerville, I. (2015). Software engineering (10th ed.). Pearson.

9. OpenTelemetry Authors. (2025). OpenTelemetry documentation.

10. NET Foundation. (2025). .NET documentation.

11. Richardson, L., & Amundsen, M. (2013). RESTful web APIs. O'Reilly Media.

12. Microsoft. (2025). Entity Framework Core documentation.

13. Microsoft. (2025). YARP reverse proxy documentation.

14. gRPC Authors. (2025). gRPC documentation.

15. React Team. (2025). React documentation.

16. O'Reilly, T. (2022). Modern web application development with React and TypeScript. TechPress.

17. TypeScript Team. (2025). TypeScript documentation.

18. Redux Team. (2025). Redux documentation

19. Docker Inc. (2025). Docker documentation.

20. Microsoft. (2025). Docker integration with .NET applications.

21. Redis Ltd. (2025). Redis documentation.

22. Microsoft. (2025). ASP.NET Core documentation.

23. Microsoft. (2025). SignalR documentation.

24. . Martin, R. C. (2017). Clean architecture: A craftsman's guide to software structure and design. Prentice Hall.

25. Martin, R. C. (2008). Clean code: A handbook of agile software craftsmanship. Prentice Hall.

26. Methods of monitoring internal and external loads and their relationships with physical qualities, injury, or illness in adolescent athletes: A systematic review and best-evidence synthesis. (2023). Sports Medicine.

27. Newman, S. (2021). Building microservices (2nd ed.). O'Reilly Media.

28. Kubernetes Authors. (2025). Kubernetes documentation.

29. OpenAI. (2025). GPT models documentation.

30. IEEE Computer Society. (2023). Software testing principles and practices in distributed systems.