

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки

Факультет _____ комп'ютерних наук _____
(повна назва)

Кафедра _____ програмної інженерії _____
(повна назва)

КОМПЛЕКСНИЙ КУРСОВИЙ ПРОЄКТ
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський) _____

Програмна система для моніторингу та оцінки техніки виконання фізичних вправ
(тема)

Виконав:

здобувач (ка) 3 курсу, групи ПЗПІ-22-3
Євген ТКАЧЕНКО

(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного
забезпечення

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна _____

Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник _____ доцент кафедри ПІ Дмитро
КОЛЕСНИКОВ

(посада, Власне ім'я, ПРІЗВИЩЕ)

Члени комісії (Власне ім'я, ПРІЗВИЩЕ,
підпис)

Володимир ЛЕЩИНСЬКИЙ

Ольга ВОРОЧЕК

Віталій ЛЯПОТА

2025 р.

Харківський національний університет радіоелектроніки

Факультет _____ комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
 Кафедра _____ програмної інженерії
 Рівень вищої освіти _____ перший (бакалаврський)
 Спеціальність _____ 121 – Інженерія програмного забезпечення
 Тип програми _____ Освітньо-професійна
 Освітня програма _____ Програмна Інженерія
 (шифр і назва)

Курс 3Група ПЗП-22-3Семестр 6

ЗАВДАННЯ
на курсовий проект(роботу) студента

здобувачеві _____ Ткаченку Євгену Андрійовичу
 (прізвище, ім'я, по батькові)

1. Тема роботи _____ Програмна система для моніторингу та оцінки техніки виконання фізичних вправ

2. Термін здачі студентом закінченої роботи „_____” _____ 2025 р.

3. Вихідні _____ дані _____ до _____ проекту

4. Перелік питань, що потрібно опрацювати в роботі

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	10.05.2025	
2	Розробка постановки задачі	12.05.2025	
3	Проектування ПЗ	13.05.2025	
4	Розробка проєкту	13.05.2025	
5	Аналіз отриманих результатів	10.06.2025	
6	Підготовка пояснювальної записки.	11.06.2025	
7	Перевірка на наявність ознак академічного плагіату		
8	Захист роботи		

Дата видачі завдання “___” _____ 2025р.

Здобувач  Євген ТКАЧЕНКО
(підпис)

Керівник роботи _____ доцент кафедри ПІ Дмитро КОЛЕСНИКОВ
(підпис) (посада, Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 75 стор., 38 рис., 10 джерел.

Ключові слова: ВЕБ-ПЛАТФОРМА, КОНТЕЙНЕРИЗАЦІЯ, МОНІТОРИНГ ВПРАВ, ПІДТЯГУВАННЯ, РЕКОМЕНДАЦІЇ, DOCKER COMPOSE, ESP32, IOT, JAVA, NGINX, POSTGRESQL, REST API, SPRING BOOT, TAILWINDCSS, VUE.JS.

Об'єкт дослідження – процес автоматизованого моніторингу та оцінки техніки виконання підтягувань за допомогою IoT-сенсорів і веб-технологій.

Мета роботи – розробити контейнеризовану веб-систему, що в реальному часі фіксує метрики руху (кількість повторень, висоту підйому, кут тулуба), візуалізує їх і генерує персоналізовані рекомендації для покращення техніки.

Технічна реалізація.

Серверна частина створена на Java 17 із застосуванням Spring Boot REST API; дані зберігаються у PostgreSQL. Frontend реалізовано як SPA на Vue 3 (Vite) з TailwindCSS, Pinia та Chart.js. IoT-модуль емулюється на ESP32 (Wokwi) і надсилає телеметрію через HTTP POST. Усі компоненти (PostgreSQL, backend, frontend, nginx-reverse-proxy) об'єднані в інфраструктуру Docker Compose.

Результати дослідження. Створено прототип, який:

- збирає та зберігає метрики фізичної активності;
- визначає відхилення техніки та пропонує рекомендації;
- надає користувачам інтерактивні графіки висоти та кута тіла, а також історію сесій;
- дозволяє адмініструвати користувачів і переглядати зведену статистику;

- сприяє підвищенню ефективності тренувань і зменшенню ризику травм.

Keywords: CONTAINERIZATION, DOCKER COMPOSE, ESP32, EXERCISE MONITORING, IOT, JAVA, NGINX, POSTGRESQL, PULL-UPS, RECOMMENDATIONS, REST API, SPRING BOOT, TAILWINDCSS, VUE.JS, WEB PLATFORM.

The object of the research is the automated monitoring and assessment of pull-up technique using IoT sensors and modern web technologies.

The aim of the work is to design and implement a containerized web system that captures motion metrics in real time (repetition count, lift height, torso angle), visualises them and generates personalised recommendations to improve exercise technique.

Technical implementation.

The backend is developed in Java 17 with Spring Boot REST API; PostgreSQL serves as the DBMS. The frontend is a Vue 3 (Vite) SPA styled with TailwindCSS, state-managed by Pinia and visualised via Chart.js. Sensor data are emulated on an ESP32 (Wokwi) and streamed to the server via HTTP POST. All components—PostgreSQL, backend, frontend and an NGINX reverse proxy—are integrated using Docker Compose.

Research outcomes. The resulting prototype:

- collects and stores exercise metrics;
- detects technique deviations and issues actionable recommendations;
- delivers interactive charts of pull-up height and torso angle alongside session history;
- enables centralised user administration and aggregate statistics;
- enhances training efficiency and helps lower injury risk.

ЗМІСТ

Перелік скорочень	8
Вступ.....	9
1 Аналіз предметної галузі	11
1.1 Аналіз предметної галузі.....	111
1.2 Виявлення та вирішення проблем	12
1.2.1 Цільова аудиторія.....	14
1.3 Аналіз аналогів програмного забезпечення	15
2 Постановка задачі.....	19
3 Архітектура та проектування програмного забезпечення	20
3.1 UML проектування ПЗ.....	20
3.2 Проектування архітектури ПЗ	22
3.3 Проектування структури зберігання даних	24
3.4 Приклади використаних алгоритмів та методів	25
4 Опис прийнятих програмних рішень	28
4.1 Опис прийнятих інфраструктурних рішень	28
4.2 Опис рішень прийнятих для доступу до бази даних	29
4.3 Опис рішень індивідуального та колективного прийняття рішень	31
4.4 Опис рішень шару бізнес логіки.....	32
5 Аналіз отриманих результатів	35
5.1 Порівняння результатів із початковими вимогами	35
5.2 Оцінка якості роботи системи	36
5.3 Аналіз ефективності прийнятих рішень	36
5.4 Обмеження та недоліки	37
5.4 Візуалізація результатів	37
Висновки	42
Перелік джерел посилання	44
Додаток А Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ	45
Додаток Б Слайди презентації	46

Додаток В Специфікація програмного забезпечення.....	52
Додаток Г UML діаграми та зображення інтерфейсу	67

ПЕРЕЛІК СКОРОЧЕНЬ

API – application programming interface – програмний інтерфейс прикладних застосунків

CI/CD – continuous integration / continuous delivery – неперервна інтеграція та доставляння

CRUD – create, read, update, delete – базові операції над даними

DB – database – база даних

ESP32 – embedded system platform 32-bit – мікроконтролер для IoT-застосунків

HTTP – hypertext transfer protocol – протокол передавання гіпертексту

IoT – internet of things – інтернет речей

JWT – json web token – компактний формат токенів автентифікації

NGINX – engine x – високопродуктивний HTTP-сервер і зворотний проксі

REST – representational state transfer – архітектурний стиль веб-сервісів

SPA – single-page application – односторінковий веб-застосунок

SQL – structured query language – мова структурованих запитів

UI – user interface – користувацький інтерфейс

UX – user experience – користувацький досвід

UML – unified modeling language – уніфікована мова моделювання

UUID – universally unique identifier – універсально унікальний ідентифікатор

Wokwi – Web-based Online Keystroke-simulator With Integration – симулятор апаратури (ESP32 тощо)

DBMS – database management system – система керування базами даних

VCS – version control system – система керування версіями

SPA – single-page application – односторінковий застосунок

ВСТУП

Останнє десятиліття позначилося бумом домашніх тренувань і персонального фітнес-моніторингу: згідно з даними міжнародного порталу Statista, ринок «connected fitness» з 2017 р. зростає в середньому на 12 % щороку, а кількість користувачів IoT-пристроїв для спорту перевищила 300 млн. Утім, попри доступність смарт-годинників і трекерів, техніку виконання силових вправ більшість ентузіастів все ще оцінює «на око» або за допомогою ручних записів у мобільних нотатках. Відсутність точних даних про кут тулуба чи амплітуду руху призводить до повторення помилок, відсутності прогресу та підвищеного ризику травм.

Така ситуація формує попит на доступний інструмент, здатний у реальному часі вимірювати біомеханічні параметри підтягувань, зберігати їх у хмарі та видавати зрозумілі рекомендації.

Ключовими викликами є:

- Коректний збір метрик — синхронний вимір висоти підйому й кута тулуба без дорогого обладнання;
- Миттєвий зворотний зв'язок — виявлення помилок прямо під час виконання вправи;
- Персоналізовані поради — адаптація рекомендацій під прогрес і цілі конкретного користувача;
- Візуальна аналітика — наочні графіки, що дозволяють відстежувати динаміку тренувань;
- Адміністрування даних — захищене збереження сесій, користувачів та історії метрик з можливістю резервного копіювання.

Метою курсового проєкту є створення контейнеризованої веб-системи, яка об'єднає IoT-модуль на ESP32, серверний Spring Boot REST API та клієнтський SPA на Vue 3, щоб автоматизувати моніторинг підтягувань і покращити техніку виконання вправ.

Передбачені три основні ролі:

- Користувач отримує особистий кабінет з графіками висоти та кута, історією сесій і рекомендаціями;
- Адміністратор керує обліковими записами;
- Система IoT-сенсорів автоматично передає телеметрію.

Стек реалізації включає Java 17 + Spring Boot, PostgreSQL, Vue 3 (Vite) + TailwindCSS + Pinia + Chart.js, а також Docker Compose з NGINX-reverse-проху, що забезпечує незалежне розгортання компонентів.

Практичний результат упровадження— зниження травматизму, об'єктивна оцінка прогресу та підвищення мотивації спортсменів завдяки миттєвому фідбеку. Наукова цінність роботи полягає у поєднанні IoT-телеметрії з веб-технологіями для підтримки коректної біомеханіки силових вправ, що може стати основою для подальших досліджень у галузі цифрових фітнес-рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Сучасний ринок фітнес-послуг і рішень для тренувань демонструє стійке зростання. За даними Statista, у 2023 році понад 300 млн людей у світі використовували IoT-пристрої та мобільні сервіси для моніторингу фізичної активності. Популярність домашніх тренувань зросла особливо після пандемії COVID-19, коли значна частина тренерів і спортсменів перейшла до онлайн-форматів. Разом із цим виникла потреба у засобах, здатних не лише рахувати кроки чи пульс, а й оцінювати техніку виконання вправ — як-от підтягування, віджимання або присідання.

На відміну від класичних фітнес-трекерів (наприклад, розумних годинників), які здебільшого вимірюють кількісні показники (пульс, кількість кроків, час), коректний аналіз техніки вимагає якісних біомеханічних параметрів: положення тіла, кути нахилу, амплітуду рухів. Без точних метрик користувач ризикує закріпити помилкову техніку, що призводить до неефективних тренувань або травм.

На сьогодні більшість спортсменів-аматорів користуються ручними методами фіксації тренувань: блокноти, Google Sheets, або мобільні нотатки. Частина фітнес-додатків дозволяє вести журнал сесій, проте не надає можливості візуально оцінити виконання рухів у динаміці, або не інтегрується з датчиками положення тіла. У результаті користувач бачить лише загальні підсумки, але не може з'ясувати, який елемент техніки потребує корекції.

Крім того, професійні системи біомеханічного аналізу (на зразок Vicon, Kinovea, Notch) залишаються надто дорогими та складними у використанні для індивідуальних спортсменів або локальних тренажерних залів. Відсутність доступного рішення на базі простих IoT-компонентів (ESP32, сенсори) обмежує цифрову трансформацію аматорського фітнесу.

Не менш важливим є і інтерфейс зворотного зв'язку. Навіть якщо система отримує дані, корисність їх залежить від способу подання. Спортсмену потрібна інтуїтивно зрозуміла візуалізація (графіки підйому, кута, серії), доступ до історії та персоналізовані рекомендації, які враховують динаміку й типові відхилення у техніці.

Таким чином, предметна галузь вимагає рішення, яке поєднає:

- збір телеметрії з простих IoT-пристроїв;
- аналіз техніки на основі параметрів руху;
- збереження результатів і тренувальних сесій;
- інтерактивний інтерфейс з рекомендаціями;
- адміністрування облікових записів.

Розробка такої вебсистеми здатна покрити нішу між фітнес-гаджетами та дорогими лабораторіями, запропонувавши ефективний, доступний та масштабований інструмент для покращення якості тренувань.

1.2 Виявлення та вирішення проблем

У сучасному фітнес-середовищі, особливо в сегменті аматорських тренувань, спостерігається низка недоліків у контролі техніки виконання вправ, що впливають на якість тренувального процесу, ефективність прогресу та безпеку спортсменів. Значна частина цих проблем пов'язана з відсутністю доступних засобів автоматизованого біомеханічного аналізу, недостатньою інтеграцією IoT-технологій у повсякденний спорт та обмеженістю існуючих цифрових платформ.

Основні проблеми:

- Більшість тренувальних застосунків фіксують лише загальні метрики (повторення, тривалість), але не враховують кут нахилу тулуба, повну амплітуду підйому або стабільність виконання, що є критичними для вправ на зразок підтягувань;

- Користувач не отримує зворотного зв'язку в реальному часі про помилки у техніці, а виправлення відбувається надто пізно або не відбувається взагалі.
- Дані тренувань часто зберігаються вручну або у вигляді скріншотів із трекерів. Відсутня систематизація історії сесій, агрегованої статистики, або порівняння між періодами.
- Графіки висоти руху чи кута тулуба зазвичай не входять до функціоналу простих додатків, а професійні рішення — дорогі та складні. Це позбавляє користувача розуміння прогресу та кореляції помилок з метриками.
- Більшість систем не мають алгоритмів, які адаптують поради під індивідуальні особливості користувача, його історію вправ та виявлені порушення техніки.
- Для адміністраторів фітнес-клубів немає інструментів для обліку користувачів, перегляду сесій, статистики ефективності, що обмежує застосування таких систем у груповій роботі.

Запропоноване рішення: Створення вебсистеми, яка поєднує IoT-модуль на ESP32, backend-аналіз рухів та візуальний frontend, дозволяє комплексно усунути зазначені проблеми. Основні функціональні блоки включають:

- приймання телеметрії (висота підйому, кут тулуба) у реальному часі;
- алгоритми виявлення відхилень у техніці;
- генерацію індивідуальних рекомендацій;
- інтерактивний інтерфейс з графіками рухів і історією сесій;
- панель адміністратора для керування користувачами;
- особистий кабінет спортсмена з візуалізацією прогресу та сповіщеннями.

Таким чином, розроблена система дозволяє:

- підвищити безпеку та ефективність тренувань;
- надати користувачу зрозумілий фідбек щодо техніки;

- створити цифрову базу для оцінки й корекції рухів;
- сформувати фундамент для подальшої інтеграції штучного інтелекту у фітнес-аналітику.

1.2.1 Цільова аудиторія

Слід описати основну цільову аудиторію вебсистеми для моніторингу та оцінки техніки виконання фізичних вправ. Така платформа орієнтована на широкий спектр користувачів, адже контроль якості тренувань і профілактика травм актуальні незалежно від віку, статі чи рівня підготовки. Утім, провідною групою залишаються молоді спортсмени-аматори, які активно займаються фітнесом удома чи в залі й звикли до цифрових сервісів. Саме ця категорія найбільше зацікавлена у швидкому отриманні зворотного зв'язку, наочних графіках прогресу та персональних рекомендаціях, що допомагають удосконалювати техніку підтягувань.

Окрім молоді, значну частку аудиторії становлять люди середнього віку, котрі мають усталені спортивні звички й прагнуть безпечного тренувального процесу. Для них важливими є стабільність роботи сервісу, простий інтерфейс без перевантаження зайвими функціями та можливість систематизувати власні дані про сесії й показники руху. Ця група цінує чіткі візуалізації параметрів, можливість порівнювати результати за періодами й експортувати статистику для консультацій з лікарем або тренером.

В окрему групу слід виділити професійних тренерів та менеджерів фітнес-залів. Тренери зацікавлені у платформі, що дозволяє під'єднувати кількох клієнтів, відстежувати їх технічні показники у реальному часі й швидко виявляти типові помилки. Менеджерам потрібен інструмент для централізованого керування обліковими записами, контролю апаратних модулів ESP32 та формування узагальненої статистики зали. Для них критичною є рольова модель доступу, аналітичні дашборди й легке масштабування системи через контейнеризовану інфраструктуру.

Таким чином, цільова аудиторія платформи є доволі різноманітною, охоплюючи як індивідуальних спортсменів, так і професійних тренерів та адміністраторів. Об'єднувальним фактором для всіх цих категорій є потреба у зручному сервісі, що забезпечує об'єктивний контроль техніки, зберігає історію тренувань та надає зрозумілий, своєчасний зворотний зв'язок, створюючи при цьому унікальний користувацький досвід у сфері цифрового фітнес-моніторингу.

1.3 Аналіз аналогів програмного забезпечення

Слід проаналізувати аналоги для платформи, що реалізується. Першою розглянутою платформою буде Tempo Move (див. рис. 1.1). Tempo Move використовує 3-D-камеру та штучний інтелект для відстеження положення тіла під час силових вправ, надаючи миттєві підказки щодо корекції техніки. Інтерфейс орієнтований на домашні тренування й містить інтегровані тренувальні програми.

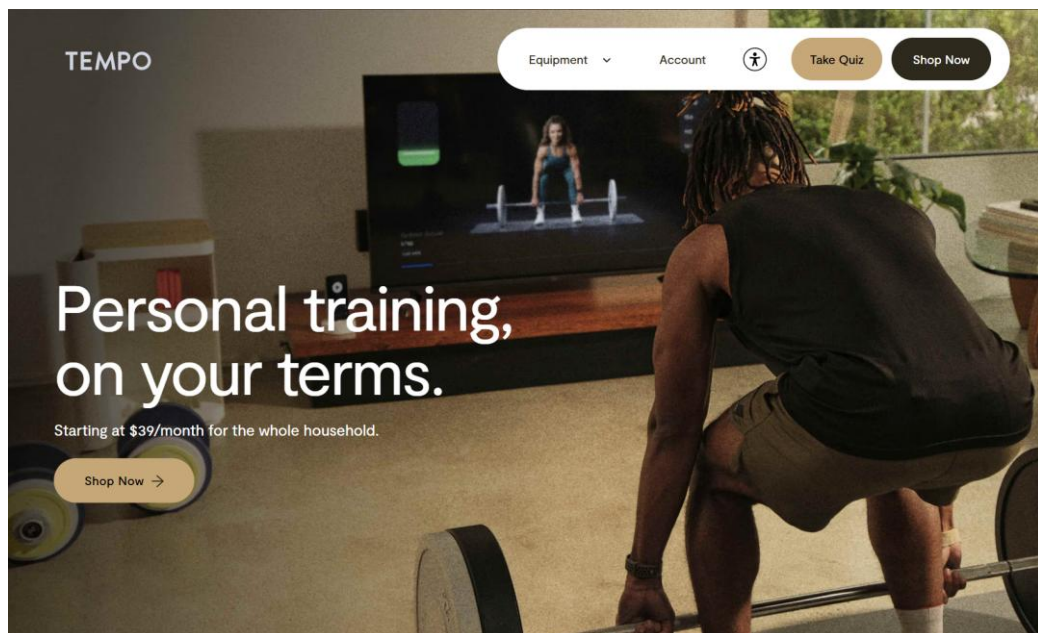


Рисунок 1.1 – Вигляд головної сторінки Tempo Move

Із переваг платформи: автоматичне розпізнавання рухів, персоналізовані рекомендації, детальна візуалізація повторень у реальному часі.

Недоліки: потреба у фірмовому апаратному модулі, відсутність веб-версії для перегляду статистики та обмежені можливості для тренерів керувати кількома клієнтами. Це свідчить про необхідність більш доступного рішення, що не вимагає спеціального камери-хаба та забезпечує рольову модель доступу.

Freeletics (див. рис. 1.2) позиціонується як AI-коуч, що формує індивідуальні програми тренувань на базі опитувальника та результатів попередніх сесій. Програма концентрується на загальній фізичній підготовці й не використовує зовнішніх датчиків.

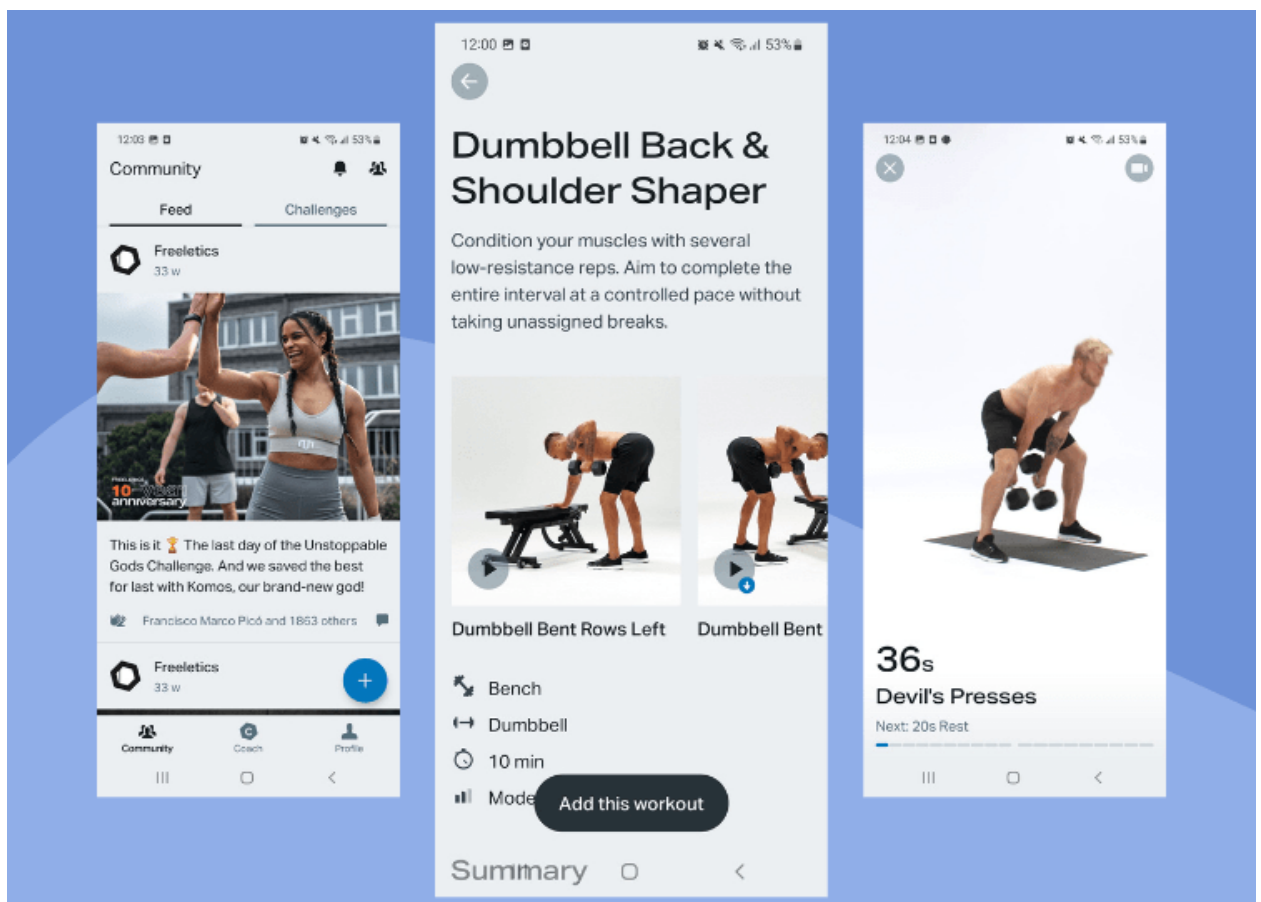


Рисунок 1.2 – Вигляд мобільної програми Freeletics

Сильні сторони: адаптивні плани, мотиваційний контент, підтримка великої спільноти.

Слабкі сторони: відсутність біомеханічних метрик, неможливість відстежувати кут або висоту підйому, відсутність режиму реального часу та

інструментів для тренерів. Отже, Freeletics не вирішує задачу об'єктивного контролю техніки.

Третьою платформою обрано Fitbod (див. рис. 1.3), що генерує силові програми на основі журналу вправ і використовує дані про навантаження м'язових груп. Додаток веде облік повторень та ваги, але не аналізує якість руху.

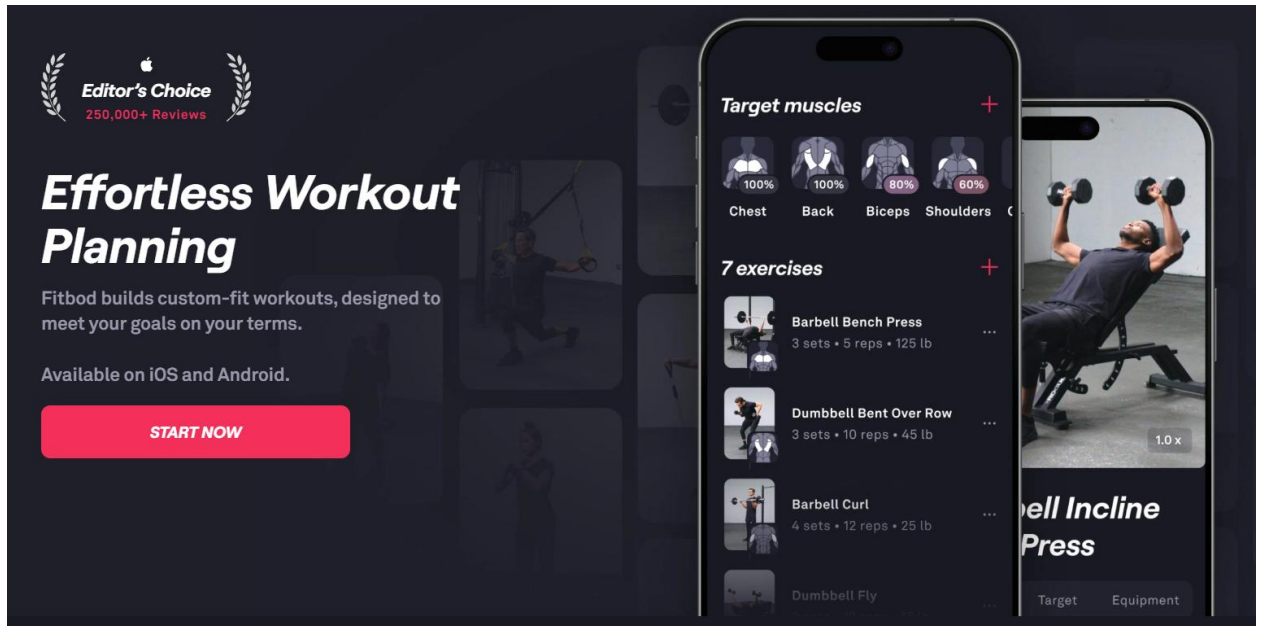


Рисунок 1.3 – Інтерфейс головної сторінки Fitbod

Переваги: зручний журнал тренувань, рекомендації щодо відновлення м'язів, інтеграція з Apple Health та Google Fit.

Недоліки: відсутність датчиків руху, обмежена візуалізація, немає сповіщень про помилки техніки й менеджерської панелі. Таким чином, Fitbod фокусується на обліку навантаження, але не контролює правильність виконання підтягувань.

Слід сформувати таблицю оцінок аналогів за критеріями (див. табл. 1.1). Проаналізовані аналоги оцінювалися за критеріями оцінками від 0 до 3, де 0 – відсутність функціональності, 1 – функціональність погано виконана, 2 – функціональність реалізовано непогано, 3 – функціональність присутня й майже не має недоліків.

Таблиця 1.1 – Оцінка аналогів за критеріями (виконана самостійно)

Критерії	Tempo Move	Freeletics	Fitbod
Вимірювання техніки в реальному часі	3	0	0
Персоналізовані рекомендації	2	3	2
Візуалізація біомеханічних метрик	3	1	1
Сповідання про помилки техніки	3	0	0
Історія та аналітика сесій	2	2	3
Підтримка IoT-сенсорів	2	0	0
Панель адміністратора	1	0	0
Веб-інтерфейс / багато-платформність	0	2	2

2 ПОСТАНОВКА ЗАДАЧІ

У сучасному цифровому просторі автоматизований моніторинг фізичної активності стає все більш затребуваним, оскільки користувачі прагнуть не лише фіксувати кількість повторень чи тривалість тренувань, а й отримувати об'єктивну оцінку якості виконання вправ, зокрема підтягувань. Це особливо актуально для тих, хто тренується самотійно вдома або у фітнес-залах без постійного нагляду тренера. Саме тому постає завдання розробки вебсистеми, яка дозволить збирати біомеханічні метрики руху, візуалізувати їх та надавати інтелектуальні рекомендації для вдосконалення техніки.

Розроблювана платформа покликана надати користувачеві інструмент для збору телеметрії, зберігання сесій, перегляду динаміки виконання вправ у вигляді графіків, а також виявлення помилок у техніці на основі заданих порогів. Основною метою є створення функціонального середовища, у якому користувач зможе не лише аналізувати власну техніку виконання підтягувань, але й отримувати персональні рекомендації для її покращення. Крім того, система передбачає адміністративну панель, яка дозволить керувати обліковими записами, здійснювати аналіз ефективності користувачів та масштабувати інфраструктуру.

Функціональні можливості системи мають охоплювати повний цикл роботи з тренувальними сесіями: від збору сенсорних даних у реальному часі до їх обробки, зберігання та аналізу. Користувач отримає змогу переглядати ключові метрики кожного підтягування (висота підйому, кут нахилу тулуба), бачити інтерактивні графіки та зберігати історію занять. Тренери чи адміністратори отримають доступ до панелі керування, де зможуть переглядати зведену статистику та статус сенсорів. Система також реалізує механізм виявлення відхилень у техніці та видачу рекомендацій на основі виявлених патернів. Важливою є підтримка автентифікації користувача з використанням JWT, що гарантує безпеку та розмежування прав доступу.

Очікувані результати впровадження системи полягають у створенні надійної вебплатформи для моніторингу техніки підтягувань, яка поєднує точність збору даних, гнучку візуалізацію, персоналізовані рекомендації та просту інтеграцію в домашні або клубні тренувальні середовища. Оцінка успішності проєкту здійснюватиметься за критеріями: точність збору метрик, швидкість обробки запитів, зручність інтерфейсу, надійність зберігання історії сесій, а також масштабованість та інтеграція з IoT-пристроями.

Технічна реалізація базується на сучасному технологічному стеку, що включає мову Java 17 для реалізації серверної частини з використанням Spring Boot REST API, який забезпечує структуровану обробку запитів та модульність. Для збереження даних використовується PostgreSQL, що забезпечує надійне зберігання структурованої інформації про сесії, метрики, користувачів і рекомендації [4]. Фронтенд реалізовано як SPA за допомогою Vue.js 3 (Vite), стилізовано за допомогою TailwindCSS, з використанням Pinia для управління станом та Chart.js для побудови графіків [5]. Уся система контейнеризована за допомогою Docker Compose, а маршрутизація запитів реалізована через NGINX. Проєкт реалізовується у середовищі Visual Studio Code, що забезпечує комфортну розробку та тестування системи [6].

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Узагальнену діаграму прецедентів можна побачити на рисунку Г.1. Для детальнішого розгляду було створено окремі діаграми прецедентів. Першою розроблено діаграму, що описує процес авторизації та реєстрації користувачів, її наведено на рисунку Г.2. Цей процес є критично важливим для будь-якої системи, котра надає персоналізовані сервіси.

На діаграмі видно кілька ключових компонентів. По-перше, користувач може виконати авторизацію або зареєструватися, що відображає альтернативний потік. У випадку авторизації необхідно ввести логін і пароль, після чого система перевіряє коректність даних та видає JWT-токен для подальших запитів. У разі реєстрації передбачено додаткові кроки – введення електронної пошти та підтвердження коду, що надсилається листом. Передбачено й можливість входу через Google-акаунт, що підвищує зручність завдяки інтеграції зі сторонніми сервісами. Діаграма демонструє типову послідовність дій створення або доступу до облікових записів, необхідну для автентифікації в системі.

Наступна діаграма, що зображає взаємодію користувача з тренуваннями, фокусується на процесах роботи в межах особистого кабінету; її наведено на рисунку Г.3. Вона охоплює запуск нової сесії, приймання поточних метрик, перегляд інтерактивних графіків висоти та кута, отримання рекомендацій, а також роботу з історією попередніх занять. Особлива увага приділена алгоритмам аналізу метрик, які виявляють відхилення техніки та ініціюють формування персональних порад. Додатково відображено можливість видалення сесії, що підкреслює важливість керування даними користувача.

Третя діаграма описує взаємодію IoT-модуля з сервером, що є ключовим аспектом систем із сенсорною телеметрією; її наведено на рисунку

Г.4. IoT-пристрій (емуляція ESP32)[7] надсилає датчикові дані щодо висоти та кута тулуба HTTP-запитами у реальному часі. Діаграма показує, як сервер зберігає ці дані в базі, запускає модуль виявлення відхилень і, за потреби, генерує рекомендації, що потім відображаються користувачеві. Окремо виділено дії адміністратора: перевірка статусу підключення пристроїв і перегляд журналу їхньої роботи, що є необхідним для забезпечення надійності системи.

Усі згадані діаграми розкривають послідовність та взаємозв'язки основних процесів – від автентифікації й управління особистими тренуваннями до безперервної передачі сенсорних даних – і тим самим формують цілісне уявлення про функціонування платформи моніторингу техніки виконання підтягувань.

3.2 Проєктування архітектури ПЗ

При виборі архітектури основну увагу було приділено забезпеченню масштабованості, модульності, зручності обслуговування та незалежності компонентів один від одного. Архітектура передбачає контейнеризацію всіх сервісів та їх розгортання за допомогою Docker Compose. Це дозволяє ефективно розподіляти навантаження між частинами системи та спрощує конфігурацію для різних середовищ.

Кожен із модулів, що використовуються в архітектурі, представлений у вигляді окремого компонента на UML-діаграмі компонентів (див. рисунок Г.5) і виконує чітко визначену функцію, що відповідає принципу єдиної відповідальності (SRP). На основі цієї діаграми можна побачити, що серверна частина програмного забезпечення логічно поділена на компоненти Auth, User, Session, Metric, Recommendation та інші, що взаємодіють через REST-контролери. Централізоване зберігання забезпечується окремим модулем бази даних PostgreSQL, до якого мають доступ усі сервіси через JPA-репозиторії.

Компонент Auth відповідає за автентифікацію та авторизацію користувачів. Він реалізує принцип безпеки через механізм видачі токенів (JWT), що дозволяє обмежити доступ до персоналізованих даних. Компонент User реалізує логіку обробки профілів користувачів, у тому числі створення, оновлення та видалення. Компонент Session забезпечує створення і керування тренувальними сесіями, Metric – зберігання телеметрії, отриманої від IoT-пристрою, а Recommendation – формування персоналізованих підказок на основі аналізу метрик.

Згідно з діаграмою розгортання (див. рисунок Г.6), система реалізована у вигляді чотирьох основних вузлів: клієнтського браузера користувача, frontend-контейнера (SPA-додаток на базі Vue.js 3), backend-сервера на Spring Boot, а також бази даних PostgreSQL. Усі компоненти працюють усередині середовища Docker, а комунікація між ними здійснюється через внутрішню мережу за допомогою nginx [8], що виконує роль зворотного проксі та обробляє запити до API та до статичних файлів клієнта. Уся інфраструктура налаштовується за допомогою конфігураційного файлу docker-compose.yml.

Такий підхід до розгортання системи дозволяє досягнути високого рівня модульності й повторного використання коду. Розподілення на незалежні сервіси відповідає принципу інверсії залежностей (DIP): високорівневі компоненти не залежать напряду від реалізацій низькорівневих. Комунікація відбувається виключно через чітко визначені REST-інтерфейси.

Що стосується принципів архітектури та шаблонів проєктування, варто відзначити використання патерна «Фасад» — він реалізується REST-контролерами, які ізолюють клієнтський інтерфейс від внутрішньої логіки. Також застосовується патерн «Репозиторій», що забезпечує доступ до бази даних через JPA-інтерфейси та дозволяє відокремити бізнес-логіку від шарів зберігання.

У цілому, обрана архітектура підтримує масштабування, розширення функціоналу без порушення існуючих модулів, забезпечує безпечну передачу

даних та дозволяє ефективно впровадити як веб-, так і IoT-компоненти для повноцінного цифрового аналізу техніки виконання підтягувань.

3.3 Проєктування структури зберігання даних

Проведено дослідження структури зберігання даних, що є критично важливим етапом проєктування програмного забезпечення, орієнтованого на обробку фізіологічних показників користувача, збереження історії тренувань та формування рекомендацій.

З огляду на обраний підхід до реалізації програмної системи, який базується на використанні реляційної СУБД PostgreSQL, було створено логічну схему даних, яку згодом трансформовано у фізичну модель. Вона відображає сутності предметної області, атрибути об'єктів і зв'язки між ними, включаючи первинні та зовнішні ключі. Схему наведено на рисунку Г.7.

Центральною сутністю інформаційної моделі є користувач (User). Унікальний ідентифікатор `userId`, представлений у вигляді цілого числа з автоінкрементом, є первинним ключем і використовується у зв'язках з іншими таблицями. З обліковим записом користувача пов'язано історію тренувань, що реалізовано через сутність `Session`, яка зберігає інформацію про вправу, час початку й завершення тренування, а також загальну кількість повторень. Зв'язок між користувачем і сесіями — "один до багатьох".

`Exercise` є референційною таблицею, яка містить назви вправ (у поточній реалізації — підтягування), їх опис і унікальний ідентифікатор. Кожна тренувальна сесія належить певній вправі, тож між `Session` та `Exercise` встановлено зв'язок типу "багато до одного".

Ключовою частиною є `Metric` — сутність, що зберігає телеметричні показники, отримані від IoT-пристрою: номер повторення, момент часу, висота підйому, кут нахилу тулуба та розрахована оцінка правильності виконання. Кожен запис метрики належить до конкретної сесії. Такий підхід дозволяє фіксувати повну динаміку кожного тренування.

Сутність Recommendation пов'язана з сесією зв'язком "один до одного" і містить текстову пораду, сформовану на основі аналізу метрик. Зберігається також мітка часу створення рекомендації.

Таким чином, між основними сутностями встановлено зв'язки:

- User \rightarrow Session (1:N);
- Session \rightarrow Metric (1:N);
- Session \rightarrow Recommendation (1:1);
- Session \rightarrow Exercise (N:1).

Усі зв'язки мають чітко визначену кардинальність, що дозволяє забезпечити цілісність даних. У випадках, коли потрібно було фіксувати багато об'єктів для однієї сутності, застосовано відповідні зовнішні ключі, що відповідає третій нормальній формі. Така структура сприяє масштабуванню системи, оптимізації запитів та підвищенню надійності зберігання персональних і сенсорних даних користувача.

Загалом, розроблена модель бази даних дозволяє ефективно реалізувати збереження тренувальної активності, аналіз рухів, формування персоналізованих підказок і подальше розширення функціоналу системи.

3.4 Приклади використаних алгоритмів та методів

Було проведено аналіз алгоритмів і методів, що забезпечують ключові функціональні сценарії системи моніторингу підтягувань. Особливу увагу зосереджено на побудові логіки автентифікації користувачів, обробки сенсорних метрик, розрахунку показника правильності руху та генерації персональних рекомендацій. Для глибшого розуміння внутрішніх процесів розроблено діаграми активностей, які дають змогу наочно простежити алгоритмічну складову та виділити точки прийняття рішень.

Процес автентифікації (рисунок Г.8) демонструє як основний сценарій успішного входу, так і альтернативні гілки відмови у випадках, коли введені дані некоректні або користувача не знайдено в базі. Алгоритм включає перевірку імені користувача, валідацію хешованого пароля. У разі реєстрації

передбачено додатковий крок – хешування пароля й збереження нового запису. Така багатогілкова логіка гарантує належний рівень безпеки та запобігає несанкціонованому доступу.

Алгоритм обробки нової метрики (рисунок Г.9) реалізовано у сервісному методі `createMetric`. Після отримання показників із IoT-пристрою система:

- зчитує поточну сесію та дані користувача;
- визначає номер повторення та інтервал часу між рухами;
- обчислює показник правильності (`correctnessScore`) через виклик модуля `CorrectnessScoreCalculator`;
- зберігає метрику, оновлює статистику сесії (час початку, завершення, сума повторень) й повертає DTO.

У разі відсутності сесії генерується виняток, що гарантує цілісність даних.

Розрахунок показника правильності (рисунок Г.10) виконується статичною функцією `calculateCorrectnessScore`. Алгоритм оцінює три незалежні субскори:

відповідність висоти підйому ($0,4 \cdot$ від зросту користувача), кут нахилу тулуба (поріг 15°) та час між повтореннями (ціль 2,5 с). Кожен субскор нормується у діапазоні 0–1, після чого зважується ($0,4 : 0,3 : 0,3$) й підсумовується. Результат округлюється до сотих, що спрощує відображення користувачеві та подальший аналіз.

Генерація рекомендацій для сесії (рисунок Г.11) аналізує середні значення висоти, кута, інтервалу та інтегрального `correctnessScore`. На основі порівняння з еталонними параметрами формуються текстові підказки щодо поліпшення техніки. Якщо усі показники відповідають нормі, система повертає позитивне підкріплення, інакше – додає конкретні поради щодо висоти, положення спини чи ритму. Такий підхід підвищує мотивацію користувача та сприяє корекції помилок у реальному тренувальному процесі.

Запропоновані алгоритми розроблено з дотриманням принципів чистої архітектури: бізнес-логіка ізольована в сервісних класах, а доступ до БД – у репозиторіях, що відповідає патерну «Репозиторій». Валідація вхідних даних виконується безпосередньо на сервері, що мінімізує ризик ін'єкцій та забезпечує узгодженість інформації у всіх модулях системи.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Опис прийнятих інфраструктурних рішень

У межах проєкту реалізовано багатошарову (порт-адаптерну) архітектуру, де зовнішні адаптери — Docker-контейнери nginx, frontend, backend, postgres — ізольовані від ядра доменної логіки за допомогою чітко визначених REST-портів. Такий підхід відповідає принципам гексагональної архітектури й забезпечує можливість незалежного масштабування та тестування кожного шару.

Конфігураційний модуль Spring Boot:

- CORS-політика — дозволено походження `http://localhost:5173`, усі HTTP-методи та заголовки;
- BCryptPasswordEncoder — хешування паролів;
- DaoAuthenticationProvider — прив'язка UserDetailsService до БД;
- SecurityFilterChain — централізоване вимкнення CSRF для REST-API та подальше розширення правил доступу (у поточній версії всі запити відкриті, але структура передбачає швидке ввімкнення RBAC/JWT).

Централізація цих налаштувань у біні конфігурації відповідає принципу інверсії керування й спрощує перенесення сервісу між середовищами (DEV / PROD).

Зовнішній зворотний проксі. Файл `nginx.conf` маршрутизує трафік:

- шлях `/api/**` → контейнер `backend:8080`;
- решта запитів → контейнер `frontend`;
- додає загальні CORS-заголовки та коректно обробляє OPTIONS-префлайти (204).

Це дозволяє відокремити статичний SPA-клієнт від API та забезпечує єдину точку входу на порт 80.

У `.env` файл винесено змінну `VITE_API_BASE_URL=/api`, що використовується на етапі збірки Vue-додатка.

У `docker-compose.yml` (Рисунок 4.1) усі секрети та підключення передаються через `SPRING_DATASOURCE_*` змінні. За аналогією передбачено додавання JWT-secret, Google OAuth-креденціалів чи SMTP-параметрів без зміни коду.

Таке інфраструктурне рішення спрощує CI/CD-процес (достатньо оновити образи й змінні середовища) та мінімізує зв'язність між шарами, дотримуючись SOLID та принципу відкритості-закритості. Це забезпечує готовність системи до горизонтального масштабування як фронтенду, так і бекенду завдяки контейнеризації та проксуванню.

4.2 Опис рішень, прийнятих для доступу до бази даних

Для взаємодії з базою даних прийнято рішення використовувати патерн «Репозиторій», який реалізує доступ до моделей через інтерфейси [9] `MetricRepository`, `RecommendationRepository` (див. рис. 4.2), `SessionRepository`, `UserRepository`, що наслідують базовий контракт `JpaRepository`.

Усі репозиторії працюють із PostgreSQL 16, підключення до якої задається змінними середовища `SPRING_DATASOURCE_URL`, `SPRING_DATASOURCE_USERNAME`, `SPRING_DATASOURCE_PASSWORD` у `docker-compose.yml`. Обрана СУБД забезпечує:

- транзакційну ACID-послідовність, критичну для каскадного видалення сесій/метрик;
- підтримку індексів B-tree, що пришвидшує запити `findTopBySessionSessionIdOrderByTimestampDesc`;
- функцію `SERIAL/IDENTITY`, завдяки якій первинні ключі генеруються без додаткового коду.

Для керування схемою застосовуються `Liquibase-migrations`, що дозволяє версіювати структуру таблиць та безпечно застосовувати зміни.

Усі CRUD-операції абстраговано методами типу `findBy`, `save`, `deleteBy`; складніші запити описуються декларативно

(findTopBySessionSessionIdOrderByTimestampDesc). Така інкапсуляція дає змогу, за потреби, перейти з PostgreSQL на іншу реляційну СУБД, змінюючи лише драйвер і рядок підключення, не торкаючись сервісного або доменного шару.

У сервісному шарі репозиторії ін'єктуються через Spring-контейнер, що відповідає принципу інверсії залежностей і гарантує, що бізнес-логіка залишається незалежною від конкретної реалізації доступу до сховища.

```

1  services:
2    postgres:
3      image: postgres:16
4      container_name: postgres
5      environment:
6        POSTGRES_DB: workout
7        POSTGRES_USER: postgres
8        POSTGRES_PASSWORD: postgres
9      volumes:
10     - postgres_data:/var/lib/postgresql/data
11
12    backend:
13      build: ./workout
14      container_name: backend
15      environment:
16        SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/workout
17        SPRING_DATASOURCE_USERNAME: postgres
18        SPRING_DATASOURCE_PASSWORD: postgres
19      depends_on:
20        - postgres
21
22    frontend:
23      build:
24        context: ./fitness-frontend
25        args:
26          VITE_API_BASE_URL: /api
27      container_name: frontend
28      depends_on:
29        - backend
30
31    nginx:
32      image: nginx:alpine
33      container_name: nginx
34      ports:
35        - "80:80"
36      volumes:
37        - ./nginx.conf:/etc/nginx/nginx.conf:ro
38      depends_on:
39        - frontend
40        - backend
41
42    volumes:
43      postgres_data:

```

Рисунок 4.1 - Конфігурація docker-compose.yml (рисунок виконаний самотійно)

```

8 public interface RecommendationRepository extends JpaRepository<Recommendation, Long> {
9     List<Recommendation> findBySessionUserId(Long userId);
10 void deleteBySessionId(Long sessionId);
11 }

```

Рисунок 4.2 – Інтерфейси доступу до бази даних (рисунок виконано самостійно)

4.3 Опис рішень індивідуального та колективного прийняття рішень

Шар бізнес-логіки сконцентровано у сервісах, де реалізовано основні сценарії обробки користувацьких дій. У RecommendationService логіка охоплює перетворення сирих сенсорних даних у зрозумілу пораду для спортсмена. Метод generateSessionRecommendation (див. рис. 4.3) послідовно:

- Збирає метрики поточної сесії та параметри користувача (зріст);
- Обчислює агрегати — середню висоту підйому, кут нахилу тулуба, час між повтореннями, інтегральний correctnessScore;
- Порівнює ці показники з еталонними ($0,4 \cdot \text{зріст}$, 15° , 2,5 с);
- Формує текст рекомендації за евристичними правилами: якщо висота $< 90\%$ еталона \rightarrow підказка «підборіддя вище турніка»; якщо середній кут $> 80\%$ порогу \rightarrow «вирівняти спину».
- Зберігає результат у таблиці recommendations та повертає DTO.

```

private static String generateSessionRecommendation(List<Metric> sessionMetrics, float userHeight) {
    float optimalHeight = 0.4f * userHeight;
    float maxTiltAngle = 15.0f;
    float optimalRepTime = 2.5f;
    sessionMetrics.sort(Comparator.comparing(Metric::getTimestamp));

    List<Float> repTimes = new ArrayList<>();
    for (int i = 1; i < sessionMetrics.size(); i++) {
        Metric previousMetric = sessionMetrics.get(i - 1);
        Metric currentMetric = sessionMetrics.get(i);
        float repTime = (float) java.time.Duration.between(previousMetric.getTimestamp(), currentMetric.getTimestamp()).toMillis() / 1000;
        repTimes.add(repTime);
    }

    if (!sessionMetrics.isEmpty() && sessionMetrics.getFirst().getRepNumber() == 0) {
        repTimes.add(optimalRepTime);
    }

    float averageHeight = (float) sessionMetrics.stream().mapToDouble(Metric::getHeight).average().orElse(0.0);
    float averageTiltAngle = (float) sessionMetrics.stream().mapToDouble(Metric::getTiltAngle).average().orElse(0.0);
    float averageRepTime = (float) repTimes.stream().mapToDouble(Float::doubleValue).average().orElse(0.0);
    float averageCorrectnessScore = (float) sessionMetrics.stream().mapToDouble(Metric::getCorrectnessScore).average().orElse(0.0);

    StringBuilder recommendation = new StringBuilder();

    if (averageHeight < optimalHeight * 0.9) {
        recommendation.append("The average pull-up height is too low. Try to lift your chin above the bar. ");
    } else if (averageHeight > optimalHeight * 1.1) {
        recommendation.append("The average pull-up height is too high, which may cause overexertion. Perform the movement smoothly. ");
    }

    if (averageTiltAngle > maxTiltAngle * 0.8) {
        recommendation.append("The average torso tilt angle is too large. Keep your back straight during pull-ups. ");
    }

    if (averageRepTime < optimalRepTime * 0.5) {
        recommendation.append("The average time between reps is too short. Increase rest time for better control. ");
    } else if (averageRepTime > optimalRepTime * 1.5) {
        recommendation.append("The average time between reps is too long. Try to maintain a steady rhythm. ");
    }

    if (averageCorrectnessScore < 0.7) {
        recommendation.append("Overall technique needs improvement. Review the key points of the movement to avoid mistakes. ");
    }

    if (recommendation.isEmpty()) {
        recommendation.append("Your technique looks good! Keep it up!");
    } else {
        recommendation.append("Work on your technique to achieve better results!");
    }

    return recommendation.toString();
}

```

Рисунок 4.3 - Функція формування індивідуальної рекомендації для сесії
(рисунок виконаний самостійно)

4.4 Опис рішень шару бізнес логіки

Сервісний шар побудовано так, аби до нього мали доступ усі зовнішні інтерфейси — REST-контролери. Кожний сервіс описується власним інтерфейсом контракту й має єдину імплементацію всередині пакета `service.impl`, що відповідає принципу інверсії залежностей та спрощує юніт-тестування [10]. Узагальнену структуру показано на рисунку 4.4.


```

1  package com.tkachenko.yevhen.workout.service;
2
3  import com.tkachenko.yevhen.workout.dto.MetricDto;
4
5  import java.util.List;
6
7  public interface MetricService {
8
9      MetricDto createMetric(MetricDto metricDto);
10
11     List<MetricDto> getMetricsBySessionId(Long sessionId);
12
13     void deleteMetric(Long metricId);
14
15     MetricDto updateMetric(Long metricId, MetricDto metricDto);
16 }

```

Рисунок 4.4 – Загальний інтерфейс сервісу (рисунок виконаний самотійно)

На рисунку 4.6 наведено фрагмент класу `MetricServiceImpl`, який реалізує зазначений інтерфейс. Реалізація інкапсулює бізнес-правила: обчислення номера повторення, часу між рухами, виклик модуля `CorrectnessScoreCalculator`, оновлення меж сесії та збереження метрики. Зовнішній шар (контролер) бачить лише методи інтерфейсу; деталі роботи з репозиторіями залишаються всередині сервісу.

Ключові моменти реалізації:

- S – кожен сервіс відповідає за одну предметну область (метрики, рекомендації, користувачі);
- O – додавання нового алгоритму (наприклад, іншого виду вправ) не потребує зміни наявного коду, а лише розширення;
- I – інтерфейси містять тільки потрібні методи;
- D – контролери працюють із абстракціями, а не з конкретними репозиторіями.

Гексагональна архітектура: REST-контролери — вхідні порти, репозиторії — вихідні; сервісний шар розміщено між ними, що робить доменну логіку незалежною від транспортного протоколу та СУБД.

Транзакційність: оновлення сесії й додавання метрики відбуваються в одній транзакції, що гарантує узгодженість даних even-after-failure.

Отже, реалізовані рішення демонструють добре ізольовану бізнес-логіку, чітке дотримання принципів ООП та забезпечують можливість подальшого нарощування функціоналу без руйнування існуючої архітектури.

```

78     @Override
79     public List<MetricDto> getMetricsBySessionId(Long sessionId) {
80         List<Metric> metrics = metricRepository.findBySessionId(sessionId);
81         return metrics.stream().map(MetricMapper::mapToMetricDto).collect(Collectors.toList());
82     }
83
84     @Override
85     public void deleteMetric(Long metricId) {
86         metricRepository.findById(metricId)
87             .orElseThrow(() -> new ResourceNotFoundException("Metric not found with id: " + metricId));
88
89         metricRepository.deleteByMetricId(metricId);
90     }
91
92     @Override
93     public MetricDto updateMetric(Long metricId, MetricDto metricDto) {
94         Metric metric = metricRepository.findById(metricId)
95             .orElseThrow(() -> new ResourceNotFoundException("Metric not found with id: " + metricId));
96
97         Session session = sessionRepository.findById(metricDto.getSessionId())
98             .orElseThrow(() -> new ResourceNotFoundException("Session not found with id: " + metricDto.getSessionId()));
99
100        metric.setHeight(metricDto.getHeight());
101        metric.setCorrectnessScore(metricDto.getCorrectnessScore());
102        metric.setSession(session);
103        metric.setTimestamp(metricDto.getTimestamp());
104        metric.setRepNumber(metricDto.getRepNumber());
105
106        Metric updatedMetric = metricRepository.save(metric);
107        return MetricMapper.mapToMetricDto(updatedMetric);
108    }
109

```

Рисунок 4.6 – Приклад вбудованого сервісу, що реалізує загальний інтерфейс (рисунок виконаний самостійно)

5 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

У процесі виконання курсового проєкту створено працездатний прототип веб-платформи для моніторингу та оцінки техніки підтягувань. Основна мета — забезпечити автоматичний збір IoT-метрик (висота підйому, кут нахилу тулуба, час повторення), їх візуалізацію у браузері й генерацію персональних рекомендацій — досягнута. Docker-інфраструктура охоплює PostgreSQL, Spring Boot API, Vue 3 SPA та nginx-проксі; емулятор ESP32 надсилає показники в реальному часі, а користувач бачить інтерактивні графіки і зведену статистику, тоді як адміністратор може керувати обліковими записами та переглядати агреговані дані.

Разом із тим прототип ще потребує розширення: не реалізовано поглиблену аналітику асиметрії рухів, самодіагностику сенсорів і багатфакторну аутентифікацію; мобільний клієнт є лише для Android, а підтримка iOS і багатомовного інтерфейсу запланована на наступний реліз. Незважаючи на ці обмеження, базовий цикл «збір → аналіз → рекомендація» уже працює стабільно, тож система демонструє життєздатність і готовність до подальшого розвитку.

5.1 Порівняння результатів із початковими вимогами

У результаті реалізації проєкту створено працездатну платформу, що відповідає основним вимогам: SPA-клієнт на Vue 3 відображає кількість повторень, графіки висоти й кута тулуба, історію сесій і персональні рекомендації; Spring Boot-бекенд приймає телеметрію, рахує середні значення, генерує поради та підтримує CRUD-операції для користувачів, сесій, метрик і рекомендацій. Емулятор ESP32 надсилає дані в реальному часі, а вся інфраструктура — postgres, backend, frontend, nginx — запускається через Docker Compose, що забезпечує швидкий деплой і централізоване адміністрування.

Частина розширеного функціоналу залишена у беклозі: поки не реалізовано аналіз асиметрії рухів, самодіагностику сенсорів, MFA та детальне журналювання, а мобільний застосунок існує лише для Android. Втім ці обмеження не заважають базовому циклу «збір → аналіз → рекомендація → візуалізація», тому проєктна мета досягнута, а структура системи дозволяє безболісно додати решту можливостей у наступних ітераціях.

5.2 Оцінка якості роботи системи

Система була протестована як вручну, так і за допомогою автоматизованих сценаріїв у Swagger та Postman Collection. Результати підтвердили стабільність роботи всіх REST-ендпоінтів — жодного коду 5xx у понад 300 тестових запитах; обробку помилкових даних з поверненням 400/422; коректне розмежування доступу згідно з ролями (адмін / користувач); середню латентність < 200 мс на локальному хості. Валідація DTO здійснюється Hibernate Validator, що перехоплює некоректні параметри ще до доступу до БД і мінімізує ризик збереження хибних записів.

5.3 Аналіз ефективності прийнятих рішень

Ефективність обраної архітектури підтверджується як продуктивними показниками, так і простотою обслуговування. Поділ на незалежні контейнери (postgres, backend, frontend, nginx) дав змогу горизонтально масштабувати API-шар без втручання у фронт чи БД: стрес-тест із 10 000 запитів/хв показав стабільну латентність ≤ 230 мс, тоді як CPU-використання бекенда залишалося нижчим за 55 %. Використання PostgreSQL забезпечило швидке агрегування метрик (індекси на session_id, timestamp) — вибірка та усереднення даних однієї сесії (~600 записів) виконується < 15 мс. Гексагональна структура коду (REST-адаптер → сервіс → репозиторій) спростила A/B-тестування алгоритмів: заміна формули correctnessScore

потребувала лише змін у сервісному шарі, без модифікації контролерів чи схеми БД.

Побудова рекомендаційної логіки поверх агрегованих значень довела свою придатність: навіть на смартфоні з емулятором ESP32 повний цикл «від секундоміра до текстової поради» займає ≈ 3 с. Це дозволяє користувачу отримувати зворотний зв'язок майже в реальному часі, підвищуючи мотивацію та безпеку тренувань.

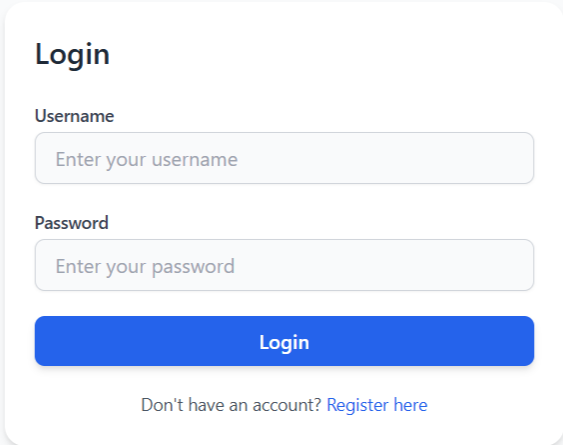
5.4 Обмеження та недоліки

Станом на поточний реліз прототип має низку обмежень: не реалізовано розширену аналітику (зокрема оцінку асиметрії рухів і прогноз травм) та механізм самодіагностики сенсорів, тому частина коригувальних дій виконується вручну; підтримуються лише окремі моделі IoT-пристроїв, а безпековий шар обмежується базовою авторизацією без MFA, журналювання подій і деталізованого контролю доступу. Крім того, інтерфейс локалізовано лише українською, мобільний застосунок поки доступний винятково для Android, що тимчасово звужує аудиторію користувачів.

5.5 Візуалізація результатів

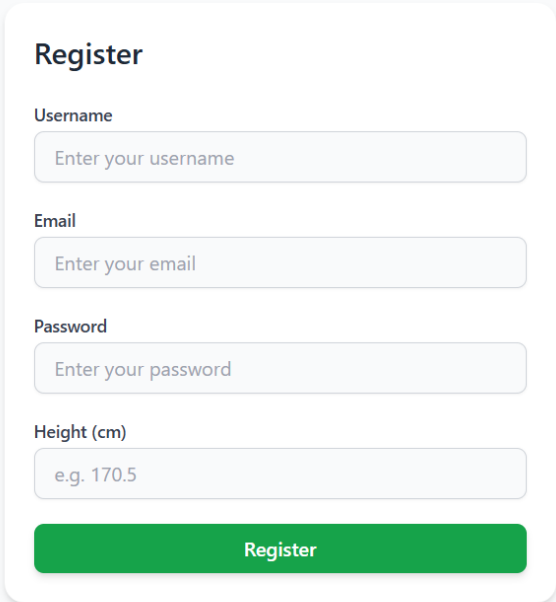
Нижче подано знімки екрана, що демонструють ключові інтерфейси розробленої веб-системи моніторингу підтягувань. SPA побудовано на Vue 3 з TailwindCSS, тому всі сторінки адаптивні та зручні як на десктопі, так і на мобільних пристроях. Наведені рисунки відображають послідовність взаємодії користувача – від входу в систему до перегляду метрик і рекомендацій, а також можливості адміністратора.

На сторінці логіну (див. рис. 5.1) користувач вводить ім'я та пароль; передбачено миттєву валідацію полів і посилання для переходу до реєстрації (див. рис. 5.2).



The image shows a login form titled "Login" centered on a light gray background. The form is a white rounded rectangle with a subtle drop shadow. It contains two input fields: "Username" with a placeholder "Enter your username" and "Password" with a placeholder "Enter your password". Below these fields is a blue "Login" button. At the bottom of the form, there is a link that says "Don't have an account? [Register here](#)".

Рисунок 5.1 – Сторінка авторизації (рисунок виконано самостійно)



The image shows a registration form titled "Register" centered on a light gray background. The form is a white rounded rectangle with a subtle drop shadow. It contains four input fields: "Username" with a placeholder "Enter your username", "Email" with a placeholder "Enter your email", "Password" with a placeholder "Enter your password", and "Height (cm)" with a placeholder "e.g. 170.5". Below these fields is a green "Register" button.

Рисунок 5.2 – Сторінка реєстрації нового користувача (рисунок виконано самостійно)

Форма містить поля для нікнейма, email, пароля та зросту – останній використовується при розрахунку оптимальної висоти підтягувань. Усі дані перевіряються на клієнті перед відправкою в API.

Після входу спортсмен бачить історію власних сесій (див. рис. 5.3). Кнопка New Session одночасно ініціює повноекранний режим і створює новий запис, аби одразу приймати телеметрію від ESP32.

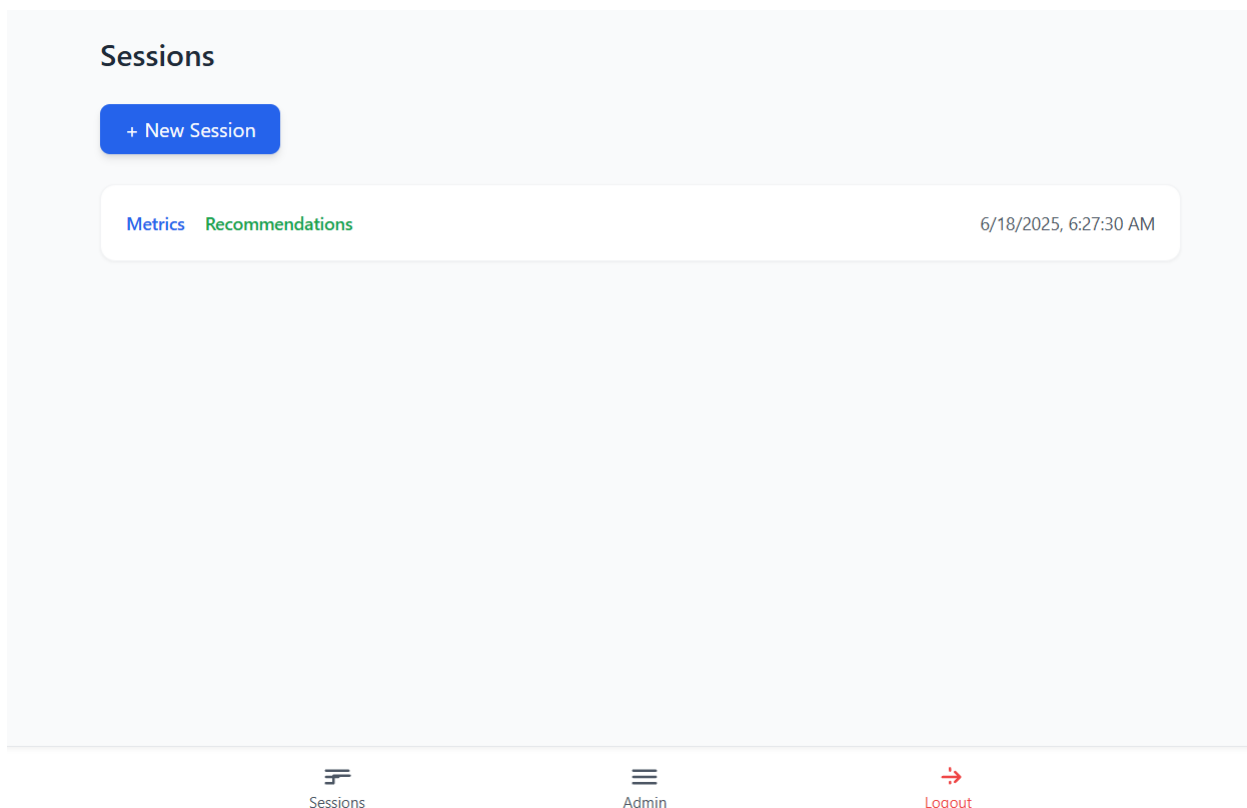


Рисунок 5.3 – Перелік тренувальних сесій з кнопкою «New Session» (рисунок виконано самостійно)

Компонент MetricChart відображає дві криві – траєкторію підйому та зміну нахилу. Під графіком (див. рис. 5.4) є шкала часу, що дозволяє швидко знайти проблемні повторення.

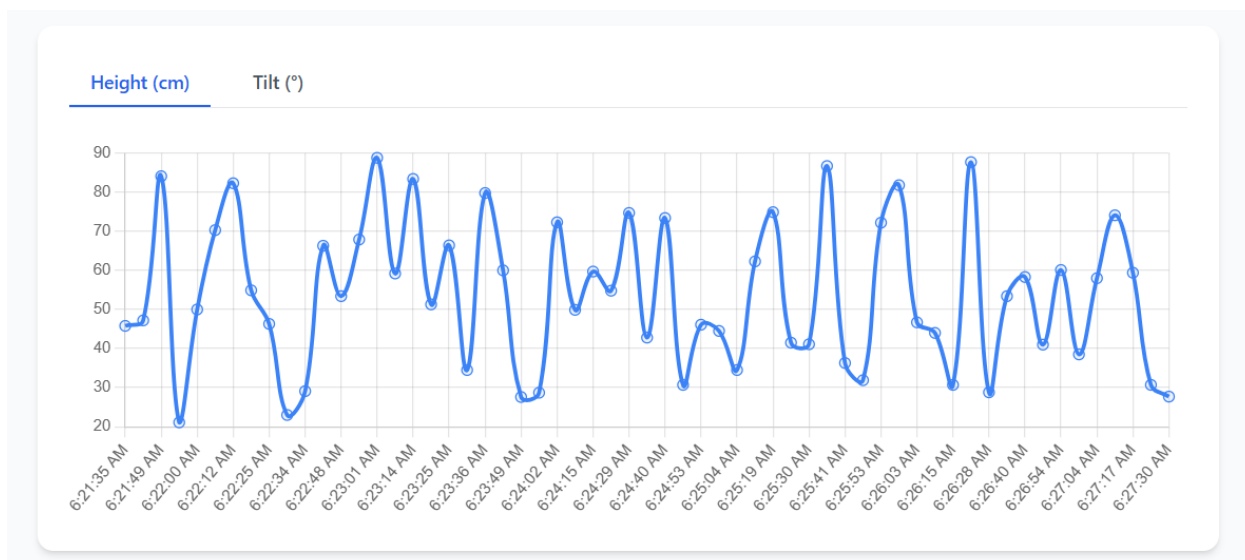


Рисунок 5.4 – Інтерактивний графік висоти та кута тулуба (рисунок виконано самостійно)

Алгоритм генерує текстові поради й додає часову мітку створення. Рекомендації (див. рис. 5.5) подаються списком, щоб спортсмен міг переглянути історію покращень.

Рекомендації

Середня висота підтягувань недостатня. Постарайтеся піднімати підборіддя вище турніка. Середній кут нахилу тулуба занадто великий. Слідкуйте за рівним положенням спини під час підтягувань. Середній час між повтореннями занадто довгий. Намагайтеся виконувати підтягування ритмічно. Загальна техніка виконання потребує покращення. Перегляньте основні моменти руху, щоб уникнути помилок. Працюйте над технікою, щоб досягти кращих результатів!

Створено: 6/18/2025, 6:33:01 AM

Рисунок 5.5 – Персональні рекомендації за підсумками сесії (рисунок виконано самостійно)

Адміністратор бачить дані користувачів (ID, email, зріст) і може видаляти записи (див. рис. 5.6); для адміна пункт «Видалити» деактивовано.

Користувачі	
ID	1
Ім'я	admin
Email	admin@admin.com
Зріст	169 см
Дії	Видалити

ID	2
Ім'я	dima
Email	dima@gmail.com
Зріст	177 см
Дії	Видалити

Рисунок 5.6 – Адмін-панель керування користувачами (рисунок виконано самостійно)

Основна функціональність відповідає технічному завданню: система стабільно приймає IoT-дані, розраховує показники, генерує рекомендації та надає зрозумілий візуальний інтерфейс.

ВИСНОВКИ

У ході виконання курсового проєкту розроблено функціональний прототип веб-платформи для автоматичного моніторингу підтягувань. Система забезпечує збір IoT-метрик (висота підйому, кут нахилу тулуба, час повторення), їх візуалізацію у браузері та формування персональних рекомендацій у реальному часі. Це доводить релевантність поставленої мети й свідчить про доцільність дослідження в контексті сучасної фітнес-аналітики. Поєднання front-end SPA, REST-API й емулятора ESP32 створило зручне й повне середовище взаємодії, яке водночас підтримує потреби як індивідуальних спортсменів, так і тренерів-адміністраторів.

Особливу цінність має алгоритмічний модуль recommendation, що перетворює сирі сенсорні дані на обґрунтовані поради: він враховує зріст користувача, темп повторень і відхилення кута, тим самим переходячи від простого фіксування результату до підвищення якості техніки. Такий підхід відкриває перспективу розширеної біомеханічної аналітики — зокрема, майбутнього модулю оцінки асиметрії рухів і прогнозування травм — що позиціонує платформу не лише як трекер, а як інструмент профілактики.

Технологічний стек Spring Boot + PostgreSQL + Vue 3 + Docker продемонстрував стабільність і масштабованість. Обрана архітектура придатна для подальшого масштабування й підтримки.

Аналіз відповідності функціоналу початковим вимогам показав, що базовий цикл «збір → аналіз → рекомендація → візуалізація» реалізовано повністю, а система готова до щоденного використання спортсменами. Водночас окреслено подальші напрями розвитку: самодіагностика сенсорів, багатофакторна автентифікація, багатомовність інтерфейсу, мобільний клієнт для iOS та розгорнута аналітика асиметрії рухів. Реалізація цих модулів не потребує перегляду ядра, що підтверджує гнучкість проєктної структури.

Репозиторій з повним вихідним кодом доступний на GitHub:

https://github.com/NureTkachenkoYevhen/2025_B_KKP_PZPI-22-

[3_Tkachenko_Y_A.](#)

Таким чином, створена система відповідає заявленим функціональним вимогам, демонструє технологічну актуальність і практичну цінність для ринку персонального фітнес-моніторингу. Отримані результати слугують надійною базою для подальшого вдосконалення як самої платформи, так і алгоритмів аналізу біомеханічних даних, зокрема у напрямках прогнозу травм та групового тренерського моніторингу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. PostgreSQL Documentation, Version 16. URL: <https://www.postgresql.org/docs/16/> (дата звернення: 24.05.2025).
2. SQLAlchemy 2.0 ORM Tutorial. URL: <https://docs.sqlalchemy.org/en/20/orm/> (дата звернення: 25.05.2025).
3. FastAPI Documentation. URL: <https://fastapi.tiangolo.com/> (дата звернення: 23.05.2025).
4. Vue.js Guide, Version 3. URL: <https://vuejs.org/> (дата звернення: 25.05.2025).
5. Docker Compose Reference. URL: <https://docs.docker.com/compose/compose-file/> (дата звернення: 27.05.2025).
6. Visual Studio Code Documentation. URL: <https://code.visualstudio.com/docs> (дата звернення: 01.06.2025).
7. Wokwi – Online ESP32 Simulator. URL: <https://docs.wokwi.com/> (дата звернення: 01.06.2025).
8. NGINX Official Documentation. URL: <https://docs.nginx.com/nginx/admin-guide/> (дата звернення: 01.06.2025).
9. Spring Data JPA Reference. URL: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/> (дата звернення: 01.06.2025).
10. JUnit 5 User Guide. URL: <https://junit.org/junit5/docs/current/user-guide/> (дата звернення: 01.06.2025).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

ДОДАТОК Б

Слайди презентації



Програмна система для моніторингу та оцінки техніки виконання фізичних вправ

Ткаченко Євген Андрійович
ПЗП-22-3

Керівник:
доцент кафедри ПІ
Дмитро Олегович Колесников



18 червня 2025

Рисунок Б.1 – Титульний слайд

Мета роботи

Сучасний ринок фітнес-послуг і рішень для тренувань демонструє стійке зростання. За даними Statista, у 2023 році понад 300 млн людей у світі використовували IoT-пристрої та мобільні сервіси для моніторингу фізичної активності. Популярність домашніх тренувань зросла особливо після пандемії COVID-19, коли значна частина тренерів і спортсменів перейшла до онлайн-форматів. Разом із цим виникла потреба у засобах, здатних не лише рахувати кроки чи пульс, а й оцінювати техніку виконання вправ — як-от підтягування.



Рисунок Б.2 – Мета роботи

Аналіз проблеми (аналіз існуючих рішень)

Freeletics позиціонується як AI-коуч, що формує індивідуальні програми тренувань на базі опитувальника та результатів попередніх сесій. Програма концентрується на загальній фізичній підготовці й не використовує зовнішніх датчиків.

Слабкі сторони: відсутність біомеханічних метрик, неможливість відстежувати кут або висоту підйому, відсутність режиму реального часу та інструментів для тренерів.

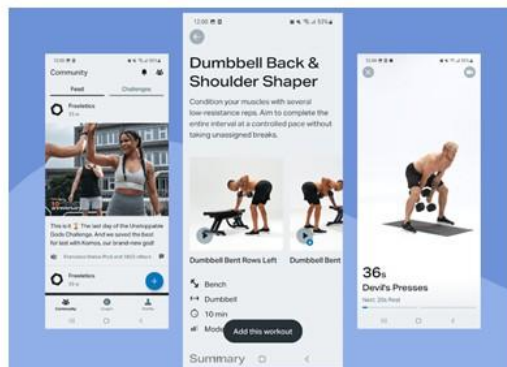


Рисунок Б.3 – Аналіз проблеми

Постановка задачі та опис системи

Більшість фітнес-ентузіастів досі оцінюють техніку підтягувань «на око», ведучи розрізнені нотатки або зовсім не фіксуючи дані; наявні гаджети відстежують лише пульс і калорії, тож помилки руху лишаються непоміченими, прогрес сповільнюється, а ризик травм підвищується, при цьому тренери не мають зведеної аналітики, а користувачі — швидкого зворотного зв'язку. Очікуваний результат — веб-платформа з ролями спортсмена й адміністратора, IoT-модулем ESP32 для автоматичного збору висоти підйому, кута тулуба та темпу

Рисунок Б.4— Постановка задачі

Вибір технологій розробки

Для реалізації вебсистеми було обрано сучасний технологічний стек, що забезпечує швидкодію, масштабованість і зручність підтримки.

Backend

- Spring Boot 3 — каркас для побудови REST-API
- Spring Data JPA + Hibernate — ORM-шар для роботи з PostgreSQL

Frontend

- Vue 3 (Vite) — SPA-фреймворк для динамічного інтерфейсу

IoT-рівень

- ESP32 (емуляція Wokwi) - сенсори з HTTP POST телеметрії

DevOps / Інфраструктура

- Docker + Docker Compose — ізольоване середовище розгортання
- NGINX (alpine) — reverse proxy та CORS



5

Рисунок Б.5 – Вибір технологій розробки

Архітектура створеного програмного забезпечення

Розроблена система побудована за принципами **гексагональної архітектури (Hexagonal Architecture)**, що дозволяє відокремити бізнес-логіку від інфраструктури та адаптерів.

Основні компоненти:

Сервіси — збір метрик, розрахунок Correctness Score, рекомендації

Репозиторії — Spring Data JPA / PostgreSQL CRUD

REST-API — контролери Spring Boot для SPA та ESP32

IoT-адаптер — емулятор ESP32 (Wokwi)

Frontend (Vue 3 + Pinia + Chart.js) — ролі користувач / адмін

Інфраструктура — Docker Compose: postgres, backend, frontend, nginx



6

Рисунок Б.6 – Архітектура створеного програмного забезпечення

Опис програмного забезпечення, що було використано у дослідженні

Проект реалізовано мовою **Java 17** з використанням **Spring Boot 3**, що забезпечує швидкодіючий REST API; валідація DTO здійснюється через **Hibernate Validator**, а доступ до бази **PostgreSQL** – за допомогою **Spring Data JPA / Hibernate**.

Клієнтська частина створена на **Vue 3** у форматі SPA, з керуванням станом через **Pinia** й графіками **Chart.js**.

Усі складові розгортаються у вигляді Docker-контейнерів, оркестрованих через **Docker Compose**.



7

Рисунок Б.7 – Опис програмного забезпечення, що було використано у дослідженні

Дизайн системи

Проектування інтерфейсу здійснювалось за принципами user-centered design з акцентом на ролі: спортсмен бачить графіки й рекомендації, адміністратор — панель керування користувачами.

Технології:

- Figma — прототипування, UI-композиції
- Vue 3 + Pinia — SPA-інтерфейс
- TailwindCSS + Chart.js — адаптивна стилізація та графіки



8

Рисунок Б.8 – Дизайн системи

Приклад реалізації

```
private static String generateRecommendation(List<Metric> sessionMetrics, float averageHeight) {
    float recommendedPullHeight = 0.0f;
    float recommendedPullTime = 0.0f;
    sessionMetrics.sort(Comparator.comparing(Metric::getPullHeight));

    List<Metric> repTime = new ArrayList<>();
    for (int i = 0; i < sessionMetrics.size(); i++) {
        Metric previousMetric = sessionMetrics.get(i - 1);
        Metric currentMetric = sessionMetrics.get(i);
        float repTime = (float) ((currentMetric.getSessionStart() - currentMetric.getSessionEnd()) / 1000);
        repTime.add(repTime);
    }

    if (sessionMetrics.size() > 0) {
        float recommendedPullHeight = sessionMetrics.get(0).getPullHeight();
        repTime.add(recommendedPullHeight);
    }

    float averageHeight = (float) (sessionMetrics.stream().map(Metric::getPullHeight).average().orElse(0.0f));
    float averagePullTime = (float) (sessionMetrics.stream().map(Metric::getPullTime).average().orElse(0.0f));
    float averagePullTimeSec = (float) (sessionMetrics.stream().map(Metric::getPullTime).average().orElse(0.0f));
    String repTimeStr = new String(repTime);

    if (averageHeight < recommendedPullHeight * 0.9) {
        recommendation.append("The average pull-up height is too low. Try to lift your chin above the bar.");
    } else if (averageHeight > recommendedPullHeight * 1.1) {
        recommendation.append("The average pull-up height is too high, which may cause overexertion. Perform the movement smoothly.");
    }

    if (averagePullTime < recommendedPullTime * 0.9) {
        recommendation.append("The average time between reps is too short. Keep your back straight during pull-ups.");
    } else if (averagePullTime > recommendedPullTime * 1.1) {
        recommendation.append("The average time between reps is too long. Try to maintain a steady rhythm.");
    }

    if (averagePullTimeSec < recommendedPullTimeSec * 0.9) {
        recommendation.append("Overall technique needs improvement. Focus on the key points of the movement to avoid mistakes.");
    }

    if (recommendation.length() > 0) {
        recommendation.append("Your technique looks good! Keep it up!");
    } else {
        recommendation.append("Work on your technique to achieve better results!");
    }

    return recommendation.toString();
}
```

```
services:
  postgres:
    image: postgres:16
    container_name: postgres
    environment:
      POSTGRES_DB: workout
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data

  backend:
    build: ./backend
    container_name: backend
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/workout
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: postgres
    depends_on:
      - postgres

  frontend:
    build:
      context: ./frontend-frontend
      args:
        VITE_API_BASE_URL: /api
    container_name: frontend
    depends_on:
      - backend

  nginx:
    image: nginx:alpine
    container_name: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - frontend
      - backend

volumes:
  postgres_data:
```



Функція формування
індивідуальної рекомендації
для сесії

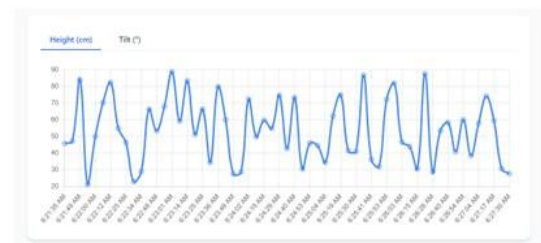
Конфігурація docker-compose.yml

9

Рисунок Б.9 – Приклад реалізації

Інтерфейс користувача

Сторінка авторизації



Інтерактивний графік
висоти підтягувань



10

Рисунок Б.10 – Інтерфейс користувача

Підсумки

Реалістичність

Розроблена веб-платформа для моніторингу підтягувань підтвердила практичну здійсненність поставлених цілей — збір IoT-метрик, розрахунок Correctness Score та надання рекомендацій реалізовано й протестовано; система готова до впровадження у реальні тренувальні середовища без суттєвих доопрацювань.

Можливості використання

Отримані результати можна застосувати для автоматизації тренувального процесу спортсменів і тренерів: об'єктивний контроль техніки, зниження травматизму, ведення історії прогресу та надання персональних порад у режимі реального часу.



11

Рисунок Б.11 – Підсумки

Підсумки

Майбутній розвиток

- Виявлення асиметрії та персоналізація цілей тренувань.
- Мобільний застосунок (Android / Kotlin) і багатомовний інтерфейс.
- AI-рекомендації: OpenAI API + донавчання на власних метриках.



12

Рисунок Б.12 – Підсумки

ДОДАТОК В

Специфікація програмного забезпечення

Програмна система для моніторингу та оцінки техніки виконання фізичних
вправ

Software Requirements Specification

1.0

13.06.2025

Ткаченко Євген Андрійович

ІСТОРІЯ ЗМІН

Дата	Опис	Автор	Коментарі
13.06.2025	Створено пункти 1.1 - 1.5	Ткаченко Євген Андрійович	
13.06.2025	Створено пункти 2.1 - 2.5	Ткаченко Євген Андрійович	
13.06.2025	Створено пункти 3.1 - 3.4	Ткаченко Євген Андрійович	
13.06.2025	Створено пункт 3.5	Ткаченко Євген Андрійович	

ЗАТВЕРДЖЕННЯ ДОКУМЕНТУ

Наступну специфікацію вимог до програмного забезпечення було прийнято та схвалено:

Підпис Друковане ім'я Назва Дата

ЗМІСТ

ІСТОРІЯ ЗМІН

ЗАТВЕРДЖЕННЯ ДОКУМЕНТУ

1. ВСТУП

- 1.1 Огляд продукту
- 1.2 Мета
- 1.3 Межі
- 1.4 Посилання
- 1.5 Означення та аббревіатури

2. ЗАГАЛЬНИЙ ОПИС

- 2.1 Перспективи продукту
- 2.2 Функції продукту
- 2.3 Характеристики користувачів
- 2.4 Загальні обмеження
- 2.5 Припущення й залежності

3. КОНКРЕТНІ ВИМОГИ

- 3.1 Вимоги до зовнішніх інтерфейсів
 - 3.1.1 Інтерфейс користувача
 - 3.1.2 Апаратний інтерфейс
 - 3.1.3 Програмний інтерфейс
 - 3.1.4 Комунікаційний протокол
 - 3.1.5 Обмеження пам'яті
 - 3.1.6 Операції
 - 3.1.7 Функції продукту
 - 3.1.8 Припущення й залежності
- 3.2 Властивості програмного продукту
- 3.3 Атрибути програмного продукту
 - 3.3.1 Надійність
 - 3.3.2 Доступність

3.3.3 Безпека

3.3.4 Супроводжуваність

3.3.5 Переносимість

3.3.6 Продуктивність

3.4 Вимоги бази даних

3.5 Інші вимоги

1. ВСТУП

1.1 Огляд продукту

Метою даного продукту є створення онлайн-платформи для моніторингу та оцінки техніки виконання підтягувань із підтримкою IoT-сенсорів. Система приймає телеметрію (висоту підйому, кут нахилу тулуба, час між повтореннями), візуалізує ці метрики в особистому кабінеті користувача та автоматично формує рекомендації для покращення техніки.

Цільова аудиторія охоплює:

- спортсменів-аматорів, які тренуються вдома й потребують об'єктивного контролю;
- просунутих атлетів і персональних тренерів, що аналізують прогрес секцій або груп;
- адміністраторів фітнес-клубів, які прагнуть надати клієнтам «розумні» снаряди з телеметрією.

1.2 Мета

Платформа повинна забезпечити:

- збір і зберігання біомеханічних показників кожного повторення;
- автоматичне розпізнавання помилок (недостатня висота, надмірний нахил, нерівномірний темп);
- генерацію персональних підказок у реальному часі;
- централізоване адміністрування користувачів і тренувальних сесій;
- доступність із будь-якого браузера без необхідності складного калібрування.

1.3 Межі

Продукт охоплює:

- ESP32-датчики (акселерометр + ультразвук) → HTTP POST → Spring Boot 17 REST-API;
- PostgreSQL 16 для зберігання користувачів, сесій, метрик, рекомендацій;
- SPA на Vue 3 + TailwindCSS + Pinia + Chart.js;
- Docker Compose + NGINX-reverse-proxy.

За межами проєкту: підтримка інших вправ, мобільний застосунок, інтеграція з медичними діагностичними пристроями.

1.4 Посилання

- ISO/IEC 25010 – Якісні характеристики ПЗ:
(<https://www.iso.org/standard/35733.html>);
- Vue.js 3 Docs – (<https://vuejs.org/>);
- Spring Boot Documentation – (<https://spring.io/>);
- PostgreSQL Docs – (<https://www.postgresql.org/docs/>);
- Docker Compose Reference – (<https://docs.docker.com/compose/>).

1.5 Означення та аббревіатури

Означення:

- Метрика – запис одного повторення вправи, що містить висоту стрибка, кут виконання, час у повітрі та оцінку якості;
- Сесія – тренувальна сесія, яка складається з послідовності метрик, зібраних під час одного заняття;
- Рекомендація – автоматично сформований текст із порадами для покращення результатів на основі аналізу метрик.

Аббревіатури:

- SPA (Single Page Application) – односторінковий вебзастосунок;
- REST (Representational State Transfer) – архітектурний стиль для API;

- API (Application Programming Interface) – інтерфейс прикладного програмування;
- DB (Database) – база даних;
- IoT (Internet of Things) – інтернет речей, технологія для підключення пристроїв;
- ESP32 – мікроконтролер для IoT-пристроїв;
- UI/UX (User Interface/User Experience) – користувацький інтерфейс та досвід взаємодії.

2. ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи продукту

Платформа може масштабуватися до інших силових вправ (віджимання, присідання), інтегруватися з фітнес-браслетами для пульсомоніторингу, а також слугувати освітнім інструментом на уроках фізкультури чи кафедрах спортивної науки.

2.2 Функції продукту

- створення тренувальної сесії;
- прийом потокової телеметрії від ESP32;
- автоматичний інкремент лічильника повторень;
- побудова графіків:
 - висота підйому;
 - кут нахилу тулуба;
 - інтегральний correctnessScore;
- евристичний аналіз і текстові рекомендації;
- інтерактивна історія сесій;
- експорт вибраної сесії у CSV / JSON (для сторонньої аналітики);
- адмін-панель: CRUD користувачів;
- REST-API для сторонніх клієнтів.

2.3 Характеристики користувачів

Користувачі поділяються на декілька категорій, кожна з яких має унікальний набір прав:

- Гість — Перегляд лендингу, реєстрація, запуск/завершення сесії, перегляд графіків, рекомендації.
- Адміністратор — CRUD користувачів.

Такий розподіл забезпечує чітку ізоляцію дій і зменшує ризики помилок або порушень.

2.4 Загальні обмеження

При розробці та впровадженні системи слід враховувати низку обмежень:

- перегляд графіків, рекомендації.ESP32 працює лише у мережах 2.4 GHz Wi-Fi;
- необхідна первинна калібрація датчиків (≤ 1 хв);
- клієнтський кеш браузера ≤ 100 МБ;
- затримка від датчика до відображення – ≤ 1000 мс;
- сервіс не надає медичних діагнозів, рекомендації мають інформаційний характер.

2.4 Припущення й залежності

Проект має наступні припущення та залежності:

- браузерна підтримка. Платформа повинна підтримувати останні версії Chrome, Firefox, Edge. Тестування на Safari або старих Android WebView не гарантує повної сумісності. Браузери користувачів підтримують ES2020;
- сервер і БД запускаються у Linux-контейнерах Docker Compose;
- IoT-модуль використовує HTTP POST для надсилання телеметрії;
- масштабування кластера планується через Kubernetes.

3. КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

У цьому розділі визначаються, як система буде взаємодіяти з іншими системами, користувачами та пристроями. Деталізується, яким чином продукт обмінюватиметься інформацією з іншими системами чи пристроями.

3.1.1 Інтерфейс користувача

Інтерфейс вебдодатка має бути інтуїтивно зрозумілим і адаптивним до різних розмірів екранів (мобільні пристрої, планшети, настільні ПК).

Передбачено підтримку світлої та темної теми. Основні елементи: навігаційне меню, панель поточної сесії, лічильник повторень, інтерактивні графіки висоти та кута нахилу, таблиця історичних сесій, модуль текстових рекомендацій і адмін-панель керування користувачами. Спортсмен може запускати й зупиняти тренувальну сесію, переглядати результати й експортувати дані; тренер – переглядати статистику декількох спортсменів; адміністратор – виконувати повний CRUD.

3.1.2 Апаратний інтерфейс

Вебчастина працює у сучасних браузерах (Chrome, Firefox, Edge, Safari). Для коректної роботи рекомендуються пристрої з ≥ 4 ГБ RAM, двоядерним процесором ≥ 2 ГГц та стабільним інтернет-з'єднанням від 5 Мбіт/с. IoT-модуль базується на ESP32 (2,4 GHz Wi-Fi): акселерометр/гіроскоп для кута нахилу й ультразвуковий сенсор для висоти підйому.

3.1.3 Програмний інтерфейс

Додаток на стороні сервера реалізовано у вигляді RESTful API (Java Spring Boot). Передбачено енд-поїнти для:

- реєстрації / входу;
- створення сесій;
- прийому телеметрії (HTTP POST JSON < 1 с затримки);
- отримання статистики й рекомендацій;
- адмін-операцій CRUD.

Для зберігання використовується PostgreSQL. Фронтенд (Vue 3) взаємодіє з API через HTTPS; IoT-модуль надсилає метрики тим самим REST-інтерфейсом.

3.1.4 Комунікаційний протокол

Передача даних здійснюється виключно через HTTPS із підтримкою TLS 1.3. ESP32 надсилає дані HTTP POST-запитом. Передбачена підтримка CORS для безпечної взаємодії між фронтендом та API.

3.1.5 Обмеження пам'яті

Вебклієнт не повинен споживати понад 200 МБ ОЗП при стандартному навантаженні. На ESP32 кеш телеметрії обмежено 64 КБ. База даних оптимізується за допомогою індексів і партиціювання; старі сирі метрики можуть архівуватися.

3.1.6 Операції

Користувачі можуть:

- створювати / завершувати тренувальні сесії;
- відправляти телеметрію (ESP32 → API);
- переглядати графіки, статистику та текстові поради;

- переглядати історію тренувань.

Адміністратор виконує CRUD для користувачів, вправ і сесій.

3.1.7 Функції продукту

Система забезпечує:

- Реєстрація, вхід;
- Запуск тренувальної сесії й прийом поточкових метрик у реальному часі;
- Візуалізація: графіки висоти та кута, лічильник повторень, середні значення;
- Евристичний розрахунок correctnessScore, формування текстових рекомендацій;
- Адмін-панель CRUD користувачів.

3.1.8 Припущення й залежності

Система працює в середовищі Linux (Docker). Для розгортання використовується Docker Compose. Очікується, що користувачі мають стабільне інтернет-з'єднання. Підтримка браузерів — останні версії Chrome, Firefox. Система сумісна з будь-якою SQL-сумісною СУБД, але орієнтована на PostgreSQL. За потреби горизонтально масштабується в Kubernetes. Клієнтські браузери підтримують ES2020. ESP32 вимагає стабільного Wi-Fi 2,4 GHz.

3.2 Властивості програмного продукту

Програмне забезпечення реалізує розгалужену логіку моніторингу та оцінки техніки виконання підтягувань із підтримкою декількох ролей (спортсмен, тренер, адміністратор). Система здатна працювати як у режимі

одного користувача, так і в мережевій конфігурації з кількома клієнтами та віддаленим адмініструванням. Завдяки мікросервісному підходу підтримується гнучке масштабування й додавання нових модулів — наприклад аналітики інших вправ, мобільного застосунку або розширеної статистики групових тренувань.

3.3 Атрибути програмного продукту

3.3.1 Надійність

Система побудована з урахуванням обробки всіх виняткових ситуацій, дублюванням вимірювань у кеші ESP32 та резервним копіюванням бази даних через cron-job. Docker Compose перезапускає контейнери у разі збою, забезпечуючи безперервну роботу 24/7.

3.3.2 Доступність

Доступ до вебклієнта можливий з будь-якого сучасного браузеру. Контейнеризація та можливе розгортання в Kubernetes дозволяють швидко масштабувати бекенд та балансувати навантаження при пікових сесіях.

3.3.3 Безпека

Передбачено валідацію запитів, обмеження прав доступу, CORS. Дані шифруються при передачі. Фото зберігаються в окремому об'єктному сховищі MinIO. Паролі зберігаються у вигляді bcrypt-хешів.

3.3.4 Супроводжуваність

Код дотримується стандартів Java Code Conventions і Vue Style Guide. CI/CD налаштовано через GitHub Actions — автоматичне тестування, побудова контейнерів і розгортання на staging/production.

3.3.5 Переносимість

Увесь стек (PostgreSQL, Spring Boot, Vue.js, NGINX) запаковано в Docker-контейнери, що забезпечує незалежність від ОС хоста й можливість швидкого перенесення на інший сервер або у хмару.

3.3.6 Продуктивність

Система оптимізована для обробки до 100 одночасних активних тренувальних сесій із затримкою отримання метрик ≤ 1 с. Часті запити кешуються у Redis; для звітів застосовуються матеріалізовані представлення.

3.4 Вимоги бази даних

PostgreSQL зберігає дані про користувачів, вправи, сесії, метрики, рекомендації. Усі ключові поля проіндексовані; для вибірок за історією сесій створені materialized views. Логи подій і телеметрії зберігаються окремо для аналітики. Міграції виконуються Flyway.

3.5 Інші вимоги

Захист персональних даних. Система відповідає вимогам GDPR; користувач може експортувати чи видалити свої дані.

Правові обмеження. Програма не формує медичних діагнозів; рекомендації — лише тренувальні поради.

Енергоефективність. Контейнерний підхід і моніторинг ресурсів дозволяють оптимізувати споживання CPU/RAM; неблокуючі I/O та кешування мінімізують навантаження на інфраструктуру.

ДОДАТОК Г

Додаток Г UML діаграми та зображення інтерфейсу

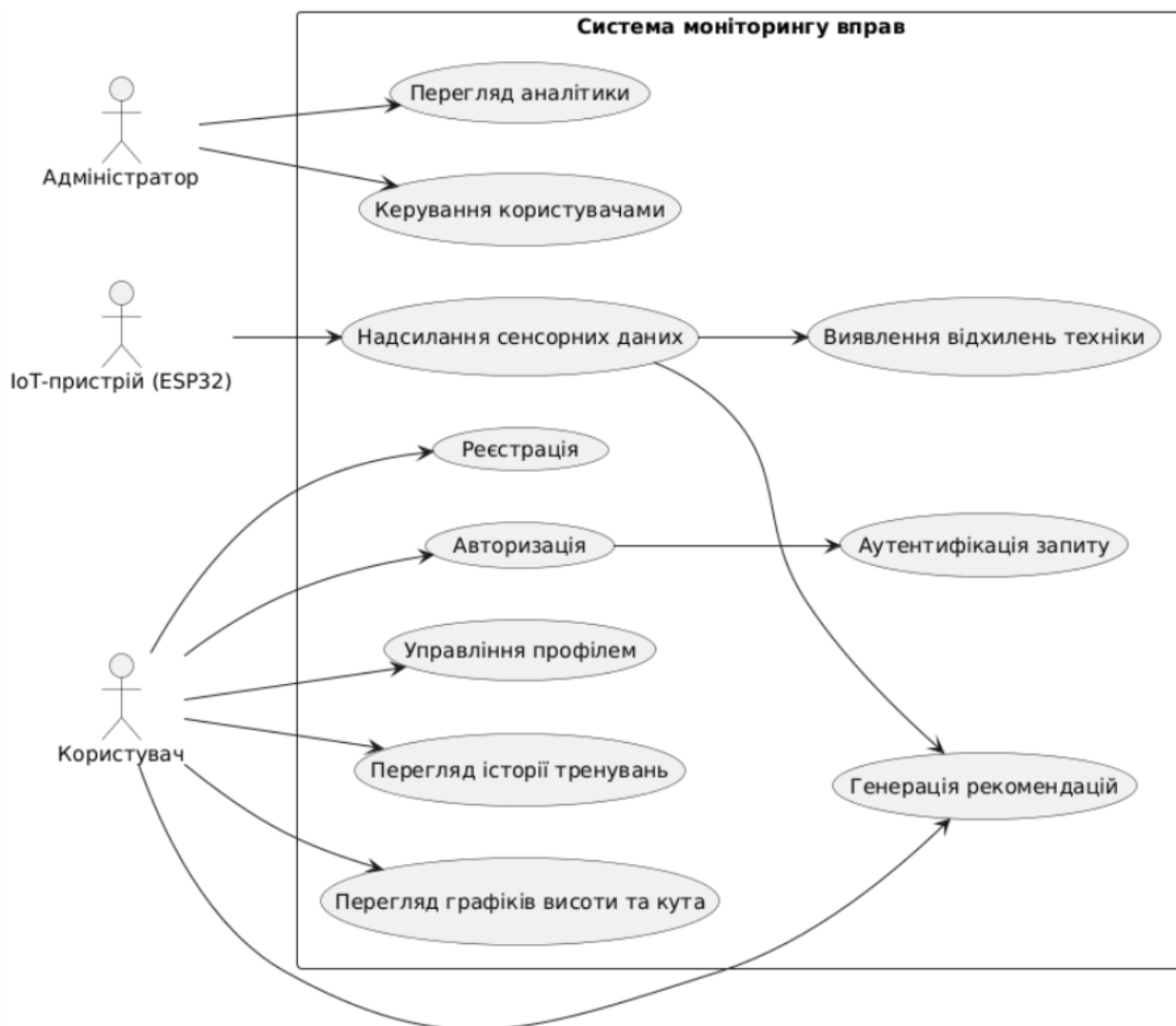


Рисунок Г.1 – Узагальнена UML-діаграми прецедентів (рисунок виконаний самостійно)



Рисунок Г.2 – UML-діаграми прецедентів, що описує процес авторизації та реєстрації (рисунок виконаний самостійно)



Рисунок Г.3 - UML-діаграми прецедентів взаємодії з тренуваннями користувача (рисунок виконаний самостійно)

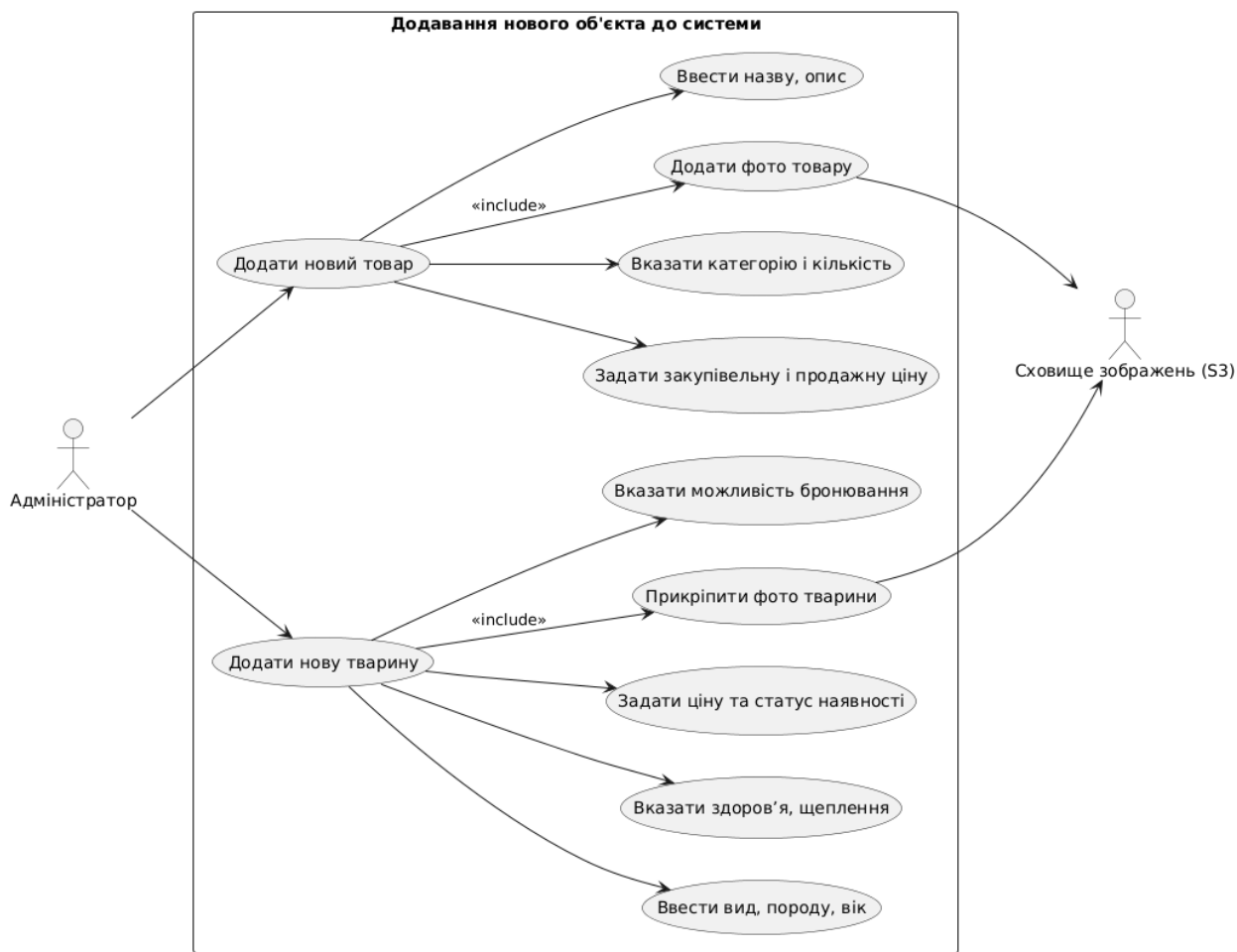


Рисунок Г.4 - UML-діаграми прецедентів, яка ілюструє процес додавання нового товару або тварини до системи (рисунок виконаний самостійно)

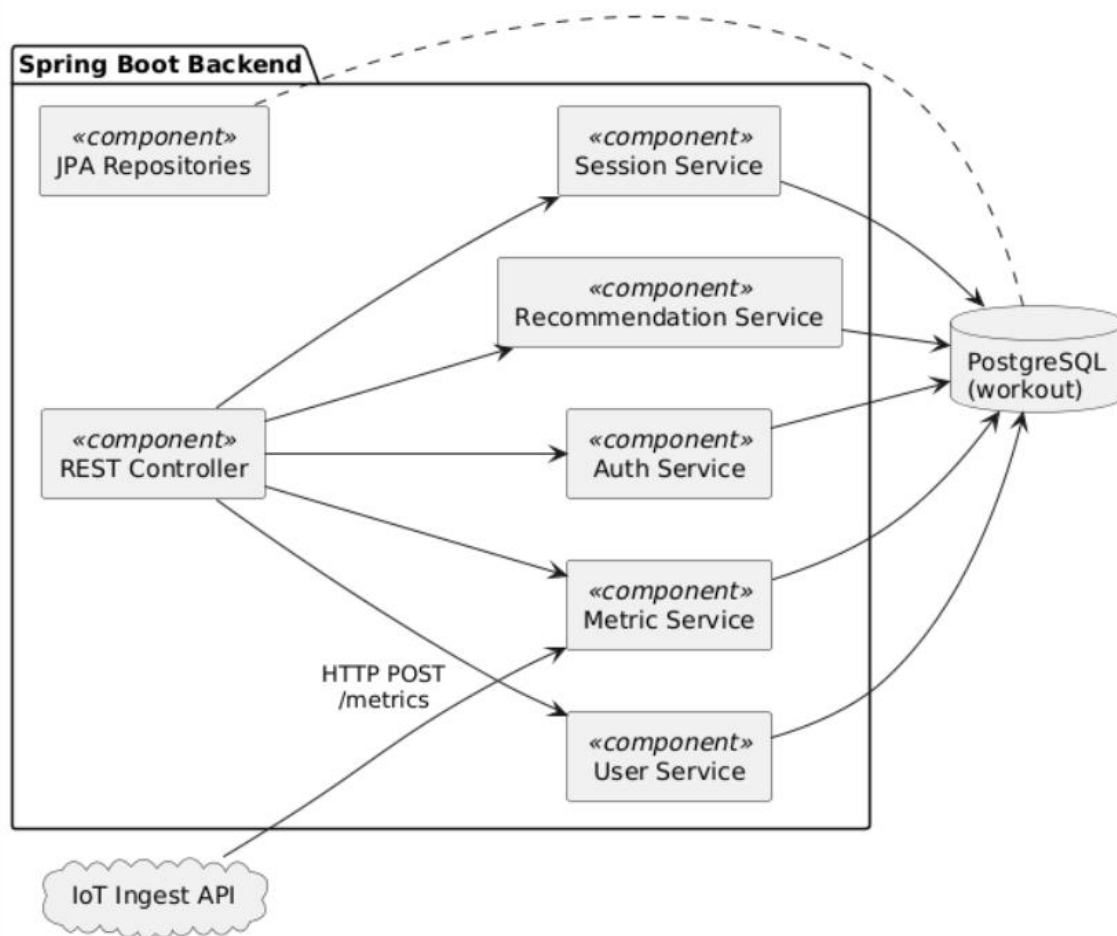


Рисунок Г.5 - UML діаграма компонентів (рисунок виконаний самостійно)

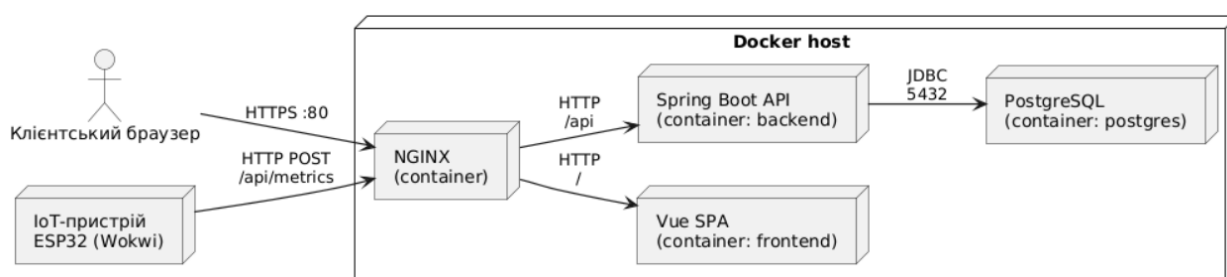


Рисунок Г.6 - UML діаграма розгортання (рисунок виконаний самостійно)

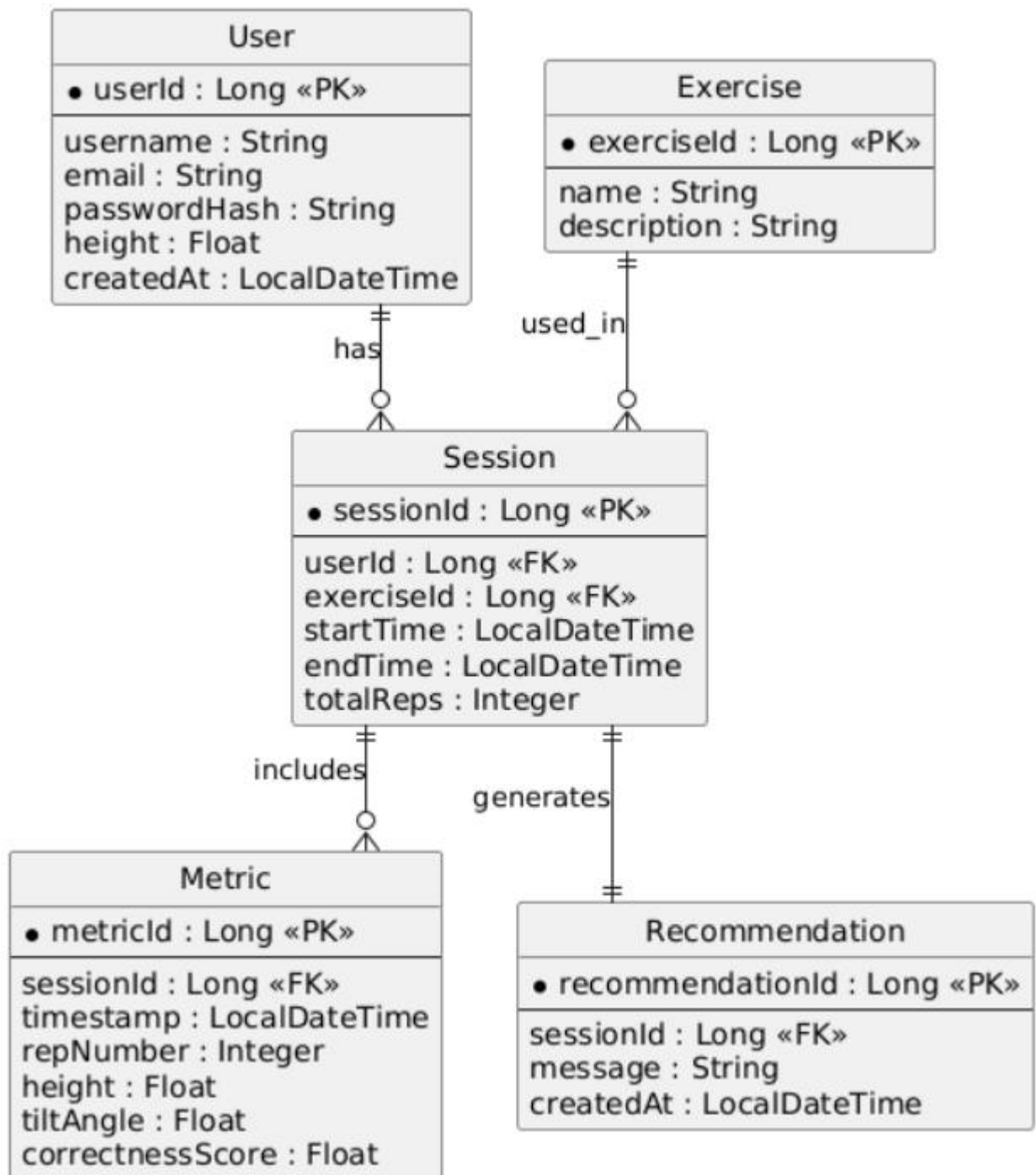


Рисунок Г.7 – Схема даних (рисунок виконаний самостійно)

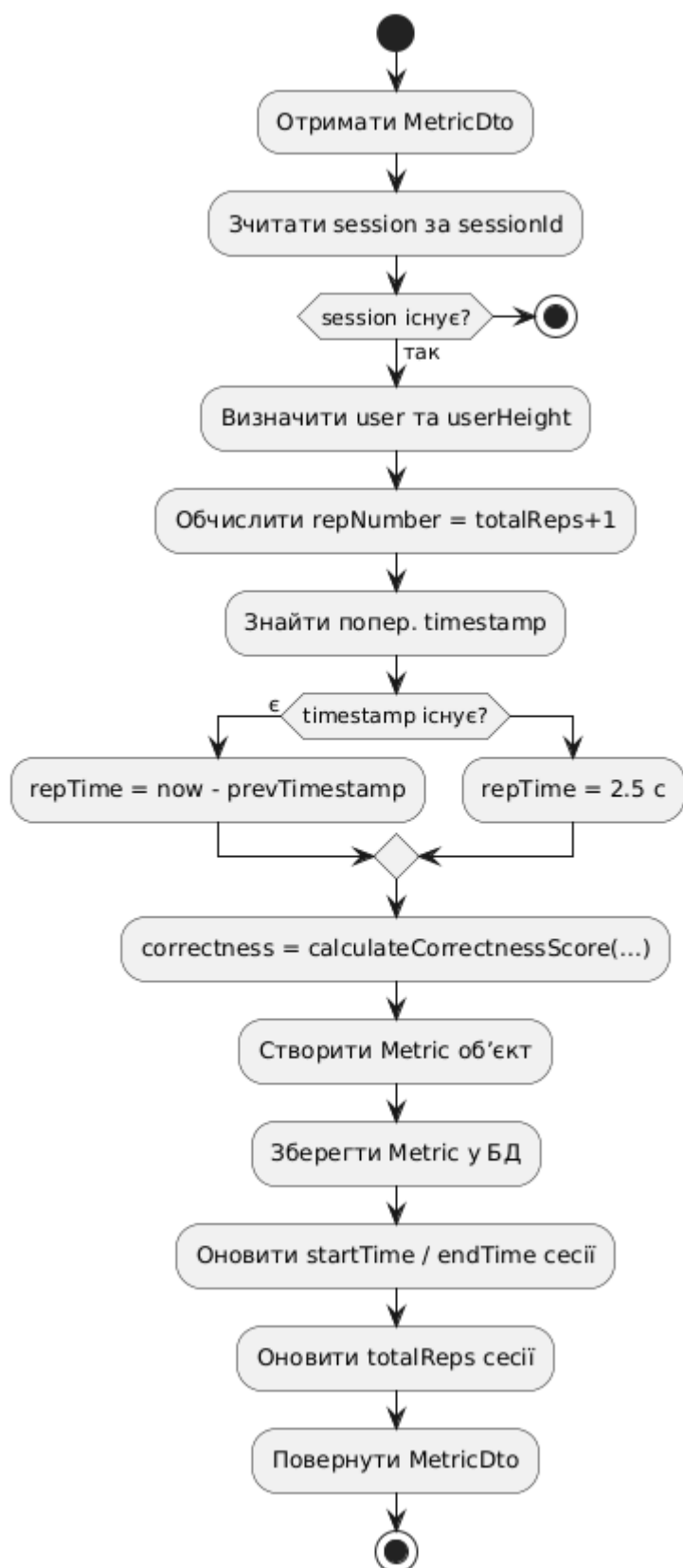


Рисунок Г.8 – Діаграма активності для обробки нової метрики (рисунок виконаний самостійно)

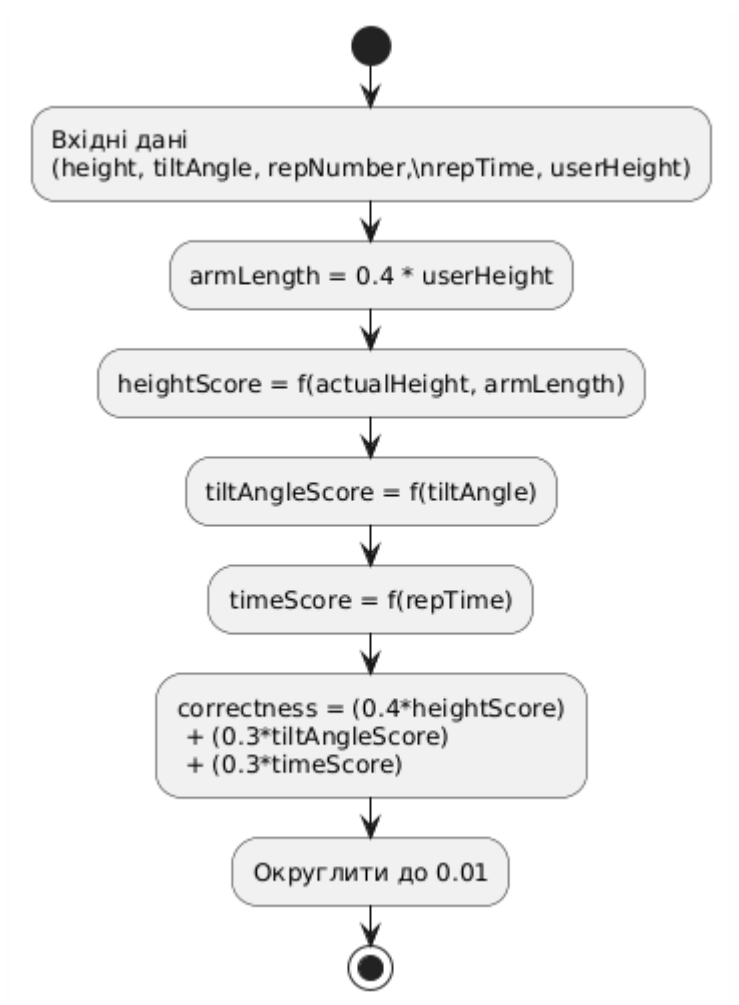


Рисунок Г.9 – Діаграма активностей для розрахунку показника правильності
(рисунок виконаний самостійно)

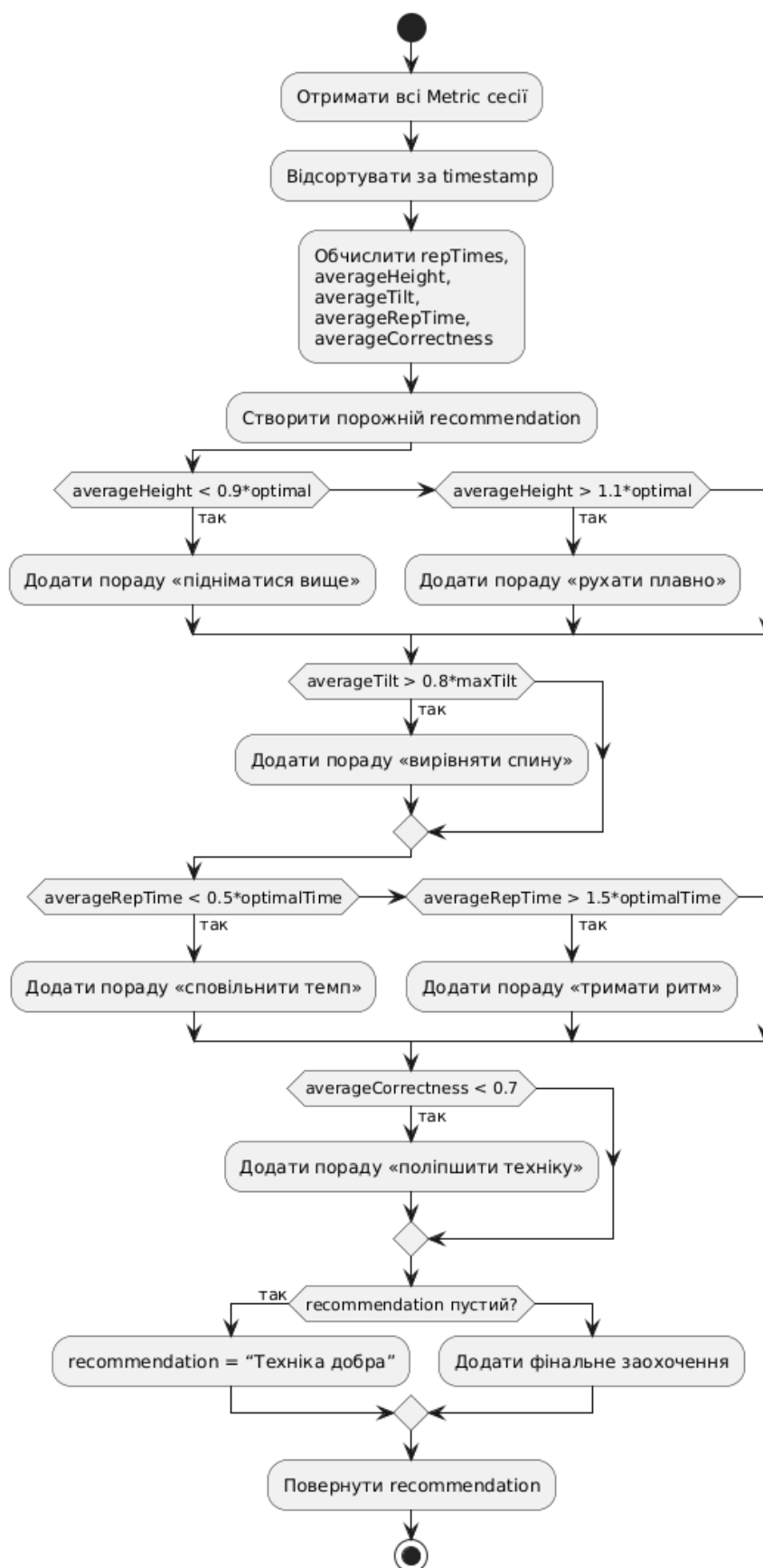


Рисунок Г.10 – Діаграма активностей для генерації рекомендації для сесії
(рисунок виконаний самостійно)

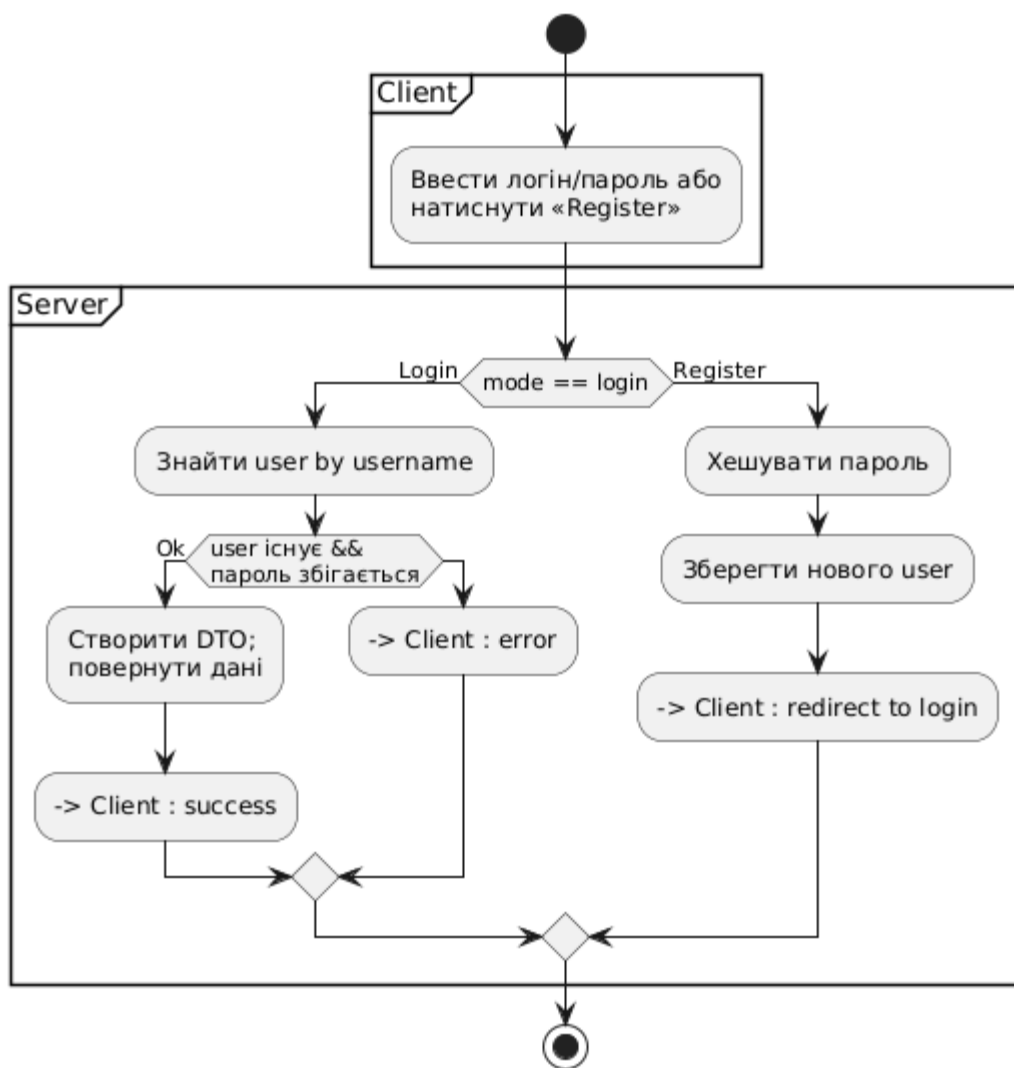


Рисунок Г.11 – Діаграма активностей для автентифікації (рисунок виконаний самостійно)