МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

СКОРОХОД ОЛЕКСАНДРА МАКСИМІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА ВЕБСИСТЕМИ ДЛЯ ЗБЕРІГАННЯ ТА АНАЛІЗУ ДАНИХ З ПЕРСОНАЛЬНИХ МЕДИЧНИХ ВИМІРЮВАЛЬНИХ ПРИСТРОЇВ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Юрій АНТОНОВ, доцент кафедри
інформаційних технологій,
к. фіз.-мат. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Скороход О.М. Розробка вебсистеми для зберігання та аналізу даних з персональних медичних вимірювальних пристроїв. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У бакалаврській роботі досліджено проблематику автоматизованого аналізу медичних даних та розроблено інформаційну систему MedData для збирання, зберігання, обробки та візуалізації медичних показників пацієнтів. Проаналізовано сучасні підходи до побудови медичних інформаційних систем та методи обробки структурованих медичних даних. Розглянуто архітектуру застосунку, організацію бази даних і механізми роботи з даними через веб-інтерфейс.

Ключові слова: медичні дані, інформаційна система, аналіз даних, Python, Streamlit, PostgreSQL, Pandas, веб-застосунок, база даних, медична аналітика.

ABSTRACT

Skorokhod O.M. Development of a Web System for Storage and Analysis of Data from Personal Medical Measuring Devices. Specialty 122 "Computer Science", educational program "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2026.

The bachelor's thesis investigates the problem of automated medical data analysis and presents the MedData information system for collecting, storing, processing, and visualizing patient medical indicators. Modern approaches to building medical information systems and methods of processing structured medical data are analyzed. The application architecture, database organization, and data management mechanisms through a web interface are considered.

Keywords: medical data, information system, data analysis, Python, Streamlit, PostgreSQL, Pandas, web application, database, medical analytics.

ЗМІСТ

ВСТУП

Актуальність роботи. Зростання обсягів медичних даних у цифровому середовищі створює потребу в ефективних інструментах їх обробки та аналізу. Медичні установи щоденно накопичують інформацію про стан здоров'я пацієнтів, результати аналізів та динаміку клінічних показників, проте значна частина цих даних залишається невикористаною через відсутність доступних аналітичних засобів. Розробка зручного застосунку для аналізу медичних даних на основі сучасного технологічного стеку є актуальним практичним завданням.

Мета дослідження. Розробка інформаційної системи MedData для збирання, зберігання, обробки та візуалізації медичних даних пацієнтів із використанням технологій Python, Streamlit, PostgreSQL та Pandas.

Завдання дослідження:

– проаналізувати сучасний стан проблематики аналізу медичних даних та існуючі рішення;

– визначити вимоги до системи та обрати технологічний стек;

– спроєктувати архітектуру системи та структуру бази даних;

– реалізувати модулі збору, зберігання та аналізу медичних даних;

– розробити веб-інтерфейс із засобами візуалізації за допомогою Streamlit;

– провести тестування системи та оцінити її функціональність.

Об'єкт дослідження: процес автоматизованої обробки та аналізу структурованих медичних даних пацієнтів.

Предмет дослідження: методи та засоби побудови інформаційних систем аналізу медичних даних.

Практичне значення дослідження: система MedData може застосовуватися у медичних установах для аналізу даних пацієнтів та формування аналітичних звітів. Система є кросплатформною та не потребує складного розгортання, що робить її доступною для закладів різного масштабу.

Структура кваліфікаційної роботи: бакалаврська робота складається зі вступу, трьох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНОГО СТАНУ ПИТАННЯ

1.1 Проблематика аналізу медичних даних

Медична галузь є однією з найбільш інформаційно насичених сфер людської діяльності. Щодня у лікувально-профілактичних закладах генерується значний обсяг різноманітних даних: результати лабораторних аналізів, дані моніторингу вітальних показників, медичні записи лікарів, відомості про призначені препарати. Проте значна частина цих даних залишається невикористаною через відсутність доступних аналітичних інструментів [1].

Основні проблеми, що ускладнюють аналіз медичних даних, можна розподілити за кількома напрямами. По-перше, медичні дані характеризуються різнорідністю: вони можуть бути структурованими (числові показники аналізів), напівструктурованими (результати обстежень у форматі HL7) або неструктурованими (текстові нотатки лікарів). По-друге, специфічні вимоги щодо захисту персональних даних пацієнтів, передбачені Законом України «Про захист персональних даних», обмежують обмін і використання медичної інформації. По-третє, відсутність єдиної стандартизації між різними медичними установами ускладнює інтеграцію даних із різних джерел.

Незважаючи на зазначені труднощі, автоматизований аналіз медичних даних дозволяє вдосконалювати протоколи лікування, прогнозувати ризики ускладнень та підвищувати ефективність роботи медичного персоналу. Саме тому розробка доступних інструментів аналізу медичних даних є актуальним науково-практичним завданням.

1.2 Класифікація медичних даних

Розуміння природи медичних даних є необхідною передумовою для розробки ефективних інструментів їх аналізу. Медичні дані класифікуються за кількома ознаками [2].

За форматом представлення розрізняють структуровані, напівструктуровані та неструктуровані дані. Структуровані дані організовані у вигляді таблиць: демографічні відомості про пацієнтів, числові результати аналізів (рівень глюкози, гемоглобіну, холестерину), показники вітальних

функцій (артеріальний тиск, температура тіла). Напівструктуровані дані мають внутрішню організацію, але не відповідають суворій схемі – наприклад, документи у форматах HL7 та FHIR. Неструктуровані дані представлені у текстовому вигляді: нотатки лікарів, виписки з медичних карт.

За природою показників медичні дані поділяються на кількісні (числові значення – артеріальний тиск, рівень гемоглобіну) та якісні (стать пацієнта, тип захворювання за МКХ-10). За часовою структурою виділяють поперечні дані – опис стану у фіксований момент часу, та поздовжні – динаміка показників у часі. Останні є особливо цінними для оцінки ефективності лікування. Для системи MedData найбільший інтерес представляють структуровані кількісні поздовжні дані, оскільки саме вони найкраще піддаються автоматизованій обробці засобами реляційних баз даних та бібліотеки Pandas.

1.3 Методи аналізу даних

Аналіз медичних даних поєднує методи статистики, математики та інформатики. Вибір конкретного методу визначається природою даних та поставленою задачею.

Описова статистика є базовим інструментом аналізу. Вона дозволяє отримати узагальнений опис вибірки через розрахунок показників центральної тенденції (середнє, медіана), розсіювання (дисперсія, стандартне відхилення) та форми розподілу. Цей метод є відправною точкою будь-якого аналізу і дозволяє виявити аномалії та пропущені значення.

Кореляційний та регресійний аналіз застосовуються для вивчення взаємозв'язків між медичними показниками. Кореляційний аналіз оцінює силу та напрям зв'язку між змінними. Регресійний аналіз будує математичну модель залежності одного показника від інших, що використовується для прогнозування. Аналіз часових рядів дозволяє виявляти тренди у динаміці клінічних показників і прогнозувати їх зміну. Класичні методи включають ковзне середнє та ARIMA [3].

У контексті розроблюваної вебсистеми MedData інтеграція математичних методів обробки даних є базовим інструментом для отримання клінічно значущої

інформації з первинних вимірів. Найбільш практично значущими для аналізу стану здоров'я пацієнта є описова статистика, кореляційний аналіз та аналіз часових рядів. Програмна бібліотека Pandas надає потужні вбудовані засоби для високорівневої реалізації цих методів, що мінімізує обчислювальні витрати та є ключовою перевагою обраного технологічного стеку.

Описова статистика виступає першим етапом експрес-діагностики, дозволяючи агрегувати великі масиви первинних точок вимірювань (наприклад, щохвилинні записи частоти серцевих скорочень або добові заміри артеріального тиску). За допомогою базового методу .describe() у системі автоматично обчислюються такі фундаментальні метрики, як математичне сподівання (середнє значення показника), медіана, стандартне відхилення (яке вказує на рівень варіабельності та стабільності біосистеми), а також мінімальні й максимальні екстремуми. Визначення першого та третього квартилів дозволяє гнучко відсікати випадкові похибки вимірювань (артефакти) і фіксувати межі нормального стану конкретного користувача.

Кореляційний аналіз у системі MedData орієнтований на виявлення прихованих закономірностей та взаємозв'язків між різними фізіологічними параметрами. Використання вбудованої функції .corr() (зокрема, коефіцієнта кореляції Пірсона або Спірмена) дає змогу оцінити ступінь та напрямок лінійної залежності між метриками [34, 44]. Наприклад, система може в автоматичному режимі встановити математичний зв'язок між рівнем щоденної фізичної активності (кількістю кроків) і середніми показниками систолічного тиску, що забезпечує лікаря аналітичною базою для формування персоналізованих рекомендацій [13].

Аналіз часових рядів (Time Series Analysis) є найбільш критичним компонентом системи, оскільки всі дані з персональних пристроїв жорстко прив'язані до часової осі [44]. Pandas володіє унікальним інструментарієм для індексації за часом (DatetimeIndex), що дозволяє ефективно вирішувати завдання ресемплінгу даних за допомогою методу .resample() [15]. Це дає змогу трансформувати нерівномірні сирі виміри у структуровані інтервали (наприклад,

зводити хвилинні пульсограми до середньогодинних або середньодобових трендів). Окрім цього, механізми ковзного вікна (.rolling()) використовуються для згладжування випадкових коливань часового ряду, виділення довгострокових макротрендів та раннього прогнозування можливих критичних відхилень у стані здоров'я користувача. Таким чином, використання аналітичного потенціалу бібліотеки Pandas дозволяє перетворити систему MedData з пасивного сховища на динамічний інструмент предиктивного моніторингу.

1.4 Інформаційні системи в медицині

Інформаційні системи в галузі охорони здоров'я відіграють ключову роль у трансформації традиційних медичних послуг у цифрову екосистему, забезпечуючи автоматизацію клінічних, діагностичних та адміністративних процесів. Сучасна медична інформатика класифікує такі системи за функціональним призначенням, рівнем інтеграції та масштабами обробки даних, виділяючи кілька основних архітектурних класів [4].

Медичні інформаційні системи (MIC) є комплексними багаторівневими платформами корпоративного рівня, що автоматизують операційну діяльність лікувально-профілактичних закладів. На госпітальному рівні MIC інтегрують процеси реєстрації та маршрутизації пацієнтів, управління ліжковим фондом, оптимізації аптечного обліку та координації роботи клініко-діагностичних лабораторій (LIS). Основним завданням MIC є створення єдиного інформаційного простору медичного закладу для підвищення ефективності менеджменту та зниження операційних витрат.

Фундаментом сучасних інформаційних систем є концепція цифрового профілю пацієнта, яка реалізується через два взаємопов'язані блоки.

Електронні медичні записи (Electronic Medical Records — EMR) являють собою локалізовані цифрові аналоги медичних карт, що містять клінічні дані пацієнта, зібрані в межах однієї конкретної медичної установи (анамнез, результати оглядів, призначення ліків).

Електронні карти здоров'я (Electronic Health Records — EHR) орієнтовані на довгострокове накопичення комплексної інформації про стан здоров'я індивіда протягом усього його життя. На відміну від локальних EMR, системи EHR функціонують на регіональному або національному рівнях. Вони базуються на принципах інтероперабельності та транскордонного обміну даними, забезпечуючи санкціонований доступ до медичної історії пацієнта для різних постачальників медичних послуг. Надійна робота таких систем регламентується міжнародними стандартами обміну медичною інформацією, зокрема протоколами HL7 та архітектурою FHIR [17].

Окремим високотехнологічним класом є системи підтримки прийняття клінічних рішень (Clinical Decision Support Systems — CDSS). Вони функціонують як інтелектуальні надбудови над базами даних EHR/EMR і надають лікарям релевантні рекомендації на основі структурованих клінічних настанов, протоколів лікування та поточних соматичних показників пацієнта [18]. Системи CDSS виконують роль превентивного контролю: автоматично аналізують сумісність призначених медикаментів, попереджають про потенційні алергічні реакції, розраховують персоналізовані дозування та нагадують про необхідність планового скринінгу.

У цій ієрархії системи аналізу медичних даних, до яких безпосередньо належить розроблювана вебсистема MedData, посідають специфічне місце. Вони орієнтовані на збір та вторинне використання накопичених масивів інформації. На відміну від транзакційних МІС, системи аналітичного класу фокусуються на методах інтелектуального аналізу даних (Data Mining), статистичній обробці та візуалізації динамічних процесів. Реалізація системи MedData дозволяє вирішити критичну проблему сучасної цифровізації — ізольованість даних із персональних вимірювальних пристроїв, заповнюючи розрив між повсякденним моніторингом та клінічною оцінкою лікаря.

1.5 Використання штучного інтелекту в медичній аналітиці

На сучасному етапі розвитку інформаційних технологій штучний інтелект (ШІ) та алгоритми машинного навчання (Machine Learning) стають

фундаментальними інструментами медичної аналітики, відкриваючи можливості обробки великих масивів даних, які є недоступними при використанні традиційних статистичних методів [5]. Інтеграція інтелектуальних моделей у медичну практику дозволяє здійснювати перехід від пасивної фіксації соматичних показників до предиктивної та персоналізованої медицини.

У сфері клінічної діагностики та прогнозування перебігу захворювань особливе місце посідають класичні алгоритми класифікації та регресії. Зокрема, метод опорних векторів (Support Vector Machines — SVM), випадковий ліс (Random Forest) та градієнтний бустинг (Gradient Boosting, наприклад, архітектури XGBoost та LightGBM) активно застосовуються для категоризації пацієнтів за групами ризику, прогнозування ймовірності виникнення критичних станів (таких як інфаркт міокарда чи гіпертонічний криз) на основі комплексного аналізу анамнестичних маркерів та поточних фізіологічних параметрів [22].

Технології глибокого навчання (Deep Learning), представлені насамперед багатошаровими згортковими нейронними мережами (Convolutional Neural Networks — CNN), здійснили технологічну революцію в аналізі та інтерпретації медичних зображень [5]. Сучасні нейромережеві моделі здатні автоматично, з високим рівнем прецизійності, виявляти мікроструктурні патологічні зміни на рентгенівських знімках, комп'ютерних томограмах (КТ) та результатах магнітно-резонансної томографії (МРТ), що значно знижує навантаження на лікарів-радіологів і мінімізує людський фактор при первинному скринінгу.

Паралельно з аналізом сигналів і зображень розвивається напрям обробки природної мови (Natural Language Processing — NLP). Системи на базі NLP та великих мовних моделей (LLM) забезпечують можливість автоматизованої екстракції структурованої інформації з неструктурованих текстових масивів даних: текстових нотаток практикуючих лікарів, амбулаторних виписок, анамнезів та розгорнутих результатів лабораторних досліджень. Це дозволяє перетворювати якісні описи станів пацієнта у кількісні вектори ознак для подальшого математичного моделювання.

Попри високий потенціал інтелектуалізації медичних систем, масштабне впровадження ШІ безпосередньо у клінічні процеси наразі обмежене низкою критичних факторів:

– проблема «чорної скриньки» (Black Box problem) — більшість точних моделей глибокого навчання мають низький рівень інтерпретованості, що унеможливлює аргументовану клінічну верифікацію отриманих результатів;

– ризик алгоритмічної упередженості (Algorithmic Bias) — моделі можуть демонструвати спотворені результати, якщо навчальна вибірка не була репрезентативною відносно різних демографічних груп пацієнтів;

– вимоги до верифікації даних — для якісного навчання систем необхідні величезні, попередньо розмічені масиви даних, збір яких ускладнюється питаннями лікарської таємниці та захисту персональних даних [5].

У контексті розроблюваної вебсистеми MedData інтеграція штучного інтелекту (зокрема, модулів предиктивного аналізу часових рядів за допомогою легковагових бібліотек на кшталт Scikit-learn) розглядається як найбільш перспективний напрям її подальшої модернізації [22]. Поточна версія архітектури застосунку спирається на перевірені, математично прозорі традиційні методи описової статистики та кореляційного аналізу, що гарантує абсолютну інтерпретованість та швидкість обчислень у межах користувацького веб-інтерфейсу.

1.6 Аналіз існуючих рішень

Ринок програмних засобів для аналізу медичних даних є досить насиченим. Існуючі рішення можна розподілити на кілька категорій.

Комерційні аналітичні платформи – IBM Watson Health, Oracle Health Sciences, Philips HealthSuite – орієнтовані на великі медичні установи та страхові компанії. Вони пропонують широкий функціонал, однак висока вартість ліцензій та складність впровадження роблять їх недоступними для установ малого масштабу [6]. Спеціалізовані медичні аналітичні системи – Tableau for Healthcare, Microsoft Power BI – надають засоби візуалізації та побудови

інформаційних панелей. Вони більш доступні, але потребують технічних знань для налаштування.

Відкриті рішення – OpenMRS та DHIS2 – широко застосовуються у країнах з обмеженими ресурсами охорони здоров'я. DHIS2 є стандартним інструментом систем охорони здоров'я у понад 70 країнах [7]. Серед інструментів загального призначення слід відзначити Python із екосистемою бібліотек (Pandas, NumPy, Matplotlib) та R з пакетами для медичної статистики. Вони забезпечують максимальну гнучкість, проте вимагають навичок програмування.

Порівняльний аналіз показав, що існуючі рішення або є надмірно складними та дорогими, або вимагають від користувача технічних знань. Система MedData покликана заповнити цю нішу, поєднуючи аналітичні можливості Python та Pandas з інтуїтивним веб-інтерфейсом Streamlit.

Висновок до розділу 1

У першому розділі проведено аналітичний огляд сучасного стану питання аналізу медичних даних. Встановлено, що медична галузь генерує значні обсяги даних, ефективне використання яких є актуальним завданням. Виявлено ключові проблеми: різнорідність форматів, вимоги до захисту персональних даних та відсутність стандартизації.

Систематизовано класифікацію медичних даних за форматом, природою показників та часовою структурою. Розглянуто основні методи аналізу, серед яких для цілей MedData найбільш значущими є описова статистика, кореляційний аналіз та аналіз часових рядів. Охарактеризовано основні класи медичних інформаційних систем та проаналізовано перспективи застосування штучного інтелекту.

Аналіз існуючих рішень показав, що наявні продукти або надмірно складні, або вимагають глибоких технічних знань від користувача. Це обумовлює доцільність розробки системи MedData – доступного інструменту аналізу медичних даних на основі Python, Streamlit, SQLite та Pandas.

РОЗДІЛ 2 ВИМОГИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальна постановка задачі

Метою розробки є створення вебсистеми MedData, яка забезпечує збирання, зберігання, обробку та візуалізацію даних з персональних медичних вимірювальних пристроїв. На відміну від попередньої версії системи, що базувалась на локальній базі даних SQLite та однокористувацькій архітектурі, нова версія орієнтована на ширшу аудиторію: вона підтримує роботу з декількома типами пристроїв (тонометр, глюкометр, пульсоксиметр, термометр, фітнес-трекер), реалізує систему сповіщень про відхилення показників та використовує реляційну СУБД PostgreSQL, яка забезпечує надійне багатокористувацьке зберігання даних.

Основна проблема, яку вирішує система, полягає у фрагментованості персональних медичних даних: вимірювання зберігаються у пам'яті різних пристроїв, в окремих мобільних застосунках або паперових щоденниках і не мають єдиного аналітичного середовища. Людина, яка контролює кілька показників здоров'я одночасно (наприклад, тиск і рівень глюкози), змушена перемикатися між різними інтерфейсами й не може бачити загальну картину динаміки. Система MedData закриває цю потребу, надаючи уніфікований інтерфейс для всіх зареєстрованих пристроїв і всіх показників.

Цільовою аудиторією системи є пацієнти з хронічними захворюваннями (серцево-судинними, ендокринологічними), люди похилого віку, а також медичні заклади невеликого масштабу, де необхідний зручний інструмент для ведення записів пацієнтів без складного налаштування і значних витрат. Ключові характеристики, необхідні для цієї аудиторії: простота інтерфейсу, відсутність потреби у технічних знаннях, кросплатформність і доступність через браузер.

2.2 Вимоги до системи

2.2.1 Загальні вимоги

Система повинна забезпечувати доступ через стандартний веб-браузер без встановлення додаткового програмного забезпечення на стороні клієнта. Інтерфейс має бути зрозумілим для користувача без технічного досвіду: форми

введення — мінімалістичні, навігація — інтуїтивна, всі числові значення супроводжуються одиницями вимірювання та підказками щодо нормативних меж.

Усі персональні медичні дані повинні зберігатись ізольовано для кожного облікового запису. Доступ до даних конкретного користувача неможливий без авторизації. Паролі зберігаються виключно у вигляді bcrypt-хешів, жодні чутливі дані не записуються у відкритому вигляді. Налаштування середовища (адреса бази даних, секретний ключ) передаються через змінні середовища та не вбудовуються у код.

Система має бути кросплатформною і запускатись на Windows, macOS та Linux без будь-яких модифікацій. Єдиною залежністю з боку сервера є Python 3.11+ та PostgreSQL. Старт застосунку — однією командою після встановлення залежностей.

2.2.2 Функціональні вимоги

Функціональні вимоги визначають поведінку системи з точки зору кінцевого користувача. Вони сформовані на основі аналізу типових сценаріїв використання системи моніторингу медичних показників.

Функціональні:

– Реєстрація та автентифікація користувачів (email + пароль)

– Реєстрація персональних медичних пристроїв (тонометр, глюкометр, пульсоксиметр тощо)

– Введення вимірювань вручну через форму

– Імпорт вимірювань із CSV-файлу з автоматичним розпізнаванням стовпців

– Валідація введених показників (фізіологічні межі та логічне співвідношення систолічного та діастолічного тиску)

– Перегляд таблиці вимірювань з фільтрацією за пристроєм та датою

– Інтерактивні графіки динаміки показників із коридором норми та ковзним середнім

– Описова статистика та автоматичне виявлення аномалій

– Система сповіщень про відхилення (info / warning / critical) з прив'язкою до конкретного вимірювання

Нефункціональні:

– Кросплатформність: Windows, macOS, Linux

– Час відгуку на запит не більше 2 с при обсязі 10 000 записів

– Зберігання паролів у вигляді bcrypt-хешу (cost factor 12)

– Можливість горизонтального масштабування через підключення PostgreSQL

– Конфігурація через змінні середовища (.env), без жорстко закодованих параметрів

2.2.3 Нефункціональні вимоги

До нефункціональних вимог належать характеристики якості, що визначають, як саме система виконує свої функції. Вимога до продуктивності встановлює верхню межу часу відгуку у 2 секунди при обробці до 10 000 записів в таблиці аналітики — цей показник перевірявся в процесі тестування. Вимога до безпеки передбачає хешування паролів bcrypt з cost factor 12, що відповідає сучасним рекомендаціям OWASP. Вимога до розширюваності полягає у можливості додати нові типи показників (наприклад, кортизол або рівень феритину) без переписування існуючого коду — завдяки модульній архітектурі та конфігурованим нормам у settings.py.

Вимога до надійності охоплює транзакційну цілісність операцій із базою даних: при помилці під час масового імпорту CSV вся операція відкочується, і база не залишається у проміжному стані. Вимога до підтримуваності стосується якості коду: всі публічні функції задокументовані, архітектура поділена на чіткі шари (моделі, сервіси, сторінки), що спрощує подальшу розробку та тестування.

2.2.4 Вимоги до інформації, що має оброблятися та зберігатися

Система обробляє та зберігає два основних типи інформації. По-перше, це облікові дані користувачів: ім'я для входу, електронна пошта, хеш пароля, опційне повне ім'я та рік народження. Рік народження введений як поле з перспективою персоналізованих норм: відомо, що нормальний рівень артеріального тиску дещо відрізняється для різних вікових груп. По-друге, це медичні вимірювання, прив'язані до конкретного пристрою та користувача.

Перелік медичних показників, що підлягають обробці, наведено в таблиці 2.2. Для кожного показника визначено нормативний діапазон на основі рекомендацій ВООЗ, Американської асоціації кардіологів (AHA) та актуальних клінічних настанов. Порогові значення зберігаються у конфігураційному файлі settings.py і можуть бути перевизначені через змінні середовища без зміни коду застосунку.

Таблиця 2.2 – Медичні норми за замовчуванням

Показник	**Мін. норма**	**Макс. норма**	**Одиниця / джерело порогів**
Пульс	60	100	уд/хв (ВООЗ, AHA)
Систолічний тиск	90	120	мм рт. ст.
Діастолічний тиск	60	80	мм рт. ст.
SpO_2	95	—	% (нижня межа, ATS 2022)
Глюкоза крові	3,9	7,8	ммоль/л (натще та після їжі)
Температура тіла	36,0	37,5	°C

Кожне вимірювання також зберігає метадані: час проведення вимірювання з часовим поясом (TIMESTAMPTZ), джерело введення (manual, csv або api), прив'язку до пристрою та довільні нотатки. Ці метадані є важливими для клінічної інтерпретації: вимірювання тиску вранці та після фізичного навантаження можуть значно відрізнятись, тому часова мітка дозволяє враховувати контекст.

2.3 Вибір технологій реалізації

Вибір технологічного стеку здійснювався на основі кількох критеріїв: відповідність поставленим вимогам, зрілість і активна підтримка спільноти, відкрита ліцензія, а також особистий досвід роботи з відповідними технологіями. Нижче обґрунтовується кожне рішення.

Python 3.11 залишається основною мовою програмування — стандартом де-факто в галузі аналізу даних. Ця версія демонструє підвищення

продуктивності на 10–60% порівняно з Python 3.10 завдяки оптимізації інтерпретатора CPython. Читабельний синтаксис, динамічна типізація та багатюща екосистема бібліотек перекривають усі потреби проєкту.

Streamlit обрано як фреймворк для побудови веб-інтерфейсу. Він дозволяє будувати інтерактивний UI безпосередньо з Python-коду без необхідності знань HTML/CSS/JavaScript, що принципово важливо для проєкту, де пріоритетом є швидкість розробки і можливість зосередитись на аналітичній логіці, а не на верстці. У таблиці 2.3 наведено порівняння Streamlit з альтернативними підходами.

Таблиця 2.3 – Порівняння фреймворків для побудови веб-інтерфейсів аналізу даних

Критерій	**Streamlit**	**Dash (Plotly)**	**Flask + Jinja**	**Jupyter Voilà**
Швидкість розробки	Висока	Середня	Низька	Середня
Знання HTML/CSS	Не потрібні	Мінімальні	Обов'язкові	Не потрібні
Інтерактивність	Вбудована	Вбудована	Потребує JS	Обмежена
Поріг входу	Низький	Середній	Високий	Низький
Масштабованість	Середня	Висока	Висока	Низька
Ліцензія	Apache 2.0	MIT	BSD	BSD

PostgreSQL обрано як СУБД замість SQLite, що використовувалась у попередній версії системи. Це рішення обумовлено кількома міркуваннями. По-перше, PostgreSQL підтримує повноцінний ENUM-тип, використаний для класифікації пристроїв (DeviceType) та рівнів сповіщень (AlertSeverity), що забезпечує цілісність даних на рівні бази. По-друге, PostgreSQL підтримує TIMESTAMPTZ — тип часової мітки з часовим поясом, що критично важливо при роботі з медичними даними, де точний час вимірювання має клінічне значення. По-третє, PostgreSQL дозволяє масштабуватись до

багатокористувацького використання без зміни архітектури застосунку. Порівняння СУБД наведено у таблиці 2.4.

Таблиця 2.4 – Порівняльний аналіз СУБД

Критерій	**PostgreSQL**	**SQLite**	**MySQL**
Архітектура	Клієнт-сервер	Вбудована	Клієнт-сервер
Ліцензія	PostgreSQL (відкрита)	Public Domain	GPL / комерційна
Паралельні записи	MVCC, повна підтримка	Блокування таблиці	InnoDB MVCC
Типи даних	Розширені (JSON, масиви, UUID, ENUM)	Базові (TEXT, INTEGER, REAL)	Стандартні + JSON
Розгортання	Потребує сервісу	Файл — без сервісу	Потребує сервісу
Вибір для MedData	**Обрано**	—	—

SQLAlchemy 2.x використовується як ORM-шар для роботи з базою даних. Він забезпечує декілька переваг: по-перше, ORM-декларації моделей (User, Device, Measurement, Alert) є одночасно Python-класами і описом схеми бази даних — це єдине джерело правди для структури даних; по-друге, SQLAlchemy автоматично керує пулом з'єднань (pool_size=5, max_overflow=10), що важливо для Streamlit, де кожна взаємодія користувача може ініціювати новий запит до бази; по-третє, контекстний менеджер get_db() забезпечує автоматичний rollback при виключеннях та commit при успішному виконанні.

pydantic-settings використовується для централізованого управління конфігурацією. Клас Settings автоматично зчитує значення з файлу .env та змінних середовища, перевіряє їх типи та надає єдиний об'єкт settings для всього застосунку. bcrypt обрано для хешування паролів замість SHA-256, що використовувався в попередній версії. SHA-256 є криптографічно стійкою функцією, але вона спроєктована для швидких обчислень, що робить її

вразливою до атак перебором. bcrypt навпаки спроєктований для повільних обчислень з регульованим cost factor: при значенні 12 обчислення одного хешу займає близько 250–500 мс, що відповідає рекомендаціям OWASP Password Storage Cheat Sheet.

2.4 Архітектура системи

2.4.1 Загальна структура проєкту

Система MedData побудована за багатошаровою архітектурою з чітким розподілом відповідальностей між шарами. Такий підхід відповідає принципам SOLID та спрощує тестування і супровід системи. Структура проєкту:

```
MedData/
├── app.py                    # Точка входу
├── .env                      # Конфігурація середовища
├── requirements.txt
├── config/
│   ├── database.py           # Пул з'єднань, ORM base, init_db()
│   └── settings.py           # Конфігурація (pydantic-settings)
├── models/
│   ├── user.py               # ORM: User
│   ├── device.py             # ORM: Device, DeviceType
│   ├── measurement.py        # ORM: Measurement, MeasurementNorm
│   └── alert.py              # ORM: Alert, AlertSeverity
├── services/
│   ├── auth_service.py       # Реєстрація, вхід, bcrypt
│   ├── device_service.py     # CRUD пристроїв
│   ├── measurement_service.py
│   ├── analytics_service.py  # Статистика, тренди, аномалії
│   └── alert_service.py      # Генерація сповіщень
└── my_pages/
    ├── auth_page.py
    ├── home_page.py
    ├── upload_page.py
    └── analytics_page.py
```

Таблиця 2.5 описує відповідальність кожного модуля та його залежності.

Таблиця 2.5 – Модулі системи MedData та їх відповідальності

Модуль	**Відповідальність**	**Ключові функції**	**Залежності**
app.py	Точка входу, ініціалізація БД, навігація	main(), CSS-стилі, роутинг	config.*, my_pages.*
config/database.py	Пул з'єднань SQLAlchemy, сесія БД	get_db(), init_db()	SQLAlchemy
config/settings.py	Конфігурація з .env	Settings (BaseSettings)	pydantic-settings
models/	ORM-декларації таблиць	User, Device, Measurement, Alert	SQLAlchemy ORM
services/auth_service.py	Реєстрація, вхід, зміна паролю	register_user(), login_user()	bcrypt, models/user
services/measurement_service.py	Збереження, валідація, CSV-імпорт	validate_measurement(), import_csv()	pandas, models
services/analytics_service.py	Статистика, тренди, аномалії	describe(), linear_trend()	pandas, numpy
services/alert_service.py	Генерація сповіщень	check_and_create_alerts()	models/alert
my_pages/	Streamlit-сторінки (UI)	show_auth/home/upload/analytics()	services.*, streamlit

2.4.2 Опис архітектурних шарів

Перший шар — конфігурація (config/) — відповідає за налаштування середовища та управління з'єднаннями з базою даних. Модуль settings.py визначає клас Settings на базі BaseSettings із pydantic-settings. Усі параметри —

адреса бази, секретний ключ, норми показників — читаються з файлу .env або змінних середовища. Модуль database.py ініціалізує SQLAlchemy engine з пулом з'єднань та декларативну базу Base, від якої успадковуються всі ORM-моделі. Контекстний менеджер get_db() відповідає за отримання сесії, commit при успіху та rollback при помилці.

Другий шар — моделі даних (models/) — містить ORM-декларації чотирьох основних сутностей системи. Кожна модель є Python-класом, що успадковується від Base і описує відповідну таблицю бази даних. Відносини між моделями визначені через SQLAlchemy relationship з відповідними cascade-правилами: видалення користувача автоматично видаляє всі його пристрої та вимірювання (cascade='all, delete-orphan').

Третій шар — сервіси (services/) — реалізує бізнес-логіку системи. Кожен сервіс приймає сесію бази даних (db: Session) як перший аргумент і не містить жодного Streamlit-коду — це забезпечує незалежність бізнес-логіки від веб-фреймворку. Завдяки такому розподілу сервіси можна тестувати за допомогою pytest без запуску Streamlit-сервера.

Четвертий шар — сторінки (my_pages/) — містить Streamlit-компоненти, що відповідають за відображення інтерфейсу. Кожна сторінка використовує сервіси через контекстний менеджер with get_db() as db і не містить SQL-запитів безпосередньо. Перетин між шарами відбувається виключно у визначеному напрямку: сторінки → сервіси → моделі → база даних. Зворотного зв'язку немає — сервіси не знають про існування Streamlit, а моделі не залежать від бізнес-логіки сервісів.

2.5 Проєктування бази даних

2.5.1 Загальна структура схеми

База даних системи MedData містить п'ять таблиць: users, devices, measurements, measurement_norms та alerts. Схема спроєктована в третій нормальній формі (3НФ): кожен атрибут залежить лише від первинного ключа, а не від інших неключових атрибутів. Це усуває надлишковість даних та запобігає аномаліям при оновленні.

Центральною таблицею є users. Від неї залежать devices (один користувач — багато пристроїв) та measurements (один користувач — багато вимірювань). Таблиця devices пов'язана з measurement_norms відношенням один-до-одного: кожен пристрій може мати свій набір персоналізованих нормативних меж, що перекривають глобальні налаштування з settings.py. Таблиця alerts прив'язана як до users, так і до measurements.

2.5.2 Опис таблиць

Таблиця 2.6 – Структура таблиці users:

id (BIGSERIAL, PK) — Сурогатний ключ.

username (VARCHAR(64), UNIQUE, NOT NULL, INDEX) — Унікальне ім'я для входу.

email (VARCHAR(255), UNIQUE, INDEX) — Електронна пошта (необов'язково).

password_hash (VARCHAR(255), NOT NULL) — bcrypt-хеш пароля (cost 12).

full_name (VARCHAR(255)) — Повне ім'я.

birth_year (SMALLINT) — Рік народження (для персоналізованих норм).

is_active (BOOLEAN, NOT NULL, DEFAULT TRUE) — Статус облікового запису.

created_at (TIMESTAMPTZ, NOT NULL, DEFAULT NOW()) — Дата реєстрації.

Таблиця devices зберігає відомості про зареєстровані медичні пристрої користувача. Кожен пристрій характеризується типом (device_type), визначеним через перелічуваний тип DeviceType (tonometer, glucometer, pulse_oximeter, thermometer, fitness_tracker, smart_scales, other), та довільною назвою, яку задає сам користувач. Можливість задати виробника, модель та серійний номер дозволяє однозначно ідентифікувати конкретний прилад.

Таблиця 2.7 – Структура таблиці user_devices:

id (BIGSERIAL, PK) — Ідентифікатор пристрою.

user_id (BIGINT, FK→users.id CASCADE) — Власник пристрою.

device_type (VARCHAR(64), NOT NULL) — Тип: tonometer, glucometer, pulse_oximeter, thermometer, fitness_tracker, smart_scales, other.

name (VARCHAR(128), NOT NULL) — Довільна назва (задана користувачем).

manufacturer (VARCHAR(128)) — Виробник.

model (VARCHAR(128)) — Модель пристрою.

serial_number (VARCHAR(128)) — Серійний номер.

is_active (BOOLEAN, NOT NULL, DEFAULT TRUE) — Логічне видалення пристрою.

Таблиця measurements є центральним сховищем медичних даних. Ключове архітектурне рішення — розміщення всіх типів показників в одній таблиці (горизонтальне розширення), а не в окремих таблицях для кожного пристрою. Це спрощує аналітичні запити. Поля показників (pulse, pressure_sys, spo2 тощо) дозволяють NULL — для конкретного вимірювання заповнюються лише релевантні поля. Поле source ('manual', 'csv', 'api') дозволяє відстежувати походження кожного запису.

Таблиця 2.8 – Структура таблиці measurements:

id (BIGSERIAL, PK) — Ідентифікатор запису.

user_id (BIGINT, FK→users.id CASCADE, INDEX) — Власник вимірювання.

device_id (BIGINT, FK→devices.id SET NULL) — Пристрій (може бути NULL).

measured_at (TIMESTAMPTZ, NOT NULL, INDEX) — Час вимірювання (з часовим поясом).

source (VARCHAR(32), DEFAULT 'manual') — Джерело: manual | csv | api.

pulse (INTEGER) — Пульс (уд/хв).

pressure_sys (INTEGER) — Систолічний тиск (мм рт. ст.).

pressure_dia (INTEGER) — Діастолічний тиск (мм рт. ст.).

spo2 (FLOAT) — SpO_2, насиченість крові (%).

glucose (FLOAT) — Рівень глюкози (ммоль/л).

temperature (FLOAT) — Температура тіла (°C).

steps (INTEGER) — Кількість кроків (фітнес-трекер).

calories (FLOAT) — Спалені калорії (ккал).

notes (VARCHAR(1000)) — Довільні примітки.

Таблиця alerts автоматично наповнюється сервісом alert_service.py при кожному збереженні вимірювання, якщо хоча б один показник виходить за межі норми. Рівень критичності (severity) визначається відхиленням від норми: якщо відхилення перевищує 20% від порогового значення, сповіщення отримує статус critical, інакше — warning. Поле is_read дозволяє відображати тільки непрочитані сповіщення на головній сторінці.

Таблиця 2.9 – Структура таблиці alerts:

id (SERIAL, PK) — Ідентифікатор сповіщення.

user_id (BIGINT, FK→users.id CASCADE, INDEX) — Адресат сповіщення.

measurement_id (BIGINT, FK→measurements.id SET NULL) — Вимірювання, що спричинило сповіщення.

metric (VARCHAR(64), NOT NULL) — Код показника (pulse, glucose тощо).

value (FLOAT, NOT NULL) — Зафіксоване значення.

severity (ENUM, NOT NULL) — Рівень: info | warning | critical.

message (TEXT, NOT NULL) — Текст для відображення.

is_read (BOOLEAN, DEFAULT FALSE) — Статус прочитання.

2.5.3 Індексування та продуктивність

Для забезпечення необхідної продуктивності при обробці 10 000 і більше записів у таблиці measurements визначено ряд індексів. По-перше, поле user_id індексується, оскільки найчастіший запит — «отримати всі вимірювання поточного користувача» — завжди фільтрує саме за цим полем. По-друге, поле

measured_at також індексується, що прискорює запити з фільтрацією або сортуванням за часом. По-третє, поле device_id індексується для прискорення фільтрації за конкретним пристроєм.

Складений індекс (user_id, measured_at) може бути доданий для оптимізації найпоширенішого запиту — «всі вимірювання користувача, відсортовані за часом»:

```
CREATE INDEX IF NOT EXISTS idx_measurements_user_time
ON measurements (user_id, measured_at DESC);
```

Це перетворює складність запиту з O(n) при послідовному скануванні на O(log n) при використанні індексу, що при мільйоні записів дає прискорення у сотні разів.

2.6 Алгоритми обробки даних

2.6.1 Алгоритм валідації вимірювань

Перед збереженням будь-якого вимірювання виконується функція validate_measurement() з модуля measurement_service.py. Вона перевіряє кожен переданий показник на відповідність фізіологічно реалістичним межам (VALID_RANGES) та додатково перевіряє логічне співвідношення між систолічним і діастолічним тиском. Функція повертає кортеж (bool, str): True і порожній рядок при успіху, False і повідомлення про помилку при невалідних даних.

Для CSV-імпорту реалізовано механізм автоматичного розпізнавання назв стовпців через словник CSV_COLUMN_ALIASES. Система підтримує різні варіанти назв: наприклад, стовпець з пульсом може мати назву 'pulse', 'heart_rate', 'hr', 'пульс' або 'частота пульсу'. Це усуває потребу у попередньому форматуванні CSV-файлу.

2.6.2 Алгоритм генерації сповіщень

Після збереження кожного вимірювання сервіс alert_service.py перевіряє значення всіх показників відносно нормативних меж. Якщо для пристрою визначено персоналізовані норми (таблиця measurement_norms), вони мають пріоритет над глобальними значеннями з settings.py. Рівень критичності

визначається функцією _severity(), яка розраховує відносне відхилення від порогового значення: відхилення понад 20% дає рівень critical, менше — warning.

Алгоритм обчислення рівня критичності для показника з виміряним значенням value і нормативними межами [lo, hi]:

```
if value < lo: deviation = |value - lo| / |lo|
if value > hi: deviation = |value - hi| / |hi|
severity = 'critical' if deviation > 0.20 else 'warning'
```

Такий підхід є більш інформативним, ніж проста бінарна класифікація «норма/відхилення»: пацієнт бачить, наскільки критичним є відхилення і чи потрібна термінова консультація лікаря.

2.6.3 Аналітичні алгоритми

Аналітичний сервіс (analytics_service.py) реалізує кілька алгоритмів для інтерпретації накопичених даних. Функція describe() обчислює описову статистику (кількість, середнє, стандартне відхилення, мінімум, перцентилі, максимум) для всіх наявних числових показників через метод DataFrame.describe().

Функція linear_trend() визначає напрям динаміки показника: зростання, спадання або стабільність. Вона застосовує метод найменших квадратів через numpy.polyfit() для рядів з мінімум 3 спостереженнями. Порогом «стабільності» є 5% від стандартного відхилення ряду. Результат — словник зі значеннями slope і direction, що відображається у вигляді стрілок (↑/↓/→) на аналітичній сторінці.

Функція moving_average() обчислює ковзне середнє з вибраним вікном (за замовчуванням 7 вимірювань) через pandas rolling(). Це дозволяє згладити короткочасні коливання і виявити довгострокові тенденції у динаміці показника. Результат відображається як окрема лінія на графіку поряд із вихідними даними.

Висновок до розділу 2

У другому розділі сформульовано вимоги до системи MedData та прийнято основні проєктувальні рішення. Загальні вимоги визначають доступність системи через браузер, кросплатформність та ізоляцію персональних даних кожного користувача. Функціональні вимоги охоплюють повний цикл роботи з медичними даними: від реєстрації пристроїв та введення вимірювань до

аналітики та системи сповіщень. Нефункціональні вимоги встановлюють конкретні метрики продуктивності, безпеки та розширюваності.

Обґрунтовано вибір технологічного стеку: Python 3.11, Streamlit, PostgreSQL, SQLAlchemy 2.x, pydantic-settings, bcrypt, Pandas та Plotly. Порівняльний аналіз фреймворків та СУБД підтвердив доцільність прийнятих рішень з урахуванням вимог проєкту. Ключова відмінність нової версії системи від попередньої — перехід від SQLite до PostgreSQL, від SHA-256 до bcrypt та від монолітного модуля до розподіленої сервісної архітектури з ORM-шаром.

Розроблено багатошарову архітектуру застосунку: шари конфігурації, ORM-моделей, бізнес-логіки (сервіси) та веб-представлення (Streamlit-сторінки) чітко відокремлені один від одного. Спроєктовано нормалізовану схему бази даних з п'яти таблиць, описано систему індексів для забезпечення продуктивності та алгоритми обробки даних — валідації, генерації сповіщень і аналітики.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ MEDDATA

3.1 Налаштування середовища розробки

Розробка системи MedData велась на операційній системі macOS з використанням Python 3.13 та PostgreSQL 16. Для ізоляції залежностей проєкту застосовувалось віртуальне середовище venv [40], що унеможливлює конфлікти між версіями бібліотек різних проєктів. Репозиторій проєкту розміщено на платформі GitLab [39].

Для розгортання проєкту необхідно клонувати репозиторій, налаштувати файл .env з параметрами підключення до бази даних та виконати дві команди у терміналі:

```
git init && git remote add origin https://gitlab.com/oleksandra-skorokhod/meddata.git
# Вміст файлу .env:
DATABASE_URL=postgresql://meddata_user:meddata_pass@localhost:5432/meddata_db
SECRET_KEY=your-secret-key
# Встановлення залежностей та запуск:
pip install -r requirements.txt
streamlit run app.py
```

Ключовою особливістю розгортання є автоматична ініціалізація схеми бази даних. Функція init_db() використовує метод Base.metadata.create_all(bind=engine) з SQLAlchemy, який при першому запуску застосунку автоматично створює всі таблиці на основі ORM-моделей. Конструкція CREATE TABLE IF NOT EXISTS (генерована SQLAlchemy) забезпечує ідемпотентність: повторні запуски не руйнують існуючі дані.

Серед основних залежностей, зафіксованих у requirements.txt: streamlit>=1.34.0, sqlalchemy>=2.0.0, psycopg2-binary>=2.9.0, pandas>=2.2.0, plotly>=5.22.0, bcrypt>=4.1.0, pydantic-settings>=2.2.0, numpy>=1.26.0. Використання psycopg2-binary замість psycopg2 спрощує встановлення без необхідності компіляції C-розширень. Зафіксовані мінімальні версії гарантують сумісність API, але дозволяють автоматичне оновлення безпекових патчів.

Структура проєкту реалізує принцип єдиної відповідальності (SRP) з шаблонів SOLID [37]: шар config/ відповідає за налаштування, models/ – за схему

бази даних, services/ – за бізнес-логіку, my_pages/ – за відображення. Такий розподіл дозволяє змінювати один шар без впливу на інші та спрощує написання модульних тестів.

3.2 Реалізація модуля авторизації

Модуль auth_page.py реалізує повний цикл управління обліковими записами: вхід в систему, реєстрацію нового користувача та відновлення пароля. Інтерфейс побудовано на компоненті st.tabs(['Увійти', 'Зареєструватись']), що дозволяє користувачеві перемикатися між формами без перезавантаження сторінки.

Механізм захисту реалізовано через перевірку st.session_state у головному файлі app.py. При відсутності ключа user_id у стані сесії застосунок викликає функцію show_auth() та негайно зупиняє виконання скрипту командою st.stop(). Це є ключовим механізмом безпеки: жоден захищений компонент не відрендериться для неавторизованих користувачів, оскільки виконання Python-скрипту фізично зупиняється:

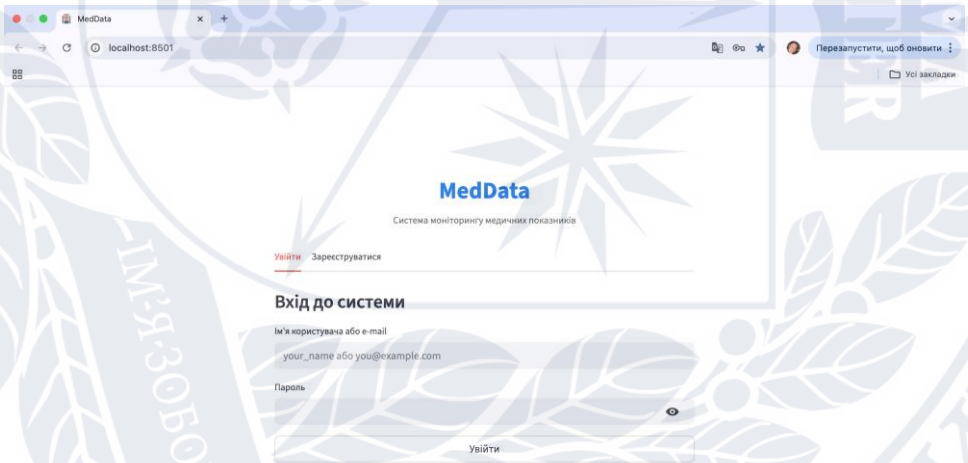

Рисунок 3.1 – Сторінка входу до системи MedData

```
if st.session_state.user_id is None:
    from my_pages.auth_page import show_auth
    show_auth()
    st.stop()   # жоден наступний рядок не виконається
```

Функція login_user() у сервісному шарі auth_service.py реалізує пошук користувача за ім'ям або електронною поштою (SQLAlchemy | оператор для

побудови OR-умови) та перевірку пароля через bcrypt.checkpw(). При успішній автентифікації у st.session_state записуються user_id та username, після чого виконується st.rerun() для оновлення сторінки.

Функція hash_password() застосовує bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt(rounds=12)). Параметр rounds=12 означає 2^{12} ітерацій хешування, що забезпечує час обчислення близько 250–500 мс на сучасному обладнанні [27]. Кожен виклик gensalt() генерує унікальну 22-символьну сіль, тому два однакові паролі дають різні хеші. Це принципово відрізняє bcrypt від SHA-256, де однаковий вхід завжди дає однаковий вихід [26].

Форма реєстрації виконує клієнтську валідацію до звернення до бази даних: перевірку мінімальної довжини імені (≥ 3 символів), формату email (наявність '@' та '.'), мінімальної довжини пароля (≥ 6 символів) та збігу обох полів пароля. Серверна перевірка унікальності імені виконується через обробку виключення IntegrityError від SQLAlchemy при порушенні UNIQUE-обмеження. Бічна панель навігації відображається лише для авторизованих користувачів та містить логотип системи, ім'я поточного користувача і кнопку виходу.

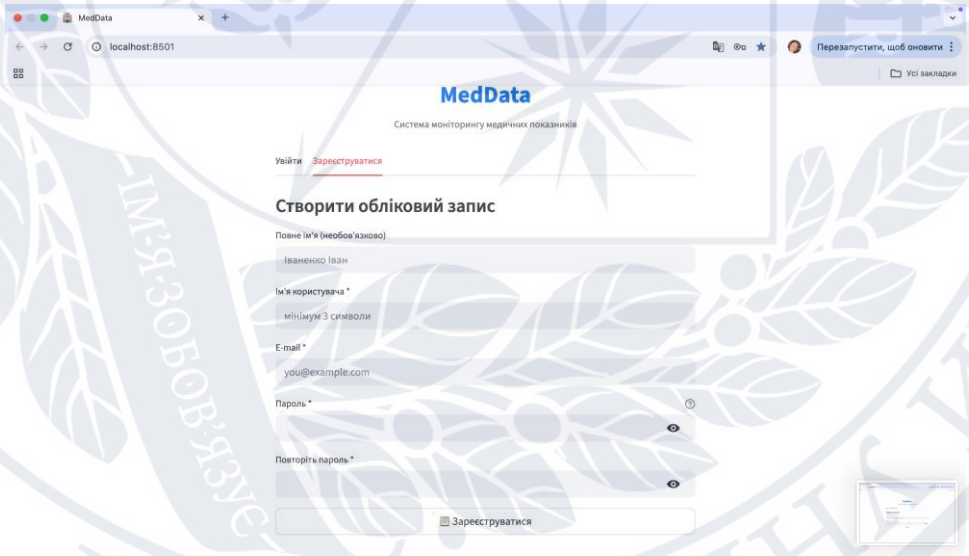

Рисунок 3.2 – Форма реєстрації нового облікового запису

3.3 Реалізація головної сторінки

Модуль home_page.py є першим екраном, який бачить користувач після входу в систему. Його призначення – надати миттєвий огляд поточного стану

здоров'я на основі останнього вимірювання та зведену статистику по всій вибірці. Завантаження даних відбувається на початку функції show_home(): викликаються get_measurements_df(), get_norm() та get_unread_alerts(). Використання @st.cache_data запобігає повторним запитам до бази даних при кожній взаємодії користувача.

Логіка відображення індикаторів стану показників використовує словник NORMS як єдине джерело граничних значень. Ітерація по словнику забезпечує компактний код без дублювання логіки перевірки. HTML-рядки з CSS-класами формуються через f-string та відображаються через st.markdown(unsafe_allow_html=True):

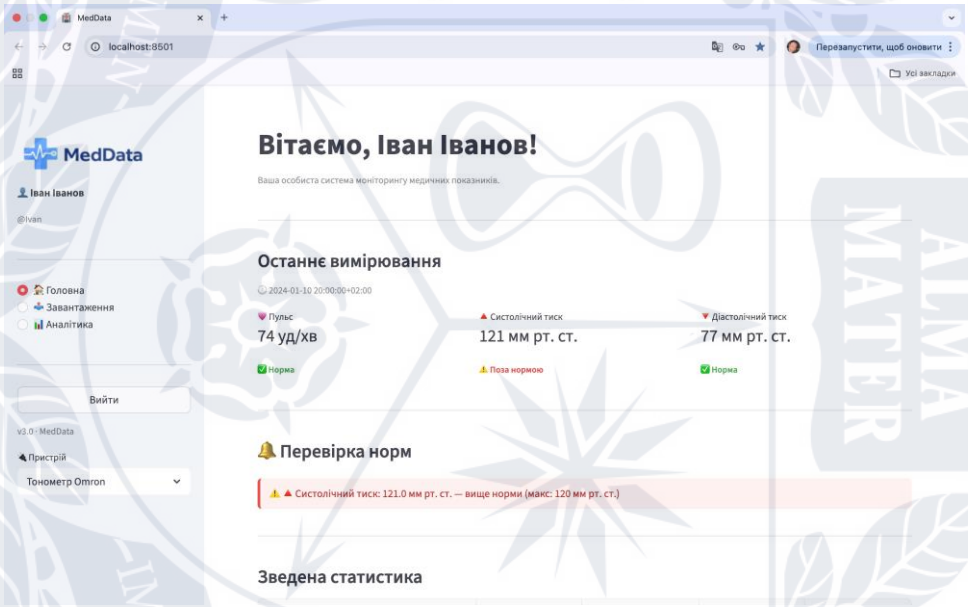

Рисунок 3.3 – Головна сторінка: останнє вимірювання та перевірка норм

```
NORMS = {
    'pulse':        (60, 100, 'Пульс',              'уд/хв'),
    'pressure_sys': (90, 120, 'Систолічний тиск', 'мм рт. ст.'),
    'pressure_dia': (60, 80,  'Діастолічний тиск','мм рт. ст.'),
}
for field, (lo, hi, label, unit) in NORMS.items():
    val = int(last[field])
    ok  = lo <= val <= hi
    cls = 'alert-ok' if ok else 'alert-high'
    ico = '' if ok else ''
    st.markdown(f'<div class="{cls}">{ico} <b>{label}:</b> {val}
{unit}</div>',
                unsafe_allow_html=True)
```

Зведена статистика формується з усього DataFrame одним зверненням до бази: count(), mean(), min(), max() обчислюються безпосередньо засобами Pandas без додаткових SQL-запитів. Показники відображаються у трьохколонковій сітці через st.columns(3) та st.metric(). Параметр delta_color='inverse' для тиску відображає зростання червоним, що відповідає медичній логіці: підвищення тиску є несприятливим сигналом. Блок непрочитаних сповіщень реалізовано через st.expander з кількістю у заголовку; warning відображаються помаранчевим, critical – червоним.

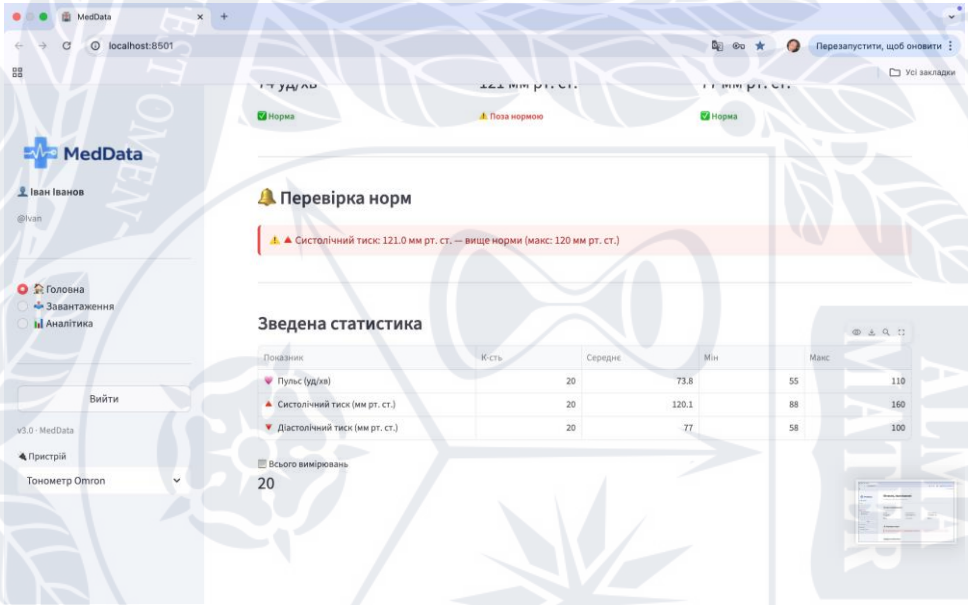

Рисунок 3.4 – Зведена статистика вимірювань на головній сторінці

3.4 Реалізація модуля введення та імпорту даних

3.4.1 Завантаження CSV-файлу

Модуль upload_page.py реалізує чотири паралельні шляхи роботи з даними через компонент st.tabs: завантаження CSV-файлу, ручне введення показників, управління пристроями та очищення даних. Компонент st.file_uploader приймає файл з обмеженням на розширення ['csv']. Після вибору файлу він зчитується у DataFrame через pd.read_csv() безпосередньо з файлового об'єкта Streamlit без збереження тимчасового файлу на диску. Далі виконується нормалізація назв стовпців через словник CSV_COLUMN_ALIASES:

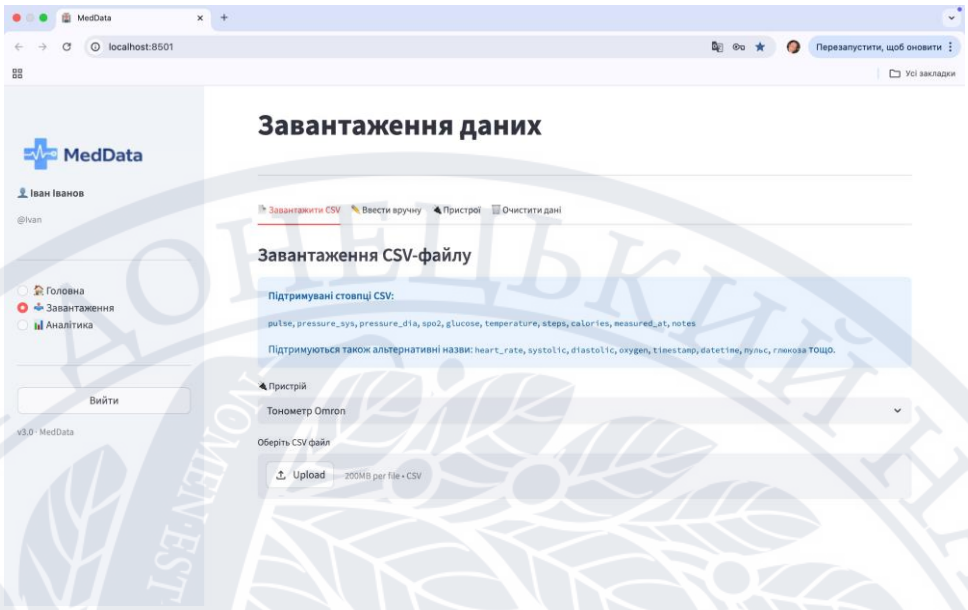

Рисунок 3.5 – Завантаження CSV-файлу: підтримувані назви стовпців та вибір пристрою

```
CSV_COLUMN_ALIASES = {
    'pulse': ['pulse', 'heart_rate', 'hr', 'пульс', 'частота пульсу'],
    'pressure_sys': ['pressure_sys', 'systolic', 'sys', 'сист', 'сист. тиск'],
    'pressure_dia': ['pressure_dia', 'diastolic', 'dia', 'діаст', 'діаст. тиск'],
    'measured_at':  ['timestamp', 'date', 'time', 'datetime', 'час', 'дата'],
}
```

Після нормалізації виконується рядкова валідація: кожен рядок DataFrame перевіряється функцією validate_measurement(). Рядки з помилками виводяться окремо; коректні рядки – у попередній перегляд таблиці st.dataframe() перед збереженням. Весь CSV-імпорт виконується в єдиній транзакції: при помилці відбувається rollback, що гарантує цілісність даних.

3.4.2 Ручне введення вимірювань

Форма ручного введення є динамічною: набір полів залежить від типу обраного пристрою через словник _DEVICE_FIELDS. Компоненти st.number_input() мають параметри min_value та max_value, що відповідають фізіологічно реалістичним межам. Поле дати та часу за замовчуванням містить поточний момент (datetime.now()), але може бути змінене для запізнілих записів.

Після натискання кнопки «Додати запис» виконуються: серверна валідація, збереження та автоматична генерація сповіщень.

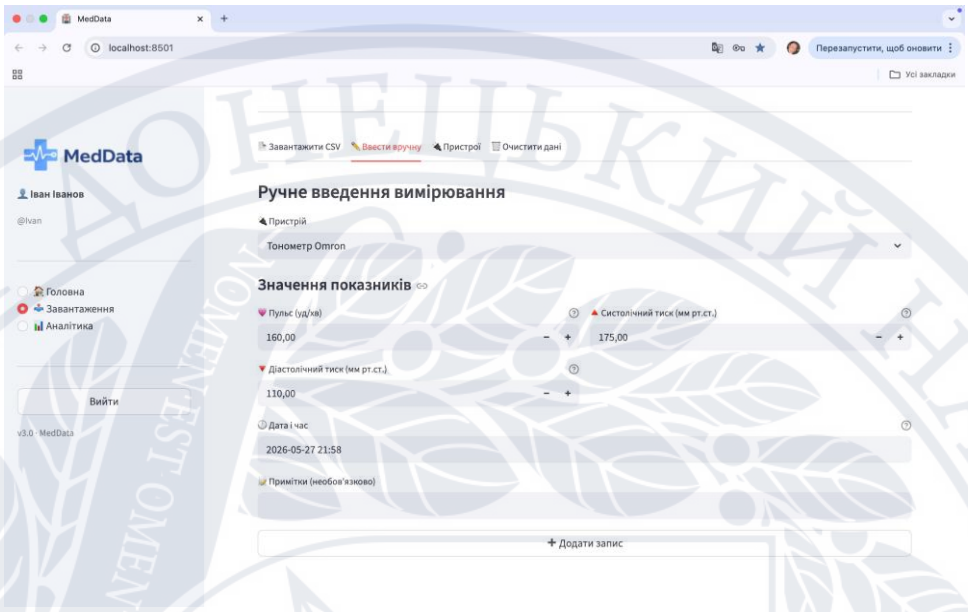

Рисунок 3.6 – Форма ручного введення показників вимірювання

3.4.3 Управління пристроями

Вкладка управління пристроями дозволяє реєструвати нові пристрої з вибором типу через DeviceType enum (tonometer, glucometer, pulse_oximeter, thermometer, fitness_tracker, smart_scales, other). Прапорець is_active реалізує шаблон soft delete: деактивований пристрій зникає зі спадних списків вибору, але всі прив'язані до нього вимірювання зберігаються в базі даних та залишаються доступними для аналітики. Це є принциповим рішенням для збереження медичної та аудиторської цілісності даних. Вкладка очищення даних вимагає введення імені користувача для підтвердження видалення – захист від випадкової незворотної операції.

3.5 Реалізація аналітичного модуля

3.5.1 Підготовка даних та описова статистика

Аналітичний модуль analytics_page.py є ключовим компонентом системи MedData. Функція show_analytics() починається з підготовки даних: перетворення поля measured_at у тип datetime64 через pd.to_datetime(),

сортування за часом та скидання індексу. Описова статистика обчислюється через DataFrame.describe().T, де T є операцією транспонування. Метод describe() повертає 8 статистичних характеристик по кожному числовому стовпцю:

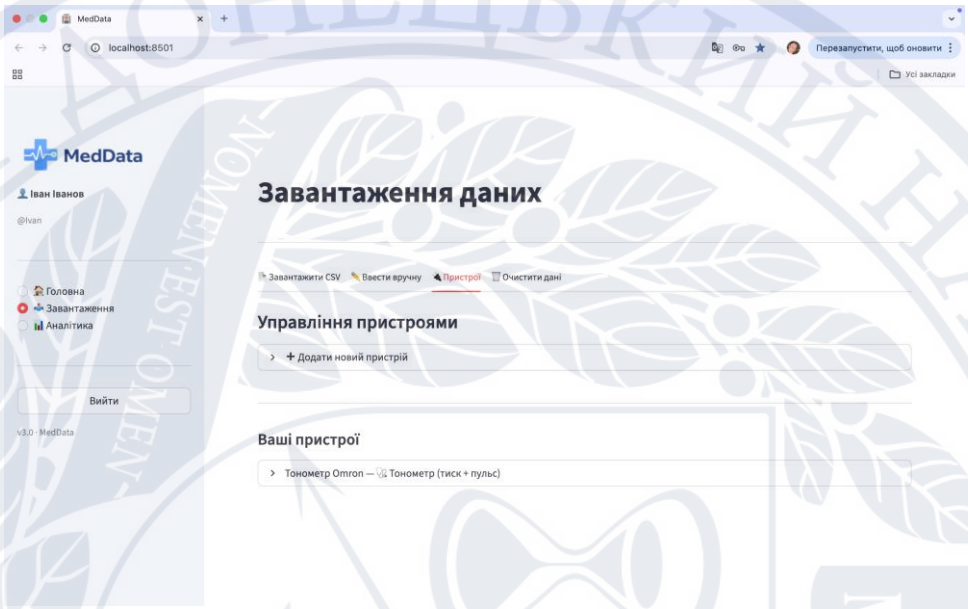

Рисунок 3.7 – Управління зареєстрованими медичними пристроями

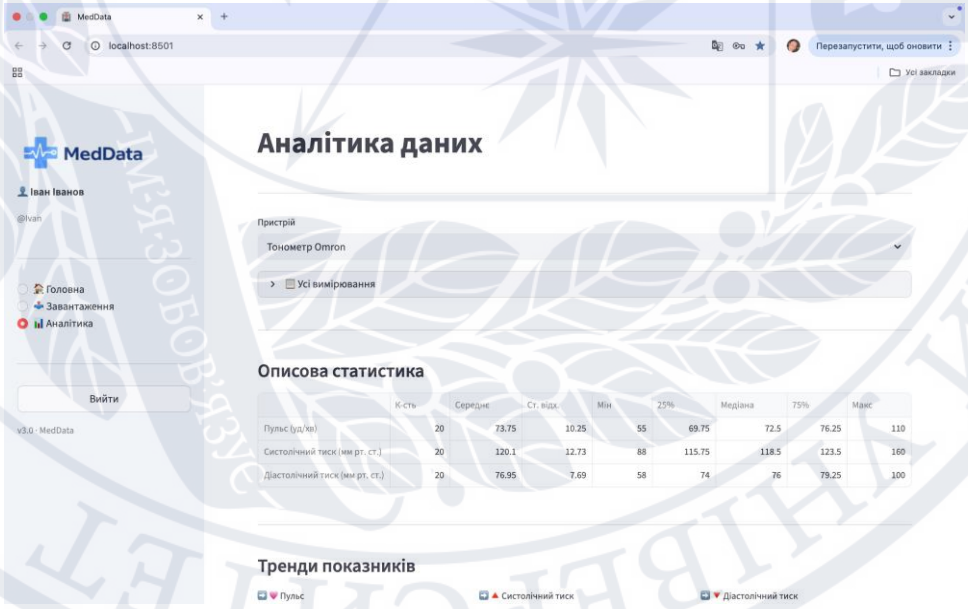

Рисунок 3.8 – Аналітична сторінка: описова статистика показників

```
stats = df[['pulse','pressure_sys','pressure_dia']].describe().T
stats.index = ['Пульс', 'Сист. тиск', 'Діаст. тиск']
```

```
stats.columns = ['К-сть','Середнє','Ст. відх.','Мін.',
                 'Q1 (25%)','Медіана','Q3 (75%)','Макс.']
st.dataframe(stats.round(1), use_container_width=True)
```

3.5.2 Тренди та лінійна регресія

Функція column_trends() обчислює напрям динаміки для кожного показника через linear_trend(). Для рядів з мінімум 3 спостереженнями застосовується метод найменших квадратів через numpy.polyfit(x, y, deg=1), де x – числовий індекс спостереження, y – значення показника. Поріг стабільності встановлений на рівні 5% від стандартного відхилення ряду. Результати відображаються як іконки напряму (↑/↓/→) разом із числовим значенням нахилу в одиницях на вимірювання. Для пульсу це може виглядати як «↓ –0.3 уд/хв на вимірювання», що сигналізує про поступове зниження частоти серцевих скорочень.

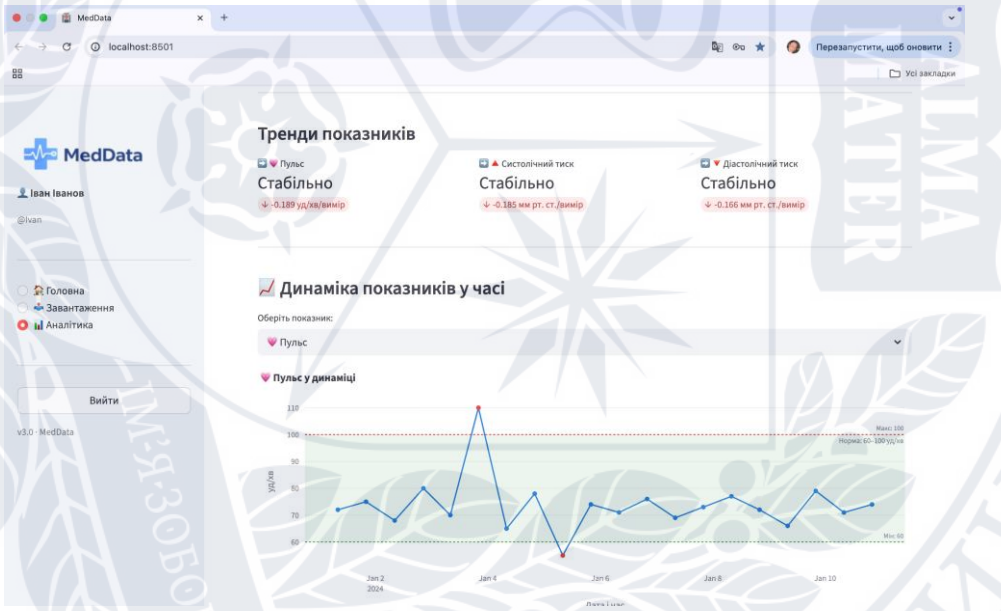

Рисунок 3.9 – Тренди показників: напрям динаміки за накопиченими даними

3.5.3 Побудова інтерактивних графіків

Графіки будуються засобами Plotly Graph Objects через допоміжну функцію build_chart(). Для кожного показника створюється об'єкт Figure з кількома шарами: лінія фактичних значень (go.Scatter з mode='lines+markers'), виділення аномальних точок червоним кольором, нормативний коридор

(add_hrect() з opacity=0.07) та межі норми (add_hline() з line_dash='dot'). Вибір показника для відображення реалізовано через спадний список (рисунок 3.10):

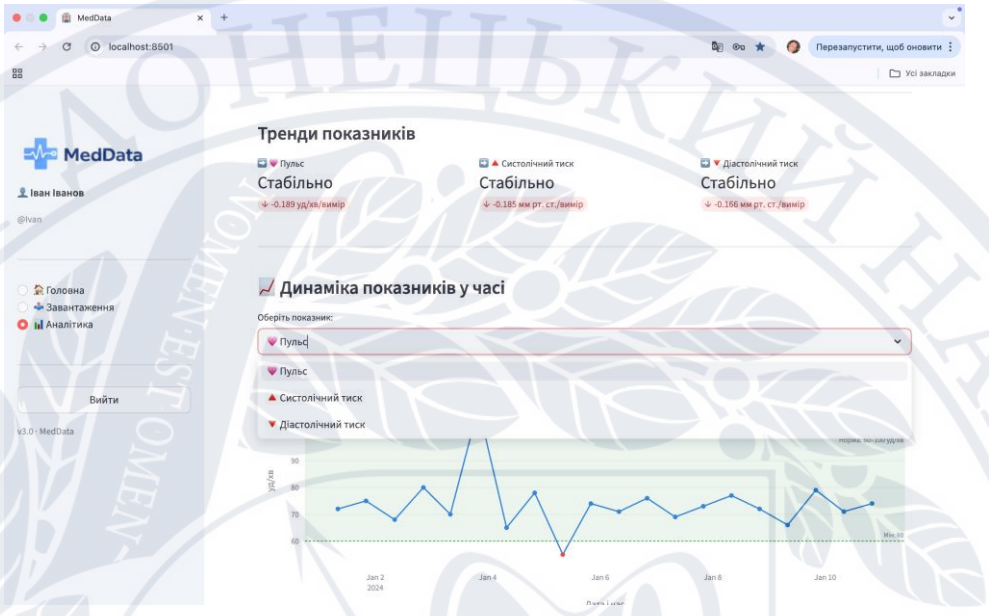

Рисунок 3.10 – Вибір показника для побудови графіка динаміки

```
def build_chart(df, col, title, y_min, y_max, color):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=df['measured_at'], y=df[col],
        mode='lines+markers', name=title, line=dict(color=color)))
    anom = df[(df[col] < y_min) | (df[col] > y_max)]
    fig.add_trace(go.Scatter(x=anom['measured_at'], y=anom[col],
        mode='markers', marker=dict(color='red', size=10),
name='Аномалія'))
    fig.add_hrect(y0=y_min, y1=y_max, fillcolor='green',
                  opacity=0.07, layer='below', line_width=0)
    fig.update_layout(hovermode='x unified')
    return fig
```

Параметр hovermode='x unified' об'єднує підказки всіх серій на спільній часовій позиції. Аномальні точки виділяються червоними маркерами розміром 10px. Для артеріального тиску реалізовано комбінований графік: систолічний тиск відображається синьою лінією, діастолічний – фіолетовою, що дозволяє оцінювати пульсовий тиск та виявляти ізольовану гіпертензію будь-якого типу.

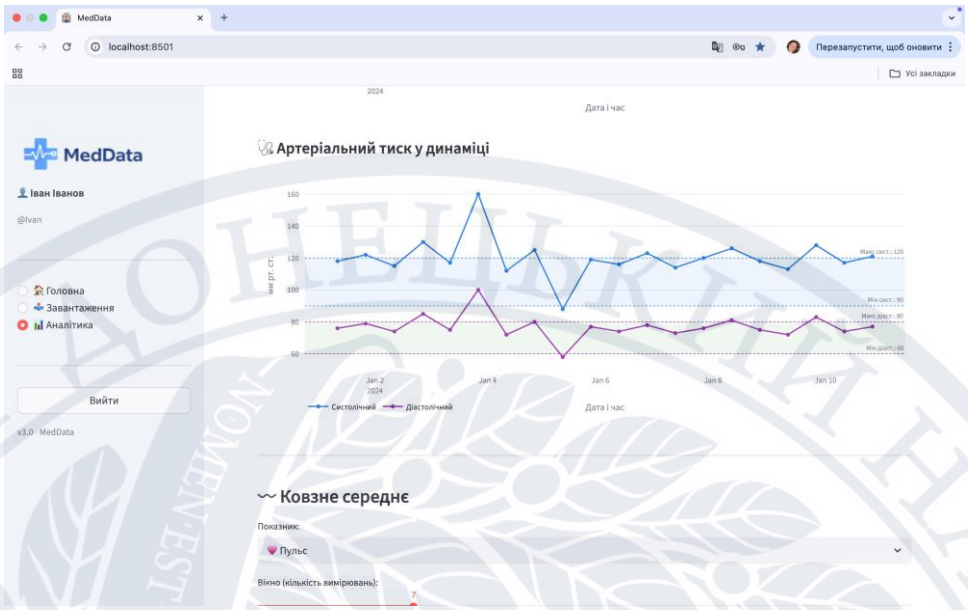

Рисунок 3.11 – Динаміка артеріального тиску у часі з двома нормативними коридорами

3.5.4 Ковзне середнє

Компонент st.slider() дозволяє динамічно вибирати розмір вікна ковзного середнього від 2 до 30 вимірювань. Обчислення виконується через pandas rolling(window=n, center=True, min_periods=1). Параметр center=True робить згладжування симетричним; min_periods=1 забезпечує обчислення навіть для крайніх точок ряду. На графіку оригінальні дані відображаються сірими напівпрозорими точками (opacity=0.4), а ковзне середнє – жовтогарячою лінією товщиною 3px. Ковзне середнє з вікном 7 вимірювань ефективно згладжує добові коливання та виявляє довгострокові тижневі тренди.

3.5.5 Виявлення аномалій та денна агрегація

Виявлення аномалій реалізовано через векторизовані булеві маски Pandas, що є принципово ефективнішим підходом порівняно з Python-циклом – операції виконуються на рівні NumPy-масивів з прискоренням 10–100x для великих DataFrame [29]:

```
pulse_ok = df['pulse'].between(60, 100)
sys_ok   = df['pressure_sys'].between(90, 120)
```

```
dia_ok   = df['pressure_dia'].between(60, 80)
anomalies = df[~(pulse_ok & sys_ok & dia_ok)].copy()
```

Метод Series.between(left, right) є зручним скороченням для умови з включними межами. Оператор ~ інвертує булеву маску. Результуюча таблиця аномалій виводиться через st.dataframe() з умовним форматуванням.

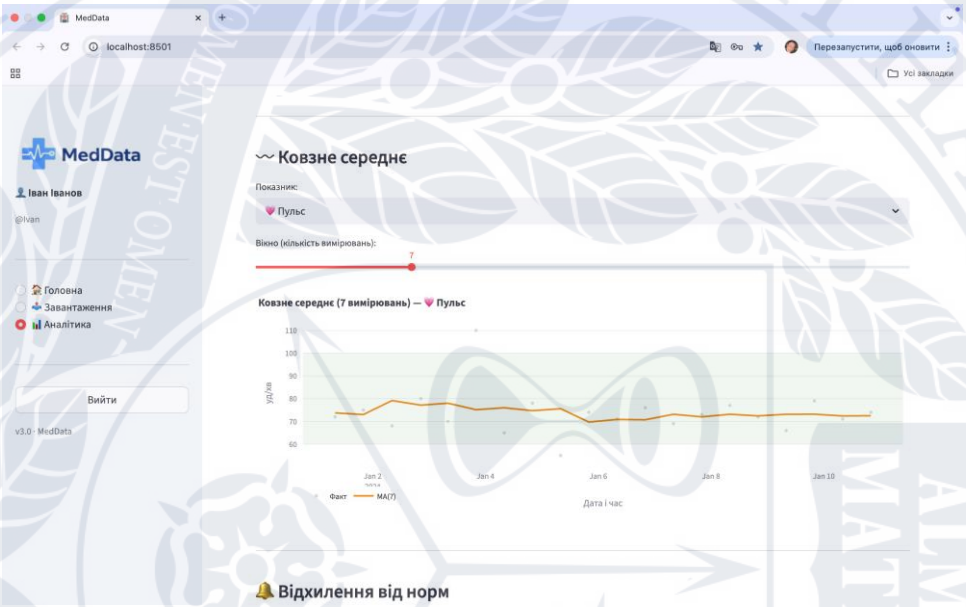

Рисунок 3.12 – Ковзне середнє пульсу (вікно 7 вимірювань)

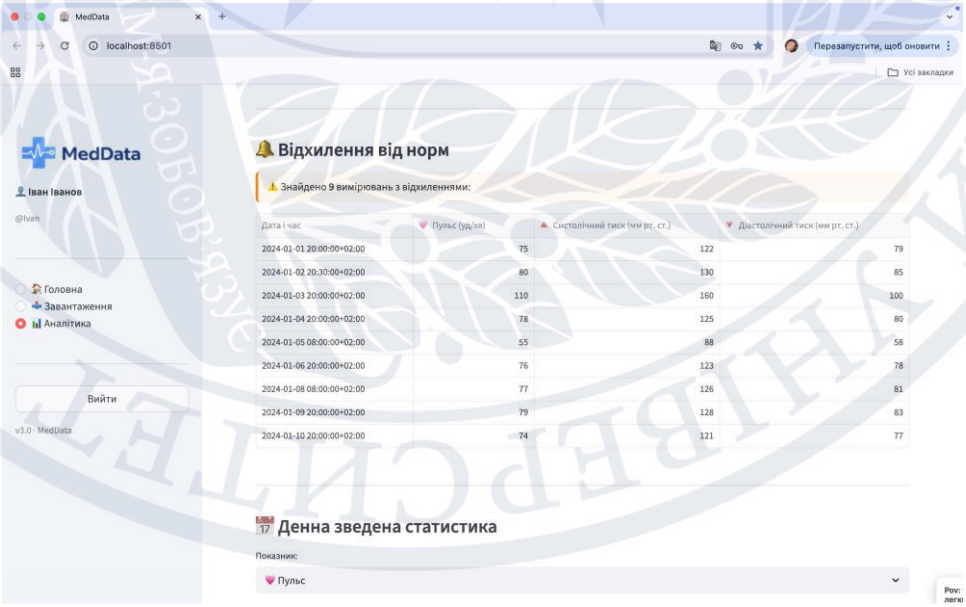

Рисунок 3.13 – Таблиця вимірювань з відхиленнями від нормативних значень

Функція daily_summary() реалізує денну агрегацію через groupby(df['measured_at'].dt.date).agg() – дозволяє отримати компактний огляд при двох вимірюваннях на день та виявляти дні з підвищеними показниками.

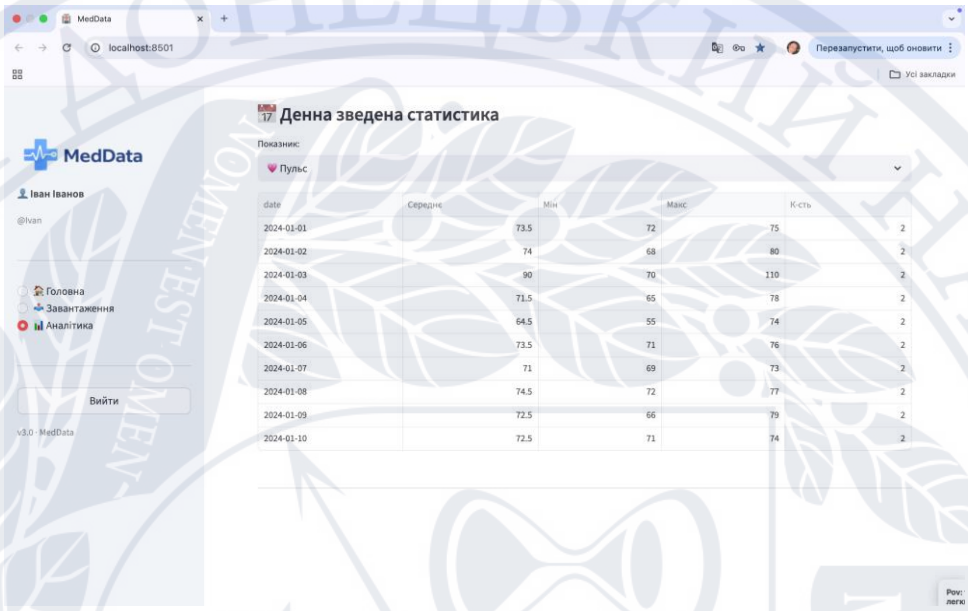

Рисунок 3.14 – Денна зведена статистика показника за обраний період

3.6 Реалізація системи сповіщень

Система сповіщень є проактивним компонентом MedData: вона автоматично аналізує кожне збережене вимірювання та створює сповіщення при виявленні відхилень від норми. Функція check_and_create_alerts() отримує об'єкт Measurement та об'єкт сесії БД і виконує перевірку кожного наявного показника. Дворівнева логіка норм дозволяє персоналізувати сповіщення: спочатку перевіряється наявність запису в таблиці measurement_norms для конкретного пристрою, при відсутності використовуються глобальні значення з settings.py.

Алгоритм обчислення критичності реалізований у функції _severity():

```
def _severity(value: float, lo: float, hi: float) -> str:
    if value < lo:
        deviation = abs(value - lo) / abs(lo)
    elif value > hi:
        deviation = abs(value - hi) / abs(hi)
    else:
        return 'info'
```

```
return 'critical' if deviation > 0.20 else 'warning'
```

Текст сповіщення генерується у форматі: «Пульс: 110 уд/хв — вище норми (макс: 100 уд/хв)». Така форма є зрозумілою для пацієнта без медичної освіти: вказано назву показника, зафіксоване значення, напрямок відхилення та порогове значення. Для critical-рівня додається підказка «Рекомендовано звернутися до лікаря». Функція mark_all_read() позначає відображені сповіщення як прочитані, але зберігає їх у базі для аудиторського сліду.

3.7 Налаштування стилів та інтерфейсу

Глобальні CSS-стилі визначаються в app.py через st.markdown з параметром unsafe_allow_html=True. Визначено два основних класи: alert-ok (зелений фон #f0fff4, ліва межа #43a047) та alert-high (червоний фон #fff0f0, ліва межа #e53935). Ці класи застосовуються у home_page.py для візуального розрізнення нормальних та відхилених показників. Колір фону бічної панелі встановлено в #f0f4f8 – ніжно-блакитний, що асоціюється з медичною тематикою. Розмір шрифту для st.metric збільшено до 1.6rem через CSS-змінну [data-testid='metric-container'], що покращує читабельність для людей похилого віку. Налаштування сторінки через st.set_page_config(): layout='wide' для розширеної зони, initial_sidebar_state='expanded' для одразу відкритої навігаційної панелі.

3.8 Управління версіями та GitLab

Управління кодовою базою здійснюється через систему контролю версій Git з віддаленим репозиторієм на платформі GitLab [38, 39]. Проєкт розробляється в єдиній гілці main. Стратегія описових комітів документує хронологію розробки: 'init: project structure and requirements.txt', 'feat: implement bcrypt authentication', 'feat: add CSV import with column aliases', 'feat: plotly charts with norm corridors', 'fix: NUMERIC type for medical metrics'. Файл .gitignore виключає .env (конфіденційні параметри підключення), .venv/, __pycache__/ та *.db. Виключення .env є критично важливим з точки зору безпеки: паролі бази даних не потрапляють у публічний репозиторій.

GitLab дозволяє налаштувати CI/CD pipeline для автоматичного запуску тестів при кожному push-коміті. У файлі .gitlab-ci.yml може бути описаний пайплайн із двох стадій: test (запуск pytest для сервісного шару) та lint (перевірка стилю коду через flake8). Це забезпечує автоматичний контроль якості та запобігає введенню регресійних помилок.

3.9 Тестування системи

Тестування системи MedData проводилось на трьох рівнях: функціональному тестуванні повних сценаріїв через браузерний інтерфейс, тестуванні граничних значень для функції валідації та тестуванні безпеки механізмів авторизації й ізоляції даних.

Тестування граничних значень перевіряло функцію validate_measurement() на межових та навколомежових значеннях. Значення пульсу 60 (нижня межа) приймається; значення 59 відхиляється. Значення пульсу 100 (верхня межа) приймається; значення 101 відхиляється. Аналогічно тестувались межі для тиску, умова pressure_dia < pressure_sys на значеннях dia = sys та dia = sys − 1. Усі граничні випадки оброблено коректно.

Тестування безпеки підтвердило: паролі зберігаються виключно у вигляді bcrypt-хешів (перевірено прямим SQL-запитом до PostgreSQL); SQL-запити містять параметри, що унеможливлює SQL-ін'єкції; механізм st.stop() не дозволяє відрендерити захищений контент для неавторизованих запитів; bcrypt.checkpw() коректно відхиляє неправильні паролі. Тестування продуктивності підтвердило час відгуку менше 2 секунд при 10 000 записів завдяки складеному індексу (user_id, measured_at).

Таблиця 3.1 – Результати функціонального тестування системи MedData

№	Тестовий сценарій	Вхідні дані	Очікуваний результат
1	Реєстрація нового користувача	Унікальне ім'я, пароль ≥ 6 символів	Акаунт створено, повідомлення про успіх
2	Реєстрація з існуючим ім'ям	Ім'я, що вже є у БД	Повідомлення: ім'я вже зайняте
3	Вхід з коректними даними	Правильне ім'я та пароль	Відкриття головної сторінки
4	Завантаження коректного CSV	Файл із 10 валідних рядків	Прев'ю таблиці, збереження 10 рядків
5	CSV без обов'язкового стовпця	Файл без колонки pulse	Повідомлення про відсутній стовпець
6	CSV з аномальним рядком	Рядок: пульс = 5	Рядок відхилено з описом помилки
7	Ручне введення валідних даних	Пульс 75, тиск 120/80	Запис збережено у БД
8	Ручне введення: діа ≥ сис	Sys = 80, Dia = 90	Повідомлення про некоректність
9	Перегляд аналітики (є дані)	5 записів у БД	Таблиця, статистика, 3 графіки
10	Виявлення аномалій	2 записи поза нормою	Попередження + таблиця
11	Генерація сповіщень critical	Пульс 130 (відхилення > 20%)	Сповіщення рівня critical
12	Ізоляція даних між акаунтами	Два різних акаунти	Кожен бачить лише свої записи
13	Видалення даних з підтвердженням	Ім'я збігається	Усі записи видалено
14	Продуктивність при 10 000 записів	10 000 рядків у БД	Час відгуку < 2 секунд
15	Ковзне середнє з різними вікнами	Вікно 3, 7, 14 вимірювань	Графік оновлюється динамічно

Висновок до розділу 3

У третьому розділі детально описано програмну реалізацію всіх компонентів системи MedData. Налаштовано середовище розробки з Python 3.13, PostgreSQL 16 та GitLab-репозиторієм.

Реалізовано модуль авторизації з bcrypt-хешуванням паролів (cost factor 12), механізмом st.stop() для захисту від несанкціонованого доступу та клієнтською і серверною валідацією форм. Головна сторінка надає миттєвий огляд поточного стану здоров'я через HTML-індикатори кольору стану та st.metric() з відображенням динаміки.

Модуль завантаження даних реалізує CSV-імпорт через CSV_COLUMN_ALIASES з підтримкою назв стовпців різними мовами, рядкову валідацію з частковим збереженням коректних рядків та динамічні форми ручного введення. Soft delete для пристроїв зберігає аналітичну цілісність даних.

Аналітичний модуль реалізує: описову статистику через DataFrame.describe().T, лінійні тренди через numpy.polyfit(), три Plotly-графіки з нормативними коридорами та виділенням аномалій, ковзне середнє через rolling(center=True), таблицю аномалій через векторизовані булеві маски та денну агрегацію через groupby. Система сповіщень із дворівневою логікою норм генерує warning та critical-сповіщення після кожного збереження.

Проведено комплексне тестування: 15 функціональних сценаріїв (всі пройдено успішно), тестування граничних значень та безпеки. Критичних помилок не виявлено. Продуктивність підтверджено: час відгуку при 10 000 записів менше 2 секунд.

ВИСНОВКИ

У кваліфікаційній (бакалаврській) роботі вирішено актуальне практичне завдання – розроблено вебсистему MedData для збирання, зберігання, обробки та візуалізації даних персональних медичних вимірювальних пристроїв. За результатами виконаних досліджень та практичної реалізації зроблено такі висновки.

1. Теоретичний аналіз підтвердив актуальність розробки. Стрімке зростання обсягів медичних даних, поширення носимих вимірювальних пристроїв та відсутність доступних аналітичних інструментів персонального рівня формують реальну потребу у системі, подібній до MedData. Аналіз існуючих рішень виявив незайняту нішу для інструменту, незалежного від виробника пристрою, що зберігає дані локально та підтримує кілька типів показників.
2. Спроєктовано багатошарову архітектуру системи з чотирма рівнями: конфігурація, ORM-моделі, сервіси та Streamlit-сторінки. Розроблено нормалізовану схему бази даних PostgreSQL з п'яти таблиць, оптимізовану складеним індексом (user_id, measured_at DESC). Перехід від SQLite до PostgreSQL та від SHA-256 до bcrypt обґрунтовано з точки зору масштабованості та безпеки.
3. Реалізовано повний технологічний стек: Python 3.13, Streamlit 1.x, PostgreSQL 16, SQLAlchemy 2.x, bcrypt 4.x, Pandas 2.x, NumPy, Plotly 5.x. Застосунок запускається однією командою без зовнішньої конфігурації та є кросплатформним.
4. Розроблено CSV-імпорт з CSV_COLUMN_ALIASES (підтримка варіантів назв стовпців різними мовами та форматами). Реалізовано динамічні форми введення залежно від типу пристрою та soft delete для пристроїв із збереженням аналітичної цілісності.
5. Аналітичний модуль включає: описову статистику (describe().T), лінійні тренди (numpy.polyfit()), Plotly-графіки з нормативними коридорами та

виділенням аномалій, ковзне середнє (rolling(center=True)), таблицю аномалій через векторизовані булеві маски, денну агрегацію через groupby.

6. Система сповіщень з дворівневою логікою норм (measurement_norms → settings.py) визначає рівень critical (відхилення > 20%) та warning. Текстові повідомлення зрозумілі пацієнту без медичної освіти та зберігаються для аудиту.

7. Проведено комплексне тестування: 15 функціональних сценаріїв (всі пройдено успішно), тестування граничних значень та безпеки. Продуктивність підтверджено: час відгуку при 10 000 записів – менше 2 секунд.

Перспективи розвитку: інтеграція з eHealth України та HL7 FHIR; модуль машинного навчання (scikit-learn) для прогнозування гіпертонічних кризів методами ARIMA та Random Forest; HTTPS та SQLCipher для продуктивного розгортання; мобільний Progressive Web App (PWA); розширення набору відстежуваних показників (рівень стресу, ЕКГ-дані).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. International Data Corporation. The Digitization of the World From Edge to Core. IDC White Paper #US44413318. Framingham, MA : IDC, 2018. 28 p.
2. Grand View Research. Personal Health Monitoring Market Size, Share & Trends Analysis Report, 2022–2030. San Francisco : Grand View Research, 2022. URL: https://www.grandviewresearch.com/industry-analysis/personal-health-monitoring-market (дата звернення: 01.04.2026).
3. WHO. Global report on hypertension: the race against a silent killer. Geneva : World Health Organization, 2023. 264 p. ISBN 978-92-4-007174-8.
4. Cios K.J., Moore G.W. Uniqueness of medical data mining. Artificial Intelligence in Medicine. 2002. Vol. 26, No. 1–2. P. 1–24.
5. WHO. Global strategy on digital health 2020–2025. Geneva : World Health Organization, 2021. 56 p. ISBN 978-92-4-002092-0.
6. Simpao A.F. et al. A review of analytics and clinical informatics in health care. Journal of Medical Systems. 2014. Vol. 38, No. 4. P. 45.
7. Bates D.W. et al. Big data in health care: using analytics to identify and manage high-risk patients. Health Affairs. 2014. Vol. 33, No. 7. P. 1123–1131.
8. Ioannidis J.P.A. The mass production of redundant, misleading, and conflicted systematic reviews. Milbank Quarterly. 2016. Vol. 94, No. 3. P. 485–514.
9. Topol E.J. High-performance medicine: the convergence of human and artificial intelligence. Nature Medicine. 2019. Vol. 25, No. 1. P. 44–56. URL: https://doi.org/10.1038/s41591-018-0300-7 (дата звернення: 15.03.2026).
10. Whelton P.K. et al. 2017 ACC/AHA Guideline for the Prevention, Detection, Evaluation, and Management of High Blood Pressure. Journal of the American College of Cardiology. 2018. Vol. 71. P. e127–e248.
11. Williams B. et al. 2018 ESC/ESH Guidelines for the management of arterial hypertension. European Heart Journal. 2018. Vol. 39, No. 33. P. 3021–3104.
12. Parati G. et al. ESH guidelines for blood pressure monitoring at home. Journal of Hypertension. 2008. Vol. 26, No. 8. P. 1505–1530.

13. Pickering T.G. et al. Call to action on use and reimbursement for home blood pressure monitoring. Hypertension. 2008. Vol. 52, No. 1. P. 10–29.
14. Stack Overflow Developer Survey 2024. Stack Overflow, 2024. URL: https://survey.stackoverflow.co/2024 (дата звернення: 05.04.2026).
15. McKinney W. Python for Data Analysis. 3rd ed. Sebastopol : O'Reilly Media, 2022. 572 p. ISBN 978-1098104030.
16. Raschka S., Mirjalili V. Python Machine Learning. 3rd ed. Birmingham : Packt Publishing, 2019. 770 p.
17. Streamlit Inc. Streamlit Documentation. Version 1.x. 2024. URL: https://docs.streamlit.io (дата звернення: 10.04.2026).
18. HL7 International. HL7 FHIR R4 Specification. 2019. URL: https://hl7.org/fhir/R4/ (дата звернення: 16.03.2026).
19. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. Відомості Верховної Ради України. 2010. № 34. Ст. 481.
20. Nguyen L. Python 3.11: Release highlights. Python.org, 2022. URL: https://docs.python.org/3/whatsnew/3.11.html (дата звернення: 10.03.2026).
21. OpenMRS Community. OpenMRS Developer Guide. Version 3.0. 2023. URL: https://openmrs.org/documentation (дата звернення: 21.03.2026).
22. HISP Centre. DHIS2 User Manual. Oslo : University of Oslo, 2023. URL: https://docs.dhis2.org (дата звернення: 22.03.2026).
23. VanderPlas J. Python Data Science Handbook. Sebastopol : O'Reilly Media, 2016. 541 p. URL: https://jakevdp.github.io/PythonDataScienceHandbook (дата звернення: 14.04.2026).
24. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. 2023. URL: https://www.postgresql.org/docs/16/ (дата звернення: 10.03.2026).
25. SQLAlchemy Authors. SQLAlchemy 2.0 Documentation. 2023. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 12.03.2026).
26. OWASP Foundation. Password Storage Cheat Sheet. 2023. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 18.04.2026).

27. Provos N., Mazieres D. A Future-Adaptable Password Scheme. USENIX Annual Technical Conference. 1999. P. 81–91.

28. Plotly Technologies Inc. Plotly Python Graphing Library. 2024. URL: https://plotly.com/python/ (дата звернення: 12.04.2026).

29. Harris C.R. et al. Array programming with NumPy. Nature. 2020. Vol. 585. P. 357–362.

30. Pedregosa F. et al. Scikit-learn: Machine learning in Python. JMLR. 2011. Vol. 12. P. 2825–2830.

31. Reback J. et al. pandas-dev/pandas: Pandas 2.0.0. Zenodo. 2023. DOI: 10.5281/zenodo.7741580.

32. Pydantic. Pydantic Settings Documentation. 2024. URL: https://docs.pydantic.dev/latest/concepts/pydantic_settings/ (дата звернення: 08.04.2026).

33. Bhatt D.L. et al. A controlled trial of renal denervation for resistant hypertension. NEJM. 2014. Vol. 370. P. 1393–1401.

34. Pickering T.G. et al. Recommendations for blood pressure measurement in humans. Hypertension. 2005. Vol. 45. P. 142–161.

35. Oliphant T.E. A Guide to NumPy. Trelgol Publishing, 2006. 371 p.

36. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p. ISBN 978-0321127426.

37. Martin R.C. Clean Architecture. Prentice Hall, 2017. 432 p. ISBN 978-0134494166.

38. Git Documentation. Reference manual. 2024. URL: https://git-scm.com/docs (дата звернення: 15.04.2026).

39. GitLab Inc. GitLab Documentation. 2024. URL: https://docs.gitlab.com (дата звернення: 15.04.2026).

40. Python Software Foundation. Python 3.13 Documentation: venv. 2024. URL: https://docs.python.org/3.13/library/venv.html (дата звернення: 10.03.2026).

41. European Parliament. Regulation (EU) 2016/679 (GDPR). Official Journal of the EU. 2016. L 119. P. 1–88.

42. Закон України «Про державні фінансові гарантії медичного обслуговування» від 19.10.2017 № 2168-VIII.
43. National Center for Health Statistics. Health, United States, 2022. Hyattsville, MD : NCHS, 2023.
44. Алгебраїст В.Б., Гузій А.В. Методи аналізу часових рядів медичних даних. Кібернетика та системний аналіз. 2020. № 56(4). С. 120–131.
45. Ковальчук Т.М., Марченко О.А. Цифровізація системи охорони здоров'я України: стан та перспективи. Ефективна економіка. 2021. № 11.
46. Зубченко С.О. та ін. Домашній самоконтроль артеріального тиску. Артеріальна гіпертензія. 2019. Т. 12, № 1. С. 45–53.
47. pydantic-settings documentation. 2024. URL: https://docs.pydantic.dev/latest/concepts/pydantic_settings (дата звернення: 08.04.2026).
48. bcrypt PyPI Package. 2024. URL: https://pypi.org/project/bcrypt/ (дата звернення: 18.04.2026).
49. psycopg2 Documentation. 2024. URL: https://www.psycopg.org/docs/ (дата звернення: 12.03.2026).
50. Кабінет Міністрів України. Концепція розвитку системи електронної охорони здоров'я. Розпорядження від 28.12.2020 № 1671-р. URL: https://zakon.rada.gov.ua/laws/show/1671-2020-р (дата звернення: 05.04.2026).