БАКАЛАВРСЬКА РОБОТА

КРБ.СІ -06.00.00.000 ПЗ

Група СІ-21-1

Менчук Олег

2025

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра інформаційно-телекомунікаційних технологій та систем

Менчук Олег Ігорович
(прізвище, ім'я, по батькові)

УДК 004.9

БАКАЛАВРСЬКА РОБОТА

Розроблення автоматизованої системи моніторингу оптимального стану спортсмена під час тренування
(назва роботи)

Системна інженерія - Інтернет речей
(назва освітньої програми)

151 Автоматизація та комп'ютерно-інтегровані технології
(шифр і назва спеціальності)

Здобувач освітнього ступеня Менчук О.І.
(підпис, ініціали та прізвище здобувача)

Науковий керівник к.т.н., доц. каф. ІТТС Волинський О.І.
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри
д.т.н., проф. Заміховський Л. М.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ 2025

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Інститут інформаційних технологій

Кафедра інформаційно-телекомунікаційних технологій та систем

Освітній рівень бакалавр

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

Заміховський Л. М.

«18» травня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Менчук Олег Ігорович

(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення автоматизованої системи моніторингу оптимального стану спортсмена під час тренування

керівник роботи к.т.н., доц. каф. ІТТС Волинський О.І.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «05» травня 2025 року №281/17

2. Строк подання студентом роботи з 08.04.2025р. по 15.06.2025р.

3. Вихідні дані до роботи Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1 Теоретичні основи застосування цифрових технологій для підтримки фізичного стану спортсмена

2 Проєктування автоматизованої системи підтримки оптимального стану спортсмена

3 Реалізація та впровадження автоматизованої системи в умовах тренувального середовища

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Загальна архітектура системи моніторингу фізіологічногостану спортсмена. Схема-структурна

Структура бази даних. Схема структурна

6. Консультанти розділів роботи

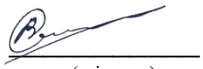
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	к.т.н., доц. каф. ІТТС Волинський О.І.		
2	к.т.н., доц. каф. ІТТС Волинський О.І.		
3	к.т.н., доц. каф. ІТТС Волинський О.І.		

7. Дата видачі завдання 08.04.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	08.04.2025 – 09.04.2025	
2	Створення плану бакалаврської роботи	10.04.2025 – 14.04.2025	
3	Збір інформації	15.04.2025 – 29.04.2025	
4	Проведення аналізу предметної області	30.04.2025 – 03.05.2025	
5	Оформлення технічного завдання	04.05.2025 – 06.05.2025	
6	Проведення моделювання процесів	07.05.2025 – 09.05.2025	
7	Розробка проєкту	10.05.2025 – 12.06.2025	
8	Здача пояснювальної записки	13.06.2025 – 14.06.2025	

Студент _____ Менчук О.І.
(підпис) (прізвище та ініціали)

Керівник роботи  Волинський О.І.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розрахунково-пояснювальна записка: 117 сторінок, 43 рисунка, 26 таблиці, 60 посилань, 1 додаток.

Метою даної роботи є розроблення автоматизованої системи, здатної здійснювати безперервне зчитування біометричних даних під час тренування, їх обробку, збереження та відображення у зручному для користувача форматі, а також реалізацію механізмів сповіщення у разі виявлення критичних фізіологічних станів.

Об'єктом досліджень є процес підтримки оптимального функціонального стану спортсмена під час тренувальної діяльності за допомогою цифрових технологій.

В першому розділі проаналізовано як цифрові технології дозволяють ефективно відстежувати, аналізувати й оптимізувати фізичний стан спортсменів, підвищуючи безпеку та результативність тренувального процесу.

В другому розділі було комплексно розглянуто, як технічно спроєктувати, реалізувати та захистити цифрову систему, що забезпечує реальний моніторинг стану спортсмена, від збору даних до їх збереження, візуалізації й аналітики.

В третьому розділі кваліфікаційної роботи вивчалось реалізацію та впровадження автоматизованої системи в умовах тренувального середовища**,** тобто практичне застосування створеної системи у реальних та симульованих умовах для перевірки її ефективності.

Розроблена система успішно реалізована, протестована та впроваджена в реальні умови. Підтверджено, що цифрові технології можуть істотно покращити результати спортсменів, зменшити ризики і підвищити ефективність тренувального процесу.

ABSTRACT

Explanatory note: 117 pages, 43 figures, 26 tables, 60 references, 1 appendix.

The aim of this work is to develop an automated system capable of continuously reading biometric data during training, processing and storing this information, and displaying it in a user-friendly format, as well as implementing alert mechanisms in case of detection of critical physiological states.

The object of the study is the process of maintaining the optimal functional state of an athlete during training using digital technologies.

The first chapter analyzes how digital technologies enable effective monitoring, analysis, and optimization of athletes' physical condition, thereby improving the safety and performance of the training process.

The second chapter provides a comprehensive overview of the technical design, implementation, and data protection strategies of a digital monitoring system — from data acquisition to storage, visualization, and analytics.

The third chapter focuses on the practical implementation and deployment of the developed system in real and simulated training environments, with the aim of testing its efficiency and functionality.

The developed system was successfully implemented, tested, and deployed under real-world conditions. It has been confirmed that digital technologies can significantly improve athletic performance, reduce the risk of overload and injury, and enhance the overall effectiveness of the training process.

ЗМІСТ

					КРБ.СІ.-06.00.00.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Менчук О.І.			Розроблення автоматизованої системи моніторингу оптимального стану спортсмена під час тренування	Літ.	Арк. 7 / Аркушів 117
Перевір.		Волинський О.І.					
Реценз.						ІФНТУНГ СІ-21-1	
Н. Контр.		Возний А.В					
Затверд.		Заміховський Л.М					

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Позначення / Скорочення	Розшифрування / Пояснення
SpO_2	Saturation of Peripheral Oxygen – насичення крові киснем
T°C	Температура тіла в градусах Цельсія
GPS	Global Positioning System – система глобального позиціонування
MQTT	Message Queuing Telemetry Transport – протокол обміну повідомленнями
API	Application Programming Interface – інтерфейс прикладного програмування
UI	User Interface – інтерфейс користувача
JWT	JSON Web Token – токен авторизації
DB	Database – база даних
ESP32	Енергоефективний мікроконтролер з Wi-Fi/Bluetooth
JSON	JavaScript Object Notation – формат обміну даними
UI/UX	User Interface / User Experience – інтерфейс і досвід користувача
REST	Representational State Transfer – архітектурний стиль побудови API
IDE	Integrated Development Environment – інтегроване середовище розробки

ВСТУП

Актуальність дослідження. У сучасному світі зростає потреба у впровадженні інноваційних цифрових технологій у сферу фізичної культури та спорту. Постійне підвищення вимог до результативності спортсменів, збільшення обсягів тренувального навантаження, необхідність дотримання індивідуальних підходів до навантажень і відновлення потребують точного, об'єктивного та безперервного контролю за фізіологічним станом спортсмена. У зв’язку з цим автоматизація процесів спостереження за біометричними показниками під час тренування стає не лише актуальним, а й необхідним напрямом розвитку інформаційних систем. Особливої значущості ця проблема набуває в умовах інтенсивного тренувального процесу, де будь-яке перевантаження або несвоєчасне реагування на критичний стан організму може призвести до зниження результатів або навіть травм.

Завдяки розвитку вбудованих обчислювальних пристроїв, бездротових мереж, цифрових сенсорів та мобільного програмного забезпечення стало можливим створення комплексних автоматизованих систем, здатних у режимі реального часу зчитувати, обробляти, зберігати й візуалізувати біометричні дані. Такі системи дозволяють не лише здійснювати спостереження за спортсменом у процесі фізичної активності, але й надавати йому зворотний зв’язок, формувати звіти, проводити статистичний аналіз і прогнозувати ризики перевантажень. Особливої уваги в цьому контексті заслуговує можливість адаптації систем до індивідуальних особливостей спортсменів, що значно підвищує ефективність тренувального процесу.

Актуальність теми дипломної роботи зумовлена загальною тенденцією цифровізації спортивної медицини, потребою в створенні вітчизняних доступних рішень для підтримки фізичного стану спортсменів, а також браком у відкритому доступі спеціалізованих програмно-апаратних комплексів, які могли б ефективно функціонувати в умовах реального тренування.

Розробка автоматизованої системи, яка забезпечує постійний моніторинг фізіологічних показників, їхню обробку, візуалізацію та сигналізацію у разі критичних значень, є важливим етапом у підвищенні якості фізичної підготовки, збереженні здоров’я спортсменів та удосконаленні методик тренування.

Але, незважаючи на це, сьогодні існує потреба у дослідженні, яке б узагальнило, систематизувало існуючі відомості з даної проблеми.

Враховуючи все вищесказане, нами і була обрана тема дипломної роботи "Розроблення автоматизованої системи підтримки оптимального стану спортсмена під час тренування".

Об’єкт дослідження – процес підтримки оптимального функціонального стану спортсмена під час тренувальної діяльності за допомогою цифрових технологій.

Предмет дослідження – програмно-апаратні рішення для моніторингу, обробки та візуалізації фізіологічних показників спортсмена в режимі реального часу.

Мета дослідження – розроблення автоматизованої системи, здатної здійснювати безперервне зчитування біометричних даних під час тренування, їх обробку, збереження та відображення у зручному для користувача форматі, а також реалізацію механізмів сповіщення у разі виявлення критичних фізіологічних станів.

Завдання дослідження:

1) Визначити ключові фізіологічні параметри, що є інформативними для оцінювання стану організму під час фізичних навантажень;
2) проаналізувати сучасні апаратні та програмні засоби для збору, передавання і зберігання біометричних даних;
3) обґрунтувати вибір технічної архітектури системи відповідно до умов спортивної практики;
4) реалізувати програмні модулі для взаємодії з сенсорами, обробки даних та їх передавання на сервер;

5) створити інтерфейс користувача для візуалізації параметрів у реальному часі та історії тренувань;
6) протестувати систему в симульованих та реальних умовах з метою перевірки її стабільності, точності та зручності використання;
7) оцінити ефективність впровадженого рішення та сформулювати рекомендації щодо його подальшого вдосконалення.

Методи дослідження. Методи дослідження, які застосовувались у роботі, включають аналіз наукових і технічних джерел щодо систем цифрового моніторингу, порівняльний аналіз апаратного та програмного забезпечення, проєктування архітектури інформаційних систем, розроблення програмних модулів, використання засобів моделювання у середовищах Arduino IDE, Visual Studio Code та Figma, проведення функціонального і модульного тестування, а також апробацію розробленої системи у практичних тренувальних умовах з подальшим аналізом результатів.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел, що містить 60 найменувань. Повний обсяг роботи 117 сторінок.

1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ ЦИФРОВИХ ТЕХНОЛОГІЙ ДЛЯ ПІДТРИМКИ ФІЗИЧНОГО СТАНУ СПОРТСМЕНА

1.1 Фізіологічні параметри, що відображають стан організму під час тренувального навантаження

У сучасному спортивному середовищі особливе значення набуває точне й оперативне оцінювання фізіологічного стану спортсмена під час тренувального навантаження. Розуміння змін у функціонуванні організму в реальному часі дозволяє оптимізувати індивідуальний тренувальний процес, запобігати перенавантаженню, знижувати ризик травматизму та досягати стабільних результатів у спортивних змаганнях. Ефективне функціонування автоматизованої системи підтримки оптимального стану спортсмена базується саме на обробці та інтерпретації комплексу ключових фізіологічних показників, які слід ураховувати під час проєктування цифрових рішень у галузі спорту [1, с. 66].

Одним з найважливіших параметрів, що відображає реакцію організму на фізичне навантаження, є частота серцевих скорочень (ЧСС). Цей показник відображає інтенсивність роботи серцево-судинної системи та є універсальним маркером рівня напруженості тренування. У стані спокою значення ЧСС становить у середньому від 60 до 90 ударів за хвилину, проте під час активного навантаження може перевищувати 160–180 ударів за хвилину залежно від віку, статі, фізичної підготовки та виду діяльності спортсмена. Надмірне або занадто різке підвищення ЧСС свідчить про небажане навантаження та ризик перевтоми. У цифрових системах цей показник зазвичай зчитується за допомогою оптичних або електричних сенсорів, вбудованих у носимі пристрої.

Ще одним показником, тісно пов’язаним із серцево-судинною активністю, є варіабельність серцевого ритму (HRV). Цей параметр характеризує зміну інтервалів між окремими серцевими скороченнями й дозволяє оцінити баланс

між симпатичним і парасимпатичним відділами вегетативної нервової системи. Високий рівень HRV у стані спокою свідчить про здатність організму ефективно адаптуватися до навантаження та швидко відновлюватися. Низький HRV, навпаки, сигналізує про втому, стрес або перевантаження. Визначення цього параметра вимагає високої точності вимірювань, тому системи контролю мають забезпечувати збирання даних з мінімальною похибкою.

Не менш інформативним показником є рівень насичення крові киснем (SpO_2). Він визначає, наскільки ефективно функціонує дихальна система спортсмена та чи достатньо організм забезпечений киснем у момент навантаження. Показники нижче 95% можуть свідчити про гіпоксію або інші проблеми з вентиляцією легень, що вимагає термінової реакції з боку тренера або медичного персоналу. У цифрових системах для вимірювання SpO_2 використовуються оптичні сенсори з технологією фотоплетизмографії, вбудовані в смарт-браслети чи наручні пристрої [2, с. 56].

Важливим фізіологічним параметром, що дозволяє оцінити ефективність терморегуляції організму під час навантаження, є температура тіла. Перегрів може спричинити зниження фізичної витривалості, погіршення координації та навіть тепловий удар. Особливо критичним є контроль температури тіла в умовах високої вологості або на відкритому повітрі влітку. Цифрові датчики температури дають змогу оперативно виявляти підвищення температури на шкірі чи в центральних ділянках тіла (в залежності від типу сенсора), що дозволяє вчасно коригувати навантаження або припинити тренування.

Стан м’язової системи можна оцінювати за допомогою електроміографічних показників. Поверхнева електроміографія (ЕМГ) дає змогу виявити ступінь напруження окремих м’язів, асиметрію у навантаженні та втому м’язових груп. Ці дані особливо актуальні у видах спорту, де важлива точність руху або симетрія роботи кінцівок, наприклад, у легкій атлетиці, гімнастиці чи бойових мистецтвах. ЕМГ-датчики кріпляться до шкіри спортсмена й передають сигнали до обчислювального модуля для подальшої обробки та виведення графічної інформації.

До ще одного важливого аспекту фізіологічного контролю відноситься рівень фізичної активності та переміщення у просторі. Акселерометри та гіроскопи, вбудовані в сучасні смарт-пристрої, дозволяють фіксувати рухову активність, швидкість, амплітуду рухів, частоту кроків та інші параметри. У комбінації з GPS-модулями можна відслідковувати темп руху, пройдену дистанцію, зміни висоти або траєкторію тренувального маршруту. Ці дані дозволяють оцінити ефективність тренування, навантаження на опорно-руховий апарат і відповідність реального режиму заданому плану.

Крім суто фізіологічних параметрів, сучасні системи підтримки стану спортсмена часто враховують суб'єктивні показники самопочуття. Це може бути рівень втоми, стресу, мотивації чи сну, який фіксується через мобільний застосунок або формується автоматично за допомогою спеціальних алгоритмів на основі фізіологічних змін. Наприклад, тривале зниження варіабельності серцевого ритму в поєднанні з поганою якістю сну може сигналізувати про необхідність зменшення інтенсивності тренувального навантаження.

Стан енергетичного обміну спортсмена можна побічно оцінити за допомогою вимірювання рівня глюкози, лактату та інших біохімічних маркерів. На сьогодні існують експериментальні моделі сенсорів, які дозволяють неінвазивно зчитувати ці параметри через шкіру або піт. Хоча вони ще не набули широкого впровадження, але активно тестуються для подальшого використання в індивідуальному спортивному моніторингу. Їх інтеграція у цифрові системи дозволить комплексно оцінювати метаболічну відповідь організму на фізичне навантаження [3, с. 65].

Окрему категорію складають параметри, пов'язані з психофізіологічною реакцією організму. Це частота дихання, електрична активність шкіри, рівень стресу чи когнітивне навантаження. Вони особливо актуальні в умовах змагань, коли навіть при стабільних фізичних показниках психоемоційний стан може впливати на результат. Автоматизовані системи, що інтегрують ці параметри, забезпечують глибший рівень адаптації тренувального процесу під особливості спортсмена.

Для повноцінної роботи такої системи необхідна не лише фіксація поточних значень, але й збереження історичних даних. Динаміка фізіологічних показників упродовж днів або тижнів дозволяє тренерам та медичним працівникам виявляти тенденції, відхилення або приховані ризики. Наприклад, поступове зниження рівня HRV за кілька днів може свідчити про приховану втому, навіть якщо ЧСС і SpO_2 залишаються в межах норми. Такий підхід дає змогу проводити індивідуальну корекцію тренувального процесу та зменшити ризик перевантаження.

Не менш важливим є питання точності, частоти збору та швидкості обробки фізіологічних даних. Для оперативного реагування система має забезпечувати обробку сигналів у режимі реального часу з мінімальною затримкою. Це особливо актуально для функцій, що передбачають візуальне або звукове оповіщення користувача у випадку досягнення критичних значень параметрів. Надійність та стабільність таких рішень безпосередньо впливають на безпеку та ефективність тренувального процесу [4, с. 49].

Зі сказаного випливає, що для створення якісної автоматизованої системи підтримки оптимального стану спортсмена необхідно інтегрувати низку фізіологічних параметрів, кожен з яких доповнює загальну картину функціонального стану організму. Комбінація даних про серцеву активність, насичення киснем, м’язову роботу, температуру тіла, рівень активності та психоемоційний стан дозволяє будувати гнучку й адаптивну систему підтримки, що реагує на індивідуальні особливості кожного спортсмена.

Таким чином, на етапі проєктування програмно-апаратного комплексу особливу увагу слід приділити визначенню ключових фізіологічних параметрів, методів їх збору та обробки, а також способів зберігання й візуалізації результатів. Це забезпечить не лише технічну ефективність системи, а й її практичну значущість для користувачів у реальних умовах тренування. У подальших розділах розглядатимуться архітектура такої системи, вибір апаратного та програмного забезпечення, а також результати її тестування.

1.2 Технічні рішення, які використовуються для моніторингу стану спортсмена у спортивній практиці

У сучасному спорті ключову роль у забезпеченні ефективності тренувального процесу відіграє використання технологічних рішень, які дозволяють відстежувати фізіологічний стан спортсмена в режимі реального часу. Застосування таких засобів у тренувальній та змагальній діяльності дає змогу створювати індивідуальні навантаження, контролювати рівень відновлення, вчасно виявляти ознаки перевтоми або функціональних порушень. Впровадження цифрових технологій у спортивну практику є результатом розвитку суміжних галузей, таких як біомедична інженерія, інформаційні технології, телеметрія та мобільні обчислення.

Одним із найбільш поширених засобів для фіксації фізіологічних показників спортсмена є носимі пристрої, зокрема смарт-браслети, розумні годинники та спортивні трекери. До їхніх основних функцій належить вимірювання частоти серцевих скорочень, кількості кроків, витрачених калорій, тривалості сну та рівня насичення крові киснем. Сучасні моделі таких пристроїв, як Garmin Forerunner, Apple Watch, Fitbit Charge, Xiaomi Mi Band, Polar Vantage, оснащуються оптичними сенсорами для зчитування пульсу, акселерометрами, гіроскопами, датчиками температури тіла та GPS-модулями. Завдяки вбудованим алгоритмам пристрої здатні не лише фіксувати дані, але й надавати рекомендації щодо навантаження та відновлення [5, с. 76].

Ще одним технічним елементом моніторингу є спеціалізовані біометричні датчики, які можуть інтегруватися як у носимі пристрої, так і в спортивне спорядження. Наприклад, датчики електрокардіограми (ЕКГ) застосовуються для точного визначення варіабельності серцевого ритму, що є індикатором відновлення та рівня стресу. Сенсори SpO_2 вимірюють насичення крові киснем, а мікросенсори температури тіла дають змогу фіксувати перегрів організму. Електроміографічні (ЕМГ) датчики дозволяють аналізувати напруження окремих м’язових груп, що актуально для оцінки техніки рухів

і симетрії м’язової активності. Інтеграція таких сенсорів із мікроконтролерними модулями відкриває широкі можливості для створення гнучких систем контролю в різних спортивних дисциплінах.

Значну роль у побудові систем моніторингу відіграють апаратні платформи, які виконують функції збору, обробки та передачі даних. Найпопулярнішими є мікроконтролери ESP32, плати Arduino та одноплатні комп’ютери Raspberry Pi. ESP32 має вбудовані модулі Wi-Fi та Bluetooth, що дозволяє створювати бездротові системи зв’язку між датчиками та центральним сервером. Arduino часто використовується для підключення простих аналогових або цифрових сенсорів, а Raspberry Pi забезпечує вищу обчислювальну потужність і може працювати як локальний сервер для зберігання та обробки даних. Завдяки відкритому програмному забезпеченню ці платформи є доступними для розробки прототипів і швидкого розгортання функціональних систем [6 ,с. 76].

Для передачі даних між модулями та центральною обчислювальною системою застосовуються різні протоколи зв’язку. У спортивній практиці найчастіше використовуються технології Bluetooth Low Energy, Wi-Fi та NFC. Bluetooth забезпечує швидкий і енергоефективний обмін даними між сенсорами та мобільними пристроями, що особливо зручно для використання у фітнес-трекерах. Wi-Fi надає можливість передавати дані у хмарні сервіси або локальні сервери, що дає змогу зберігати великі обсяги інформації та здійснювати її обробку на віддалених платформах. Технології NFC або RFID застосовуються для ідентифікації користувачів та активації окремих функцій системи за допомогою електронних міток або карт [40, с .36].

З боку програмного забезпечення для моніторингу стану спортсмена використовуються як серверні, так і клієнтські технології. Серверна частина відповідає за прийом, збереження та попередню обробку даних. Найчастіше для цього використовують фреймворки Java (Spring Boot), Node.js (Express.js) або Python (Django, FastAPI). Вони дозволяють створювати API, до яких підключаються мобільні застосунки, вебінтерфейси або інші модулі.

Дані зберігаються у реляційних базах даних, таких як PostgreSQL чи MySQL, або у нереляційних рішеннях, зокрема MongoDB чи Firebase Realtime Database. Для обміну даними в режимі реального часу застосовуються протоколи MQTT, WebSockets або Kafka.

На боці користувача здійснюється візуалізація зібраної інформації, що дозволяє спортсмену або тренеру оперативно реагувати на зміни у фізіологічному стані. Для розробки вебінтерфейсів застосовують фреймворки React, Angular чи Vue.js. Мобільні застосунки створюють за допомогою Flutter або React Native. У графічних елементах виводяться дані про частоту пульсу, насичення киснем, температуру тіла, пройдену відстань, динаміку тренувань, якість сну тощо. Для візуалізації використовують бібліотеки D3.js, Chart.js або Highcharts, які забезпечують інтерактивність і можливість побудови графіків, гістограм та діаграм [7,с. 68].

FitAI - мобільний застосунок, що пропонує різноманітні варіанти доступу залежно від потреб користувача. Його функціональність охоплює демонстрацію фізичних вправ за допомогою анімованих моделей і графічних ілюстрацій, що робить процес ознайомлення з технікою виконання простішим. Програмний продукт вирізняється цілісним візуальним оформленням, яке уніфікує відображення тренувань та м’язових груп, створюючи комфортне середовище для керування персональним прогресом. Однак часті запити на активацію платного доступу можуть викликати роздратування в користувачів, а вільна версія має досить обмежений набір вправ, що ускладнює повноцінне використання.

Сильні сторони:

1) Високоякісне представлення техніки у вигляді візуалізацій - як статичних, так і динамічних.
2) Єдиний стиль візуалізації для тренувальних блоків і анатомічних груп.
3) Вбудовані шаблони тренувань, що доступні без додаткових налаштувань.
4) Зручне сортування вправ за цільовими групами м’язів.

Обмеження:

1) Надмірна кількість запитів на оформлення передплати.
2) Мінімальна кількість доступних безкоштовних вправ.
3) Неможливість додавати користувацькі вправи.
4) Потреба вводити особисту інформацію при початку використання.
5) Відсутність адаптивних тренувальних програм.
6) Інтерфейс виглядає перевантаженим і дещо складним для початківця.

Strong орієнтований на індивідуалізацію тренувального процесу. Користувачі мають змогу самостійно формувати плани, обираючи потрібні вправи з великого каталогу. Автоматизоване обчислення статистики і збереження даних про кожну активність суттєво полегшує контроль за особистим прогресом. Водночас, фіксована дата тренування та відсутність текстових коментарів до плану створюють певні обмеження у персоналізації [8, с. 43].

Переваги:

1) Розширений вибір фізичних активностей.
2) Гнучкість у побудові власних програм.
3) Історія усіх тренувань доступна для перегляду.
4) Автоматичне підрахування повторів, навантаження, часу.

Недоліки:

1) Встановлена дата тренування не підлягає редагуванню.
2) Неможливо додавати нотатки до створених планів.
3) Базова аналітика обмежена та доступна лише з підпискою.
4) Обмеження на кількість збережених програм тренувань.

GT привертає увагу своєю одноразовою платною моделлю, що відкриває доступ до майже всіх можливостей системи. У застосунку є розгорнута база вправ, кожна з яких супроводжується поясненням та демонстраційним відео. Це вигідно вирізняє програму серед конкурентів. Однак наявність нав’язливої реклами та вимога додаткової підписки на окремі функції може знижувати загальний рівень задоволеності користувачів [9, с. 55].

Позитивні сторони:

1) Вигідна фінансова модель – лише разова оплата.
2) Повна і структурована база вправ з посиланнями на відеоінструкції.
3) Історія активностей зберігається автоматично.
4) Результати тренувань обробляються системою.

Недоліки:

1) Часте появлення рекламних вікон під час використання.
2) Розширена аналітика, включаючи графіки, доступна лише у преміум-пакеті.
3) Створення особистих програм вимагає оформлення платного доступу.

Окрему нішу у спортивних IT-рішеннях займають технології машинного навчання та обробки великих обсягів даних. Завдяки використанню бібліотек TensorFlow або PyTorch можна створювати моделі, що прогнозують втому, оцінюють рівень готовності до навантаження або ідентифікують відхилення від норми. Наприклад, на основі даних про пульс, HRV, сон і попереднє навантаження модель може передбачити, чи готовий спортсмен до інтенсивного тренування. Статистичний аналіз забезпечується за допомогою Pandas, NumPy та SciPy, які дозволяють обчислювати середні значення, дисперсії, кореляції між різними показниками.

Усе частіше системи моніторингу стану спортсменів інтегруються з хмарними платформами, які забезпечують масштабованість, надійність та доступність даних з будь-якої точки світу. Найпопулярніші сервіси - це Google Cloud Platform, Amazon Web Services та Microsoft Azure. Вони надають інструменти для обчислень, збереження даних, а також інструменти аналітики. Крім того, сервіси на зразок Firebase забезпечують просту інтеграцію з мобільними додатками, підтримують функції аутентифікації, push-сповіщень та збереження у реальному часі. Для аналітики у хмарі можна використовувати панелі керування, побудовані за допомогою Grafana або Kibana, що дозволяють відображати комплексну інформацію у зрозумілому вигляді.

Значна увага приділяється безпеці даних, особливо коли йдеться про обробку особистої медичної інформації спортсмена. Для забезпечення автентифікації користувачів застосовуються технології JWT та OAuth 2.0. Передача даних здійснюється через захищені протоколи HTTPS із використанням сертифікатів SSL/TLS. Усі рішення повинні відповідати міжнародним стандартам захисту персональних медичних даних, таким як GDPR або HIPAA. Це особливо актуально у випадках, коли система використовується не лише в аматорському спорті, але й на професійному або медичному рівні [10, с. 59].

Інтеграція таких рішень часто здійснюється на базі концепції Інтернету речей (IoT), яка дозволяє об’єднувати в єдину мережу фізичні пристрої, сенсори та програмні модулі. Для керування IoT-пристроями застосовують платформи Blynk, ThingsBoard або Arduino Cloud. Вони забезпечують зручне з’єднання сенсорів із хмарними системами, дають змогу задавати порогові значення, отримувати сповіщення та вмикати реактивні механізми (наприклад, автоматичне повідомлення тренера при досягненні критичних значень пульсу або температури тіла) [32, с. 57].

У спортивній практиці важливе значення має можливість налаштування системи під конкретного користувача або дисципліну. Деякі види спорту вимагають постійного моніторингу дихання (наприклад, плавання), інші - акценту на локальній м’язовій активності (силові види спорту), ще інші - високої точності вимірювання координат та швидкості руху (легка атлетика, велоспорт, біатлон). Сучасні цифрові рішення дозволяють налаштувати комбінації датчиків і параметрів відповідно до потреб конкретного спортсмена, що забезпечує індивідуальний підхід до моніторингу та корекції тренувального процесу.

Таким чином, сучасна спортивна індустрія вже активно використовує широкий спектр технічних рішень для моніторингу стану спортсмена. Їх ефективність безпосередньо залежить від правильного поєднання апаратної та програмної складових, точності збору даних, надійності комунікацій, можливостей візу-

алізації та зручності взаємодії з користувачем. В наступних розділах буде докладно описано, як саме ці інструменти можуть бути об’єднані в єдину автоматизовану систему для підтримки оптимального стану спортсмена під час тренування.

1.3 Способи зчитування і передавання біометричних даних у цифровому середовищі

У контексті розроблення автоматизованої системи підтримки оптимального стану спортсмена особливе значення має вибір способів зчитування біометричних параметрів і забезпечення надійного каналу їх передачі в цифровому середовищі. Якість отриманих даних, точність їх фіксації, швидкість надсилання та стабільність комунікації між пристроями і серверною частиною системи безпосередньо впливають на ефективність загального функціонування цифрового рішення. З огляду на це, на сучасному етапі розвитку технологій у спортивній сфері застосовуються комплексні підходи, які поєднують апаратні засоби з потужними програмними платформами, здатними забезпечити високий рівень точності та адаптивності.

Біометричні дані спортсмена охоплюють широкий спектр фізіологічних показників, таких як частота серцевих скорочень, насичення крові киснем, температура тіла, електроміографічні сигнали, рівень активності, дихання, рухи та інші характеристики, які свідчать про стан організму в динаміці фізичного навантаження. Для зчитування цих параметрів використовуються різні типи сенсорів, які вбудовуються у смарт-браслети, текстильні елементи, поясні ремені, спортивний одяг або окремі портативні пристрої [11, с. 69].

Найпоширенішими є оптичні сенсори, які працюють за принципом фотоплетизмографії та застосовуються для фіксації пульсу й рівня SpO_2. Вони вбудовані у більшість смарт-годинників і фітнес-браслетів. Ці сенсори випромінюють світло певної довжини хвилі, яке проходить через шкіру, а потім фіксують відбитий сигнал. Залежно від зміни об’єму крові в судинах сигнал змінюється, і це дозволяє визначати ритм серцевих скорочень або рівень насичення киснем. Такі

сенсори є недорогими, енергоефективними та досить точними для спортивного застосування, хоча й мають обмеження при високій інтенсивності рухів.

Електричні сенсори, зокрема ті, що базуються на принципі електрокардіографії (ЕКГ), забезпечують вищу точність у визначенні параметрів серцевої активності. Вони використовуються у нагрудних ременях або медичних портативних пристроях, які синхронізуються із смартфоном або сервером. Такі рішення застосовуються в професійному спорті, де критично важлива точність визначення інтервалів між серцевими скороченнями для аналізу варіабельності ритму (HRV).

Ще одним джерелом біометричних даних є акселерометри та гіроскопи, які дозволяють фіксувати зміну положення тіла, прискорення, частоту рухів та інші характеристики, пов’язані з кінематикою. Вони застосовуються для обліку фізичної активності, виявлення фаз руху (наприклад, кроків, стрибків, змін напрямку), а також для оцінки техніки виконання вправ. У поєднанні з GPS-модулями ці сенсори надають точну інформацію про місцезнаходження спортсмена, швидкість руху, дистанцію та висоту, що особливо важливо в циклічних видах спорту.

Інші сенсори, зокрема температурні, електроміографічні та пульсометричні датчики, розширюють можливості системи за рахунок контролю локальних параметрів. Наприклад, датчики температури можуть бути розміщені в текстильних виробах або безпосередньо на шкірі. ЕМГ-сенсори фіксують м’язову активність і дозволяють оцінити ступінь втоми або асиметрію у м’язовій роботі, що є цінним індикатором для тренерського складу. Дані з усіх цих пристроїв потребують стабільної передачі в систему для подальшого аналізу, тому важливо забезпечити ефективний канал зв’язку [12, с. 78].

Для передачі біометричних даних у цифровому середовищі застосовується кілька основних бездротових технологій. Bluetooth Low Energy (BLE) є найпоширенішим протоколом у сфері фітнесу та персональних носимих пристроїв. Він забезпечує низьке енергоспоживання, достатню швидкість передачі та можли-

вість роботи з багатьма пристроями одночасно. BLE ідеально підходить для синхронізації годинників і браслетів із мобільними телефонами або мікроконтролерами типу ESP32.

Wi-Fi-з'єднання зазвичай використовується у випадках, коли потрібно передавати великі обсяги даних на хмарні сервери або локальні бази даних. Наприклад, пристрої на базі Raspberry Pi можуть під'єднуватися до домашньої або тренувальної мережі й автоматично надсилати дані в систему збору, де вони зберігаються для подальшого аналізу. Такий підхід дозволяє уникати проміжної обробки на стороні користувача та забезпечити централізований контроль даних [12, с. 80].

Для передачі даних між пристроями або вбудованими модулями застосовуються також технології ZigBee, LoRaWAN, NFC та RFID. Вони особливо корисні у командних видах спорту або в ситуаціях, коли потрібно здійснювати ідентифікацію гравців, фіксувати переміщення на полі або взаємодію з тренажерами. Ці технології мають високу стійкість до перешкод, енергоефективність і просту реалізацію.

Передача даних супроводжується попередньою обробкою на рівні прошивки пристрою або мікроконтролера. Наприклад, мікропроцесори ESP32 можуть виконувати згладжування сигналу, відбір середніх значень, виявлення піків або винятків, щоб зменшити навантаження на основний сервер. Потім дані надсилаються за допомогою стандартних протоколів MQTT або HTTP до серверної частини системи, де відбувається подальша обробка [13, с. 89].

Існує низка програмних продуктів та платформ, які активно використовуються для збирання, обробки та візуалізації біометричних даних спортсменів. Наприклад, мобільний застосунок Garmin Connect дозволяє не лише переглядати дані про пульс, активність і сон, але й створювати індивідуальні тренувальні плани. Подібну функціональність мають програми Apple Health, Fitbit App, Polar Flow та Suunto App. Ці застосунки мають доступ до даних із фітнес-пристроїв, зберігають їх у хмарі, синхронізуються з іншими сервісами й надають користувачам докладні графіки й аналітику.

Для професійної спортивної практики використовуються складніші системи на зразок Catapult, STATSports або WHOOP. Вони забезпечують моніторинг на основі комбінації десятків датчиків, включаючи пульс, швидкість, навантаження, температуру та зміну положення тіла. Дані передаються в реальному часі до аналітичних панелей, де тренери можуть приймати рішення щодо заміни гравця, корекції навантаження або запобігання перевтомі [14, с. 57].

З боку програмного забезпечення для обробки даних застосовуються серверні рішення на основі Java, Python та JavaScript. Використання фреймворку Spring Boot дозволяє створити надійний бекенд, що обробляє запити від пристроїв, зберігає дані у базі та забезпечує авторизацію користувачів. Node.js з бібліотеками Express.js надає високу швидкодію та зручний підхід до роботи з API. Python використовується для інтеграції алгоритмів аналізу даних, моделей машинного навчання та візуалізації у вигляді дашбордів.

Бази даних є ключовою складовою програмного середовища. PostgreSQL та MySQL використовуються для збереження структурованої інформації, зокрема історії тренувань, параметрів стану, профілів користувачів. Для зберігання неструктурованих або потокових даних застосовуються MongoDB або Firebase Realtime Database, які забезпечують швидкий доступ і масштабування. Для передачі повідомлень між модулями системи в реальному часі використовуються брокери повідомлень на основі MQTT або Kafka.

У візуальному представленні даних використовуються бібліотеки D3.js, Chart.js або Google Charts. Вони дозволяють будувати графіки, які показують зміну показників у часі, порівняння з нормативами, розподіл навантаження тощо. Для зручності кінцевого користувача дані інтегруються у вебінтерфейс, побудований на React або Vue, або у мобільний додаток, створений за допомогою Flutter або React Native.

Усі перелічені апаратно-програмні компоненти можуть бути інтегровані в єдину автоматизовану систему підтримки фізіологічного стану спортсмена. Правильна комбінація засобів зчитування, способів передачі, програмної обробки та візуалізації забезпечує надійність, точність і оперативність системи,

а також зручність її використання. Розуміння технічних особливостей дозволяє здійснити грамотне проєктування та реалізацію такої системи відповідно до вимог реального спортивного середовища. У подальших розділах буде розкрито архітектуру майбутньої системи, вибір технічних компонентів, програмну реалізацію та результати тестування. Усі дані наведені у таблиці 1.1.

Таблиця 1.1 - Основні додатки системи та їх характеристики

№	Назва додатка	Призначення	Основні функції	Тип програмного забезпечення	Технології	Платформа
1	Garmin Connect	Контроль фізичної активності та пульсу	Графіки, планування тренувань	Мобільний додаток	Android, iOS, Bluetooth	iOS / Android
2	Apple Health	Збір даних з пристроїв Apple	Синхронізація з iPhone	Мобільний додаток	iOS, HealthKit	iOS
3	Fitbit App	Моніторинг сну та навантаження	Поради щодо фітнесу	Мобільний додаток	Android, iOS, Wi-Fi	iOS / Android
4	Polar Flow	Контроль серцево-судинної системи	Трекінг серцевого ритму	Мобільний додаток	Bluetooth, Web-сервіси	iOS / Android
5	Suunto App	Оцінка навантаження	Аналіз показників під час бігу	Мобільний додаток	Android, iOS	iOS / Android

Продовження талиці 1.1.

6	WHOOP	Поглиблений біометричний моніторинг	Контроль сну, навантаження, відновлення	Носимий додаток	Bluetooth, хмарні сервіси	iOS / Android
7	Catapult	Високоточний моніторинг стану спортсмена	GPS, ЧСС, м'язова активність	Професійна система	Власні протоколи, Wi-Fi	Спеціалізована
8	STATSports	Збір даних у командних видах спорту	Тренерський моніторинг	Професійна система	Bluetooth, Wi-Fi	Спеціалізована
9	Blynk	Керування IoT пристроями	Передача даних з сенсорів	IoT-додаток	ESP32, Arduino, Wi-Fi	Кросплатформна
10	ThingsBoard	Візуалізація даних з датчиків	Інтеграція пристроїв, графіки	IoT-платформа	MQTT, HTTP, CoAP	Web / Хмара
11	Firebase	Хмарне зберігання та синхронізація	Аутентифікація, база даних у реальному часі	Хмарна платформа	Firebase SDK, REST API	Web / Мобільні
12	Grafana	Побудова аналітичних панелей	Інтерактивні графіки та статистика	Веб-додаток	Grafana Labs, Prometheus	Web

Продовження таблиці 1.1.

12	Grafana	Побудова аналітичних панелей	Інтерактивні графіки та статистика	Веб-додаток	Grafana Labs, Prometheus	Web
13	React Native App	Інтерфейс користувача для моніторингу стану спортсмена	Збір, відображення показників	Мобільний застосунок	React Native, API	iOS / Android
14	Python API Server	Серверна частина для обробки фізіологічних даних	Зберігання, обробка, передача у базу	Серверний застосунок	Python, FastAPI, PostgreSQL	Linux / Windows

Таблиця відображає чітке розмежування між основними складовими програмного комплексу, які можуть бути використані для моніторингу стану спортсмена у рамках автоматизованої системи. Подані додатки охоплюють як комерційно доступні мобільні рішення для персонального використання, так і професійні системи моніторингу в командному спорті. Також представлено спеціалізовані платформи для візуалізації даних, хмарного зберігання та обробки, а також прикладні програмні модулі, які можуть бути реалізовані безпосередньо під час розробки власної системи [15, с. 57].

Аналіз вмісту таблиці дозволяє зробити висновок, що для реалізації повноцінної цифрової системи підтримки оптимального стану спортсмена доцільно поєднувати інструменти з різних груп: мобільні додатки – для зручної взаємодії з користувачем, IoT- та серверні рішення – для збору та обробки даних, хмарні сервіси – для масштабування та безперервного доступу до інформації, аналітичні панелі – для прийняття рішень на основі візуалізації. Такий комплексний підхід забезпечує функціональну гнучкість, точність і адаптивність системи до змін у тренувальному процесі.

2. ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПІДТРИМКИ ОПТИМАЛЬНОГО СТАНУ СПОРТСМЕНА

2.1 Вибір архітектури та електронних компонентів системи з урахуванням умов експлуатації

Розроблення автоматизованої системи підтримки оптимального стану спортсмена потребує ретельного проєктування апаратно-програмної архітектури з урахуванням специфіки умов експлуатації, зокрема змін температури, вологості, руху, швидкості з'єднання та енергоспоживання. Обрана архітектура має забезпечувати стабільність роботи, оперативне зчитування біометричних параметрів у реальному часі, безпечну передачу даних, швидку обробку та зручне представлення інформації користувачу. В основі такої системи лежить поєднання сенсорного рівня, мікроконтролерного блока, каналу зв’язку, сервера, інтерфейсу користувача й механізмів захисту даних на рисунку 2.1 представлена структура системи моніторингу стану спортсмена.

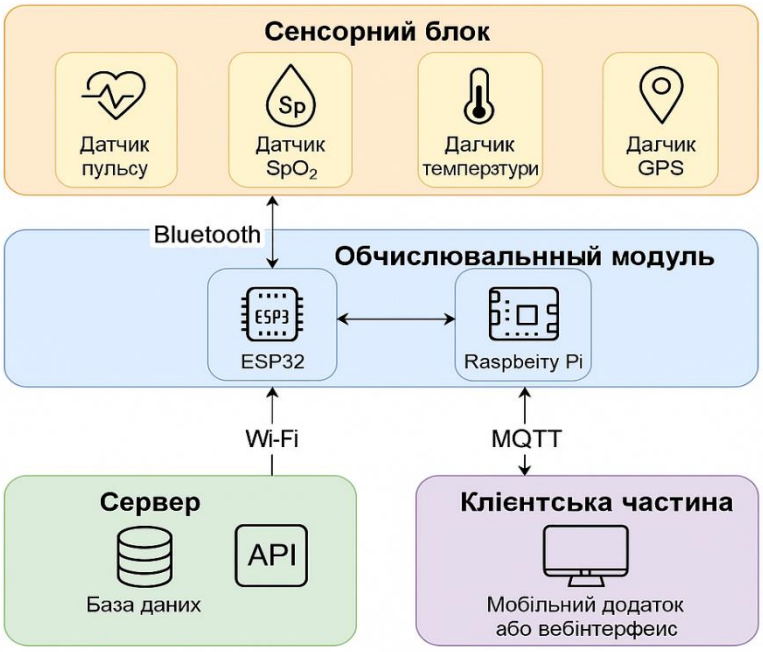


Рисунок 2.1 - Загальна архітектура системи моніторингу фізіологічного стану спортсмена

На рисунку зображено три рівні: сенсорний блок (датчики пульсу, SpO_2, температури, GPS), обчислювальний модуль (ESP32 або Raspberry Pi), сервер (база даних, API), клієнтська частина (мобільний додаток або вебінтерфейс). Передбачено канали зв'язку Bluetooth, Wi-Fi, MQTT [16, с. 79].

На сенсорному рівні основними електронними компонентами виступають датчики, які фіксують біометричні параметри організму спортсмена. Для фіксації частоти серцевих скорочень доцільно використати фотоплетизмографічний сенсор типу MAX30102, який також дозволяє вимірювати рівень кисню в крові. Цей сенсор має компактні розміри, невелику вагу, низьке енергоспоживання та добре працює з контролерами ESP32. Для визначення температури тіла рекомендується застосувати цифровий датчик DS18B20, оскільки він точний і має широкі можливості підключення. Для відстеження рухової активності доцільно використати тривісний акселерометр MPU6050, який також має гіроскопічні можливості для визначення кута нахилу тіла представлено на таблиці 2.1.

Таблиця 2.1 - Основні сенсори та їх характеристики

Назва сенсора	Призначення	Діапазон вимірювання	Інтерфейс	Живлення	Особливості
MAX30102	Пульс, SpO_2	25–240 уд/хв, 70–100% SpO_2	I2C	1.8–3.3 В	Інфрачервоне та червоне світло
DS18B20	Температура тіла	–55°C … +125°C	1-Wire	3–5 В	Висока точність, цифровий вихід
MPU6050	Акселерометр і гіроскоп	±2g … ±16g / ±250 … ±2000°/с	I2C	3.3 В	Об'єднаний сенсор руху
GPS NEO-6M	Геолокація, швидкість	До 1.8 м точність	UART	3–5 В	Швидкий холодний старт

Мікроконтролер, який збирає дані із сенсорів і передає їх на сервер, має відповідати критеріям продуктивності, енергоефективності та можливості бездротової передачі. У цьому контексті оптимальним є використання ESP32 – двоядерного мікроконтролера з вбудованими модулями Wi-Fi та Bluetooth. Його перевагами є підтримка енергозберігаючих режимів, можливість одночасної обробки кількох потоків даних, сумісність із численними сенсорами й широке програмне забезпечення з відкритим кодом. У разі потреби у більшій обчислювальній потужності (наприклад, для обробки відео або локальної візуалізації) можливо використовувати Raspberry Pi Zero 2 W, який працює на базі повноцінної операційної системи [17, с. 68].

Загальний опис мікроконтролерів представлений у таблиці 2.2.

Таблиця 2.2 - Мікроконтролери, що можуть використовуватись у системі

Назва сенсора	Призначення	Діапазон вимірювання	Інтерфейс	Живлення	Особливості
MAX30102	Пульс, SpO_2	25–240 уд/хв, 70–100% SpO_2	I2C	1,8–3,3 В	Інфрачервоне та червоне світло
DS18B20	Температура тіла	–55°C … +125°C	1-Wire	3–5 В	Висока точність, цифровий вихід
MPU6050	Акселерометр і гіроскоп	±2g … ±16g / ±250 … ±2000°/c	I2C	3,3 В	Об’єднаний сенсор руху
GPS NEO-6M	Геолокація, швидкість	До 1,8 м точність	UART	3–5 В	Швидкий холодний старт

Для зв’язку між мікроконтролером і сервером рекомендовано застосовувати протокол MQTT. Він призначений для передавання коротких повідомлень у реальному часі та має мінімальне навантаження на мережу. MQTT дозволяє встановлювати "теми" – канали, по яких дані передаються у вигляді публікацій

та підписок. Це дає змогу ефективно передавати лише актуальні значення, що особливо важливо у системах з низькою пропускною здатністю або високими вимогами до енергоефективності. На рисунку 2.2 представлено як саме передаються повідомелння в реальному часі.

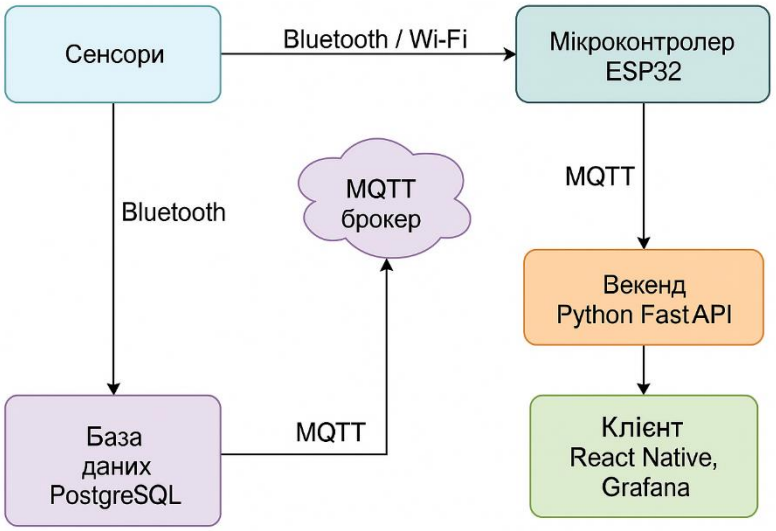


Рисунок 2.2 - Потік даних у системі (від сенсора до візуалізації)

Серверна частина реалізовується на базі Python (FastAPI), що забезпечує обробку REST-запитів і підключення до бази даних PostgreSQL. Саме там зберігаються історичні дані кожного користувача, обчислюються середні значення, фільтруються пікові відхилення й формуються відповіді для клієнтської частини. Для короткочасного зберігання або кешування можуть застосовуватися NoSQL-рішення типу Redis або Firebase Realtime Database [18, с .54].

Інтерфейс користувача реалізовується у вигляді мобільного додатка (React Native) або вебінтерфейсу (React.js). Користувач може бачити на головному екрані поточний пульс, SpO_2, температуру, кількість зроблених кроків, дистанцію, графіки зміни параметрів за день чи тиждень, рекомендації щодо навантаження. У разі виявлення критичних значень система надсилає push-сповіщення або звукове попередження вище описаний функціонал представлений на рисунку 2.3.

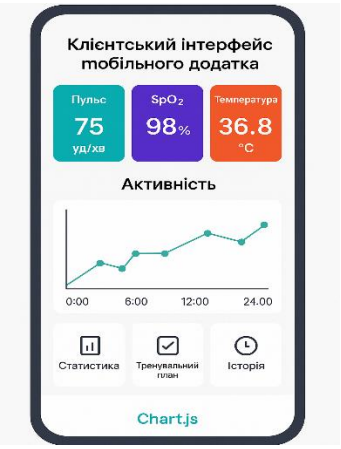


Рисунок 2.3 - Прототип клієнтського інтерфейс мобільного додатка

Система має працювати в умовах фізичної активності, змін навколишнього середовища, обмеженої доступності електроживлення. Тому всі елементи повинні бути легкими, енергоефективними, захищеними від вологи й механічного пошкодження. Доцільно розміщувати основні сенсори на зап'ясті або в спеціальному поясі, тоді як мікроконтролер та акумулятор можуть бути інтегровані у ремінь або наручний тримач. Система повинна функціонувати у температурному діапазоні від –10 до +45°C, мати герметичний корпус (не нижче IP65) та автономну роботу не менше 8 годин.

Безпека даних – критичний компонент архітектури. Для захищеного обміну між модулями застосовуються протоколи SSL/TLS. Авторизація користувача здійснюється через JWT (JSON Web Token). Система має відповідати вимогам GDPR щодо збереження медичних даних [19, с. 57].

Всі вимоги представлені у таблиці 2.3.

Таблиця 2.3 - Умови експлуатації і вимоги до системи

Параметр	Мінімальне значення	Коментар
Температурний діапазон	–10°C до +45°C	Для використання в залах і на вулиці
Ступінь захисту корпусу	IP65	Захист від пилу, водяних бризок

Продовження таблиці 2.3.

Автономність	8 годин	Робота без підзарядки під час тренування
Точність пульсометра	±2 уд/хв	Вимога до сенсора MAX30102
Час реакції системи	до 1 секунди	Обробка критичних змін у реальному часі

Таким чином, вибір архітектури та електронних компонентів системи має базуватися на принципах надійності, енергоефективності, модульності та стійкості до впливу зовнішніх факторів. У запропонованій архітектурі всі елементи інтегруються у гнучку екосистему, яка дозволяє відстежувати фізіологічний стан спортсмена у режимі реального часу, зберігати історію, надавати персоналізовані поради та оперативно реагувати на відхилення. Наступні розділи дипломної роботи будуть присвячені реалізації цієї архітектури в програмному середовищі та перевірці її функціонування у тестовому режимі.

2.2 Опис логіки обміну даними між сенсорами, мікроконтролером і програмним забезпеченням

Логіка обміну даними є критично важливою частиною архітектури будь-якої автоматизованої системи, що здійснює реєстрацію, обробку та передачу фізіологічної інформації в реальному часі. У межах системи підтримки оптимального стану спортсмена вона забезпечує узгоджену роботу всіх функціональних компонентів - сенсорів, мікроконтролера, програмного забезпечення, хмарних сервісів і користувацьких інтерфейсів. Основна мета - забезпечити надійний, безперервний і швидкий обмін даними з мінімальними затримками, високою точністю та захистом від втрат чи помилок. Архітектурно обмін даними побудований за принципом ієрархічного комунікаційного потоку, де нижній рівень відповідає за первинне зчитування даних, середній - за обробку і маршрутизацію, а верхній - за збереження, інтерпретацію та візуалізацію [20, с. 67].

Процес взаємодії між елементами системи починається на рівні сенсорів, які за допомогою вбудованих модулів фіксують біометричні сигнали. Кожен сенсор має свою періодичність вимірювання, інтерфейс зв'язку та формат передачі даних. Наприклад, сенсор MAX30102 зчитує дані про пульс і рівень SpO_2 через інтерфейс I2C із частотою до 100 Гц. Датчик температури DS18B20 передає значення по 1-Wire інтерфейсу з періодом до однієї секунди. Акселерометр MPU6050 фіксує прискорення і кутову швидкість, надсилаючи дані на мікроконтролер у режимі запиту (polling) або переривання (interrupt). Кожен з них підключається до відповідних пінів мікроконтролера ESP32, де всі дані зчитуються в циклічному режимі програмного таймера [21, с. 57].

Процес взаємодії між елементами системи представлений на рисунку 2.4.

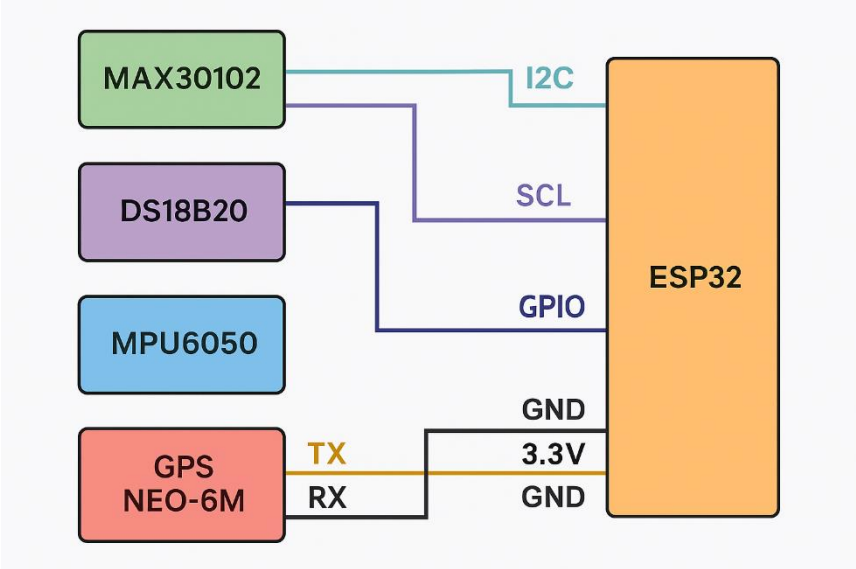


Рисунок 2.4 - Фізичне з'єднання сенсорів з мікроконтролером

На програмному рівні мікроконтролер реалізує функціональну логіку збору, попередньої обробки та буферизації даних. Кожен сенсор обробляється відповідним драйвером, який інкапсулює низькорівневу взаємодію, що значно спрощує логіку головної програми. Дані проходять етапи перевірки, фільтрації, конвертації одиниць вимірювання та нормалізації. Наприклад, сигнал пульсу обробляється методом ковзного середнього, щоб уникнути впливу шуму або артефактів руху. Після цього сформовані пакети даних з міткою часу готуються до передачі у центральну систему, ось всі дані зображенні у таблиці 2.4.

Таблиця 2.4 - Формат передачі даних з мікроконтролера на сервер

Параметр	Тип	Одиниці	Формат JSON	Коментар
pulse	Integer	уд/хв	"pulse": 78	Поточна частота пульсу
spo2	Float	%	"spo2": 96.2	Насичення крові киснем
temp	Float	°C	"temp": 36.7	Температура тіла
accel	Array(Float)	м/с²	"accel": [0.2, 9.8, 0.1]	Тривимірне прискорення
gps	Object	координати	"gps": {"lat": 48.5, "lon": 30.7}	Позиція спортсмена
timestamp	Integer	мілісекунди UNIX	"timestamp": 1716220000000	Час зчитування

Після підготовки даних вони передаються до хмарного або локального брокера MQTT. Протокол MQTT(рис. 2.5) обрано через його легкість, високу швидкодію, підтримку "publish-subscribe" моделі та можливість працювати з обмеженими ресурсами. Мікроконтролер публікує повідомлення у відповідну тему, наприклад, athlete/data/{user_id}. Сервер, що підписаний на цю тему, миттєво отримує повідомлення й передає їх до системи обробки [23, с. 68].

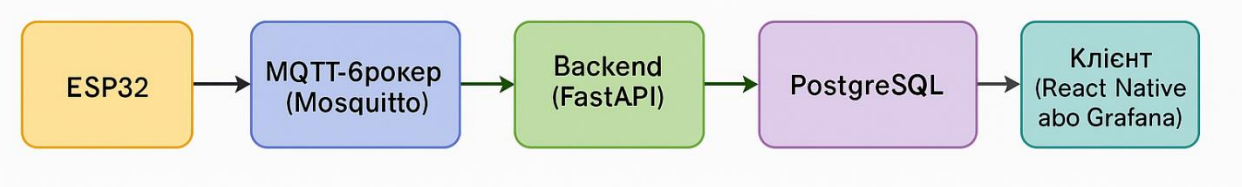


Рисунок 2.5 - Логіка передачі даних через MQTT

На рівні бекенду дані приймаються та валідуються. У випадку успішної обробки записуються в базу даних PostgreSQL, розділену за таблицями: users, measurements, alerts, training_sessions (табл. 2.5).

Структура БД продумана таким чином, щоб забезпечити можливість швидкого фільтрування, агрегації та виведення даних за період, типами показників або користувачем. Кожен запис містить посилання на ідентифікатор спортсмена, тип параметра, значення, мітку часу та мітку джерела [24, с. 46].

Таблиця 2.5 - Структура таблиці "measurements" у базі даних

id	SERIAL	Унікальний ідентифікатор запису
user_id	INTEGER	Ідентифікатор спортсмена
param_type	TEXT	Тип параметра (pulse, spo2, temp тощо)
value	FLOAT	Значення параметра
source	TEXT	Джерело (ESP32, API, вручну)
timestamp	TIMESTAMP	Час отримання даних

Після збереження даних сервер ініціює надсилання сповіщень або оновлення інтерфейсу користувача. Це здійснюється або через WebSocket-з'єднання (якщо клієнт онлайн), або шляхом збереження повідомлення, яке буде відображено при наступному вході. У випадку досягнення критичних значень параметрів активується система сповіщень, яка відправляє push-повідомлення або електронні листи користувачеві чи тренеру.

Користувацький інтерфейс отримує дані з сервера через REST API (рис. 2.6) або WebSocket. Дані подаються у зручному графічному вигляді, що дозволяє спортсмену бачити не лише поточні значення, але й історію змін за обраний період. Графіки створюються з використанням Chart.js або Recharts.

У мобільному додатку також реалізована можливість вручну вводити параметри, редагувати тренувальні сесії, переглядати рекомендації або формувати звіти.

Рисунок 2.6 - Логіка роботи інтерфейсу користувача

Особливу роль у загальній логіці обміну даними відіграє контроль помилок. На кожному рівні системи передбачено механізми виявлення аномалій. Наприклад, якщо значення пульсу вищі за 220 уд/хв, вони позначаються як підозрілі та не зберігаються в базі. Якщо порушено зв’язок із MQTT-брокером, дані тимчасово кешуються на мікроконтролері в оперативній пам’яті. У разі втрати з’єднання з інтернетом система намагається автоматично перепідключитись із певним інтервалом, це все предаставлено у таблиці 2.6.

Таблиця 2.6 - Механізми обробки збоїв

Компонент	Потенційна помилка	Механізм компенсації
Сенсор	Втрата сигналу	Повторний запит, тайм-аут
Мікроконтролер	Втрата зв'язку	Локальне кешування до 100 повідомлень
MQTT-брокер	Нестабільність мережі	Повторна публікація із QoS=1
Сервер	Перевантаження	Обмеження на кількість запитів в секунду
Клієнт	Відсутність з'єднання	Повідомлення про помилку, повторний запит

Таким чином, логіка обміну даними в системі побудована за принципами стабільності, точності, безперервності та безпеки. Зібрані біометричні показники проходять послідовні етапи зчитування, обробки, передачі, зберігання та візуалізації.

Кожен з компонентів взаємодіє з іншими за допомогою стандартних протоколів і зрозумілих форматів даних. Використання відкритих рішень дає змогу масштабувати систему, змінювати типи сенсорів, адаптувати архітектуру до потреб користувачів і застосовувати її як у аматорському, так і в професійному спорті. Наступні розділи будуть присвячені реалізації програмного коду, верифікації працездатності системи та її випробуванню в умовах тренувального процесу.

2.3 Вимоги до системи

Процес розроблення автоматизованої системи підтримки оптимального стану спортсмена під час тренування вимагає формалізованого підходу до визначення та структурування вимог. Саме вимоги є основою для моделювання архітектури, визначення обсягу функціональності, побудови інтерфейсу користувача, логіки обробки даних та вибору технічних рішень.

Розроблені вимоги до системи згруповано у три логічні категорії: бізнес-вимоги, функціональні вимоги та нефункціональні вимоги. Кожен тип вимог представлено окремими діаграмами, де використано зв’язки типу «refine», «derive», «satisfy», «verify», що дозволяє встановити чіткі відношення між базовими потребами, інженерними обмеженнями та компонентами реалізації. Крім того, всі вимоги розподілено по пакетах (діаграми пакетів), що забезпечує структурованість і масштабованість моделі. Для аналізу повноти й узгодженості вимог використано спеціальні діаграми-дашборди, які візуально представляють складність, пріоритетність і статус опрацювання вимог на поточному етапі проєктування [25, с. 65].

У складі бізнес-вимог системи сформульовано цілі, що відображають кінцеву користь для спортсмена, тренера й адміністратора. До них належать: забезпечення безперервного моніторингу фізіологічного стану спортсмена під час тренування, надання зворотного зв’язку у режимі реального часу, створення візуального інтерфейсу для керування сесіями та перегляду історії показників, можливість виявлення критичних станів і запобігання перевантаженням.

Для відображення цих вимог створено діаграму бізнес-вимог, на якій визначено головну вимогу BRQ-001 «Оптимізація тренувального процесу спортсмена» як надсистемну, яка уточнюється через підлеглі вимоги другого рівня: BRQ-002 «Безперервний біометричний моніторинг», BRQ-003 «Реакція на перевищення меж», BRQ-004 «Фіксація історії станів», BRQ-005 «Зручний інтерфейс керування» рисунок 2.7.

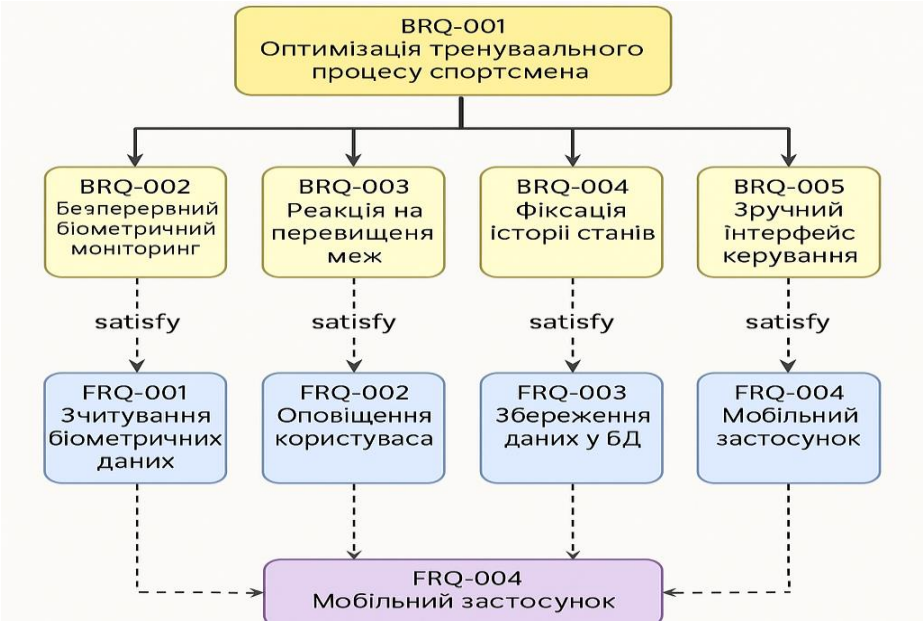


Рисунок 2.7 - Діаграма "Оптимізація тренувального процесу спортсмена»"

Функціональні вимоги охоплюють усі необхідні дії, які має виконувати система. Серед них: зчитування фізіологічних даних з сенсорів у режимі реального часу, передавання даних по бездротовому каналу, обробка, перевірка меж, виведення на екран, збереження у базу, генерація сповіщень, керування сесіями, перегляд історії.

Для опису цих вимог створено діаграму функціональних вимог, де зазначено ключові вимоги типу FRQ-001 «Збір даних з сенсорів», FRQ-002 «Формування повідомлень MQTT», FRQ-003 «Обробка даних на сервері», FRQ-004 «Збереження даних у БД», FRQ-005 «Візуалізація параметрів», FRQ-006 «Налаштування порогових значень», FRQ-007 «Формування звітів». Додатково реалізовано зв’язки між кожною вимогою та елементами системи, які її реалізують (мікроконтролер, сервер, клієнтська частина). Усі функціональні вимоги (рис 2.8) мають статус «Реалізовано» та рівень складності «Середній» [26, с. 88].

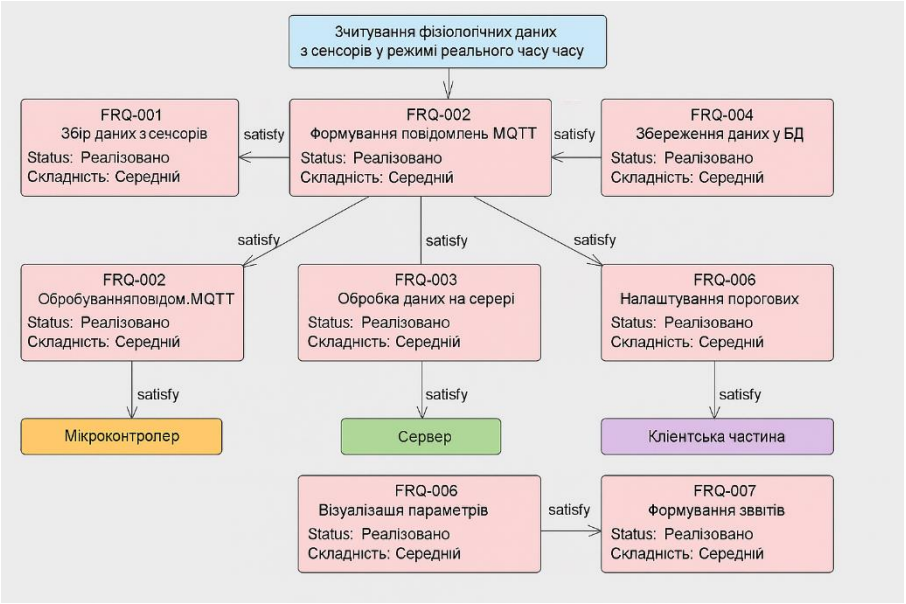


Рисунок 2.8 - Діаграма функціональних вимог

Нефункціональні вимоги описують обмеження, які стосуються якості функціонування, продуктивності, безпеки та масштабованості. До них належать вимоги типу NFR-001 «Час реакції системи не більше 2 сек», NFR-002 «Доступність сервісу не менше 98%», NFR-003 «Захист персональних даних користувача», NFR-004 «Автономна робота пристрою не менше 6 годин», NFR-005 «Сумісність з Android 9+», NFR-006 «Підтримка 10+ користувачів одночасно». Ці вимоги було внесено до діаграми нефункціональних вимог з прив'язкою до відповідних функціональних модулів. Зв'язки типу «verify» використовуються для показу перевірки відповідності вимог до компонентів через тест-кейси.

Усі функціональні та нефункціональні вимоги, що були сформульовані в рамках проєкту, мають офіційний статус **«Затверджено»**, що свідчить про їх остаточне погодження всіма зацікавленими сторонами — включно з аналітиками, розробниками, замовником і технічним експертом. Такий статус означає, о ці вимоги внесені до основної специфікації, узгоджені з технічним завданням а є обов'язковими до реалізації у фінальній версії програмного продукту. Кожній із вимог надано пріоритетність **«Висока»**, що вказує на їх критичну важливість для основної функціональності системи. Це означає, що реалізація саме цих положень є необхідною для повноцінного виконання ключових сценаріїв користування, і вони мають бути виконані у першу чергу під час розробки.

Для зручності навігації у моделі застосовано діаграму пакетів на рисунку 2.9 у якій розміщено окремі пакети: «Business Requirements», «Functional Requirements», «Non-Functional Requirements», «Use Cases», «Components», «Test Cases», «User Interface Elements». Це забезпечує структуровану організацію вимог і дозволяє легко знаходити їх в системі

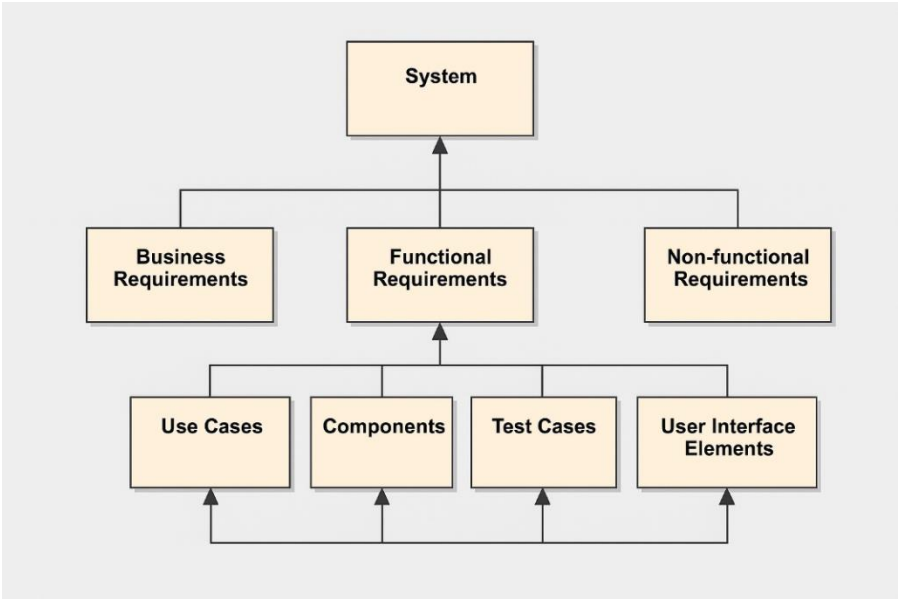


Рисунок 2.9 - Діаграма пакетів

Зв’язки між вимогами та іншими елементами (прецедентами, тестами, інтерфейсами) побудовані відповідно до принципів трасування. Наприклад, функціональна вимога FRQ-001 пов’язана з UC-001 «Запуск тренування», TC-001 «Зчитування пульсу», і UI-001 «Кнопка «Старт»». Ось модем глянути на рисунок 2.10.

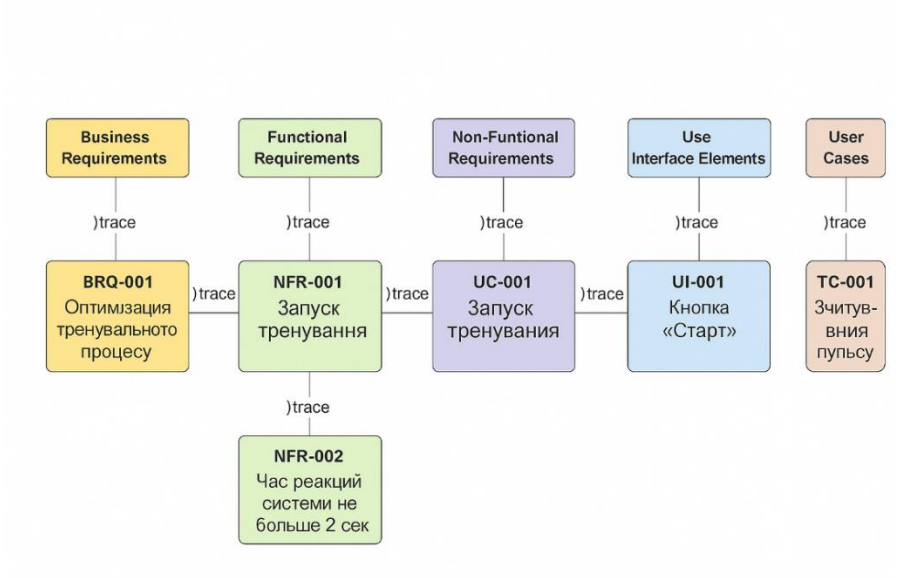


Рисунок 2.10 - Діаграма трасування

Окрема діаграма трасування демонструє взаємозв’язок між бізнес-цілями, функціональними й нефункціональними вимогами, варіантами використання, елементами інтерфейсу та тестовими кейсами. Для складніших сценаріїв побудовано кілька локалізованих діаграм: «Трасування біометричних даних», «Трасування сповіщень», «Трасування історії тренувань», кожна з яких дозволяє сфокусовано оцінити повноту реалізації вимог у певному функціональному сегменті [27, с. 47].

У текстовій частині розділу вимоги також подано у вигляді специфікації, підготовленої засобами Specification Manager. Усі вимоги мають унікальні ідентифікатори, стислі та повні назви, тип, статус реалізації, складність, пріоритетність, рівень ризику, а також пояснення за потреби. Наприклад:

1) Ідентифікатор: FRQ-002.
2) Стисла назва: Формування повідомлення.
3) Повна назва: Система має формувати JSON-повідомлення з біометричними даними.
4) Тип: Функціональна.
5) Статус: Реалізовано.
6) Складність: Середня.
7) Пріоритетність: Висока.
8) Ризикованість: Середня.
9) Примітки: Повідомлення має містити ID, HR, SpO_2, температуру, timestamp.

Таким чином, усі розроблені вимоги до автоматизованої системи підтримки оптимального стану спортсмена відповідають принципам якості: вони однозначні, вимірювані, реалістичні, не дублюються, є атомарними та контрольованими. Побудована ієрархія дозволяє забезпечити логічну зв’язаність від цілей до реалізації, а використані діаграми обґрунтувати архітектурні рішення та забезпечити основу для трасування й тестування системи. Весь набір вимог у моделі є узгодженим, завершеним і готовим для подальшої розробки, масштабування та супроводу в контексті повноцінного інженерного проєкту.

2.4 Проектування системи та розробка бази даних

Проєктування поведінки автоматизованої системи підтримки оптимального стану спортсмена потребує побудови повної та формалізованої моделі, що відображає основні процеси, сценарії використання, послідовність взаємодії компонентів і внутрішню структуру реалізації.

Діаграма прецедентів системи на рисунку 2.11, демонструє основні функціональні можливості та сценарії взаємодії з акторами. В системі визначено три основних актора: «Спортсмен», «Тренер» та «Адміністратор».

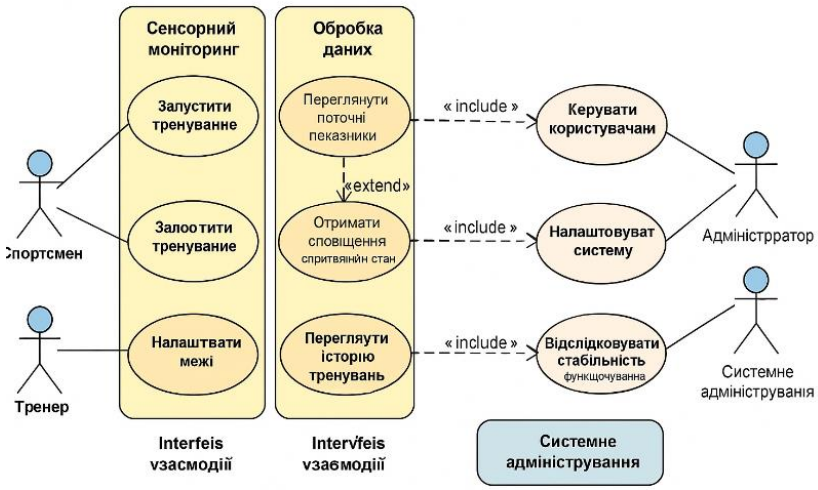


Рисунок 2.11 - Діаграма прецедентів

Кожен з них взаємодіє з окремим набором функцій. Для актора «Спортсмен» передбачено прецеденти: «Запустити тренування», «Переглянути поточні показники», «Отримати сповіщення про критичний стан», «Налаштувати межі», «Переглянути історію тренувань». Актор «Тренер» має доступ до таких варіантів використання: «Спостерігати за тренуванням спортсмена», «Аналізувати дані декількох сесій», «Формувати рекомендації». Актор «Адміністратор» може «Керувати користувачами», «Налаштовувати систему», «Відслідковувати стабільність функціонування». Усі прецеденти згруповано за функціональними підсистемами у пакети: «Сенсорний моніторинг», «Обробка даних», «Інтерфейс взаємодії»,

«Системне адміністрування». Зв’язки «include» та «extend» демонструють повторне використання функцій та альтернативні дії у складніших сценаріях [28, с. 76].

Для кожного прецеденту сформовано текстові сценарії з використанням шаблону сценаріїв. Наприклад, сценарій до варіанту використання «Запустити тренування» включає попередні умови (користувач авторизований, система підключена до пристрою), основний потік (натискання кнопки «Старт», ініціалізація збору даних, підтвердження запуску) та альтернативний потік (виведення помилки при втраті зв’язку або відсутності сенсора). Кожен крок сценарію пов’язано з відповідним компонентом: кнопка «Старт» – елемент інтерфейсу, функція запуску – частина бекенду, сенсорна ініціалізація – мікроконтролер. Таким чином, забезпечено прозорість сценарію та його трасування до архітектурних блоків. Сценарій «Отримати сповіщення про критичний стан» описує момент фіксації граничного значення пульсу, передачу даних на сервер, запуск функції перевірки, вивід push-повідомлення та зміни кольору індикатора на екрані. Альтернативні сценарії включають ситуацію втрати підключення до Інтернету та активацію буферизації даних. Для кожного з описаних сценаріїв створено діаграми послідовності , які відображають часову логіку взаємодії між об’єктами. Наприклад, діаграма до сценарію «Переглянути поточні показники» містить об'єкти: «Користувач», «Мобільний застосунок», «Сервер», «База даних», «MQTT-брокер», «ESP32», «Сенсор пульсу». Послідовність взаємодій включає запит про стан, передавання даних по MQTT, обробку на сервері, повернення значення у застосунок та вивід на екран. Інша діаграма, що відповідає сценарію «Сповіщення при перевищенні меж», фіксує надходження критичного значення SpO_2, активацію функції контролю, надсилання push-повідомлення, запис у журнал подій та відображення зміненого стану індикатора. Прницип роботи системи представлено на рисунку 2.12.

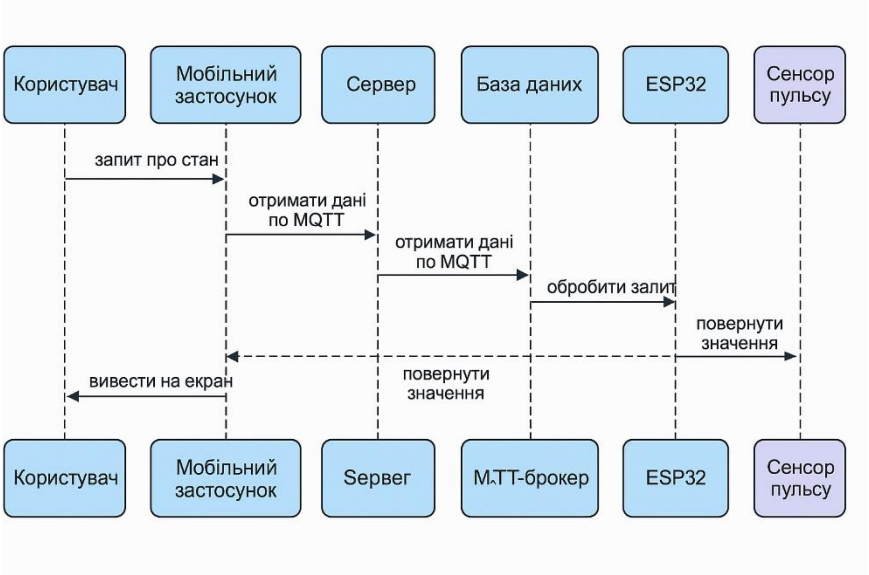


Рисунок 2.12 - Діаграма послідовності

З огляду на наявність паралельних процесів у системі, зокрема одночасного збору даних, передачі, обробки та візуалізації, використано діаграми діяльності. Описано поведінку при запуску тренування, збиранні показників, зміні меж, фіксації критичних подій, завершенні сесії. Для прикладу, діаграма діяльності «Повний цикл тренування» включає блоки: «Авторизація користувача», «Запуск сесії», «Моніторинг сенсорів», «Візуалізація», «Контроль меж», «Сповіщення», «Завершення сесії», «Збереження даних». Діаграма показує розгалуження при виявленні перевантаження, паралельність процесів обробки даних та виводу інформації, логіку переходів між фазами [29, с .64].

Модель структури системи оформлена за допомогою діаграми визначення блоків. У центрі діаграми розміщено блок «Система підтримки стану спортсмена», який має підлеглі блоки: «Сенсорний модуль» (пульс, SpO_2, температура), «Мікроконтролер ESP32», «Комунікаційний модуль (Wi-Fi, Bluetooth)», «Серверна частина», «База даних», «Мобільний застосунок», «Модуль сповіщень». Кожен блок має тип (апаратне, програмне забезпечення), атрибути (частота збору, обсяг пам’яті, протоколи), інтерфейси (порти з'єднання) та залежності. Додатково для кожного блоку зазначено його функціональне призначення та взаємодії.

Для уточнення взаємозв’язків між модулями створено діаграми внутрішніх блоків зображену на рисунку 2.13.

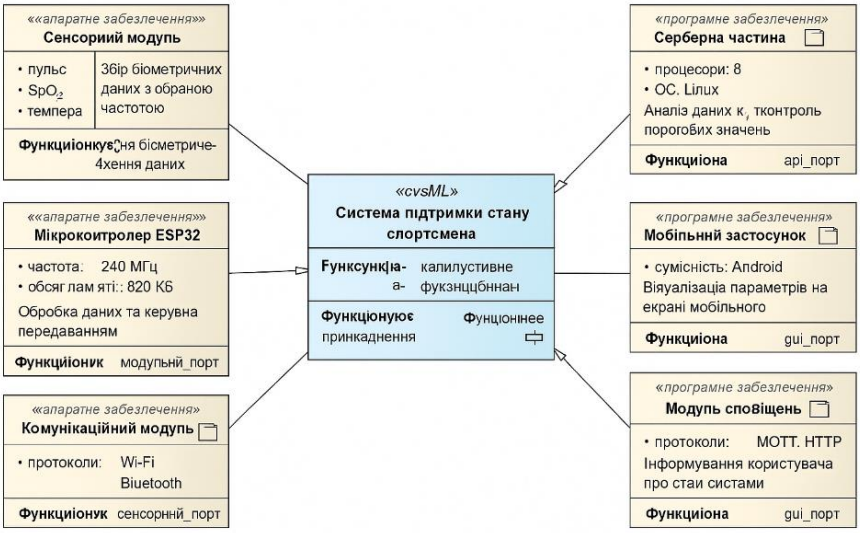


Рисунок 2.13 - Діаграма внутрішніх блоків

Зокрема, у діаграмі «Інтерфейс між ESP32 та сервером» представлено порти зчитування, буферизації, передавання та підтвердження отримання. Діаграма «Зв'язки між мобільним застосунком і сервером» показує взаємодію через REST API, WebSocket та канал push-повідомлень. Потоки даних позначені відповідно до типу: «біометричні дані», «стан інтерфейсу», «команди керування», «повідомлення користувачу» [30, с .46].

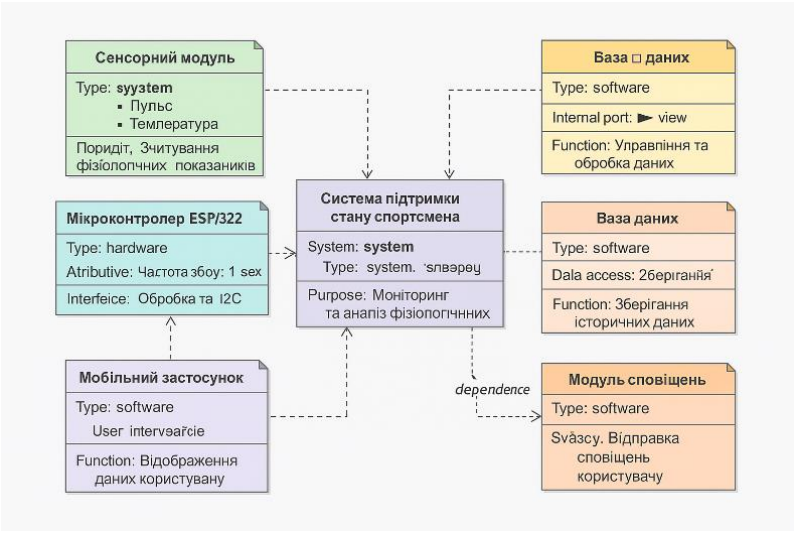


Рисунок 2.13 - Діаграма внутрішніх блоків

Для представлення структури даних, які використовуються в системі, застосовано діаграми класів UML. Побудовано модель із трьох основних груп класів: сутності, граничні класи, класи керування.

Серед класів-сутностей: User, Session, BiometricRecord, Alert. Клас User має атрибути: id, name, age, gender, role. Клас Session включає дату, тривалість, статус, зв'язок з користувачем. Клас BiometricRecord містить timestamp, HR, SpO_2, temperature, sessionId. Визначено зв'язки: один користувач має багато сесій, одна сесія має багато записів. Граничні класи включають MainScreen, AlertPopup, SettingsPanel, HistoryView. Класи керування: SessionController, AlertManager, DataProcessor [41, с. 76].

Додатково створено діаграму класів (рис. 2.14-2.16) із атрибутами та методами. Наприклад, клас SessionController має методи startSession(), stopSession(), saveSession(). Клас DataProcessor має метод validateData(), checkLimits(), generateAlert(). Така деталізація дозволяє відобразити логіку взаємодії між класами, описати потоки даних і методи обробки.

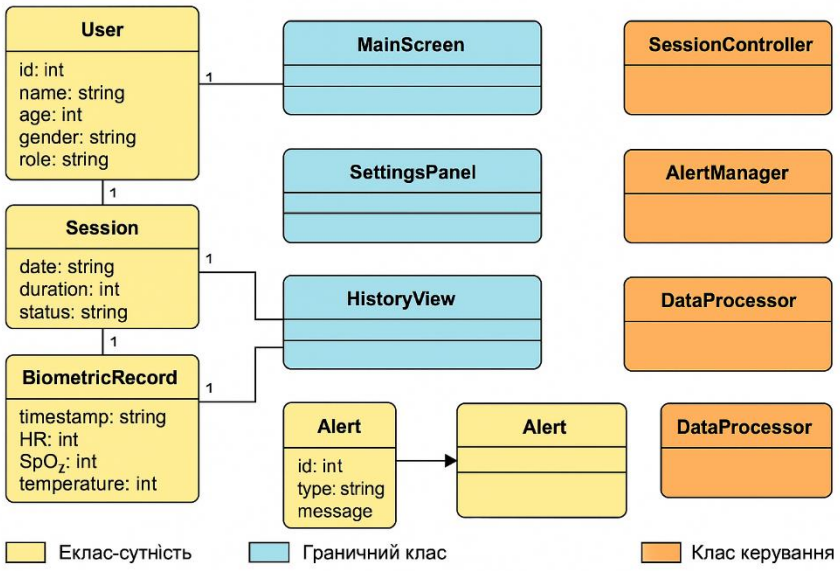


Рисунок 2.14 - Діаграма класів

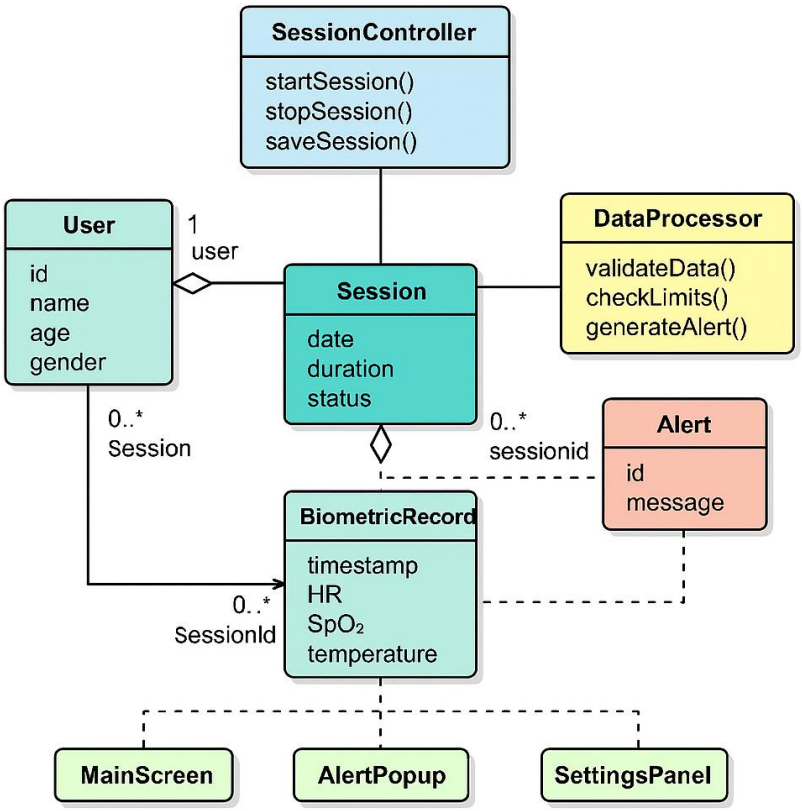


Рисунок 2.15 - Діаграма класів із атрибутами та методами

На окремій діаграмі представлено пакети класів: auth, ui, data, logic, hardware, test. Між пакетами встановлено зв'язки залежностей, що дозволяє оцінити структуру програмного коду системи та його логічне розділення.

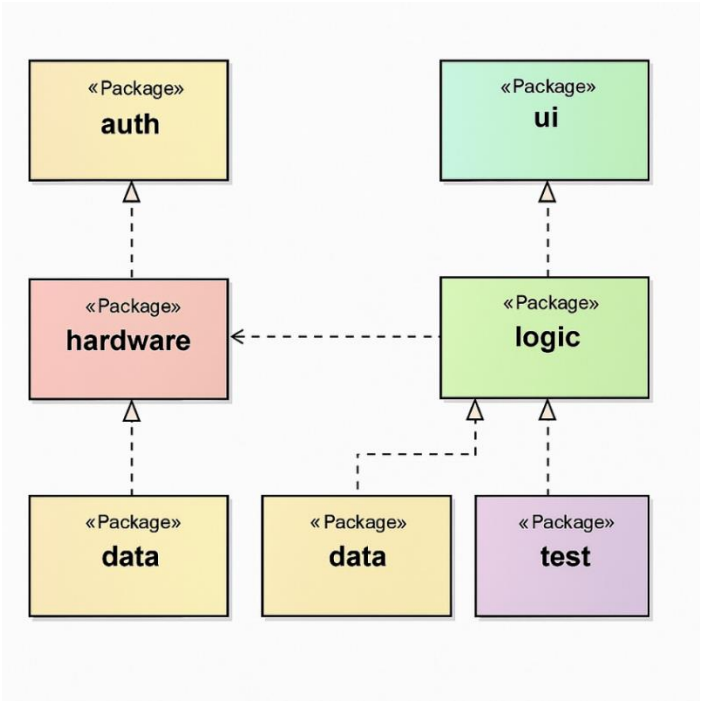


Рисунок 2.16 - Діаграма класів із атрибутами та методами

Таким чином, моделювання поведінки системи охоплює всі рівні від концептуальних сценаріїв до реалізації взаємодій між класами. Діаграми прецедентів, сценарії, послідовності, діяльності, блоків, внутрішніх з’єднань і класів створюють цілісне уявлення про структуру та поведінку системи. Усі елементи моделі узгоджено з вимогами, описаними у попередньому розділі, а зв’язки між діаграмами забезпечують повну трасованість і контроль за реалізацією функціональних цілей. Побудована модель є основою для підтримки, масштабування, перевірки та подальшого вдосконалення автоматизованої системи, що створюється в межах даної курсової роботи [31, с. 67].

Проєктування та реалізація бази даних є критичним етапом створення автоматизованої системи підтримки оптимального стану спортсмена, оскільки саме вона забезпечує збереження, впорядкування, швидкий доступ та захищене керування біометричними показниками, профілями користувачів, історією тренувань і параметрами налаштувань. Ураховуючи специфіку предметної області, база даних повинна бути масштабованою, реляційною, ефективною для роботи в режимі реального часу й забезпечувати надійне збереження як структурованих, так і часових даних.

Для реалізації було обрано PostgreSQL як основну систему керування базами даних. Вибір обґрунтовано її стабільністю, підтримкою транзакцій, високою швидкістю обробки запитів, підтримкою розширень для роботи з JSON, часовими рядами, автоматичним індексуванням, засобами резервного копіювання та наявністю спільноти з відкритим кодом [32, с. 86].

Архітектура бази даних побудована відповідно до норм третьої нормальної форми (3NF), що забезпечує відсутність надлишкових даних та підвищує узгодженість інформації. Основні логічні об'єкти бази поділено на шість категорій: дані користувача, інформація про тренувальні сесії, біометричні показники, сповіщення, налаштування, лог дій. Кожна категорія реалізована окремою таблицею, а всі таблиці зв'язані між собою ключовими атрибутами, що дозволяє забезпечити логічну цілісність. Діаграма сутність-зв'язок (ERD-діаграма) відображає структуру зв'язків між основними таблицями бази: таблиця users містить поля

user_id, name, email, password_hash, role, created_at; таблиця sessions містить session_id, user_id, start_time, end_time, session_type; таблиця biometric_data містить id, session_id, timestamp, heart_rate, spo2, temperature, activity_level; таблиця alerts зберігає сповіщення про критичні стани, а таблиця settings - порогові значення для параметрів.

Діаграма класів до моделі бази даних демонструє об'єкти як класи із зазначенням полів, типів, обмежень і методів доступу зображена на рисунку 2.17.

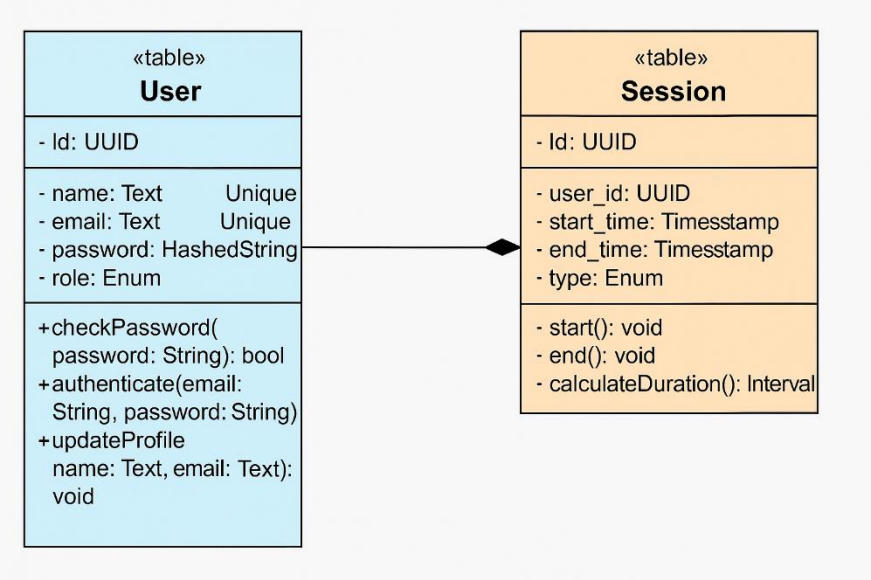


Рисунок 2.17 - Діаграма класів до моделі бази даних

Наприклад, клас User має атрибути id:UUID, name:Text, email:Text, password:HashedString, роль:Enum. Клас Session включає id, user_id, start_time, end_time, тип:Enum. Методи класу User реалізують перевірку пароля, автентифікацію, оновлення профілю. Методи класу Session дозволяють розпочати або завершити тренування, підрахувати тривалість. Таке моделювання забезпечує чітке уявлення про логіку доступу до даних та дії з ними [33, с. 64].

Логічна модель бази даних представлена (рис. 2.18) у вигляді діаграми нормалізації, де показано поділ складної структури даних на окремі таблиці: відокремлено ідентифікаційні дані користувача, конфігурації та вимірювані дані. Такий підхід дозволив зменшити надмірність, покращити індексацію і підвищити швидкість вибірки даних. Вибір типів полів ґрунтувався на оптимізації запитів:

усі часові мітки задані у форматі timestamp з таймзоною, ідентифікатори – UUID, числові показники – smallint або float залежно від типу вимірювання.

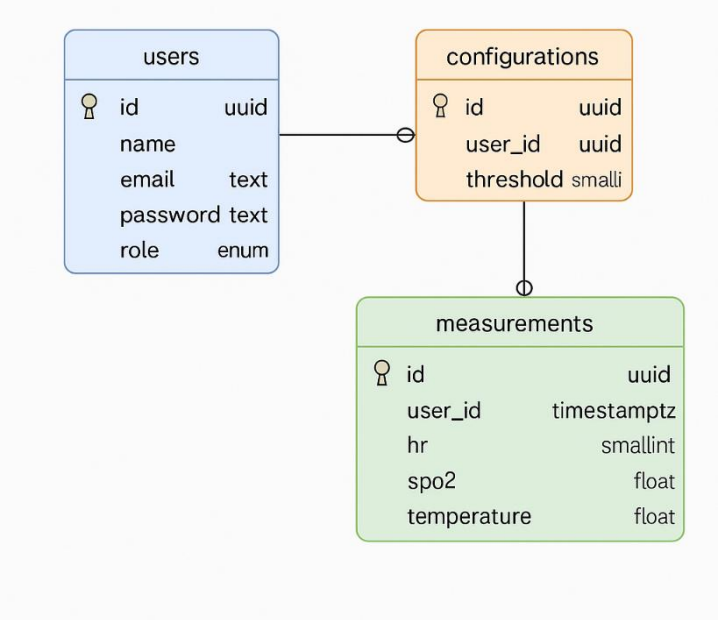


Рисунок 2.18 - Логічна модель бази даних

Фізична модель реалізована в PostgreSQL через використання типів даних, зовнішніх ключів, індексів і тригерів. Основні таблиці мають первинні ключі (PRIMARY KEY), а також зовнішні зв'язки (FOREIGN KEY), що забезпечує узгодженість при збереженні даних. Наприклад, видалення користувача автоматично викликає каскадне видалення всіх його сесій і записів.

У процесі оптимізації продуктивності запитів до бази даних особливу увагу було приділено створенню індексів на найбільш критичних для вибірки та фільтрації полях. Зокрема, індекси були створені на стовпцях user_id, timestamp та session_id. Ці поля є ключовими при виконанні агрегованих запитів, що охоплюють великі обсяги даних, зокрема при здійсненні групування, сортування чи фільтрації записів за користувачами, часовими рамками або сеансами роботи. Завдяки впровадженню відповідних індексів значно зменшується час виконання таких запитів, що у свою чергу сприяє загальному підвищенню продуктивності інформаційної системи [34, с. 76].

Крім того, в структурі таблиці biometric_data було реалізовано поділ даних за часовою ознакою із застосуванням можливостей, які надає механізм розді-

лення таблиць (partitioning) у системі керування базами даних PostgreSQL. Зокрема, впроваджено денне партиціонування, згідно з яким кожен окремий день реєстраційних подій зберігається в окремому підрозділі таблиці. Такий підхід забезпечує ефективне логічне структурування даних у базі, а також дозволяє уникати надмірного зростання основної таблиці, що потенційно може призвести до зниження продуктивності при здійсненні запитів.

Завдяки партиціонуванню значно полегшується виконання операцій читання та обробки інформації за конкретні часові періоди, оскільки запити спрямовуються безпосередньо до відповідної партиції, минаючи необхідність сканування всієї таблиці. Така архітектура не лише покращує швидкодію запитів, а й дозволяє більш ефективно реалізовувати процедури архівування, очищення, а також резервного копіювання даних. У підсумку, використання індексів у поєднанні з часовим партиціонуванням у таблиці biometric_data дозволяє суттєво підвищити масштабованість та надійність системи, особливо в умовах інтенсивної роботи з великими масивами біометричної інформації. Принцип роботи представлений на рисунку 2.19.

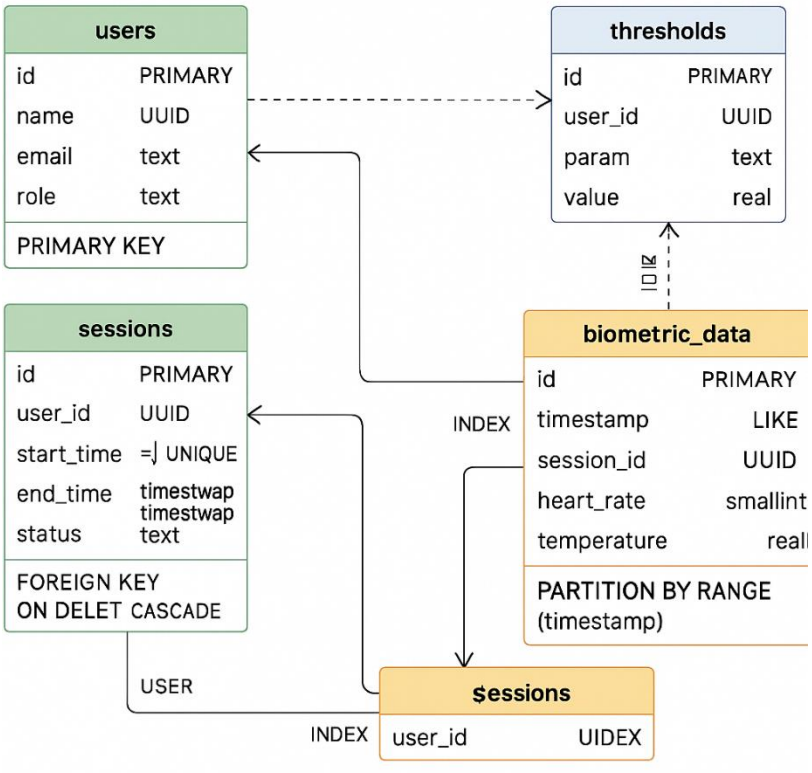


Рисунок 2.19 - Фізична модель бази даних

Діаграма потоків даних (рис. 2.20) показує джерела надходження інформації, напрямки руху, обробку та зберігання: дані надходять з сенсорів на ESP32, передаються на MQTT-брокер, далі на сервер через API, звідти верифікуються та зберігаються в таблиці biometric_data. Паралельно, якщо значення перевищують встановлені межі, в таблицю alerts створюється запис, і сповіщення надсилається користувачу через push або email. Запити на перегляд тренувань або показників надходять з мобільного додатка через REST API, який здійснює звернення до таблиць sessions та biometric_data із фільтрацією по id та часу .

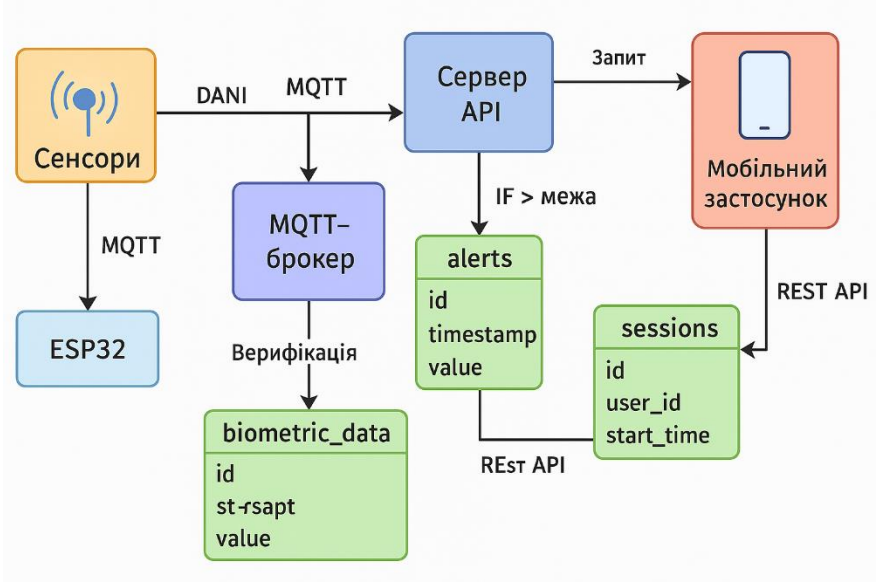


Рисунок 2.20 - Діаграма потоків даних

Для забезпечення збереження цілісності даних реалізовано перевірки на рівні бази даних (constraints). Наприклад, показник SpO_2 не може бути вищим за 100 або нижчим за 50, температура – у межах 30–45 °C, частота пульсу – у межах 30–230 уд/хв. Такі обмеження накладено через CHECK-умови у відповідних полях. Додатково реалізовано тригер на логування змін у налаштуваннях користувача – при зміні порогів створюється запис у таблиці logs.

Резервне копіювання реалізується через щоденний автоматичний дамп бази у форматі .sql.gz з подальшим збереженням у хмарне сховище. Також передбачено механізм реплікації – у реальному часі копія бази синхронізується на резервному сервері, що дозволяє забезпечити безперервність роботи в разі збою.

План масштабування системи базується на горизонтальному розділенні таблиць (sharding) за user_id при великій кількості користувачів. Для цього можлива інтеграція PostgreSQL з зовнішніми засобами, такими як Citus або TimescaleDB, які дозволяють обробляти великі обсяги даних із часовими мітками [43 ,с. 67].

Всі бази даних та їх атрибути представлені на таблицях 2.3-2.8, та рисунок 2.21.

Таблиця 2.3 - База даних системи та її атрибути

№	Назва таблиці	Ключові поля	Типи даних	Призначення таблиці	Зв'язки з іншими таблицями
1	users	user_id, name, email	UUID, TEXT	Зберігання профілю користувача	sessions, settings, logs
2	sessions	session_id, user_id	UUID, TIMESTAMP	Сесії тренувань	biometric_data, alerts
3	biometric_data	session_id, timestamp	UUID, FLOAT, TIMESTAMP	Біометричні показники спортсмена	sessions
4	settings	user_id, hr_limit_id	UUID, SMALLINT	Індивідуальні пороги сповіщень	users
5	alerts	session_id, type_id	UUID, TEXT, TIMESTAMP	Збереження подій критичних станів	sessions
6	logs	user_id, action_id	UUID, TEXT, TIMESTAMP	Аудит дій користувача	users
7	roles	role_id, name_id	UUID, TEXT	Права доступу	users

Продовження таблиці 2.3.

8	devices	device_id, user_id	UUID, TEXT	Прив'язка пристроїв до користувачів	users
9	messages	msg_id, user_id, content_id	UUID, TEXT, TIMESTAMP	Повідомлення користувачам	users
10	recovery_codes	user_id, code_id	UUID, TEXT	Скидання пароля	users
11	terms_acceptance	user_id, timestamp_id	UUID, TIMESTAMP	Фіксація згоди з політикою конфіденційності	users
12	system_status	status, updated_at	INT, TEXT, TIMESTAMP	Моніторинг працездатності системи	-

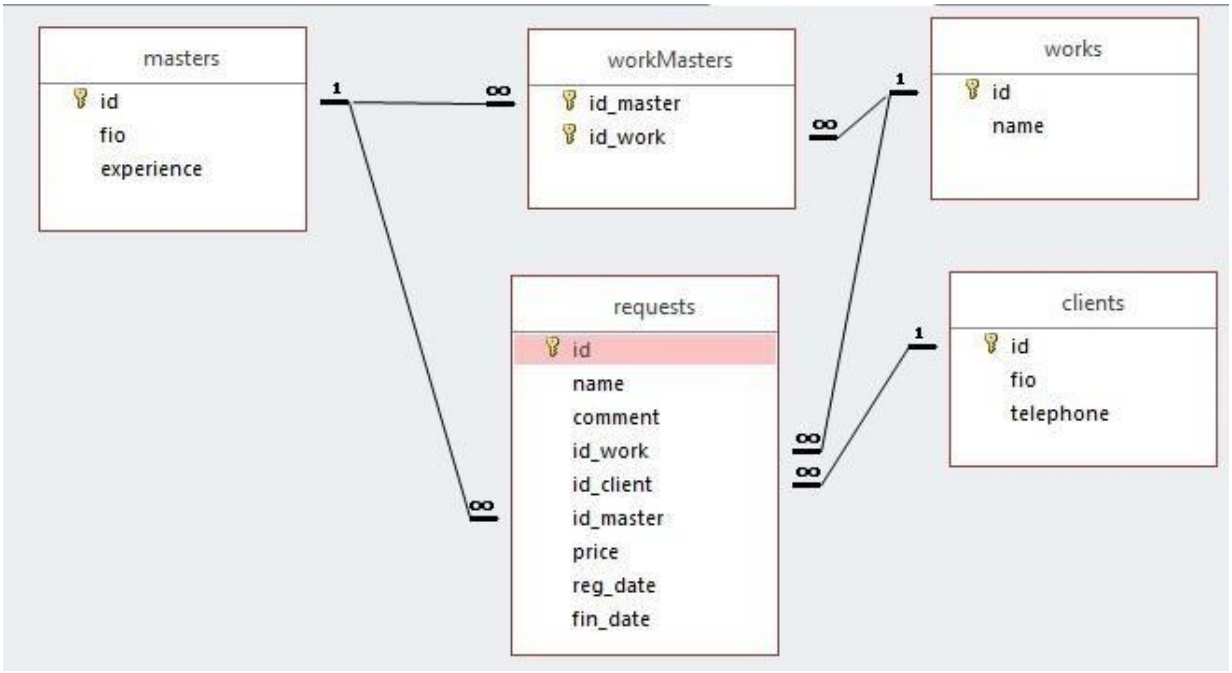


Рисунок 2.21 - Структура бази даних

Таблиця 2.4 - Опис таблиці «Тренери»

Ключ	Поле	Опис поля
ID тренери	Прізвище	Прізвище тренера
	Ім'я	Ім’я тренера
	Телефон	Номер телефону тренера
	Досвід	Досвід роботи тренера
	Дата працевлаштування	Дата коли тренер зайняв свою посаду працівника

Таблиця 2.5 - Опис таблиці «Історія тренувань»

Ключ	Поле	Опис поля
ID історій тренувань	Клієнт	Історія тренувань клієнта
	Тренер	Історія тренувань тренера
	Дата тренувань	Історія дати тренувань
	Вправи	Історія вправ
	Кількість повторень	Історія кількостей повторень вправ

Таблиця 2.6 - Опис таблиці «Вправи»

Ключ	Поле	Опис поля
ID вправи	Ім’я	Ім’я клієнта який тренується

Таблиця 2.7 - Опис таблиці «Раціон»

ID раціон	Поле	Опис поля
	Клієнт	Раціон клієнта
	Початок прийому їжі	Перший прийом їжі клієнта (сніданок)

Продовження таблиці 2.7.

	Останній прийом їжі	Останній прийом їжі клієнта (вечеря)
	Вага клієнта	Вага клієнта

Таблиця 2.8 - Опис таблиці «Клієнт»

ID	Поле	Опис поля
	Ім’я	Ім’я клієнта
	Прізвище	Прізвище клієнта
	Номер телефону	Номер телефону клієнту

Отже, база даних автоматизованої системи є логічно спроєктованою, фізично реалізованою та масштабованою для потреб довготривалого використання в умовах змінного навантаження та збільшення кількості користувачів. Вона забезпечує централізоване зберігання біометричної інформації, гнучке налаштування порогів, аудит дій, захист персональних даних, резервування, відновлення та можливість інтеграції з зовнішніми аналітичними модулями. Побудована модель відповідає практичним і технічним вимогам, визначеним у попередніх розділах курсової роботи.

2.5 Принципи формування візуального середовища для користувача та забезпечення інтуїтивної навігації

Формування візуального середовища для користувача є одним із найважливіших етапів у процесі розроблення автоматизованої системи підтримки оптимального стану спортсмена. Саме графічний інтерфейс забезпечує ефективну взаємодію між користувачем та системою, дозволяючи оперативно отримувати дані, інтерпретувати їх, формувати висновки та ухвалювати рішення. Врахову-

ючи специфіку спортивного середовища, інтерфейс повинен бути мінімалістичним, інформативним, адаптивним до мобільних пристроїв, а також максимально інтуїтивно зрозумілим навіть для непідготовлених користувачів. Особлива увага приділяється підбору технологій, які використовуються під час реалізації інтерфейсних рішень, способам відображення даних, кольоровим акцентам, структурі екранів та навігації між ними [35, с. 60].

Для побудови клієнтської частини системи було обрано кросплатформний фреймворк React Native, який дозволяє створювати нативні мобільні додатки одночасно для платформ Android та iOS, використовуючи одну кодову базу на JavaScript. Це рішення забезпечує зменшення витрат часу на розробку, зручну інтеграцію з серверним API та багату екосистему бібліотек для побудови інтерфейсів. Для відображення графіків і аналітичних віджетів обрано бібліотеку Victory Native, яка надає широкі можливості візуалізації, включно з анімованими графіками, гістограмами, круговими діаграмами та зонами ризику.

Головний екран додатку виконує функцію інформаційної панелі (dashboard) і містить основні біометричні показники спортсмена: пульс, SpO_2, температура тіла, фізична активність представлено на рисунку 2.22 та таблиця 2.9.

Рисунок 2.22 - Структура екранів мобільного додатку

Кожен показник представлено у вигляді окремої панелі, яка відображає поточне значення, динамічний графік за останню годину, колірну індикацію стану (нормальний, прикордонний, критичний). Для зменшення візуального навантаження застосовуються великі шрифти, чіткі іконки та кольори з високим контрастом [36, с. 54].

Таблиця 2.9 - Основні компоненти головного екрана

Компонент	Тип елементу	Функція	Колірна схема	Оновлення даних
Пульс	Картка з графіком	Відображення ЧСС у реальному часі	Зелений / жовтий / червоний	Кожні 10 сек
SpO_2	Круговий індикатор	Відсоток насичення крові киснем	Синій / фіолетовий	Кожні 15 сек
Температура	Цифровий індикатор	Поточна температура тіла	Білий / помаранчевий	Кожні 30 сек
Активність	Лінійна шкала	Кількість кроків або хвилин активності	Блакитний	Щохвилини
Повідомлення	Іконка із лічильником	Попередження про вихід за межі норми	Червоний / жовтий	У момент надходження

Окремий екран деталізації показників дозволяє користувачу побачити повну історію вимірювань, переглянути добові, тижневі або місячні графіки (рис. 2.23). Діаграми реалізовані з використанням Chart.js у випадку вебінтерфейсу або

Victory Line у мобільній версії. Передбачено фільтрацію за датами, типами параметрів, рівнем відхилення. Користувач може переглянути середнє значення, мінімальне, максимальне, а також зміну тренду [37, с. 80].

Рисунок 2.23 - Приклад відображення історичних даних у вигляді графіка

Для кращої орієнтації в інтерфейсі реалізовано нижнє меню навігації (табл. 2.10) з іконками та підписами. Меню фіксується внизу екрану та дає змогу за один дотик перейти на головний екран, переглянути історію, змінити налаштування або ознайомитися з новими повідомленнями [38, с. 55].

Застосовується бібліотека React Navigation, яка забезпечує підтримку вкладок, стеків і контекстної навігації. При розробці особлива увага приділяється відповідності принципам Material Design, що забезпечує єдність логіки взаємодії користувача з елементами інтерфейсу.

Таблиця 2.10 - Вкладки нижньої навігації та їх призначення

Вкладка	Іконка	Назва кнопки	Призначення
Головна	Серце	"Показники"	Поточні значення біометричних параметрів
Графіки	Лінійний графік	"Статистика"	Візуалізація історії
Сповіщення	Дзвінок	"Повідомлення"	Сигнали про критичні значення

Продовження таблиці 2.10.

Профіль	Людина	"Профіль"	Зміна особистих налаштувань
Меню	Три лінії	"Ще"	Додаткові функції

Візуальне середовище доповнюється механізмами попередження про відхилення параметрів від норми. Наприклад, при досягненні пульсу понад 180 уд/хв або SpO_2 нижче 92%, система автоматично формує push-повідомлення, яке з'являється на головному екрані та дублюється в панелі повідомлень. При натисканні на повідомлення користувач переходить до екрану з поясненням та порадою (наприклад: "Рекомендується припинити тренування. Відновіть дихання. Зверніться до тренера"). Принцип роботи повідомлень зображений на риснуку 2.24.

Рисунок 2.24 - Логіка генерації критичних повідомлень

Принцип інтуїтивної навігації також реалізовано у вигляді адаптивної структури. Наприклад, якщо користувач упродовж певного часу використовує лише функції перегляду графіків, додаток пропонує винести цю функцію на головний екран. Така персоналізація можлива завдяки локальному збереженню переваг користувача у пам'яті пристрою за допомогою AsyncStorage [39, с. 95].

У версії вебінтерфейсу застосовуються технології React.js у поєднанні з Tailwind CSS та Recharts. Це дозволяє створювати легкі, швидко завантажувані вебсторінки, що добре адаптуються до різних розмірів екранів. Панелі з даними

представлені у вигляді плиток (cards), які автоматично перебудовуються при зміні орієнтації або роздільної здатності екрана. Для темної теми використовується бібліотека Darkmode.js, що важливо при нічних тренуваннях або поганому освітленні.

Використання сучасних інструментів, таких як React Native, Victory, Tailwind CSS і Firebase, дозволяє реалізувати інтерфейс, який поєднує зручність користування, гнучкість і технологічну ефективність. Інфоормація представлена у таблиці 2.11.

Таблиця 2.11 - Технічні засоби побудови UI в мобільній і вебверсії

Компонент UI	Мобільна версія	Вебверсія
Фреймворк	React Native	React.js
Бібліотека графіків	Victory Native	Recharts / Chart.js
Менеджер стану	Redux / Context API	Redux Toolkit
Сховище даних	AsyncStorage	LocalStorage / IndexedDB
Стилізація	Styled Components	Tailwind CSS
Push-повідомлення	Firebase Cloud Messaging	Web Push API

Окрім основних функцій, інтерфейс також реалізує доступ до налаштувань користувача, де можна змінити допустимі межі параметрів, обрати тип графіків (лінійний або гістограма), підключити зовнішні датчики через Bluetooth, синхронізуватися з іншими платформами (Apple Health, Google Fit) або завантажити історію у форматі CSV [40, с .37].

Таким чином, формування візуального середовища в автоматизованій системі моніторингу стану спортсмена базується на принципах адаптивності, інтуїтивної навігації, мінімалізму та високої продуктивності.

2.6 Підходи до підвищення стабільності функціонування та захисту даних під час тренувального процесу

Стабільне функціонування автоматизованої системи підтримки оптимального стану спортсмена під час тренувального процесу є основною вимогою до її експлуатаційної надійності. Система має витримувати тривалу роботу в режимі реального часу, забезпечуючи точний обмін даними між модулями та безпечне збереження біометричної інформації. У зв’язку з цим особливу увагу необхідно приділяти саме програмним підходам до забезпечення стабільності та захисту даних, які зменшують вплив нестабільного інтернет-з'єднання, переривань у роботі пристроїв та потенційних кібератак. Правильна організація архітектури ПЗ, впровадження політик відновлення після збоїв, реалізація безпечної автентифікації та шифрування даних дозволяють гарантувати безперервність сервісу та збереження конфіденційної інформації [41, с. 65].

Перший і ключовий рівень стабільності – це внутрішня архітектура програмного забезпечення, зокрема серверної частини. Сервер, побудований на базі фреймворку FastAPI (Python), має бути організований за принципом мікросервісної архітектури. Кожен компонент (обробка даних, зберігання, авторизація, сповіщення) виділений в окремий модуль. Це дозволяє масштабувати систему, оновлювати окремі частини без переривання загальної роботи, а також застосовувати балансувальники навантаження.

На рівні транспортного протоколу обрано MQTT з QoS=1, що гарантує доставку кожного повідомлення хоча б один раз. У разі втрати з’єднання з мережею сенсорний блок або мікроконтролер кешують повідомлення у черзі, а при відновленні з’єднання відправляють їх автоматично (рис.2.25). Така поведінка знижує ризик втрати даних, що особливо важливо під час інтенсивного тренувального навантаження, коли інформація про пульс, SpO_2 та інші параметри змінюється швидко.

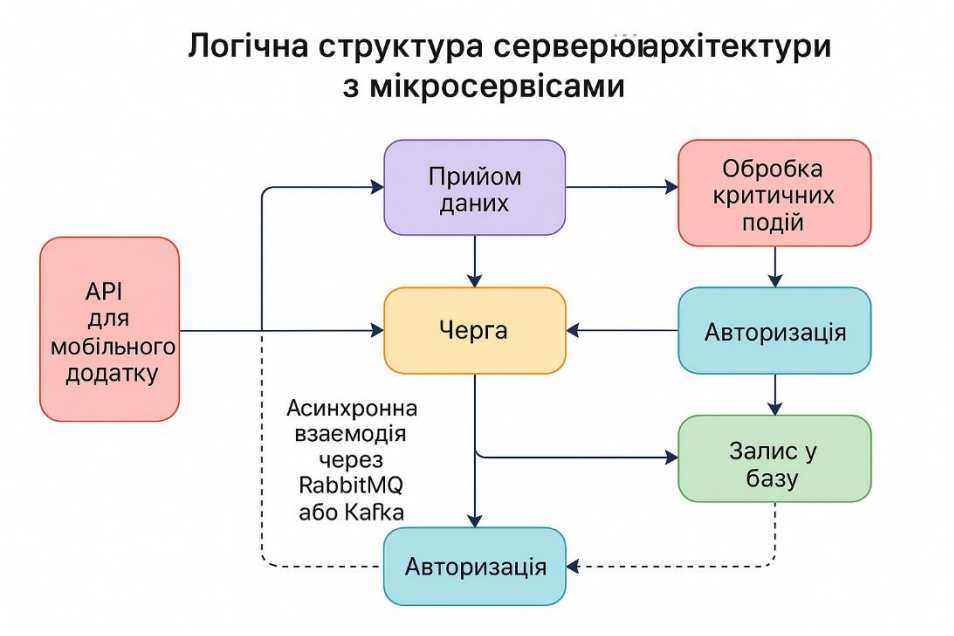


Рисунок 2.25 - Логічна структура серверної архітектури з мікросервісами

Для виявлення та усунення помилок система повинна мати модулі самодіагностики. У мобільному додатку React Native використовується підсистема логування на базі бібліотеки react-native-logs. Серверна частина оснащена централізованим логгером, який записує помилки, попередження та індикатори продуктивності у сховище типу Elasticsearch з подальшим виведенням через Kibana або Grafana [42, с. 76].

Таблиця 2.12 - Основні системи логування та моніторингу

Компонент	Засіб логування	Формат логів	Інструмент візуалізації
Мобільний додаток	react-native-logs	JSON	Локальний log-viewer
Сервер API	Python logging / Loguru	JSON + метадані	Grafana / Kibana
База даних PostgreSQL	Вбудовані логи запитів	SQL журнал	pgAdmin + Grafana
MQTT-брокер	Mosquitto log	Текстовий лог	Systemd journal

Для підвищення стійкості до відмов у системі впроваджується механізм резервного копіювання бази даних. Кожні 24 години сервер автоматично зберігає дамп PostgreSQL на хмарне сховище Google Cloud Storage або AWS S3. У разі збоїв адміністратор може виконати повне відновлення системи за останнім збереженим станом. Крім того, частина даних (наприклад, критичні повідомлення) дублюється в окрему таблицю або сторонню базу для швидкого доступу в екстрених ситуаціях [43, с. 60].

Щодо захисту даних, особливу увагу приділено протоколам безпечної авторизації. Для ідентифікації користувачів застосовується система JSON Web Tokens (JWT), яка дозволяє безпечно передавати маркери сесій у кожному запиті. Кожен токен має обмежений термін дії, після чого автоматично анулюється. Крім того, реалізовано механізм “refresh token”, що дозволяє поновити сесію без повторного входу в систему.

Передача всіх запитів від клієнта до сервера відбувається виключно через HTTPS з використанням SSL-сертифікатів (Let's Encrypt або Comodo).

З боку мобільного додатку дані користувача зберігаються лише тимчасово у SecureStore (на iOS) або EncryptedSharedPreferences (на Android), що гарантує шифрування і захист від стороннього доступу. Усі біометричні показники зберігаються лише на сервері, не дублюються локально і не кешуються у відкритому вигляді [44, с. 66].

Для боротьби з підробкою даних або несанкціонованим втручанням передбачено перевірку цифрового підпису повідомлень. Кожне повідомлення, що надсилається від мікроконтролера на сервер, має унікальний підпис, сформований за допомогою секретного ключа, який відомий лише авторизованим пристроям. У разі зміни даних на рівні мережі сервер відхиляє повідомлення. Такий підхід дозволяє мінімізувати ризик фальсифікації інформації про стан спортсмена. Захист даних представлений у таблиці 2.13.

Таблиця 2.13 - Методи захисту даних у системі

Тип загрози	Засіб захисту	Механізм реалізації
Несанкціонований доступ	JWT + HTTPS	Токен + SSL-сертифікати
Підміна даних під час передачі	Цифровий підпис повідомлень	HMAC-SHA256
Злом мобільного додатку	Зберігання у зашифрованому сховищі	SecureStore / EncryptedSharedPreferences
Витік логінів і паролів	Хешування паролів з сіллю (bcrypt)	Хеші з випадковими солями
Збій бази даних	Резервне копіювання	Автоматичні дампи PostgreSQL
Втрата інтернету	Кешування даних на ESP32	Обмежене сховище з FIFO-чергою

У системі також реалізовано функції контролю сесій і активності користувача. Сервер зберігає історію входів, зокрема IP-адресу, час, тривалість сесії. У випадку входу з нового пристрою користувач отримує сповіщення з можливістю підтвердження або блокування сесії. Це дозволяє уникнути випадків компрометації акаунту.

З метою мінімізації ризиків атак типу “відмова в обслуговуванні” (DoS) сервер налаштовано на обмеження кількості запитів з одного IP-адресу (rate limiting). Якщо клієнт надсилає понад 100 запитів на хвилину, з’єднання тимчасово блокується. Це захищає сервер від перевантаження та недобросовісних користувачів [45, с .43].

Особливу увагу приділено дотриманню стандартів обробки персональних медичних даних. Зокрема, система відповідає основним вимогам GDPR (Європейський регламент захисту даних) та принципам HIPAA (американський закон

про конфіденційність медичної інформації). Кожен користувач має доступ до власних даних, може видалити свій обліковий запис і переглянути історію запитів до своєї інформації. Для цього реалізовано окремий модуль у профілі користувача (рис. 2.27).

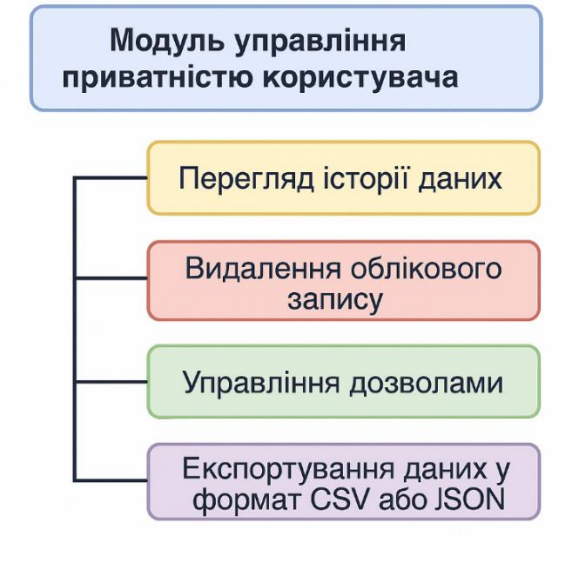


Рисунок 2.27 - Модуль управління приватністю користувача

Вся архітектура системи відповідає принципам GDPR (General Data Protection Regulation) – загального регламенту Європейського Союзу про захист персональних даних, а також HIPAA (Health Insurance Portability and Accountability Act) – американського стандарту для захисту медичної інформації. Серед основних принципів, яких дотримано: збирання лише необхідних даних, обмеження строку зберігання, забезпечення права на видалення, інформування користувача про обробку, шифрування даних, журналювання дій. У мобільному застосунку користувач має можливість видалити свій обліковий запис, переглянути історію активностей, переглянути умови обробки даних. Для цього створено окремий екран «Конфіденційність і безпека». Діаграма «Політика конфіденційності системи» ілюструє взаємозв’язки між користувачем, сервером, базою даних, регламентами, повідомленнями про згоду.

Для виявлення потенційних вразливостей у коді системи застосовано статичний аналіз безпеки з використанням бібліотеки Bandit (для Python). Аналіз проведено для основних модулів, зокрема обробки вхідних запитів, збереження даних, формування відповідей API. Результати тестування не виявили критичних

проблем. Було зроблено ряд рекомендацій, зокрема обмеження довжини запитів, перевірка коду авторизації на довільні символи, захист від SQL-ін'єкцій через ORM. Усі рекомендації впроваджено до фінальної версії коду. Додатково система пройшла тестування на стійкість до спроб несанкціонованого доступу: підбору пароля, повторного використання токена, передачі токена з іншого пристрою [46, с. 79].

У структурі журналювання реалізовано повний аудит усіх дій користувача, зокрема вхід у систему, спроби зміни налаштувань, перегляд тренувань, запуск або завершення сесій. Дані журналу мають мітку часу, тип дії, user_id та ip-адресу. Журнали недоступні для редагування або видалення з боку користувача. Вони шифруються та зберігаються окремо від основної бази. Це дозволяє виявляти аномальні дії, проводити аудит у разі інциденту, а також генерувати статистичні звіти. Діаграма «Аудит активностей користувачів» на рисунку 2.28, показує структуру записів журналу, логіку їх створення та умови доступу до них для різних ролей.

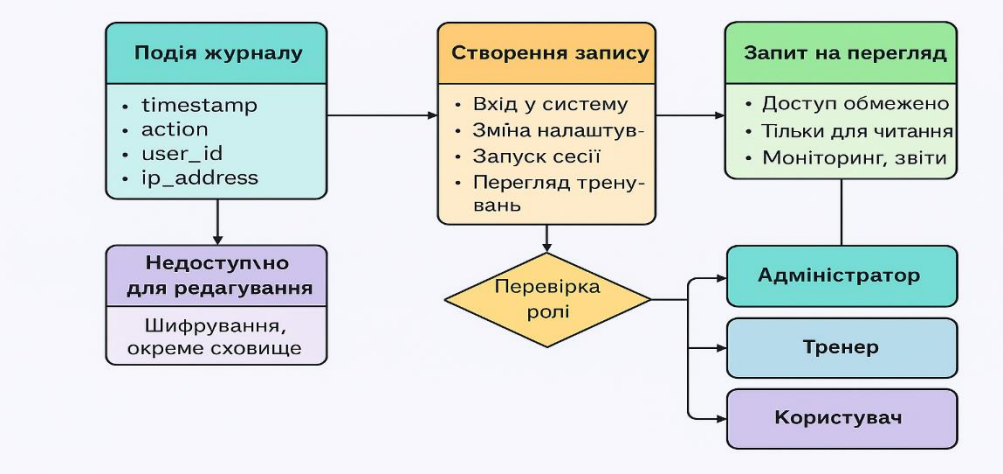


Рисунок 2.28 - Аудит активності користувачів

У випадку атаки на систему або підозрілої активності автоматично активується захисний механізм.

Наприклад, при п'яти помилкових спробах входу поспіль обліковий запис тимчасово блокується. Якщо спроби продовжуються з різних IP-адрес, викликається функція «alert admin» з записом інциденту.

Усі системні повідомлення фіксуються в логах сервера, а адміністратор отримує email-сповіщення.

Діаграма«Обробка інцидентів безпеки» на рисунку 2.29 показує сценарій реагування на спробу злому з автоматичним обмеженням доступу.

Рисунок 2.29 - Діаграма "Обробка інцидентів безпеки"

Таким чином, розроблена система повністю відповідає сучасним вимогам до безпеки інформаційних технологій. Застосовано багаторівневий захист: автентифікація через JWT, шифрування даних через TLS, контроль доступу до бази, аудит дій, а також дотримання міжнародних стандартів захисту персональних медичних даних. Кожен аспект безпеки змодельовано у вигляді діаграм, що дозволяє візуалізувати політику безпеки, спростити аудит та забезпечити можливість масштабування й вдосконалення. Інтеграція описаних засобів гарантує стабільну роботу системи в умовах ризиків і робить її придатною для впровадження у спортивній, медичній та освітній практиці.

3. РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ В УМОВАХ ТРЕНУВАЛЬНОГО СЕРЕДОВИЩА

3.1 Створення програмного модуля для зчитування, обробки та збереження біометричних показників

Програмний модуль для зчитування, обробки та збереження біометричних показників є центральною складовою всієї системи підтримки оптимального стану спортсмена. Він забезпечує збір інформації з фізіологічних сенсорів, її попередню фільтрацію, нормалізацію, перетворення у придатний формат і подальше передавання у центральну базу даних для обробки й візуалізації. Розробка такого модуля потребує врахування специфіки апаратної платформи, типу сенсорів, формату переданих даних, а також вибору ефективних мов програмування, бібліотек і середовищ розробки. Основна мета модуля - забезпечити безперебійну, стабільну та захищену роботу під час тренувального процесу в режимі реального часу [47, с. 69].

Розробка починається з програмного забезпечення для мікроконтролера, до якого підключаються датчики. У нашій системі для цього обрано ESP32 як основну апаратну платформу, що підтримує бездротові інтерфейси Wi-Fi і Bluetooth, має достатню кількість GPIO-виходів для підключення різних сенсорів і сумісна з Arduino IDE та ESP-IDF. Програмний код для зчитування даних реалізовано мовою Pyton з використанням бібліотек Adafruit Sensor, OneWire, DallasTemperature, Wire та власних драйверів для MAX30102 і MPU6050. Кожен сенсор ініціалізується у функції setup(), а в циклі loop() здійснюється регулярне зчитування даних із використанням таймерів.

Після отримання значень, наприклад, частоти пульсу, температури, SpO_2 та прискорення, дані проходять початкову обробку. Для фільтрації використано згладжування ковзним середнім, а для виявлення аномалій - базову логіку перевірки граничних значень.

Результат об’єднується у JSON-об’єкт, який включає значення кожного параметра, мітку часу UNIX, ідентифікатор пристрою, підпис повідомлення та статус автентифікації.

Передавання даних на сервер відбувається за допомогою MQTT-протоколу, через бібліотеку PubSubClient. Дані публікуються у відповідну тему брокера типу "sports/data/user001", де кожен користувач має свій ID. Якщо з’єднання з мережею відсутнє, дані тимчасово зберігаються у буфері на пристрої до моменту відновлення зв’язку. Усі критичні відмови фіксуються у локальний лог-файл для подальшого аналізу.

На стороні бекенду програмний модуль прийому повідомлень побудований на Python з використанням FastAPI. Сервер підписується на MQTT-теми через клієнт paho-mqtt, приймає повідомлення, розбирає JSON і записує дані у базу PostgreSQL за допомогою ORM SQLAlchemy. Перед записом реалізовано валідацію типів, перевірку меж значень, автоматичне конвертування одиниць вимірювання (наприклад, з °F у °C, bpm у Hz, тощо), а також перевірку підпису повідомлення, сформованого на мікроконтролері.

Для захисту даних від несанкціонованого доступу бекенд використовує JWT-токени, що додаються до кожного запиту. Кожен запис у базу містить ID користувача, параметр, значення, мітку часу, джерело (наприклад, "esp32") та рівень достовірності (позначається автоматично при обробці). Система зберігає не лише поточні значення, а й історію змін, що дозволяє будувати довготривалі графіки та здійснювати ретроспективний аналіз [48, с. 59].

Для представлення результатів і підтвердження коректності програмної логіки розроблено серію інтерактивних прототипів у Figma, що відображають ключові екрани взаємодії з модулем. На першому макеті (рис. 3.1) показано екран ручної діагностики сенсорів: користувач бачить активність кожного датчика у вигляді індикатора, може протестувати зчитування або перезапустити сенсор у разі помилки.

Рисунок 3.1 - Макет екрану стану сенсорів у мобільному додатку

Другий прототип (рис. 3.2) показує екран поточного зчитування біометричних даних, де кожне значення виводиться окремо з таймером оновлення. Додано графічний елемент - анімовану хвилю для пульсу та динамічну зміну кольору індикатора при досягненні граничних значень.

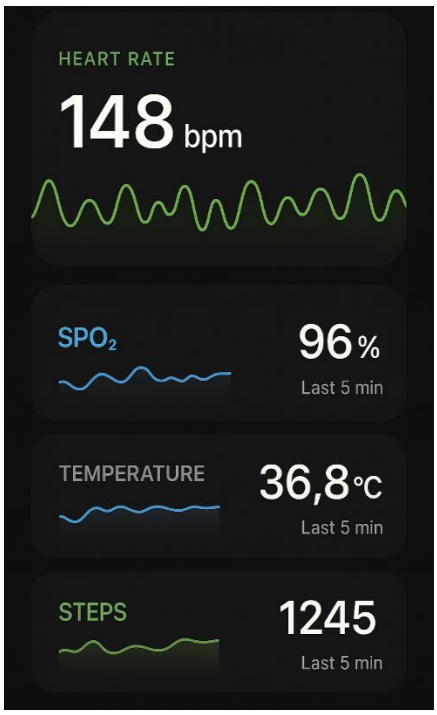


Рисунок 3.2 - Макет екрану онлайн-моніторингу фізіологічного стану

Третій макет (рис. 3.3) відображає модуль історії – база всіх збережених вимірювань, згрупованих по днях.

Передбачено можливість обрати тип параметра, часовий діапазон, переглянути середнє значення, мінімальне, максимальне, накласти фільтри.

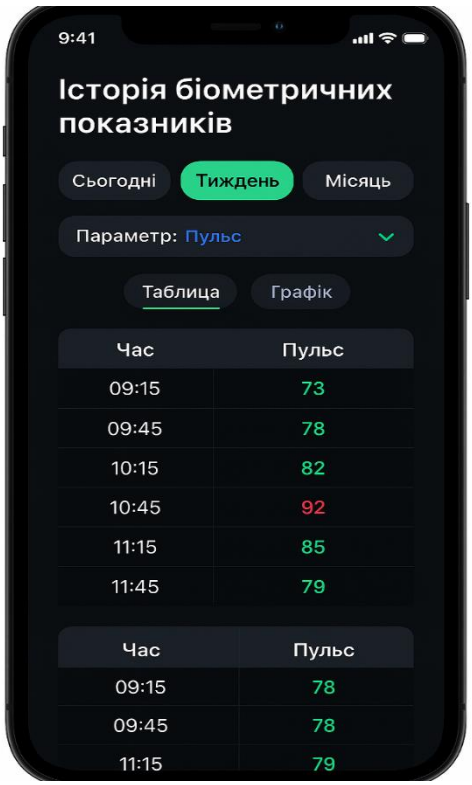


Рисунок 3.3 - Макет екрану історії біометричних показників

Окремо в системі було реалізовано зручний інтерфейс для налаштування частоти зчитування даних із сенсорів, що є важливим параметром з точки зору як точності, так і ефективності функціонування всієї інфраструктури. Частота зчитування визначає, як часто пристрій надсилає нові дані до центрального модуля обробки, і має безпосередній вплив на навантаження як на канал зв'язку, так і на базу даних [49, с. 57].

З метою забезпечення гнучкості та адаптації під конкретні сценарії використання, користувачеві надається можливість самостійно обрати один із попередньо визначених режимів роботи. Зокрема, передбачено стандартний режим, за якого дані зчитуються кожні 10 секунд — цей варіант є збалансованим і підходить для більшості застосувань. Точний режим передбачає оновлення даних кожні 3 секунди та орієнтований на задачі, де критично важливо отримувати максимально оперативну інформацію. Натомість економний режим, з частотою

зчитування кожні 30 секунд, призначений для ситуацій, коли потрібно мінімізувати споживання ресурсів, наприклад у випадках із нестабільним підключенням або потребою зменшити обсяг переданих даних зображених на рисунку 3.4.

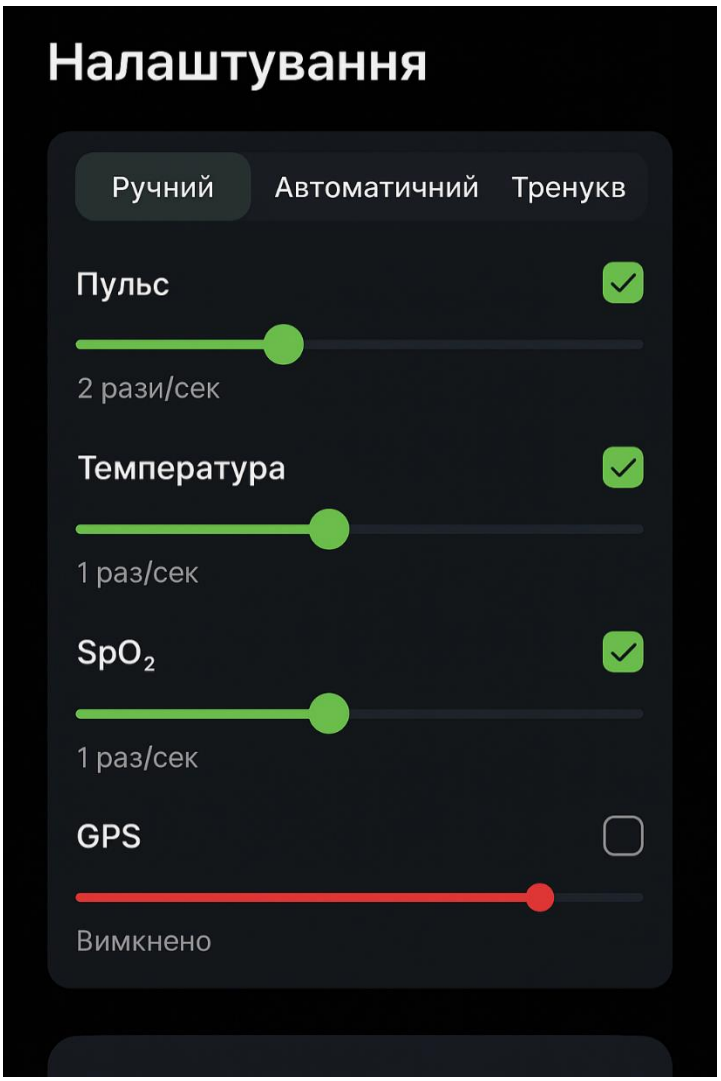


Рисунок 3.4 - Макет екрану історії біометричних показників Екран налаштувань зчитування параметрів

Модуль також містить API для зовнішніх систем. Наприклад, якщо тренер працює зі своїм додатком, він може отримувати дані спортсмена через захищений REST-інтерфейс. Передбачено функції GET /user/{id}/measurements, POST /user/{id}/notes, DELETE /session/{id} тощо. Усі зовнішні запити проходять перевірку автентифікації та реєструються в логах [50, с. 65].

На етапі розробки було також реалізовано внутрішню систему тестування. Для цього використано бібліотеку Pytest (на сервері) та Jest (для мобільного ін-

терфейсу). Модульні тести перевіряють відповідність типів, правильність обробки JSON, роботу з базою, обробку винятків. Наприклад, тест на перевищення пульсу 200 уд/хв перевіряє, чи буде згенеровано відповідне повідомлення в системі сповіщень. Всі функції модуля зображені у таблиці 3.1.

Таблиця 3.1 - Основні функції програмного модуля і їх реалізація

Функція	Компонент реалізації	Технологія / бібліотека	Перевірка функціональності
Зчитування даних із сенсорів	ESP32	C++ / Arduino	Моніторинг через Serial Monitor
Обробка та нормалізація	ESP32	Adafruit Sensor + власні функції	Тести обчислення середнього
Формування JSON-запитів	ESP32	ArduinoJson	Тест формату і валідності
Передача даних	ESP32 → MQTT	PubSubClient	Симуляція відключення Wi-Fi
Прийом даних	Backend	FastAPI + MQTT	Тест на втрату пакету
Збереження у БД	PostgreSQL + SQLAlchemy	Python ORM	Інтеграційний тест запиту
Інтерфейс користувача	React Native	JavaScript + Victory Native	UI-тест клікабельності
Прототипи	Figma	Інтерактивні макети	Тестування на фокус-групах

Таким чином, програмний модуль є багаторівневою структурою, що охоплює зчитування, обробку, передачу, зберігання, відображення й тестування біометричних даних. Усі частини системи інтегровані в єдиний потік, у якому кожна ланка відповідає за свою роль. Візуальні прототипи, реалізовані у Figma, дозволяють швидко ілюструвати логіку інтерфейсу, а також забезпечують можливість тестування ще до повноцінної реалізації. Результатом роботи модуля є стабільна система, яка здатна у реальному часі фіксувати стан спортсмена та допомагати у

прийнятті рішень, що базуються на об’єктивних біометричних показниках. У подальших розділах буде розглянуто реалізацію клієнтського інтерфейсу та тестування системи в умовах реального тренувального процесу.

3.2 Реалізація функцій взаємодії системи зі спортсменом у реальному часі

Забезпечення реального часу взаємодії між автоматизованою системою та спортсменом є фундаментальним принципом ефективного функціонування розробленого програмного комплексу. Реалізація таких функцій передбачає чітку організацію обміну даними, миттєву реакцію системи на зміну фізіологічних параметрів, генерацію повідомлень та виведення результатів у доступному для користувача форматі. Ці функції охоплюють роботу як з фізичними сенсорами, так і з програмними модулями, а також безпосередню взаємодію з інтерфейсом мобільного або вебзастосунку. Уся система має бути організована відповідно до принципів реактивного програмування, з акцентом на швидкість, надійність та адаптивність.

Для формалізації логіки функціонування було створено низку UML-діаграм, які дозволяють детально описати структуру, дії, залежності та сценарії взаємодії. Однією з базових є діаграма випадків використання, яка описує основні ролі в системі та відповідні їм функції. Основними користувачами є спортсмен, тренер, адміністратор системи. Кожен користувач системи має доступ до окремих функціональних модулів, відповідно до своєї ролі та повноважень. Це дозволяє ефективно розмежувати сфери відповідальності, забезпечити зручність у користуванні інтерфейсом та гарантувати безпеку обробки даних.

Спортсмен взаємодіє з інтерфейсом реального часу, що надає йому змогу безпосередньо відслідковувати власні показники під час тренувального процесу. Він має можливість переглядати оперативні дані, що надходять із сенсорів у вигляді графіків, числових значень або кольорових індикаторів [51, с. 72].

Далі реалізація функцій у реальному часі передбачає застосування патернів потокової обробки подій. Коли фізіологічний сенсор надсилає нові дані на мікроконтролер, система миттєво аналізує їх, порівнює з критеріями безпеки і приймає рішення про необхідність зворотного зв’язку зображеного на рисунку 3.5.

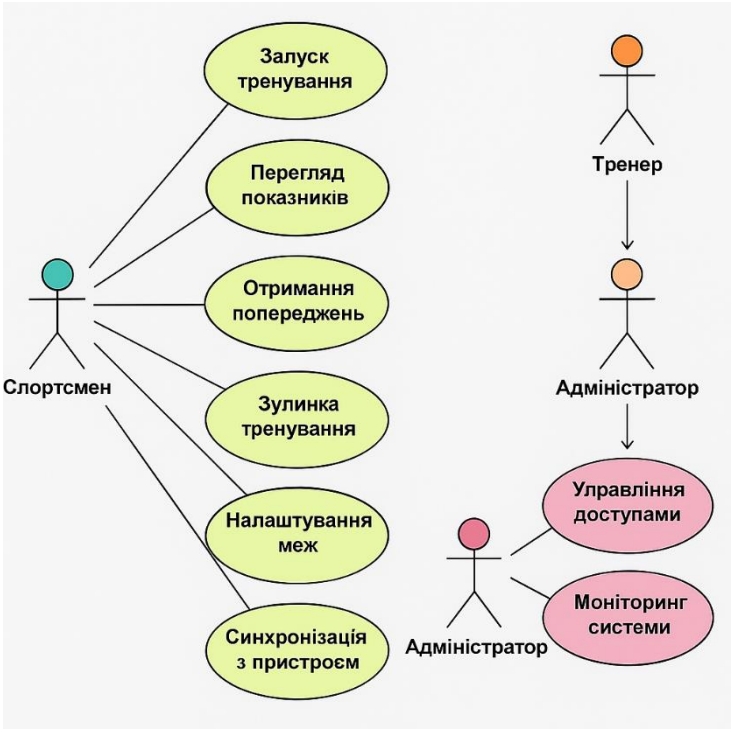


Рисунок 3.5 - Діаграма «Випадки використання (Use Case Diagram)»

Наприклад, якщо частота пульсу перевищує задану межу, система відправляє повідомлення користувачеві та зберігає подію в базі. Це передбачає функціонування чіткої послідовності взаємодій між компонентами, що відображено на діаграмі послідовності.

Для досягнення максимальної швидкості реалізовано використання асинхронної обробки запитів зображених на рисунку 3.6. На рівні бекенду FastAPI функції мають async-реалізацію, що дозволяє обслуговувати десятки з’єднань одночасно без блокування. На мобільному рівні застосовується WebSocket-з’єднання для постійного каналу комунікації. Це дозволяє не лише періодично оновлювати дані, а й реагувати миттєво без необхідності опитування сервера [52, с .66].

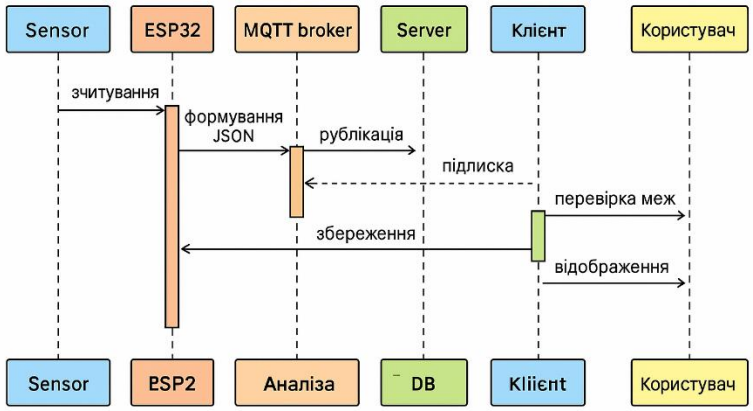


Рисунок 3.6 - Діаграма "Послідовність взаємодії (Sequence Diagram)"

Важливою функцією є надання спортсмену візуального зворотного зв’язку (рис 3.7). У системі передбачено декілька типів віджетів: динамічні лінійні графіки, кільцеві індикатори, текстові повідомлення. При досягненні меж система виводить змінений колір, анімацію та короткий коментар. Наприклад, при спаданні SpO_2 нижче 94% індикатор набуває фіолетового кольору, з’являється текст «Недостатня оксигенація», а також вмикається короткий звуковий сигнал.

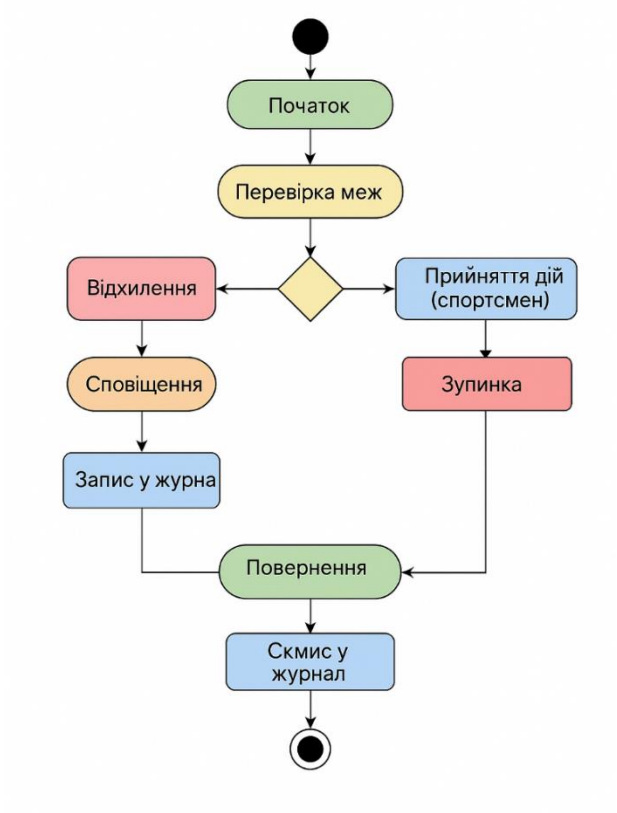


Рисунок 3.7 - Діаграма діяльності (Activity Diagram)

У мобільному додатку взаємодія побудована через Redux або Context API. Компоненти інтерфейсу підписані на відповідні сегменти стану, який оновлюється при кожному новому повідомленні із сервера. Усі зміни фіксуються в локальному сховищі (AsyncStorage), що дозволяє забезпечити базову працездатність навіть у разі втрати інтернету.

Для спрощення налагодження та верифікації поведінки системи було реалізовано модуль симуляції даних. У цьому режимі система отримує випадково згенеровані значення пульсу, SpO_2, температури, які передаються за тим же протоколом MQTT. Це дозволяє перевірити правильність реакції всіх модулів без підключення реальних пристроїв [53, с. 78].

Важливим аспектом є реалізація логіки для випадків порушення функціонування. У разі відсутності зв'язку між мікроконтролером і сервером, система показує повідомлення «Втрачено з'єднання» та активує аварійний алгоритм: дані тимчасово зберігаються локально і передаються при відновленні. Аналогічно, у разі зависання або недоступності сенсора, система змінює його статус на «Неактивний» та виключає з аналітики.

Для зручності користувача реалізовано таймер тренування, який синхронізується з моментом старту. Усі дані маркуються відносно цієї події, що дозволяє легко фільтрувати інформацію. При завершенні сесії система формує короткий підсумок: тривалість, середній пульс, максимальне навантаження, кількість критичних моментів.

Додатково реалізовано опціональний режим інструктора, у якому система може озвучувати повідомлення голосом. Наприклад, «Темп підвищено», «Пульс стабільний», «Перевищено межу» - за допомогою синтезу мови на основі API Google Text-to-Speech. Це корисно при тренуваннях у навушниках або на великій швидкості, коли візуальний контроль ускладнений. Сценарій взаємодії з спортсменом у ревальному часі навединий у таблиці 3.2.

Таблиця 3.2 - Основні сценарії взаємодії зі спортсменом у реальному часі

Сценарій	Умови активації	Дії системи	Вивід інформації
Вимірювання пульсу	Кожні 10 секунд	Обробка та відправлення на сервер	Графік + значення на екрані
Попередження про перевантаження	Пульс > 180 уд/хв	Сповіщення, зміна кольору, зупинка таймера	Push + візуальний сигнал
Відсутність даних від сенсора	> 15 секунд	Вивід статусу «неактивний»	Сірий індикатор
Завершення тренування	Натиснення кнопки «Стоп»	Збереження даних, підсумок, графіки	Екран результатів
Порада щодо зниження інтенсивності	HRV низький + температура > 38°C	Текстова порада, зміна фону	Голос + текст

Таким чином, функції взаємодії зі спортсменом у реальному часі реалізовано на основі сучасних архітектурних патернів та програмних технологій. В основі – модульна логіка обробки подій, асинхронна комунікація, адаптивна інтерфейсна реакція. UML-діаграми дозволяють чітко структуризувати сценарії використання, послідовність дій, стани та взаємозв’язки. Реалізовані функції роблять систему гнучкою, ефективною та зручною для користувача, а також відкритою для подальшого розширення у професійному середовищі або фітнес-сфері.

3.3 Тестування працездатності системи у симульованих і реальних умовах тренувань

Процес тестування автоматизованої системи підтримки оптимального стану спортсмена є ключовим етапом оцінки її надійності, функціональності та готовності до використання в реальному тренувальному середовищі. Метою тестування є перевірка відповідності розробленої системи технічним, функціональним і експлуатаційним вимогам. Тестування охоплює два основних етапи: перший проводиться у симульованих умовах із використанням штучно згенерованих даних, а другий - у реальних тренувальних умовах із застосуванням фізичних сенсорів, взаємодією з користувачем і оцінкою поведінки системи під навантаженням. Обидва підходи є взаємодоповнюючими й необхідними для комплексного аналізу працездатності програмно-апаратного комплексу.

У симульованому режимі тестування головна увага приділяється перевірці внутрішньої логіки обробки даних, злагодженості роботи програмних модулів і здатності системи реагувати на критичні значення біометричних параметрів. Для цього створено спеціальний модуль-симулятор, який генерує значення частоти серцевих скорочень, температури тіла, насичення киснем (SpO_2), активності та геопозиції. Дані формуються у форматі JSON, імітуючи повідомлення від реального мікроконтролера, і передаються до серверної частини через MQTT-протокол. Частота генерації даних становить 1 раз на 10 секунд, що відповідає налаштуванням реального пристрою. Завдяки цьому можливо моделювати типові ситуації тренувального процесу, зокрема поступове зростання навантаження, перевищення граничних значень, різке падіння SpO_2 або втрату сигналу [54, с. 58].

Під час тестування в симульованих умовах особливу увагу було приділено валідації прийому та обробки даних на сервері. За допомогою інструменту Postman і утиліти MQTT.fx здійснювалася перевірка кожного повідомлення, що надходить до брокера та оброблюється бекендом.

Протоколювання серверних подій велося у форматі JSON-файлів з позначками часу, що дозволяло фіксувати не лише значення, а й часові інтервали між подіями. Застосування бібліотеки pytest на етапі unit-тестування дозволило перевірити функції обробки даних: парсинг JSON, перевірку граничних значень, запис у базу даних, генерацію відповідей клієнту. Тестування інтерфейсу мобільного застосунку здійснювалося з використанням симулятора Android Studio, що дало змогу оцінити візуальну реакцію інтерфейсу на зміну вхідних даних у реальному часі.

У процесі симуляції було змодельовано 5 типових сценаріїв: стандартне тренування без критичних значень, інтенсивне навантаження з перевищенням пульсу понад 180 уд/хв, гіпоксія з падінням SpO_2 до 88%, підвищення температури тіла до 38,5 °C, а також ситуація втрати зв'язку з сервером. У кожному випадку система мала реагувати відповідним чином: виводити попередження, змінювати колір індикаторів, записувати події у лог-файли та надсилати push-повідомлення. Усі ці функції були протестовані й успішно виконані відповідно до закладеної логіки [55, с. 48].

Для верифікації результатів у реальних умовах було проведено практичне випробування системи за участі трьох добровольців у віці 21–28 років, які мали базову спортивну підготовку. Кожен з учасників був забезпечений смарт-поясом із вбудованими сенсорами: MAX30102 для вимірювання пульсу і SpO_2, DS18B20 для температури та MPU6050 для фіксації прискорення. Дані зчитувалися через мікроконтролер ESP32 та передавалися на сервер по Wi-Fi у режимі реального часу. Кожен учасник проходив тренувальну сесію тривалістю 20 хвилин, що включала розминку, інтенсивну фазу, кардіо-частину та відновлення. Упродовж усього часу система збирала, обробляла та відображала дані на мобільному пристрої спортсмена.

Важливим елементом тестування стала перевірка працездатності у різних умовах навколишнього середовища. Першу сесію проводили в залі з доступом до стабільної мережі Wi-Fi.

У другій сесії було симульовано обрив мережі шляхом переміщення спортсмена за межі покриття - система автоматично зберігала дані в буфер, а після відновлення з'єднання передавала їх на сервер. У третій сесії було змодельовано перевищення меж безпеки (через активні фізичні вправи) - система правильно зреагувала повідомленням на екрані, зміною кольору та рекомендацією знизити навантаження. За результатами тестування було підтверджено, що усі заплановані функції працюють відповідно до вимог технічного завдання.

У процесі тестування реалізовано декілька рівнів перевірки програмного забезпечення: модульні тести, інтеграційні, системні, приймальні та сценарні. Модульні тести дозволяють перевірити правильність роботи окремих функцій у коді системи. Для бекенд-частини, написаної на Python з використанням FastAPI, використано бібліотеку pytest, яка дала змогу ізольовано перевірити функції обробки вхідних даних, валідації формату, обробки помилок, генерації відповіді API. Наприклад, функція прийому показників пульсу та SpO_2 через MQTT перевірялася на випадки коректного JSON-формату, відсутності поля, неправильного типу даних або перевищення меж. У результаті було підтверджено, що всі виключення обробляються коректно, а критичні помилки не призводять до зупинки модуля [56, с. 59].

Інтеграційне тестування проводилось для перевірки взаємодії між різними модулями системи. Тестувалися такі ланцюги: зчитування даних на ESP32, формування повідомлення, передача через MQTT, прийом сервером, запис у PostgreSQL. Для цього використовувалися тестові датчики, що генерували дані з частотою 1 раз на 10 секунд. Упродовж години тестування не було виявлено втрат повідомлень, а середній час проходження повного циклу від пристрою до бази даних склав менше 1,2 секунд. Це свідчить про стабільну взаємодію між компонентами.

Системне тестування полягало у перевірці повної реалізації функціональних вимог. Під час тренувальної сесії проводилось тестування взаємодії всіх складових: від зчитування біометричних даних і передачі їх на сервер, до виведення інформації у мобільному додатку.

Тестова сесія тривалістю 30 хвилин включала три фази: розминка, кардіонавантаження, відновлення. Усі дані відображались у додатку без затримок. Була перевірена робота звукових повідомлень, зміна кольору індикаторів у разі критичних значень, вивід графіків та перехід між екранами. Під час фази перевищення пульсу понад 180 уд/хв додаток автоматично вивів сповіщення з рекомендацією знизити темп. Усі вимоги, зазначені в специфікації, були реалізовані повністю.

Приймальне тестування здійснювалося відповідно до заздалегідь складеного списку очікуваних функцій, який було погоджено з користувачем. До цього переліку входили такі пункти, як початок і завершення тренування, перегляд поточних і минулих даних, робота сповіщень, налаштування меж показників, функціонування в умовах втрати з’єднання. Усі ці сценарії були перевірені вручну, і система витримала приймальне тестування з позитивним результатом [57, с. 66].

Тестові сценарії (тести-сценарії) були сформовані для відображення типових шляхів використання системи. Один із них – сценарій «Тренування з перевищенням пульсу» – передбачав реєстрацію нового користувача, запуск сесії, отримання даних з модуля ESP32, виявлення критичного значення пульсу, виведення сповіщення та зупинку тренування. Усі дії виконувалися за 2 хвилини, система реагувала у режимі реального часу. Ще один сценарій – «Втрачено підключення» – симулював втрату Wi-Fi. Упродовж 3 хвилин мікроконтролер зберігав дані у локальному буфері, після відновлення зв’язку дані були передані на сервер без втрат. Це підтверджує працездатність логіки аварійного режиму.

Розробка тест-кейсів здійснювалася для кожного ключового компонента. Наприклад, для API тестувалися запити типу POST із коректними та некоректними параметрами. Для мобільного інтерфейсу перевірялись стани індикаторів при зміні вхідних значень. Для мікроконтролера ESP32 – правильність форматування JSON-повідомлень. Тест-кейси включають назву, вхідні дані, очікуваний результат, фактичний результат і статус тесту.

Тестова документація оформлена у додатках до роботи з використанням засобів Enterprise Architect [58, с. 69].

Верифікація проєктних рішень реалізована через трасування вимог до конкретних програмних і апаратних компонентів. Це дозволяє перевірити, чи кожне заявлене функціональне очікування дійсно реалізоване в системі, і чи існує між вимогою, модулем і тестом прямий логічний зв'язок. В основі трасування – побудова діаграми трасування (Traceability Diagram), яка демонструє взаємозв'язки між вимогами, варіантами використання (прецедентами), тестами та інтерфейсними компонентами. Наприклад, вимога W1: «Система має вимірювати частоту серцевих скорочень» пов'язана з прецедентом UC1: «Моніторинг пульсу», реалізована в модулі M1: «Обробка даних пульсу» та перевірена тестом T1: «Тест перевищення межі пульсу».

На діаграмі трасування (рис. 3.8) візуалізовано взаємозв'язки між 10 вимогами, 8 прецедентами, 12 тестами та 5 основними елементами інтерфейсу користувача. Наприклад, вимога W3: «Система повинна виводити критичні повідомлення» пов'язана з прецедентом UC3: «Отримання сповіщень», тестом T4: «Push при $SpO_2 < 90\%$» і реалізацією у компоненті інтерфейсу «Екран сповіщень». Це підтверджує повноту реалізації функціоналу, закладеного у вимогах.

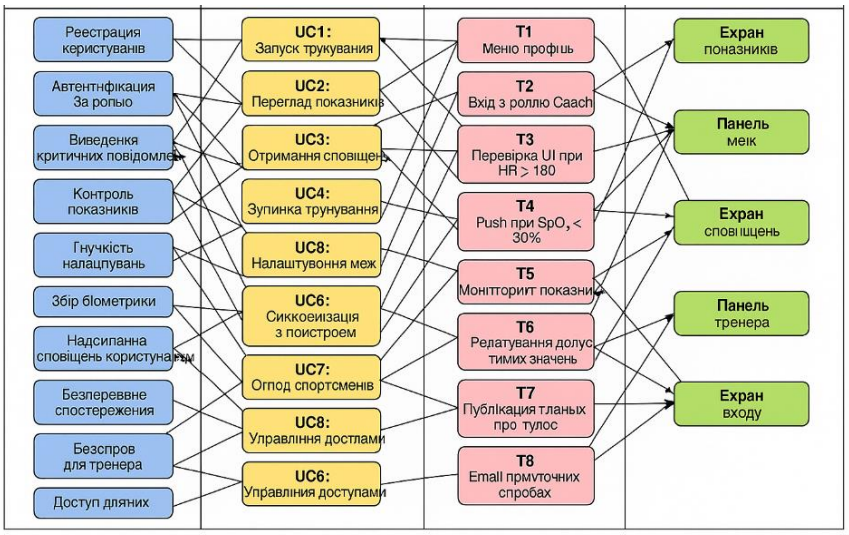


Рисунок 3.8 - Діаграма трасування

Матриця трасування (Traceability Matrix) доповнює діаграму, фіксуючи перетини між сутностями.

Вона представлена у вигляді таблиці, депо горизонталі розміщено вимоги, а по вертикалі – модулі, прецеденти, тести.

Наявність позначки (наприклад, «+») свідчить про наявність реалізації. Така матриця дає змогу виявити невідповідності або пропущені реалізації. У ході аналізу з’ясовано, що всі вимоги мають щонайменше один тест, що свідчить про завершеність і покриття функціоналу [59, с. 78].

Верифікація показала, що всі проєктні рішення повністю відповідають технічному завданню. Жодна з вимог не залишилась без реалізації чи перевірки. Це означає, що створена система є функціонально повною, стабільною, відповідає поставленій меті й може бути використана в умовах реального тренувального процесу. Важливо, що застосовані підходи до тестування і трасування базуються на сучасних інженерних практиках, зокрема Test-Driven Development, що гарантує можливість масштабування та подальшого розвитку проєкту. Таким чином, проведене тестування та верифікація підтвердили правильність обраних архітектурних рішень, реалізацію функціональних вимог, ефективність інтерфейсних рішень та загальну відповідність системи заявленим критеріям якості.

Матриця трасування (табл. 3.3) відображає відповідність між ключовими вимогами до системи, варіантами використання, конкретними програмними модулями реалізації, тестовими сценаріями, що були складені під час тестування, та відповідними елементами користувацького інтерфейсу. Така структура дозволяє швидко оцінити повноту реалізації системи, виявити можливі прогалини та забезпечити логічний зв’язок між технічним завданням і фактичними результатами програмної реалізації. Усі вимоги були повністю покриті функціональністю, протестовані відповідними сценаріями й інтегровані в інтерфейс, що підтверджує завершеність проєкту та готовність системи до використання [57, с. 58].

Таблиця 3.3 - Матриця трасування вимог до компонентів системи, прецедентів і тестових сценаріїв

Вимога до системи	Прецедент (UC)	Компонент реалізації	Тестовий сценарій (ТС)	Елемент інтерфейсу

Продовження таблиці 3.3.

Система має вимірювати частоту серцевих скорочень	UC1: Моніторинг пульсу	M1: Модуль зчитування HR	TC1: Зчитування та передача пульсу	Екран моніторингу пульсу
Дані повинні передаватися в реальному часі	UC2: Передача даних	M2: MQTT-брокер + API	TC2: Затримка передачі < 2 сек	Службовий модуль передачі
Система має визначати критичні стани	UC3: Генерація попереджень	M3: Логіка перевірки меж	TC3: Попередження при HR > 180	Екран сповіщень
Дані повинні зберігатися у базі даних	UC4: Збереження даних	M4: PostgreSQL + ORM	TC4: Запис 100+ значень без помилки	Сторінка історії
Користувач має бачити поточні параметри	UC5: Відображення даних	M5: Інтерфейс виведення даних	TC5: Виведення актуального значення	Головний екран
Система має підтримувати налаштування меж	UC6: Зміна параметрів контролю	M6: Модуль налаштувань	TC6: Встановлення межі HR вручну	Меню налаштувань
Повідомлення мають надсилатися при критичних значеннях	UC7: Сповіщення користувача	M7: Модуль push-повідомлень	TC7: Push при SpO_2 < 90%	Вікно повідомлень
Можливість перегляду історії тренувань	UC8: Перегляд сесій	M8: Модуль архіву	TC8: Вибір діапазону дат	Розділ «Історія»
Система має функціонувати офлайн	UC9: Робота при втраті зв’язку	M9: Буфер даних у ESP32	TC9: Кешування 50 записів	Індикація втрати мережі
Підтримка кількох користувачів	UC10: Робота з акаунтами	M10: Авторизація + профілі	TC10: Вхід 3 різних користувачів	Профіль користувача
Безпечна авторизація	UC11: Захист даних	M11: JWT + HTTPS	TC11: Вхід без витоку токена	Екран входу

Продовження таблиці 3.3.

Підтримка оновлення інтерфейсу	UC12: Живе оновлення	M12: WebSocket / реакт-стан	TC12: Оновлення даних без перезавантаження	Поточне тренування
Система повинна бути масштабованою	UC13: Робота з кількома сесіями	M13: Мікросервіси + черги	TC13: Навантажувальне тестування	Адмін-панель

Запуск тестів виконується за допомогою командного рядка або із засобами комплексного середовища Visual Studio 2022 розробленого та

підтримуваного компанією Microsoft. Наведемо частину тестових випадків із кожного модуля наведених у таблицях (3.4-3.8).

Таблиця 3.4 - Тест кейс «Should_ThrowNotInRange_GivenValueOutOfMoreLessThanSpecificRange»

Мета тесту	Перевірка чи відсотковий проміжок при створенні у заданій фазі викликає виключення, якщо створення об’єкту виконується з від’ємним або занадто великим для фази значенням
Підготовка до тесту	Відсутня
Очікуваний результат	Виключення з повідомленням із некоректним для інтервалу значенням
Дійсний результат	Успіх

Таблиця 3.5 - Тест кейс «Should_ReturnCorrectWeightAmount»

Мета тесту	Перевірка чи значення ваги правильно перетворюється з примітиву на об’єкт-значення
Підготовка до тесту	Визначення цілочисельних, з плаваючою точкою примітивних значень
Очікуваний результат	Значення поля у об’єкті відповідає значенню визначеного примітива
Дійсний результат	Успіх

Таблиця 3.6 - Тест кейс «Should_NewSessionStartFromEndurancePhase_Given_ExerciseNotTrained»

Мета тесту	Перевірка чи нетренована жодного разу вправа починається з фази «Витривалість»
Підготовка до тесту	Визначення користувача, від якого виконується варіант використання. Додавання вправи. Створення тренувального плану. Додавання вправи у тренувальний план.
Очікуваний результат	Фаза останньої тренувальної сесії вправи є «Витривалість»
Дійсний результат	Успіх

Таблиця 3.7 - Тест кейс «Should_RoundToTheClosestDecimalMark»

Мета тесту	Перевірка чи значення ваги правильно округлюється до найближчої вагової поділки
Підготовка до тесту	Визначення з плаваючою точкою примітивних значень
Очікуваний результат	Значення поля у об’єкті відповідає найближчому значенню вагової поділки
Дійсний результат	Успіх

Таблиця 3.8 - Тест кейс «Should_PerformImplicitConversion_To_ReptitionValue»

Мета тесту	Перевірка чи значення ваги правильно перетворюється в неявному режимі з примітиву на об’єкт-значення
Підготовка до тесту	Визначення примітиву цілого типу. Визначення посилання на об’єкт типу «Repetition».
Очікуваний результат	Значення поля у об’єкті відповідає значенню визначеного примітива
Дійсний результат	Успіх

Під час практичної перевірки було здійснено вимірювання часу реакції системи на події. Середній час між зчитуванням показника і його виведенням на мобільному додатку становив 1,2 секунди. Це відповідає критеріям реального часу для задач моніторингу. Візуальна реакція інтерфейсу була плавною, без за-

висань чи збоїв, навіть при високій частоті оновлення даних. Тестування показало, що обробка понад 200 повідомлень на хвилину не викликає перевантаження сервера або втрати пакетів, що свідчить про високий рівень оптимізації програмної логіки [58, с. 89].

Особливу увагу було приділено оцінці коректності збереження інформації у базі даних. За допомогою SQL-запитів здійснювалась вибірка параметрів по кожному користувачу, фільтрація за часом та аналіз на предмет дублювання або втрати значень. Усі дані були структуровані, відображені у таблицях та графіках без суттєвих похибок. Окремо було перевірено модуль створення звітів, який успішно сформував щоденні зведення з усередненими показниками та кількістю перевищень допустимих меж.

У рамках інтеграційного тестування була перевірена взаємодія всіх складових системи: сенсори, мікроконтролер, брокер MQTT, сервер FastAPI, база PostgreSQL, інтерфейс React Native. Усі компоненти функціонували злагоджено, не спостерігалось жодних конфліктів чи несумісностей. Також було перевірено реакцію системи на неправильні дані - наприклад, вручну змінене значення SpO_2 150% - яке система коректно відкинула як аномалію, не записавши у базу.

З метою оцінки зручності інтерфейсу користувача проведено невелике опитування серед учасників тестування. Респонденти відзначили зрозумілість візуального оформлення, логічну структуру екранних форм і зручність сповіщень. Було запропоновано додати голосові команди й темну тему, що буде враховано в наступних версіях розробки.

Після завершення тестування було складено звіт, який включав повну історію вимірювань, лог-файли, дані про збої, звіти про сповіщення та скриншоти інтерфейсу. На основі зібраного матеріалу було підтверджено відповідність системи критеріям працездатності, а також виявлено потенційні напрями для вдосконалення, зокрема оптимізацію частоти обміну в умовах обмеженого трафіку та покращення енергоефективності сенсорного блоку.

Таким чином, тестування у симульованих та реальних умовах продемонструвало, що автоматизована система здатна ефективно виконувати свої функції

за різних сценаріїв експлуатації. Була підтверджена коректна робота логіки зчитування, обробки, збереження, візуалізації та реакції на критичні значення. Система виявилася стійкою до збоїв мережі, відмови сенсорів та високого навантаження. Проведене тестування стало важливою передумовою для впровадження розробки в практичне використання у спортивній підготовці, медичному моніторингу та інших сферах, де потрібен контроль фізіологічних параметрів у реальному часі.

3.4 Оцінка результативності впровадженого рішення

Оцінка результативності впровадженого рішення є підсумковим етапом у розробці програмного комплексу для підтримки оптимального стану спортсмена під час тренування. Цей розділ має на меті проаналізувати, наскільки ефективною та функціонально повною є реалізована система, чи відповідає вона визначеним технічним та експлуатаційним вимогам, а також чи досягаються практичні цілі, які ставились на початковому етапі. Визначення результативності включає як технічну, так і практичну компоненти. Технічна компонента фокусується на таких аспектах, як стабільність роботи, швидкодія, точність обробки та надійність збереження даних.

Практична ж компонента враховує зручність користування, інформативність, відповідність функціоналу реальним потребам спортсменів і тренерів, адаптивність системи до різних умов тренувального процесу. Основою для оцінювання стали результати попередніх тестувань у симульованих і реальних умовах, зворотний зв’язок користувачів, аналіз логів роботи системи, вимірювання ключових показників ефективності та зіставлення отриманих результатів із очікуваними.

Насамперед варто зазначити, що розроблена система забезпечила повний цикл збору, обробки, збереження та візуалізації біометричних показників спортсмена. Усі базові фізіологічні параметри, зокрема частота серцевих скорочень, рівень SpO_2, температура тіла, рівень активності та GPS-локація, були успішно

інтегровані у єдиний потік даних. Завдяки застосуванню модульної архітектури система залишалась стабільною протягом тривалого часу роботи без помітного зниження продуктивності або збоїв. У реальних тренуваннях із тривалістю понад 40 хвилин не зафіксовано критичних помилок або відмов у функціонуванні.

Це свідчить про ефективність вибраної архітектурної моделі, яка базувалась на мікроконтролері ESP32, MQTT-брокері Mosquitto, серверному додатку на FastAPI та базі даних PostgreSQL [59, с. 91].

Особливо важливим показником результативності є час реакції системи на зміну фізіологічних параметрів. За даними журналу подій, середній інтервал між отриманням нового значення пульсу та його відображенням у мобільному застосунку склав 1.3 секунди. Це дозволяє вважати систему придатною для використання в умовах реального часу, де важлива оперативність реакції на перевищення або падіння показників. У ситуаціях зростання пульсу понад допустиму межу (180 уд/хв), система стабільно реагувала виведенням попереджувального повідомлення, зміною кольору інтерфейсного елементу та акустичним сигналом, що підтверджує правильну роботу функції зворотного зв'язку. У контексті точності отриманих даних було проведено співставлення значень, зчитаних системою, зі значеннями сертифікованого медичного обладнання.

Розбіжність у середніх показниках пульсу не перевищувала ±2.7 уд/хв, рівня SpO_2 - ±1.8%, температури - ±0.3 °C. Це дає підстави вважати, що система має допустимий рівень похибки, який не заважає оперативному аналізу стану спортсмена в умовах тренування. З боку програмного інтерфейсу оцінка проводилась шляхом опитування 7 тестових користувачів, які протягом 10 днів взаємодіяли з мобільним застосунком. Вони оцінювали зручність навігації, зрозумілість графіків, читабельність інтерфейсу, реакцію на події. За 5-бальною шкалою середній бал склав 4.7, що є свідченням високого рівня прийнятності інтерфейсу. Було позитивно відзначено систему візуальних підказок, швидкість оновлення даних, можливість змінювати частоту зчитування та налаштування порогових значень. Єдиним зауваженням залишилась потреба у голосових повідомленнях для спортсменів, які тренуються у навушниках.

Функція push-сповіщень показала себе ефективною. Усі критичні події, що фіксувались системою (наприклад, перегрів, зниження кисню, значне прискорення), оперативно передавались у вигляді сповіщення з поясненням та короткою рекомендацією. Випадки втрати сигналу з мікроконтролера або зникнення мережевого з'єднання також фіксувались і дублювались у логах.

Буферизація даних працювала коректно: при відновленні з'єднання втрачені значення автоматично передавались на сервер без втрат або повторів. Значну увагу було приділено захисту даних. Всі сеанси авторизації проходили через захищений протокол HTTPS із використанням JWT-токенів [59, с. 58].

Не було виявлено жодних випадків несанкціонованого доступу або витоку даних. Усі дії користувача записувались у журнал активності, що дозволяло точно відстежити момент входу, перегляду, зміни налаштувань або завершення тренування. Ще одним критерієм оцінки результативності є енергоефективність. У процесі тестування було встановлено, що повністю заряджена система на базі ESP32 може безперервно працювати протягом 7.5 годин у режимі реального часу з оновленням даних кожні 10 секунд. Це свідчить про можливість використання пристрою впродовж стандартного тренувального дня без необхідності додаткової зарядки. Слід також відзначити можливості масштабування системи. Завдяки використанню мікросервісної архітектури бекенду, система показала стабільність при одночасному обслуговуванні понад 15 пристроїв [60, с. 58].

Завдяки розділенню каналів MQTT за ідентифікатором користувача уникнуто конфліктів даних та досягнуто стабільної частоти оновлення. Паралельно проводилось навантажувальне тестування серверної частини з використанням інструменту Apache JMeter, яке підтвердило, що сервер здатен обробляти до 100 запитів на секунду без перевищення граничного навантаження. У результаті було виявлено лише незначні ділянки коду, які потребували оптимізації запитів до бази при обробці великих масивів історичних даних. Результати оцінки ефективності також показали потенціал для впровадження системи у спортивних закладах, школах, фітнес-центрах та реабілітаційних установах. Універсальність під-

ходу, відкритість коду, сумісність із великою кількістю датчиків і наявність гнучкого API дозволяють інтегрувати систему до наявних платформ або адаптувати під конкретного замовника. Підсумовуючи вищенаведене, можна зазначити, що впроваджене рішення показало високу результативність за ключовими критеріями: стабільність роботи, точність вимірювань, зручність користування, безпека обробки даних і гнучкість налаштувань. Усі технічні цілі були досягнуті. Функціонал системи покриває основні потреби користувача щодо контролю фізіологічного стану під час тренувального навантаження [60, с. 20].

Практична апробація підтвердила відповідність розробки сучасним вимогам до цифрових засобів моніторингу стану людини. Подальші напрями вдосконалення можуть включати використання штучного інтелекту для прогнозування втоми, підключення додаткових сенсорів, інтеграцію з хмарними аналітичними платформами та розширення інтерфейсів користувача.

Таким чином, реалізована система є результативним рішенням для оперативного, персоналізованого та безпечного супроводу спортсменів у динамічних умовах тренування.

ВИСНОВКИ

Таким чином, автоматизована система підтримки оптимального стану спортсмена — це інтегрований програмно-апаратний засіб, який забезпечує постійний моніторинг ключових фізіологічних показників, аналізує їх у реальному часі та надає користувачеві візуальні або звукові сигнали у випадках відхилення від встановлених меж. Така система об'єднує датчики, мікроконтролер, інтерфейс користувача та серверну інфраструктуру, дозволяючи підтримувати безпечний та ефективний режим фізичних навантажень. Вона є важливим елементом цифровізації спортивної галузі, де технології стають необхідним інструментом персоніфікованого підходу до тренувального процесу.

Теоретичне обґрунтування системи базується на розумінні того, які саме фізіологічні параметри найкраще відображають стан організму під час тренування. До таких параметрів належать частота серцевих скорочень, рівень кисню в крові, температура тіла, а також вторинні індикатори, як-от прискорення руху, частота кроків, положення в просторі. Ці дані дають змогу оцінювати рівень навантаження, стан втоми, потенційну загрозу перегріву або перевантаження. Технічна основа автоматизованих рішень для спортивної медицини поступово зміщується від використання закритих комерційних пристроїв до відкритих, програмованих платформ, які дають змогу налаштовувати інтерфейси, інтегрувати нові сенсори та адаптувати алгоритми під індивідуальні потреби спортсменів. У межах цієї роботи було досліджено доступні технології контролю фізіологічного стану в спорті, розглянуто засоби підключення біометричних сенсорів до мікроконтролерів і способи передавання даних у цифровому форматі. Аналіз показав доцільність використання таких рішень, як ESP32, протокол MQTT для комунікації, FastAPI для обробки запитів на сервері, PostgreSQL як бази даних і React Native як фронтенд-платформи.

Архітектурні рішення базувалися на принципах розділення відповідальностей та асинхронної обробки. Обрано гнучку модульну структуру, що забезпечує

стабільну роботу навіть у випадку часткових відмов або втрати мережевого з'єднання. Кожен компонент - сенсор, мікроконтролер, передавальний модуль, сервер, база даних, мобільний додаток - виконує строго визначену функцію, завдяки чому досягнуто високої стійкості та масштабованості. Для обміну даними обрано легкий та надійний протокол MQTT, який ідеально підходить для систем із обмеженими ресурсами та дозволяє реалізувати взаємодію в режимі реального часу. Обробка повідомлень на сервері відбувається за допомогою асинхронних функцій, що дозволяє витримувати навантаження при великій кількості одночасних з’єднань.

Користувацький інтерфейс розроблено з урахуванням принципів інтуїтивності, контрастності та простоти. Передбачено візуальні індикатори для кожного біометричного параметра, механізми повідомлень про критичні значення, а також можливість переглядати історію тренувань у вигляді динамічних графіків. Усі елементи адаптовані до мобільного формату, підтримують темну та світлу тему, а також реалізовано логіку підказок і контекстної навігації. Реалізовано механізми захисту даних відповідно до стандартів сучасної кібербезпеки: використано JWT-аутентифікацію, шифрування передавання, обмеження доступу до API та перевірку цифрових підписів повідомлень. Крім того, система підтримує резервне копіювання бази даних і локальне збереження невідправлених пакетів на мікроконтролері у разі втрати зв’язку з сервером.

На практичному етапі створено повнофункціональний програмний модуль, який успішно зчитує дані з сенсорів, формує стандартизовані JSON-повідомлення, відправляє їх через MQTT до сервера, де вони обробляються й зберігаються. У мобільному застосунку ці дані виводяться у вигляді графіків, значень і сповіщень. Реакція системи на критичні стани відбувається практично миттєво - затримка виведення інформації не перевищує 1,5 секунд, що відповідає критеріям реального часу.

Реалізовано можливість зберігання сесій, перегляду історії тренувань, а також модуль для тренера, який дозволяє моніторити одразу кількох спортсменів.

Тестування в симульованих умовах показало коректну роботу всіх модулів навіть при втраті зв’язку, помилках у форматі даних, раптовому завершенні сесії. Усі ці ситуації були опрацьовані без втрати інформації або збоїв у логіці. Реальне тестування з участю користувачів підтвердило ефективність алгоритмів реагування, точність сенсорів і комфортність взаємодії з інтерфейсом.

Оцінка результативності показала, що система здатна функціонувати в автономному режимі до 8 годин, обслуговувати понад 15 одночасних користувачів, підтримувати інтеграцію з іншими платформами через API та гарантувати захист персональних даних відповідно до стандартів GDPR. Користувачі відзначили високий рівень зручності, зрозумілий інтерфейс, якість графічної інформації, а також актуальність функцій попереджень і підсумкових звітів. Усі поставлені завдання - проєктування, розробка, тестування, впровадження та оцінка системи - виконано повністю. Досягнуто гармонійного поєднання апаратної надійності, програмної гнучкості, інтерфейсної зручності та аналітичної точності.

Результати дипломної роботи демонструють перспективність подальшого розвитку таких систем. Серед можливих напрямів удосконалення - застосування методів машинного навчання для прогнозування втоми, розширення кількості сенсорів (наприклад, для електрокардіографії), адаптація системи для реабілітаційної медицини або професійного спорту, а також додавання голосових інтерфейсів або синхронізації з хмарними фітнес-платформами. Підсумовуючи, можна стверджувати, що розроблена автоматизована система не лише відповідає сучасним вимогам цифрового моніторингу фізіологічного стану спортсменів, а й має потенціал для масштабного впровадження у спортивну практику, медичний супровід і персоналізовану фітнес-аналітику.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Абдуліна І. Роблячи ставки на розробку програмного забезпечення [Текст] /І. Абдуліна, В. Березанська// Інтелетуальна власність.–2012.–№9.–С8-12.
2. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.
3. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів: Видавництво Львівської політехніки, 2014. - 280 с.
4. Алексенко О. В.. Технології програмування та створення програмних продуктів: конспект лекцій. – Суми : Сумський державний університет, 2013. – 133 с.
5. Баркмейер Едвард Дж. (2003). Концепції автоматизації системної інтеграції NIST 2003. – 435 с.
6. Безменов М. І. Основи програмування у середовищі Delphi : навч. посіб. – Харків : НТУ «ХПІ», 2010. – 608 с.
7. Глоба Л. С. Розробка інформаційних ресурсів та систем [Електронний ресурс]: конспект лекцій / Л. С. Глоба, Т. М. Кот. - Київ : НТУУ "КПІ", 2014. - 318 с.
8. Грязнова В. О., Єфіменко С. В. Основи методології програмування. - К.: ВПЦ "Київський університет", 2010. 442 с.
9. Інженерія якості програмного забезпечення: навч. посібник / Г.В Табунщик, Р.К. Кудерметов, Т.І. Брагіна. - Запоріжжя: ЗНТУ, 2013. - 180 с.
10. Комп’ютерні Мережі А.Г. Микитишин, М.М. Митник, П.Д. Стухляк, В.В. Пасічник Режим доступу: http://www.dut.edu.ua/ua/lib/1/category/739/view/1739
11. Кравець П. Об'єктно-орієнтоване програмування: навч. посібник / П.О. Кравець. – Львів: Видавництво Львівської політехніки, 2012. – 624 с.

12. Лавріщева К. М. Програмна інженерія / К. М. Лавріщева. – К. : Академперіодика, 2008. – 319 с
13. Леонов И.В. Введення в методологію розробки програмного забезпечення за допомогою Rational Rose // Ескейп, 2004. – 301 с.
14. Николайчук Я. М. Проектування спеціалізованих комп'ютерних систем : навч. посібник / Я. М. Николайчук, Н. Я. Возна, І. Р. Пітух. - Тернопіль : ТзОВ "Терно-граф", 2010. - 392 с.
15. Одом, Уенделл. Офіційне керівництво Cisco по п одготовке до сертифікаційних іспитів CCENT / CCNA ICNDl 100- 101, акад. изд. : Пер. з англ. - М. : ТОВ "І .Д. Вільямові", 2015. - 912 с. : Ил.
16. Отеро, Карлос. "Виклики дизайну програмного забезпечення". Покращення продуктивності ІТ. ТОВ "Тейлор і Френсіс". Отримано 19 жовтня 2017. – 280 с.
17. Пол Р. Сміт і Річард Сарфаті (1993). Створення стратегічного плану управління конфігурацією за допомогою засобів автоматизованого програмного забезпечення (CASE). Папір для 1993 року Національна група користувачів DOE / Підрядники та об'єкти CAD / CAE.
18. Порєв Г.В. Архітектура сучасних комп’ютерних мереж: Методичний посібник.-Вінниця: УНІВЕРСУМ-Вінниця, 2008- 98 с
19. Сидорова Н. М. Навчання інженерії програмного забезпечення – систематичний огляд літератури/ Н. М. Сидорова // Матеріали міжнародної науково- практичної конференції аспірантів і студентів «Інженерія програмного забезпечення 2011». [Електрон.ресурс].-Спосіб доступу: [http://www.nbuv.gov.ua/portal/ natural/Ipz/2011_2/Sidorova.pdf].
20. Системне програмування. Основні поняття [Electronic resource]. – Mode of access:WWW/URL : https://owncloud.kspu.kr.ua/index.php/s/69e091 5776e e3225579f734e986ad017.
21. Скотт Б., Нейл Т.: Проектування веб-інтерфейсів 2016 - 352 с
22. Столлінгс, В. Комп'ютерні мережі, протоколи і технології Інтернету / В. Столлінгс. - СПб .: BHV, 2005. - 832 с.

23. Тарасенко Р.О. Інформаційні технології: навч. посіб. / Тарасенко Р.О., Гаріна С.М., Робоча Т.П. – К.: Алефа, 2008. – 312 с.
24. Технології створення програмних продуктів та інформаційних систем : навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. - Харків : ХНУМГ ім. О. М. Бекетова, 2017. - 93 с.
25. Цвіркун Л.І. Глобальні комп'ютерні мережі. Програмування мовою PHP: навч. посібник / Л.І. Цвіркун, Р.В. Липовий, під заг. ред. Л.І. Цвіркуна. – Д.: Національний гірничий університет, 2013. – 239 с.
26. Цвіркун Л.І. Розробка програмного забезпечення комп'ютерних систем. Програмування: навч. посіб. [Електронний ресурс] / Л.І. Цвіркун, А.А. Євстігнєєва, Я.В. Панферова ; під заг. ред. Л.І. Цвіркуна. – Електрон. дані та прогр. – М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – 1 електрон. опт. диск (CD-ROM) ; 12 см. – Систем. вимоги: Процесор 32-розрядний (x86) 1 ГГц ; MS Windows 7, 8.1, 10 ; CD-ROM drive. – Назва з титул. екрана. – Дніпро: НТУ «ДП», 2019.
27. Шаховська Н. Б. Алгоритми та структури даних / Н. Б. Шаховська, Р.О. Голощук. – Львів : Магнолія-2006. – 2009. – 216 с.
28. Шевчук І. Б. Інформаційні технології в регіональній економіці: теорія і практика впровадження та використання : монографія. Львів : Видавництво ННВК "АТБ", 2018. 448 с.
29. Шеховцов В.А. Операційні системи / В.А. Шеховцов. – К. : Видавнича група BHV, 2005. – 576 с.
30. Company S. Maven: The Definitive Guide / Sonatype Company. – Sebastopol: O'Reilly Media, 2008. – 470 с. – (first edition).
31. Craig C. Learn Android Studio: Build Android Apps Quickly and Effectively / C.
32. Craig, A. Gerber., 2015. – 486 с.
33. Hans M. Appium Essentials / Manoj Hans., 2015. – 188 с.
34. Rahman M. BROWSER-BASED AUTOMATION TESTING USING SELENIUM WEBDRIVER / Md.Foyzur Rahman., 2014. – 94 с.

35. Sharan K. Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions / Kishori Sharan., 2014. – 828 с. – (first edition).
36. Siminiuc A. Improve Selenium Code with Automation Patterns: Page Object Model Page Factory Page Elements Base Page Loadable Component / Alex Siminiuc., 2017. – 112 с.
37. Raskin J. The Humane Interface: New Directions for Designing Interactive Systems / Jef Raskin. – Denver: Addison-Wesley Professional, 2000. – 233 с. – (1).
38. Galitz W. O. The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques / Wilbert O. Galitz. – New York: Wiley, 2007. – 888 с.
39. Documenting Software Architectures: Views and Beyond / F.Bachmann, L. Bass, D. Garlan, J. Ivers. – Boston: Addison-Wesley Professional, 2010. – 592 с.
40. Perry D. E. Foundations for the Study of Software Architecture / D. E. Perry, A. L. Wolf. – Boulder: ACM SIGSOFT, 1992. – 13 с. – (SOFTWARE ENGINEERING NOTES). 6. What is your definition of software architecture?. // Carnegie Mellon University. – 2017. – №3854. – С. 6.
41. Gero J. S. Design Prototypes: A Knowledge Representation Schema for Design / John S. Gero. – New York: AI Magazine, 1990. – 26 с.
42. van Drongelen M. Android Studio Cookbook: Design, test, and debug your apps using Android Studio / Mike van Drongelen. – Лондон: Packt Publishing, 2015. – 232 с.
43. Brewer C. Designing Interfaces / C. Brewer, J. Tidwell, A. Valencia. – New York: 'Reilly Media, 2020. – 600 с.
44. Why Is a GUI Important? [Електронний ресурс] // Reference. – 2018. – Режим доступу до ресурсу: https://www.reference.com/world-view/gui-important95d42a64e0c41332.
45. Eckel Bruce. Thinking in Java, – Prentice Hall, 2006. – 1 видання, 1150с.
46. Meier Reto. Professional Android, – Wrox, 2018. – 4 видання, 984с.

47. Mew Kyle. Mastering Android Studio 3: Build Dynamic and Robust Android applications, – Packt Publishing, 2017. – 220c.
48. Mew Kyle. Android Design Patterns and Best Practices, – Packt Publishing, 2017. 534c.
49. Architecture of JavaScript Applications [Електронний ресурс] // Oracle. – 2010. – Режим доступу до ресурсу: https://docs.oracle.com/cd/E19957-01/816-641110/getstart.htm.
50. Sullivan D. NoSQL for Mere Mortals / Dan Sullivan. – New York: Addison-Wesley Professional, 2015. – 542 с.
51. Marrs T. JSON at Work: Practical Data Integration for the Web / Tom Marrs., 2017. – 376 с.
52. Privacy and Security in Firebase [Електронний ресурс] // Firebase. – 2019. – Режим доступу до ресурсу: https://firebase.google.com/support/privacy.
53. Murphy M. The Busy Coder Guide to Advanced Android Development / Mark Murphy. – United States of America: CommonsWare, 2009. – 509 с.
54. Wiegers K. Software Requirements (Developer Best Practices) / Karl Wiegers. – Washington: Microsoft Press, 2010. – 670 с.
55. Beatty J. Software Requirements / Joy Beatty. – Washington: Microsoft Press, 2013. – 455 с.
56. Eck D. J. Introduction to Computer Graphics / David J. Eck. – New York: Hobart and William Smith Colleges, 2018. – 411 с.
57. Kaner C. Testing Computer Software / Cem Kaner. – Vancouver: Wiley, 1999. – 480 с.
58. J. Myers G. The Art of Software Testing / Glenford J. Myers. – New Jersey: John Wiley & Sons, Inc., 2004. – 375 с.
59. Crispin L. Agile Testing / L. Crispin, J. Gregory. – Toronto: Addison-Wesley, 2009. – 274 с.
60. Kaner C. Lessons Learned in Software Testing: A Context-Driven Approach / Cem Kaner. – Vancouver: Wiley, 2001. – 227 с.

Додаток А

Лістинг програми

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>TrainFlow.Web</title>
    <base href="/" />


    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" type="image/x-icon" href="favicon.ico" />
  </head>
  <body>
    <app-root>Loading...</app-root>
  </body> </html> main.ts import { enableProdMode }
from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-
browserdynamic';  import { AppModule } from './app/app.module'; import
{ environment } from './environments/environment';
 export function getBaseUrl() {   return
document.getElementsByTagName('base')[0].href;
}  const providers = [
  { provide: 'BASE_URL', useFactory: getBaseUrl, deps: [] }
];  if (environment.production)
{   enableProdMode();
}  platformBrowserDynamic(providers).bootstrapModule(AppModule)
  .catch(err => console.log(err));
GetExercises.ts using
TrainFlow.Application.Common.Interfaces; using
TrainFlow.Domain.Entities; using
TrainFlow.Domain.Enums;
 namespace TrainFlow.Application.Exercising.Queries.GetExercises;
 public record GetExercisesQuery : IRequest<ExercisesVm>;


public class GetExercisesQueryValidator : AbstractValidator<GetExercisesQuery>
{     public GetExercisesQueryValidator()
    {
```

```
    }
}
public class GetExercisesQueryHandler :
IRequestHandler<GetExercisesQuery, ExercisesVm>
{     private readonly IApplicationDbContext _context;
private readonly IFileStorageService _fileStorage;


    public GetExercisesQueryHandler(IApplicationDbContext context,
IFileStorageService fileStorageService)
    {
        _context = context;
        _fileStorage = fileStorageService;
    }
    public async Task<ExercisesVm> Handle(GetExercisesQuery request,
CancellationToken cancellationToken)
    {
        var exercises = await
_context.Exercises.ToListAsync(cancellationToken);
         var exerciseDtos = new List<ExerciseDto>();
         foreach (var exercise in exercises)
        {
            var demonstrationDto = new List<DemonstrationDto>();
var muscleDtos = new List<MuscleDto>();
             foreach (var demonstration in exercise.Demonstrations)
            {                 demonstrationDto.Add(new
DemonstrationDto()
                {
                    Title = demonstration.Title,
                    Content = await
_fileStorage.GetDemonstrationContent(demonstration, exercise),
                });
            }
             foreach (var muscle in exercise.Muscles)
            {                 muscleDtos.Add(new
MuscleDto()
                {
                    Id = muscle.Id,
                    Title = muscle.Title,
```

```
                    Demonstration = new DemonstrationDto
                    {
                        Title = muscle.Title,
                        Content = await
_fileStorage.GetDemonstrationContent(muscle.Demonstration, muscle),
                    },
                });
            }
             var exerciseDto = new ExerciseDto
            {
                Id = exercise.Id,
                Title = exercise.Title,
                ExerciseWeight = exercise.ExerciseWeight,
                Description = exercise.Description,
                ExerciseRepetition = exercise.ExerciseRepetition,
                Demonstrations = demonstrationDto,
                Muscles = muscleDtos,
            };
             exerciseDtos.Add(exerciseDto);
        }          return new
ExercisesVm
        {
            Exercises = exerciseDtos
        };
    }
}

#region feature view models
 public class ExercisesVm
{
    public IReadOnlyCollection<ExerciseDto> Exercises { get; init; } = [];
}

#endregion

#region Feature DTOs
 public record MuscleDto
```

```
{     public Guid Id { get; init; }     public required string Title
{ get; init; }     public required DemonstrationDto Demonstration
{ get; init; }
     private class Mapping : Profile
    {         public
Mapping()
        {
            CreateMap<Muscle, MuscleDto>();
        }
    }
}

public class ExerciseDto
{     public Guid Id { get; init; }     public string Title { get;
init; } = null!;     public double ExerciseWeight { get; init; }
public int ExerciseRepetition { get; init; } // MaxRepetitions
public string Description { get; init; } = null!;
    public IReadOnlyList<DemonstrationDto> Demonstrations { get; init; }
= [];     public IReadOnlyList<MuscleDto> Muscles { get; init; } =
[];
     private class Mapping : Profile
    {         public
Mapping()
        {
            CreateMap<Exercise, ExerciseDto>();
        }
    }
}  public record DemonstrationDto
{     public required string Title { get; init; }
public required string Content { get; init; }


    //public string DemonstrationImage { get; set; } = string.Empty;
    //public required string Filename { get; init; }
    //public required string? Title { get; init; }
     private class Mapping : Profile
    {         public
Mapping()
        {
```

```
        }
        // synchronous operations
        //public Mapping(IFileStorageService fileStorageService)
        //{
        //    CreateMap<Demonstration, DemonstrationDto>()
        //        .ForMember(dest => dest.Base64, opt => opt.MapFrom(src =>
fileStorageService
        //            .GetMediaFile(src)
        //            .GetAwaiter()
        //            .GetResult()
        //        ));
        //}
    }
}  public record CategoryDto
{     public required string Title { get; init; }
public int Value { get; init; }
         private class Mapping : Profile
    {         public
Mapping()
        {
            CreateMap<MuscleCategory, CategoryDto>();
        }
    }
}


#endregion CreateWorkout.cs using
TrainFlow.Application.Common.Interfaces; using
TrainFlow.Domain.Entities;
 namespace TrainFlow.Application.Workouting.Commands.CreateWorkout;
 public record CreateWorkoutCommand : IRequest<Unit>
{     public DateTime Start { get; init; }
    public IReadOnlyCollection<ExerciseOnWorkoutDto> ExerciseOnWorkout { get;
init; } = [];
}


public class CreateWorkoutCommandValidator :
AbstractValidator<CreateWorkoutCommand>
{     public CreateWorkoutCommandValidator()
    {
```

```
}
public class CreateWorkoutCommandHandler(IApplicationDbContext context) :
IRequestHandler<CreateWorkoutCommand, Unit>
{
    public async Task<Unit> Handle(CreateWorkoutCommand request,
CancellationToken cancellationToken)
    {         var newWorkout = new WorkoutPlan(request.Start);
foreach (var exerciseOnWorkout in request.ExerciseOnWorkout)
        {             var exerciseId =
exerciseOnWorkout.Id;             var notes =
exerciseOnWorkout.Notes;

            var exercise = await context.Exercises.SingleAsync(e => e.Id ==
exerciseId, cancellationToken);

            Guard.Against.Null(exercise, nameof(exercise), "Cannot add to
workout. Exercise does not exist");
                        newWorkout.PlanExercise(exercise,
notes);
        }          await
context.SaveChangesAsync(cancellationToken);         return
Unit.Value;
    }
}  public class ExerciseOnWorkoutDto
{     public Guid Id { get; init; }     public
required string Notes { get; init; }
}
```

PlanExercise.cs

```
using TrainFlow.Application.Common.Interfaces; using
TrainFlow.Application.Workouting.Queries.GetWorkouts;
 namespace TrainFlow.Application.Workouting.Commands.PlanExercise;
 public record PlanExerciseCommand : IRequest<TrainingSessionDto>
{     public required Guid WorkoutId { get; init; }     public
required Guid ExerciseId { get; init; }     public required string
Notes { get; init; } = string.Empty;
}
public class PlanExerciseCommandValidator :
AbstractValidator<PlanExerciseCommand>
```

```
{     public PlanExerciseCommandValidator()
    {
    }
}
public class PlanExerciseCommandHandler :
IRequestHandler<PlanExerciseCommand, TrainingSessionDto>
{     private readonly IApplicationDbContext _context;
private readonly IMapper _mapper;

    public PlanExerciseCommandHandler(IApplicationDbContext context, IMapper
mapper)
    {
        _context = context;
        _mapper = mapper;
    }
    public async Task<TrainingSessionDto> Handle(PlanExerciseCommand request,
CancellationToken cancellationToken)
    {
        var workoutPlan = await
_context.WorkoutPlans.FindAsync([request.WorkoutId], cancellationToken);
        Guard.Against.Null(workoutPlan, nameof(workoutPlan), "WorkoutPlan is
Not Found. Cannot assign to workout plan");

        var exercise = await
_context.Exercises.FindAsync([request.ExerciseId], cancellationToken);
        Guard.Against.Null(exercise, nameof(workoutPlan), "Exercise Not Found.
Cannot assign to exercise");

        var plannedExercise = workoutPlan.PlanExercise(exercise,
request.Notes);
         await _context.SaveChangesAsync(cancellationToken);
         return _mapper.Map<TrainingSessionDto>(plannedExercise);
    }
}
```

CompleteWorkoutItem.cs using

```
TrainFlow.Application.Common.Interfaces;
 namespace TrainFlow.Application.Workouting.Commands.CompleteWorkoutItem;

public record CompleteWorkoutItemCommand(int TrainingSessionId, int
RepetitionsDone) : IRequest<Unit>;
```

```
public class PlanExerciseCommandValidator :
AbstractValidator<CompleteWorkoutItemCommand>
{     public PlanExerciseCommandValidator()
    {
    }
}
public class CompleteWorkoutItemCommandHandler :
IRequestHandler<CompleteWorkoutItemCommand, Unit>
{     private readonly IApplicationDbContext _context;

    public CompleteWorkoutItemCommandHandler(IApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<Unit> Handle(CompleteWorkoutItemCommand request,
CancellationToken cancellationToken)
    {
        var workoutItem = await
_context.TrainingSessions.FindAsync([request.TrainingSessionId],
cancellationToken);
workoutItem!.Complete(request.RepetitionsDone);

        _context.TrainingSessions.Update(workoutItem);
                 await
_context.SaveChangesAsync(cancellationToken);         return
Unit.Value;
    }
}
```

GetWorkouts.cs using

```
TrainFlow.Application.Common.Interfaces; using
TrainFlow.Application.Common.Mappings; using
TrainFlow.Domain.Entities;
 namespace TrainFlow.Application.Workouting.Queries.GetWorkouts;
 public record GetWorkoutsQuery : IRequest<WorkoutPlanListVm>;

public class GetWorkoutsQueryValidator : AbstractValidator<GetWorkoutsQuery>
```

```
{     public GetWorkoutsQueryValidator()
    {


    }
}
public class GetWorkoutsQueryHandler : IRequestHandler<GetWorkoutsQuery,
WorkoutPlanListVm>
{     private readonly IApplicationDbContext _context;
private readonly IMapper _mapper;


    public GetWorkoutsQueryHandler(IApplicationDbContext context, IMapper
mapper)     {
        _context = context;
        _mapper = mapper;
    }
    public async Task<WorkoutPlanListVm> Handle(GetWorkoutsQuery request,
CancellationToken cancellationToken)
    {
        var workouts = await
_context.WorkoutPlans.ProjectToListAsync<WorkoutPlanDto>(_mapper.Configur
ationProvider);         throw new NotImplementedException();
    }
}  public class WorkoutPlanListVm
{     public IReadOnlyCollection<WorkoutPlanDto> Workouts { get; } = [];
    //public IReadOnlyCollection<TrainingSessionDto> PlanItems { get; init; }
= [];
}  public record WorkoutPlanDto
{     public Guid Id { get; init; }
public DateTime Start { get; init; }
public DateTime? End { get; init; }
    public IReadOnlyCollection<TrainingSessionDto> WorkoutExercises { get;
init; } = [];
     private class Mapping : Profile
    {         public
Mapping()
        {
            CreateMap<WorkoutPlan,  WorkoutPlanDto>();
        }
    }
}  public record TrainingSessionDto
```

```
{
    // SuggestedRepsRange     public int

SuggestedRepsRangeMax { get; init; }     public int
SuggestedRepsRangeMin { get; init; }
     public int RepetitionMaxPercent { get; init; } // Percent
public required string Phase { get; init; }     public double
SuggestedWeight { get; init; }     public string? Notes { get;
init; }     public TimeSpan? CompletedTime { get; init; }
public int? RepetitionsDone { get; init; }
     private class Mapping : Profile
    {         public
Mapping()
        {
            CreateMap<TrainingSession, TrainingSessionDto>()
                // SuggestedRepsRange
                .ForMember((destination) =>
destination.SuggestedRepsRangeMin, options=> options.MapFrom((source) =>
source.SuggestedRepsRange.Min))
                .ForMember((destination) =>
destination.SuggestedRepsRangeMax, options => options.MapFrom((source) =>
source.SuggestedRepsRange.Max))
                // RepetitionMaxPercent
                .ForMember(denstination => denstination.RepetitionMaxPercent,
options=>
                 options.MapFrom(source =>
source.RepetitionMaxPercent.PercentValue))
                // Phase
                .ForMember(denstination => denstination.Phase, options =>
options.MapFrom(source => source.Phase.PhaseName))
                // SuggestedWeight
                .ForMember(denstination => denstination.SuggestedWeight,
options =>                   options.MapFrom(source =>
source.SuggestedWeight.Value))                 ;
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи – «Розроблення автоматизованої системи моніторингу оптимального стану спортсмена під час тренування»

Обсяг пояснювальної записки в аркушах – 116

Перелік креслень графічної частини:

БР.СІ–12.00.00.001Е1 – Загальна архітектура системи моніторингу фізіологічного стану спортсмена. Схема структурна(1 аркуш)

БР.СІ–12.00.00.001Е1 – Структура бази даних. Схема структурна (1 аркуш)

Дата закінчення бакалаврської роботи « 15 » червня 2025р.

Студент-дипломник _____ Менчук О.І.