ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)
«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА
на тему

ВЕБ-ДОДАТОК ДЛЯ ВІДСТЕЖЕННЯ ТА ПРОГНОЗУВАННЯ СТАНУ ЗДОРОВ'Я НА ОСНОВІ ОБРОБКИ БІОМЕДИЧНИХ ДАНИХ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Дзяткевич Максим Олегович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.ф.-м.н. Черненко О. О.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 75 с., 13 рис., 1 таблиця, 1 додаток, 13 джерел.

БІОМЕДИЧНІ ДАНІ, ПРОГНОЗУВАННЯ, МАШИННЕ НАВЧАННЯ, AI-АНАЛІЗ, FASTAPI, NEXT.JS

Об’єктом розробки є веб-система для відстеження стану здоров’я користувача, яка забезпечує збирання, збереження, аналіз та прогнозування біомедичних показників у режимі реального часу.

Предметом розробки є програмна реалізація модулів введення та обробки медичних даних, алгоритмів машинного навчання для прогнозування змін показників, а також інтерактивний інтерфейс, що дозволяє аналізувати історію вимірювань, візуалізувати статистику та отримувати рекомендації від AI-асистента.

Метою роботи є створення інтелектуального веб-додатка Health Tracker, який поєднує сучасні інструменти веб-розробки із моделями машинного навчання для автоматичного аналізу біомедичних даних, виявлення аномалій та формування персоналізованих прогнозів стану здоров’я.

У результаті виконання роботи було розроблено повноцінну систему, що включає модулі:

• введення та редагування медичних вимірювань;

• імпорту даних з CSV-файлів;

• автоматичного розрахунку індексу маси тіла (BMI), середнього артеріального тиску (MAP) та інших похідних метрик;

• побудови інтерактивних графіків на основі історичних даних;

• прогнозування показників за допомогою моделі Prophet та алгоритмів машинного навчання;

• формування AI-аналізу, виявлення ризиків і надання персональних рекомендацій;

• експорту звітів у форматах CSV, Excel та PDF.

AI-модуль аналізує часові ряди медичних показників, виявляє відхилення від норми, оцінює тренди та формує короткострокові прогнози на основі історичних

вимірювань. Також реалізовано інтерактивний AI-чат, який дозволяє ставити питання щодо стану здоров’я та отримувати персоналізовані пояснення і рекомендації. Модуль прогнозування створено на основі бібліотеки Prophet, а аналітичні алгоритми — із застосуванням Scikit-learn.

Веб-додаток містить такі основні розділи:

• Dashboard — огляд ключових показників та швидка статистика;

• Measurements — додавання, редагування та імпорт вимірювань;

• Analytics — графічний аналіз медичних даних;

• AI Doctor — детальний AI-аналіз, оцінка ризиків, трендів та рекомендації;

• Forecasts — прогнозування стану здоров’я на 3–7 днів.

Система пройшла комплексне тестування: перевірено роботу API, коректність обробки медичних даних, точність прогнозів, стабільність AI-аналітики та відповідність логіки доступу вимогам безпеки. Результати тестування підтвердили стабільну роботу всіх компонентів, коректність алгоритмів та готовність продукту до практичного використання.

Розроблений Health Tracker може бути застосований як персональний інструмент моніторингу здоров’я, як основа для телемедичних сервісів, як прототип системи прогнозної медицини, а також як навчальний або дослідницький інструмент для аналізу біомедичних даних.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
API	Інтерфейс прикладного програмування для взаємодії між клієнтом і сервером
AI	Штучний інтелект, модуль аналізу та прогнозування
ML	Машинне навчання, використовується для прогнозів стану здоров’я
BMI	Body Mass Index – індекс маси тіла
MAP	Mean Arterial Pressure – середній артеріальний тиск
JWT	JSON Web Token – токен для аутентифікації користувачів
ORM	Object-Relational Mapping – технологія взаємодії з базою даних
CSV	Формат файлів "Comma-Separated Values" для імпорту/експорту даних
PDF	Portable Document Format – формат документів для експорту звітів
SQLite	Легка реляційна база даних, використана в проєкті
FastAPI	Python-фреймворк для створення бекенд-сервісів
Next.js	React-фреймворк для створення фронтенд-частини застосунку
Prophet	Модель прогнозування часових рядів, використана для передбачення показників

Scikit-learn	Бібліотека ML для класифікації, регресії та обробки даних
Axios	HTTP-клієнт для відправлення запитів з фронтенду
Recharts	Бібліотека для побудови графіків і діаграм
Health Index	Інтегральний індекс здоров’я, розрахований AI-модулем
Measurement	Запис вимірювання конкретного показника користувача

ВСТУП

Стрімкий розвиток цифрових технологій суттєво вплинув на сферу охорони здоров’я, відкривши нові можливості для персоналізованого моніторингу стану організму. Сучасні веб-системи дозволяють користувачам фіксувати медичні показники, аналізувати їх динаміку, отримувати рекомендації та своєчасно виявляти потенційні ризики. Проте більшість існуючих рішень обмежені вузькою функціональністю, не забезпечують комплексного аналізу даних, не підтримують прогнозування та не враховують індивідуальні особливості користувача. Саме тому розробка інтелектуальної системи для аналізу й прогнозування біомедичних параметрів набуває особливої актуальності.

Актуальність дослідження зумовлена тим, що своєчасне виявлення відхилень стану здоров’я дозволяє попередити ускладнення та значно зменшити ризики розвитку хронічних захворювань. За даними світових медичних організацій, понад половина критичних станів може бути виявлена на ранніх етапах у разі регулярного контролю важливих показників, таких як артеріальний тиск, частота серцевих скорочень, глюкоза крові та інші фізіологічні параметри. У поєднанні з алгоритмами штучного інтелекту аналіз даних дає змогу не лише фіксувати стан організму, а й прогнозувати можливі зміни у майбутньому.

Об’єктом дослідження є процес відстеження та аналізу біомедичних показників користувача в цифровому середовищі.

Предметом дослідження виступають методи, програмні засоби та алгоритми обробки, аналізу і прогнозування біомедичних даних у веб-додатку для моніторингу стану здоров’я.

Визначення цих елементів дозволяє чітко окреслити межі дослідження та обґрунтувати вибір технологій, на основі яких будується система.

Метою дипломної роботи є розробка веб-додатку для відстеження та прогнозування стану здоров’я на основі біомедичних даних із застосуванням сучасних методів аналізу, алгоритмів машинного навчання та інтелектуальних моделей. Для досягнення поставленої мети передбачено виконання таких завдань:

аналіз існуючих рішень та методів обробки показників, проєктування архітектури системи, розробка клієнтської та серверної частин, впровадження алгоритмів аналізу й прогнозування, тестування застосунку та оцінка його функціональності.

Результатом роботи є створення функціонального веб-додатку Health Tracker, який забезпечує збір, зберігання та обробку медичних показників, їх статистичний аналіз, побудову графіків, розрахунок індексів здоров’я, прогнозування метрик на основі моделей часових рядів, а також формування персоналізованих рекомендацій за допомогою штучного інтелекту. Система має інтуїтивний інтерфейс, підтримує понад 50 медичних параметрів, містить модулі AI-аналізу, прогнозування та інтерактивну взаємодію з користувачем.

Створений програмний продукт може використовуватися як персональна система моніторингу здоров’я, як допоміжний інструмент для лікарів або як основа для розробки масштабованих медичних інформаційних платформ, що підкреслює практичну значущість проведеного дослідження.

1. ПОСТАНОВКА ЗАДАЧІ

Розробка веб-додатку для відстеження та прогнозування стану здоров’я потребує чіткого формулювання задачі, визначення її меж, критеріїв ефективності та суттєвих технічних і наукових аспектів. У сучасних умовах значний обсяг біомедичних даних формує потребу в системах, здатних коректно збирати, структурувати, аналізувати та інтерпретувати фізіологічні показники користувача. Це обумовлює необхідність створення платформи, яка поєднує зручний інтерфейс, інтелектуальні алгоритми обробки даних та прогностичні моделі, що дозволяють підвищити точність оцінки стану здоров’я.

Метою проєкту є створення веб-додатку Health Tracker, який забезпечує повний цикл роботи з біомедичними даними: від збору та структурування інформації до аналізу, прогнозування та формування персоналізованих рекомендацій. Система має враховувати індивідуальні особливості користувача, підтримувати обробку різних типів медичних метрик, забезпечувати візуалізацію динаміки показників та застосовувати сучасні алгоритми машинного навчання.

Для досягнення поставленої мети необхідно розв’язати такі основні завдання:

– дослідити сучасні методи цифрового моніторингу здоров’я та підходи до аналізу біомедичних даних;

– провести огляд існуючих веб-рішень і визначити їх ключові переваги та недоліки;

– розробити структуру даних, яка забезпечить зберігання великої кількості показників та гнучкість розширення системи;

– спроєктувати архітектуру веб-додатку з поділом на фронтенд, бекенд та модулі машинного навчання;

– реалізувати серверну частину з API для роботи з метриками, вимірюваннями, статистикою, аналітикою й прогнозами;

– створити клієнтську частину з інтуїтивним інтерфейсом для взаємодії користувача з даними та AI-асистентом;

– впровадити алгоритми аналізу, оцінювання ризиків та прогнозування

часових рядів;

– забезпечити безпеку зберігання даних, аутентифікацію та захист доступу до системи;

– провести тестування програмного забезпечення для оцінки коректності, стабільності та продуктивності роботи.

Розв’язання цих завдань дає змогу побудувати комплексну систему, здатну забезпечити користувача достовірною інформацією про власний фізіологічний стан, показати динаміку змін і виявити потенційні ризики. Розробка веб-додатку Health Tracker спрямована на досягнення високого рівня інтерактивності, гнучкості та аналітичних можливостей, що дозволяє розглядати його як ефективний інструмент персонального цифрового здоров’я.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Сучасні підходи до цифрового моніторингу здоров’я

Цифровий моніторинг здоров’я є одним із найдинамічніших напрямів розвитку сучасної медицини, оскільки поєднує можливості інформаційних технологій, аналітики даних та персоналізованого підходу до оцінки фізіологічного стану людини. Використання мобільних пристроїв, носимих сенсорів, хмарних технологій та веб-сервісів створює умови для постійного збору та аналізу біологічних сигналів, що забезпечує виявлення відхилень у реальному часі. Основним трендом є перехід від епізодичного контролю стану здоров’я до безперервного моніторингу, який створює об’єктивну картину фізіологічних процесів та дозволяє своєчасно реагувати на зміни.

Одним із ключових підходів є використання носимих сенсорів — таких як фітнес-браслети, розумні годинники та медичні трекери — які автоматично збирають дані про частоту серцевих скорочень, рівень активності, сон, насичення крові киснем та інші показники. Ці дані синхронізуються з мобільними або веб-застосунками, де користувач може переглядати статистику, тренди та отримувати рекомендації. Важливою особливістю такого підходу є мінімальне залучення користувача: більшість процесів відбувається автоматично, що підвищує точність та регулярність збору інформації. [1]

Іншим поширеним підходом є використання хмарних платформ, які забезпечують зберігання великих обсягів біомедичних даних, їх обробку та доступ з будь-якого пристрою. Хмарні технології дозволяють інтегрувати дані з різних джерел — лабораторних аналізів, домашніх вимірювань, медичних пристроїв — створюючи єдиний цифровий профіль користувача. Такі платформи часто використовують стандартизовані протоколи обміну медичними даними (FHIR, HL7), що полегшує взаємодію між системами та медичними установами.

Важливим напрямом є впровадження аналітичних моделей та алгоритмів штучного інтелекту, що дають змогу автоматично оцінювати стан здоров’я, виявляти відхилення, прогнозувати ризики та формувати персоналізовані

рекомендації. Сучасні системи аналізують поведінкові патерни, динаміку показників, поєднують фізіологічні та контекстні дані, враховують режим сну, активності, харчування. Алгоритми машинного навчання дозволяють проводити класифікацію станів, виявляти аномалії та будувати моделі прогнозування, що забезпечує проактивний підхід до управління здоров'ям. [2]

Окреме місце займають системи телемедицини, які забезпечують дистанційну взаємодію між користувачем та медичним спеціалістом. Такі системи дозволяють лікарям відслідковувати показники пацієнтів, переглядати їх динаміку, визначати ризики та коригувати лікування без необхідності особистих візитів. Поєднання телемедицини та автоматизованої аналітики створює гібридні рішення, які значно розширюють можливості цифрового моніторингу.

Серед сучасних тенденцій також спостерігається перехід до персоналізованої медицини, де рекомендації формуються на основі індивідуальних характеристик користувача, його історії хвороб, способу життя та генетичних факторів. Цей підхід забезпечує більш точні прогнози та адаптовані рекомендації, підвищуючи ефективність самоконтролю та лікування.

Загалом сучасні підходи до цифрового моніторингу здоров'я базуються на інтеграції автоматизованого збору даних, хмарних технологій, аналітики та моделей штучного інтелекту. Це забезпечує більш повне та своєчасне розуміння фізіологічного стану користувача, дозволяє виявляти ризики на ранніх стадіях і сприяє формуванню відповідального ставлення до власного здоров'я.

2.2. Підхід обробки та аналізу біомедичних даних

Обробка та аналіз біомедичних даних є ключовими елементами сучасних систем моніторингу стану здоров'я, оскільки саме від правильності їх інтерпретації залежить якість оцінювання фізіологічного стану, точність прогнозів та ефективність рекомендацій. Біомедичні дані, що включають фізіологічні параметри, результати лабораторних досліджень, часові ряди та показники активності, характеризуються високою варіативністю, чутливістю до артефактів та залежністю

від зовнішніх факторів. Тому їх аналіз потребує застосування спеціалізованих підходів та алгоритмів, які враховують специфіку медичних сигналів.

Першим етапом є збір та попередня обробка даних, що передбачає фільтрацію шумів, видалення аномальних значень, стандартизацію одиниць вимірювання, перетворення часових міток та нормалізацію даних для подальшої аналітики. Для багатьох фізіологічних метрик важливо забезпечити коректний формат представлення даних, зокрема приведення до рівномірних інтервалів у часових рядах або усереднення показників для зменшення випадкових коливань. На цьому етапі формується основа для достовірного статистичного аналізу.[3]

Наступним кроком є статистичний аналіз, який дозволяє визначити базові характеристики даних: середні значення, медіану, мінімальні та максимальні показники, варіацію та стандартне відхилення. Ці параметри дозволяють оцінити загальний стан користувача, виявити критичні відхилення та визначити динаміку змін. Для більш складних метрик застосовуються методи кореляційного аналізу, які дозволяють оцінити взаємозв’язки між показниками, наприклад зв’язок між рівнем глюкози, масою тіла чи артеріальним тиском.

Особливо важливими для цифрової медицини є методи аналізу часових рядів, що дозволяють оцінювати динамічні процеси в організмі. До них належать ковзні середні, експоненціальне згладжування, сезонні моделі та алгоритми виявлення трендів. Часові ряди є основою для прогнозування, оскільки саме їх структура дозволяє моделі виявити закономірності, сезонність, циклічність та можливі відхилення. У сучасних системах часто застосовуються моделі Prophet, ARIMA та алгоритми машинного навчання, адаптовані до медичних даних.

Для оцінювання стану здоров’я важливе значення має виявлення аномалій, що дає змогу фіксувати значення, які виходять за межі нормальних діапазонів або не відповідають типовим патернам користувача. Методи виявлення аномалій можуть бути статистичними (z-score, IQR), кластерними (k-means, DBSCAN) або ґрунтуватися на моделях машинного навчання. Виявлення аномалій є основою для раннього попередження про ризики. [4]

Важливою складовою аналізу є розрахунок медичних індексів — таких як

BMI, MAP, пульсовий тиск чи індекс здоров’я. Ці показники формують додатковий рівень узагальнення, що дозволяє користувачу отримати цілісне уявлення про свій стан. Комбінування індексів з базовими метриками забезпечує багаторівневий підхід до оцінки здоров’я.

Завершальним етапом є інтерпретація даних, яка передбачає формування висновків, попереджень та рекомендацій. На цьому рівні використовуються алгоритми машинного навчання, які аналізують історичні дані, прогнозують майбутні значення та визначають рівень ризику. Поєднання аналітичних методів із системами штучного інтелекту дозволяє створювати персоналізовані висновки, що враховують індивідуальні показники користувача.

Таким чином, сучасний підхід до обробки та аналізу біомедичних даних ґрунтується на інтеграції статистичних методів, аналізу часових рядів, алгоритмів машинного навчання та технологій виявлення аномалій. Це забезпечує комплексний погляд на фізіологічний стан користувача та створює основу для проактивного моніторингу здоров’я.

2.3. Огляд існуючих веб-рішень та мобільних застосунків

Сучасний ринок цифрових систем моніторингу здоров’я представлений великою кількістю веб-додатків та мобільних застосунків, що забезпечують збір, аналіз і візуалізацію фізіологічних даних. Ці рішення орієнтовані на різні категорії користувачів — від людей, які прагнуть контролювати основні життєві показники, до спортсменів, пацієнтів із хронічними захворюваннями та лікарів, які здійснюють дистанційний нагляд. Незважаючи на широкий вибір продуктів, більшість із них реалізує окремі аспекти моніторингу здоров’я, не забезпечуючи повноцінного комплексного аналізу чи прогнозування. Це визначає важливість вивчення їхнього функціоналу для формування вимог до нового програмного забезпечення.

Одним із найпоширеніших рішень є Google Fit, який забезпечує базовий моніторинг активності, частоти серцевих скорочень, кроків, тривалості сну та рівня навантаження (див. рис. 2.1). Перевагою платформи є автоматичний збір даних із

різних пристроїв та їхня візуалізація у вигляді графіків. Проте система не надає глибоких медичних інтерпретацій, аналізу ризиків чи прогнозування стану здоров’я, що обмежує її застосування у медичному контексті.

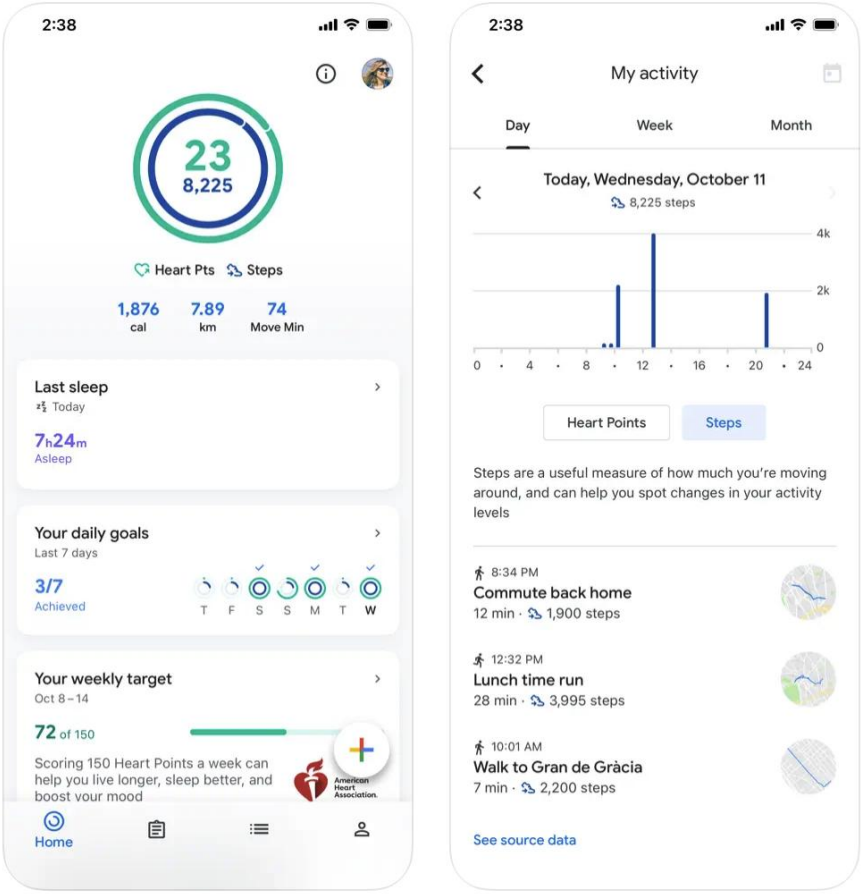

Рисунок 2.1 – Моніторинг «Google Fit»

Подібним за структурою є сервіс Apple Health, який збирає великий спектр біомедичних показників, зокрема дані про серцевий ритм, фізичну активність, цикл сну, харчування та інші параметри (див. рис. 2.2). Система дозволяє інтегрувати медичні записи, однак не пропонує інструментів машинного навчання або аналітики, що виходить за межі базової статистики. Головним недоліком є відсутність глибоких діагностичних можливостей та механізмів прогнозування.

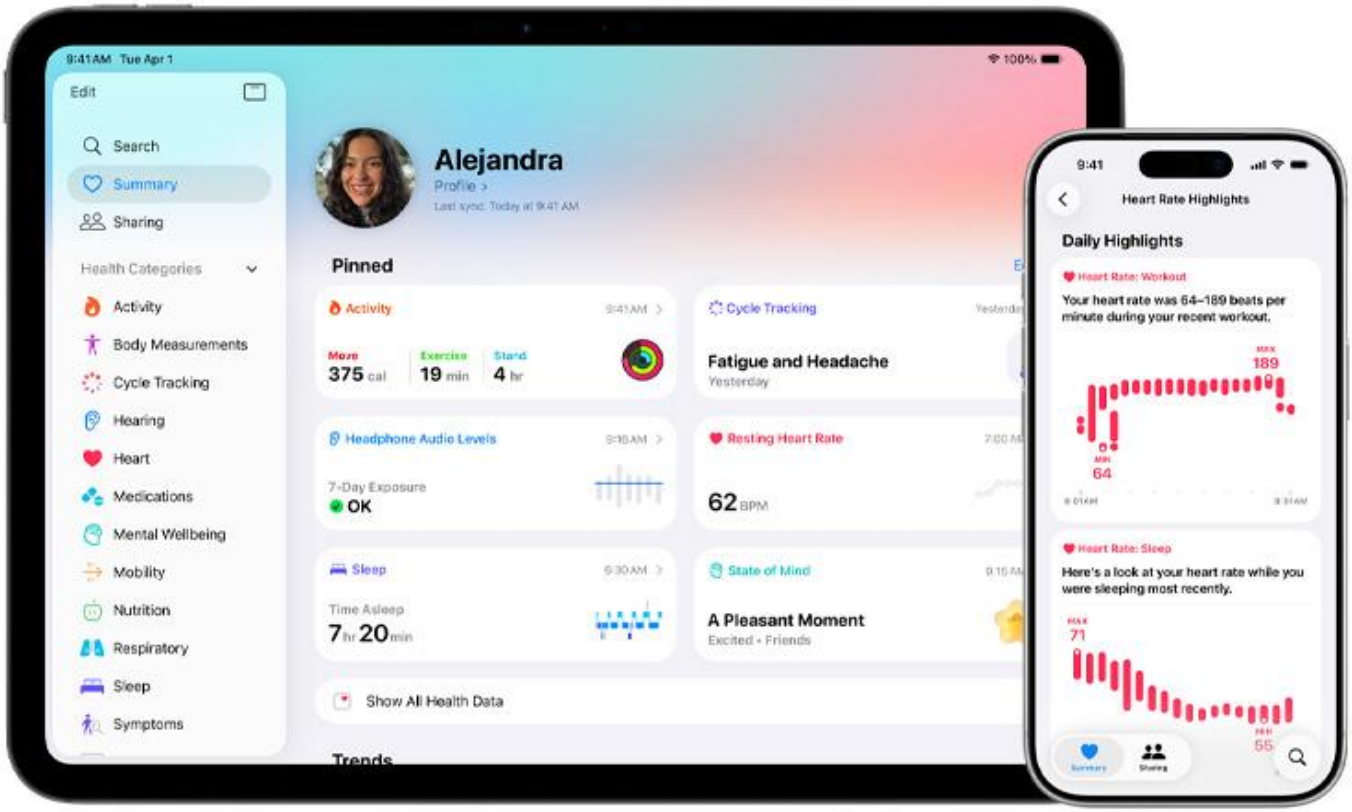

Рисунок 2.2 – Моніторинг «Apple Health»

У категорії спортивно-медичних застосунків широке поширення отримав Samsung Health, який, окрім фіксації основних метрик, пропонує тренувальні програми та рекомендації щодо способу життя (див. рис. 2.3). Проте система також орієнтована переважно на фітнес-функціональність і не забезпечує розширених можливостей аналізу медичних показників, таких як глюкоза, гормони чи біохімічні параметри.

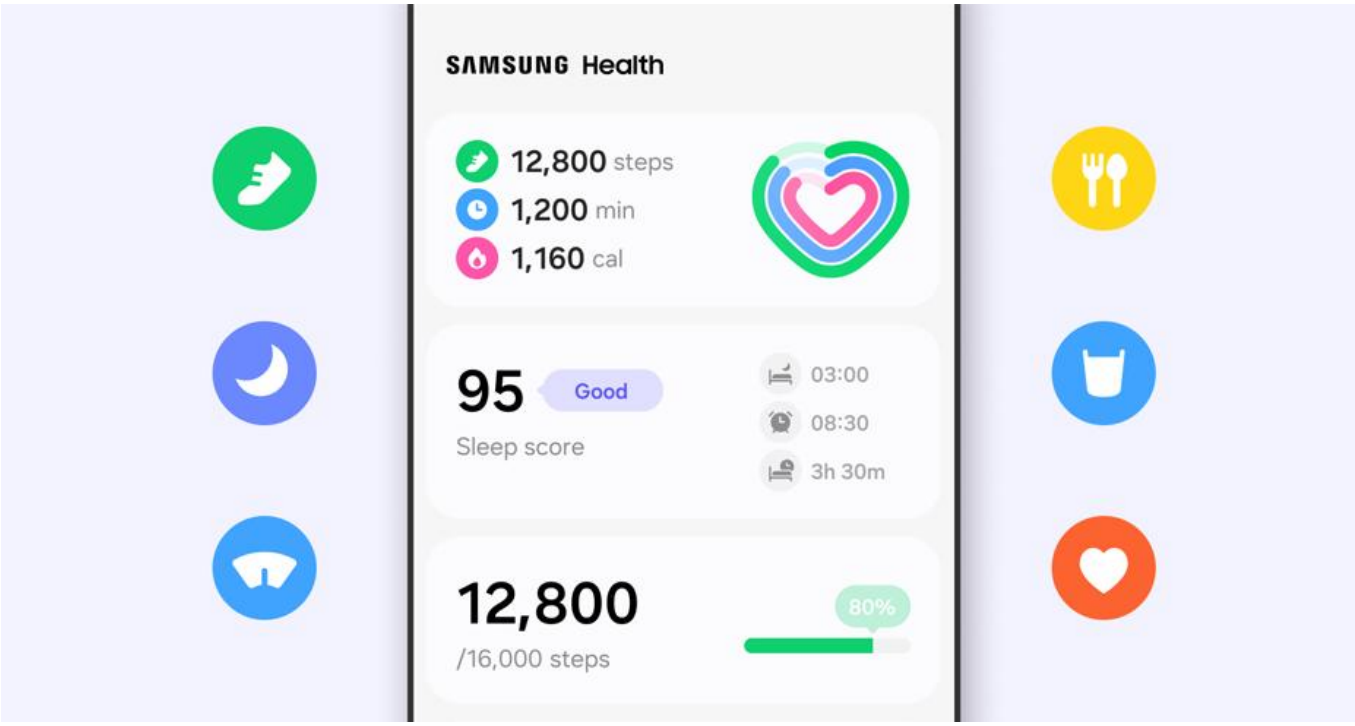

Рисунок 2.3 – Моніторинг «Samsung Health»

Аналізуючи існуючі рішення, можна зробити висновок, що ринок цифрових систем здоров’я орієнтований або на загальний фітнес-контроль, або на вузькоспеціалізовані медичні параметри. Недостатньою залишається інтеграція статистичного аналізу, алгоритмів машинного навчання, механізмів прогнозування та інтелектуальної інтерпретації даних. Більшість систем не забезпечує одночасної роботи з широким спектром біомедичних показників, не об’єднує їх у єдину модель здоров’я та не формує комплексні індекси ризику.

Таким чином, існує потреба у створенні веб-додатку, який би поєднував можливості різних типів систем, забезпечував комплексний аналіз біомедичних даних, підтримував прогнозування та формував індивідуальні рекомендації. Запропонований у цій роботі додаток Health Tracker спрямований на заповнення саме цієї прогалини, пропонуючи багаторівневу аналітику та інтеграцію сучасних методів машинного навчання.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Математичні та медичні основи аналізу біомедичних показників

Аналіз біомедичних показників ґрунтується на поєднанні медичних знань про фізіологію людини та математичних методів, що дозволяють коректно обробляти, інтерпретувати та порівнювати дані. Біомедичні параметри — такі як частота серцевих скорочень, артеріальний тиск, рівень глюкози, показники крові та інші — є кількісними величинами, динаміка яких відображає стан організму. Їх правильна інтерпретація неможлива без математичного аналізу, оскільки дані часто коливаються, містять артефакти, нерівномірно розподілені в часі та взаємопов’язані між собою.

Одним із базових елементів аналізу є визначення нормальних діапазонів, які встановлюються на основі клінічних рекомендацій та статистичних досліджень. Норма для кожної метрики визначається діапазоном значень, що відповідає здоровому функціонуванню організму. Наприклад, нормальний діапазон артеріального тиску становить приблизно 120/80 мм рт. ст., нормальний пульс — 60–100 уд/хв, рівень глюкози натще — 3.9–5.5 ммоль/л. Порівняння фактичних значень із нормальними інтервалами дозволяє визначити ступінь відхилення та оцінити ризики. [5]

Для узагальнення інформації часто застосовуються розрахункові індекси, що комбінують декілька показників. До таких належать індекс маси тіла (BMI), середній артеріальний тиск (MAP) і пульсовий тиск. BMI визначається за формулою

$$BMI = \frac{m}{h^2},$$

де m — маса тіла в кілограмах, h — зріст у метрах. MAP розраховується як

$$MAP = \frac{P_{sys}+2P_{dia}}{3},$$

де $P\square_{\gamma}\square$ та P_d_{ia} — систолічний та діастолічний тиск. Ці індекси дозволяють узагальнити дані та сформувати комплексну оцінку стану серцево-судинної системи.

Важливим аспектом є динамічний аналіз, який ґрунтується на вивченні змін біомедичних параметрів у часі. У медичній практиці значення показника в конкретний момент часу має менше значення, ніж його динаміка. Погіршення або покращення значень дає можливість оцінити реакцію організму, ефективність лікування чи появу нових ризиків. Для цього застосовуються математичні методи аналізу часових рядів: ковзні середні, лінійна апроксимація, обчислення похідних та виявлення трендів. Динамічні моделі дають змогу виявляти циклічність, сезонність та аномальні зміни у фізіологічних процесах.

Медичні показники часто пов’язані між собою, тому значну роль відіграє кореляційний аналіз, що дозволяє дослідити взаємозалежності між метриками. Наприклад, підвищений індекс маси тіла може корелювати з підвищеним рівнем глюкози або артеріальним тиском. Виявлення таких закономірностей створює основу для прогнозування ризиків та визначення факторів, що найбільше впливають на здоров’я користувача.

Ще одним важливим аспектом є виявлення аномалій, що передбачає пошук значень, які значно відрізняються від типових для конкретного користувача або від встановлених норм. Аномальні значення можуть свідчити про фізіологічні збої, помилки вимірювання або вплив зовнішніх факторів. Математичні методи, такі як z-score, міжквартильний розмах (IQR) чи аналіз відстаней у багатовимірному просторі, дозволяють автоматизувати процес виявлення таких відхилень.

Багато біомедичних показників мають нестаціонарний характер, тому для їх аналізу застосовуються методи згладжування, нормалізації та трансформацій. Оскільки дані можуть бути нерівномірно зібрані, важливо коректно працювати з часовими мітками, забезпечувати впорядкування вимірювань і обробку пропусків. Ці підходи дозволяють отримати структуровані й придатні для аналізу дані.[6]

Таким чином, математичні та медичні основи аналізу біомедичних показників охоплюють визначення нормальних діапазонів, розрахунок індикаторів, аналіз динаміки, виявлення аномалій та моделювання взаємозв’язків між параметрами.

3.2. Методи прогнозування та моделі машинного навчання для здоров’я

Прогнозування біомедичних показників є важливим елементом сучасних систем цифрового здоров’я, оскільки дозволяє передбачати можливі зміни фізіологічного стану користувача та виявляти ризики на ранніх етапах. На відміну від статичного аналізу, який працює лише з поточними даними, прогностичні моделі враховують історичну динаміку, закономірності, сезонність і реакцію організму на зовнішні фактори. Тому застосування алгоритмів машинного навчання у сфері медицини стає одним із ключових напрямів розвитку персоналізованого моніторингу здоров’я.

Одним із найефективніших методів прогнозування часових рядів у медичній сфері є модель **Prophet**, розроблена компанією Meta. Її перевага полягає у здатності автоматично враховувати тренди, сезонні коливання та нерівномірність даних без складного попереднього налаштування. Це робить Prophet придатним для роботи з фізіологічними показниками, які часто змінюються нерівномірно та мають індивідуальні патерни. Модель генерує прогноз із інтервалом довіри, що особливо важливо у медичних застосуваннях, де оцінка ризику має враховувати можливу похибку.

Окрім Prophet, у біомедичних задачах широко застосовуються класичні статистичні моделі, такі як **ARIMA** та її модифікації. Моделі ARIMA добре працюють для стаціонарних часових рядів та дозволяють аналізувати залежності між попередніми та майбутніми значеннями показників. Вони підходять для прогнозування параметрів, що мають виражені тренди або циклічність, наприклад артеріального тиску чи рівня глюкози. Проте ці моделі потребують ретельної підготовки даних і ручного налаштування параметрів, що обмежує їх використання у широких споживчих застосунках.

Для оцінювання стану здоров’я важливу роль відіграють **класифікаційні моделі**, які дозволяють визначати рівень ризику або присвоювати фізіологічним показникам статуси «норма», «попередження» чи «критично». До таких моделей належать логістична регресія, дерева рішень, градієнтний бустинг та метод опорних

векторів. Вони дають можливість будувати інтелектуальні правила аналізу на основі великого обсягу історичних даних, враховуючи не лише поточне значення метрики, а й її динаміку та кореляції з іншими показниками.

У контексті виявлення фізіологічних відхилень ефективними є методи **виявлення аномалій**, які дозволяють визначати значення, що не відповідають нормальним закономірностям або типовим станам користувача. Такі методи застосовують алгоритми кластеризації (k-means, DBSCAN), статистичні підходи (IQR, z-score), а також алгоритми машинного навчання, орієнтовані на відхилення, наприклад Isolation Forest. Виявлення аномалій є одним з основних механізмів ранньої діагностики та автоматичного формування попереджень у системах цифрового здоров'я.

Для аналізу зв'язків між показниками використовуються методи **регресійного аналізу** та **кореляційного моделювання**, які дозволяють визначити, як зміна одного параметра впливає на інші. Наприклад, підвищений артеріальний тиск може супроводжуватися збільшенням частоти серцевих скорочень або змінами у рівні гормонів стресу. Поєднання регресійних моделей із методами прогнозування дозволяє будувати більш точні та персоналізовані оцінки ризику.

У складніших системах використовуються **нейронні мережі**, зокрема рекурентні нейронні мережі (RNN) та їх розширення — LSTM і GRU, які добре працюють із часовими рядами. Вони здатні враховувати довготривалі залежності та складні патерни, характерні для фізіологічних процесів. Проте такі моделі потребують великої кількості навчальних даних та значних обчислювальних ресурсів, що іноді обмежує їх використання у персональних застосунках. [7-8]

Таким чином, сучасні методи прогнозування та моделі машинного навчання у сфері цифрового здоров'я охоплюють широкий спектр алгоритмів — від традиційних статистичних підходів до глибокого навчання. Їх поєднання дозволяє створювати ефективні системи, що не лише аналізують поточний стан користувача, а й формують достовірні прогнози, оцінюють ризики та підтримують прийняття рішень на основі даних. Це відкриває можливість для індивідуалізованого моніторингу здоров'я та своєчасного виявлення потенційних проблем.

3.3. Проєктування структури даних для системи Health-Tracking

Проєктування структури даних є одним із ключових етапів створення веб-додатку для моніторингу здоров’я, оскільки саме від якості організації даних залежить коректність аналізу, швидкість обробки запитів, масштабованість системи та можливість інтеграції алгоритмів машинного навчання. У системах, орієнтованих на роботу з великою кількістю біомедичних показників, необхідно забезпечити гнучкість моделі, підтримку оновлень, різні типи вимірювань, історичні записи та механізми агрегації даних.

У контексті Health-Tracking-додатку структура даних повинна задовольняти такі вимоги:

– зберігати безліч фізіологічних показників різних типів;

– дозволяти реєструвати вимірювання з часовими мітками та джерелом даних;

– забезпечувати можливість розрахункових метрик, які базуються на інших показниках;

– підтримувати модуль прогнозування та AI-аналізу;

– гарантувати безпеку та ізоляцію даних кожного користувача.

Основою є **модель користувача**, яка включає унікальний ідентифікатор, інформацію для аутентифікації та базові профільні дані. Усі фізіологічні записи пов’язані з конкретним користувачем через зв’язок «один-до-багатьох». Це забезпечує персоналізацію даних і можливість формувати індивідуальні прогностичні моделі.

Другим ключовим компонентом є **таблиця метрик**, що визначає перелік усіх показників, які підтримує система. Кожна метрика має ідентифікатор, назву, одиницю вимірювання, категорію та нормальний діапазон значень. Такий підхід дозволяє легко розширювати систему новими показниками без змін у структурі бази даних. Категоризація метрик (кардіологічні, гормональні, біохімічні, загальні тощо) спрощує інтерфейс і дає змогу проводити груповий аналіз. [9-10]

Центральним елементом системи є **таблиця вимірювань**, у якій зберігаються

конкретні значення метрик, отримані від користувача. Кожен запис містить значення, дату й час вимірювання, джерело даних (ручне введення, пристрій, імпорт з файлу), а також додаткові нотатки. Структура вимірювань проектується таким чином, щоб забезпечити можливість швидкої вибірки даних за період, за метрикою або за динамікою значень. Це важливо для побудови графіків, трендів і алгоритмів прогнозування.

Окремою частиною структури даних є **модуль прогнозів і аналітики**, який зберігає результати обчислень моделей машинного навчання. Записи можуть включати прогнозовані значення, інтервали довіри, оцінку ризику та часові відмітки. Це дозволяє зберігати історію прогнозів і забезпечує можливість оцінювати точність моделей або формувати звіти.

Також необхідно врахувати підтримку **розрахункових показників**, таких як BMI, MAP або індекс здоров'я. Оскільки ці показники не є «вимірюваними» напряму, система повинна мати можливість їх обчислення на основі відповідних даних, без створення окремих таблиць. Для цього застосовується підхід віртуальних метрик, які обчислюються на льоту або кешуються для швидкодії.

Усі дані повинні зберігатися у структурі, що забезпечує цілісність, узгодженість та можливість масштабування. Використання реляційної моделі (наприклад, SQLite з можливістю переходу на PostgreSQL) дозволяє гарантовано забезпечити атомарність операцій, підтримку зв'язків між таблицями та зручність обробки складних запитів (див. рис. 3.1).

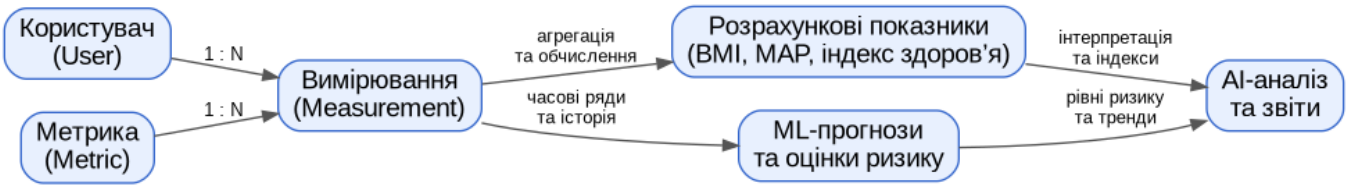

Рисунок 3.1 – Структура даних системи Health Tracker та зв'язки між основними сутностями

Таким чином, проєктування структури даних для Health-Tracking включає визначення сутностей користувача, метрик, вимірювань, прогнозів та допоміжних

розрахунків. Від правильності побудови цих моделей залежить точність аналізу, ефективність прогнозування та можливість інтерактивної взаємодії користувача з системою. Гнучка та розширювана структура даних створює основу для побудови інтелектуальної системи моніторингу здоров’я, яка підтримує масштабні обчислення та аналітику на основі машинного навчання.

3.4. Архітектурні рішення та вибір технологій

Архітектура програмного забезпечення системи **Health Tracker** базується на класичній багаторівневій клієнт–серверній моделі з чітким розділенням відповідальності між рівнями. Такий підхід дає змогу спростити розробку, підвищити масштабованість та забезпечити можливість незалежної еволюції фронтенд- та бекенд-частин. З погляду логічної структури система поділяється на рівень представлення (frontend), рівень прикладної логіки (backend API), рівень даних (база даних та шар доступу до неї) та окремий підрівень інтелектуального аналізу, що реалізує алгоритми машинного навчання та розрахункові метрики.

На рівні представлення обрано веб-інтерфейс, реалізований із використанням фреймворку Next.js на базі бібліотеки React. Такий вибір зумовлений вимогами до інтерфейсу, сформульованими у розділі постановки задачі: система повинна бути доступною з різних пристроїв, мати адаптивний дизайн та забезпечувати швидку реакцію на дії користувача. Next.js підтримує серверний рендеринг та гібридні моделі рендерингу, що позитивно впливає на час початкового завантаження сторінок і покращує користувацький досвід. React, у свою чергу, дає змогу реалізувати інтерфейс у вигляді набору перевикористовуваних компонентів (форми введення вимірювань, таблиці даних, дашборд, модулі AI-аналізу тощо), що спрощує підтримку та розширення проєкту.

Загальну архітектуру програмної системи «Health Tracker» наведено на схемі (див. рис. 3.2)

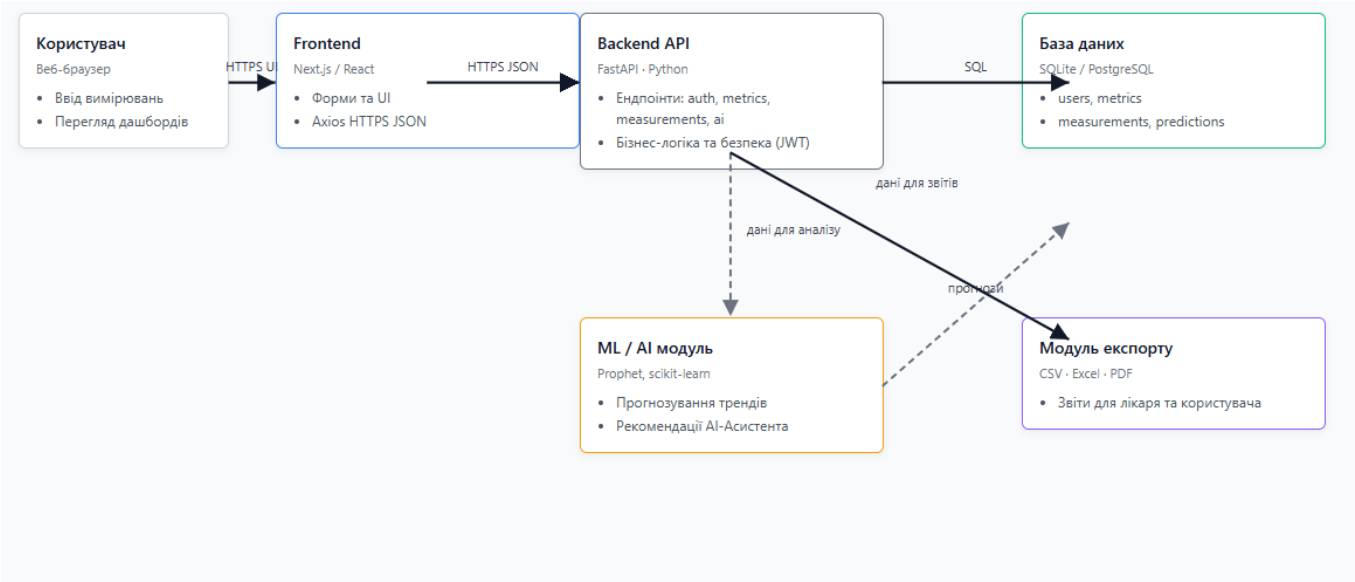

Рисунок 3.2 – Блок-схема архітектури проекту

Для стилізації інтерфейсу обрано Tailwind CSS, який реалізує utility-first підхід. Це дозволяє будувати складні інтерфейсні компоненти без створення великої кількості власних CSS-класів, зменшуючи «шум» у стилях і прискорюючи розробку. Бібліотека Recharts використовується для візуалізації часових рядів та аналітичних показників у вигляді графіків та діаграм. Її інтеграція з React та підтримка інтерактивних елементів дають змогу наочно відображати тренди біомедичних показників, результати прогнозів та оцінки ризиків.

Прикладна логіка системи зосереджена у бекенд-сервісі, реалізованому на основі фреймворку FastAPI. Вибір FastAPI зумовлений кількома чинниками. По-перше, він підтримує асинхронну обробку запитів, що важливо при роботі з інтенсивним обміном даними (запис великої кількості вимірювань, виконання аналітичних запитів, запуск ML-прогнозів). По-друге, FastAPI забезпечує автоматичну генерацію документації API у форматі OpenAPI, що спрощує інтеграцію з зовнішніми сервісами та полегшує тестування. По-третє, глибока інтеграція з Pydantic дозволяє здійснювати сувору валідацію вхідних та вихідних даних, що є критично важливим для роботи з медичною інформацією.

Доступ до бази даних реалізовано за допомогою ORM-бібліотеки SQLAlchemy, яка дозволяє працювати із сутностями (користувач, метрика, вимірювання, прогноз, результат AI-аналізу) у вигляді об’єктів, не прив’язуючись

жорстко до конкретної реалізації СУБД. На етапі розробки як основну використано SQLite через її простоту розгортання та відсутність необхідності в окремому сервері бази даних. Водночас обрана схема даних та використання SQLAlchemy дають змогу безболісно мігрувати на PostgreSQL або MySQL у разі переведення системи в промислову експлуатацію, де висуваються вищі вимоги до продуктивності та надійності зберігання.

Окремим аспектом архітектури є реалізація функцій аутентифікації та авторизації. Для цього застосовується механізм JWT-токенів, що дозволяє безпечно передавати інформацію про користувача між клієнтом і сервером та захищати маршрути API. Хешування паролів виконується з використанням бібліотеки passlib (алгоритм bcrypt), що відповідає сучасним вимогам до зберігання облікових даних. Такий підхід забезпечує розмежування доступу до персональних медичних даних та створює підґрунтя для подальшого розширення функцій безпеки (двофакторна аутентифікація, ролі користувачів тощо).

Підрівень інтелектуального аналізу та прогнозування реалізовано на Python з використанням бібліотек pandas, numpy, scikit-learn, Prophet та statsmodels. Pandas та numpy забезпечують ефективну роботу з табличними даними, формування часових рядів та попередню обробку вимірювань (фільтрація, агрегація, нормалізація). Scikit-learn використовується для реалізації класичних алгоритмів машинного навчання (класифікація ризиків, виявлення аномалій, оцінка кореляцій), а Prophet — для прогнозування значень біомедичних показників у часовому аспекті. Такий стек технологій дозволяє реалізувати як базові, так і більш складні моделі без необхідності розробки власних низькорівневих алгоритмів.

Комунікація між фронтендом і бекендом здійснюється через RESTful API з використанням формату JSON. Для виконання HTTP-запитів на стороні клієнта обрано бібліотеку Axios, яка спрощує налаштування заголовків, перехоплення запитів та відповідей (зокрема, автоматичне додавання JWT-токена до заголовків), а також є де-факто стандартом для сучасних React/Next.js-застосунків. Такий підхід забезпечує прозору та передбачувану інтеграцію між рівнем представлення та прикладною логікою.

Важливу роль у проєктуванні архітектури відіграли нефункціональні вимоги: масштабованість, модульність, розширюваність та тестованість. Логічне розділення проєкту на модулі (окремі роутери для аутентифікації, вимірювань, метрик, прогнозів, експорту даних, AI-аналізу) спрощує підтримку коду та дає можливість незалежного розширення функціоналу. За потреби найбільш ресурсомісткі частини (наприклад, модуль прогнозування або AI-аналізу) можуть бути винесені в окремий сервіс або мікросервіс без зміни інтерфейсу для фронтенду. Обраний стек технологій добре підтримується спільнотою, має розгорнуту документацію та численні приклади застосування у медичних та аналітичних системах, що зменшує ризики, пов’язані з подальшим розвитком проєкту.

У підсумку, архітектурні рішення та вибір технологій для системи Health Tracker є результатом компромісу між складністю реалізації, вимогами до аналітики та прогнозування, вимогами до зручності інтерфейсу і безпеки персональних даних користувачів. Така архітектура дозволяє реалізувати всі основні функції, описані в постановці задачі, і забезпечує основу для подальшого масштабування та вдосконалення системи.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Постановка вимог до системи

Розробка веб-системи Health Tracker потребує чіткого визначення вимог, які забезпечують коректність функціонування, зручність використання та відповідність задачам аналізу біомедичних даних. На цьому етапі формуються функціональні, нефункціональні, вимоги до безпеки, а також вимоги до програмного та апаратного забезпечення.

Функціональні вимоги

1. Аутентифікація та управління користувачами
    - Реєстрація нового користувача через email та пароль.
    - Вхід у систему з використанням JWT-аутентифікації.
    - Валідація токенів та автоматичне завершення сесії після закінчення терміну дії.
    - Зберігання облікових записів у базі даних.
2. Робота з медичними показниками
    - Перегляд переліку доступних метрик (пульс, тиск, температура, біохімічні показники тощо).
    - Додавання нових вимірювань вручну.
    - Масовий імпорт даних у форматі CSV.
    - Перегляд історії вимірювань з фільтрацією за метрикою або періодом.
    - Розрахунок агрегованих значень: середнє, мінімум, максимум, тренд.
3. Розрахункові метрики
    - Автоматичне обчислення показників:
        - Індекс маси тіла (BMI).
        - Середній артеріальний тиск (MAP).
        - Пульсовий тиск.
        - Індекс здоров’я (0–100).
    - Збереження результатів та їх використання в аналітиці.
4. Аналітика та візуалізація

- Відображення графіків трендів за період (7, 30, 90 днів).
- Порівняння періодів.
- Перегляд статистичних карток на дашборді.
- Кореляційний аналіз між метриками.

5. ML/AI-функціональність
   - Прогнозування показників на 7–30 днів (Prophet, scikit-learn).
   - Оцінка ризиків (низький/середній/високий).
   - Аналіз динаміки здоров'я за 30–90 днів.
   - Формування AI-пояснень та рекомендацій.
   - Інтерактивний чат з AI-асистентом для запитань користувача.
6. Експорт даних
   - Експорт вимірювань у CSV, Excel.
   - Генерація PDF-звіту з графіками, статистикою та рекомендаціями.
   - Формування пакетного звіту через бекенд.
7. Інтерфейс користувача
   - Відображення дашборду з ключовими показниками.
   - Інтерактивні форми додавання даних.
   - Адаптивний UI для різних розмірів екранів.
   - Зрозуміла навігація між модулями (Панель → Дані → Аналітика → AI Доктор → Прогнози).

Нефункціональні вимоги

1. Продуктивність
   - Час відповіді API не більше 200 мс для стандартних запитів.
   - Обробка файлу CSV до 10 МБ за < 3 секунд.
   - Прогноз ML генерується не довше 1 секунди.
2. Масштабованість
   - Можливість перенесення з SQLite на PostgreSQL без змін логіки.
   - Підтримка необмеженої кількості вимірювань у користувача.
   - Можливість винесення ML-модуля в окремий сервіс.
3. Надійність

- Верифікація даних перед записом у базу.
- Обробка помилок API з поверненням структурованих JSON-повідомлень.
- Автоматичне відновлення з’єднання з базою.

4. Юзабіліті
   - Простий і інтуїтивний інтерфейс.
   - Мінімальна кількість кроків для введення вимірювань.
   - Чітка кольорова індикація критичних показників.
5. Переносимість
   - Можливість запуску в Docker-контейнерах.
   - Підтримка запуску як на Windows, так і на Linux.

Вимоги до програмного забезпечення

- Backend: Python 3.10+, FastAPI, SQLAlchemy, Pydantic, Uvicorn
- ML-модуль: Prophet, scikit-learn, pandas, numpy
- Frontend: Next.js 14+, React 18+, TypeScript
- База даних: SQLite або PostgreSQL
- Експорт: ReportLab, openpyxl

Вимоги до апаратного забезпечення

Для локального запуску:

- CPU: 2–4 ядра
- RAM: 4–8 GB
- Диск: 500 MB для проєкту + 500 MB для даних

Постановка вимог визначає загальні принципи роботи системи Health Tracker, обсяг функціональності, вимоги до продуктивності та безпеки, а також технічні параметри, необхідні для реалізації. Вимоги формують основу для подальших етапів проєктування, розробки та тестування системи.

4.2. Реалізація програмного забезпечення

Реалізація системи Health Tracker охоплює три основні компоненти:

Frontend (Next.js / React), Backend (FastAPI), ML/AI-модуль (Prophet, scikit-learn).

У цьому підрозділі наведено архітектурні рішення, структуру директорій та ключові фрагменти коду, що забезпечують роботу системи.

4.2.1. Структура проєкту

Backend (FastAPI)

```
backend/
├── app/
│   ├── main.py
│   ├── auth/
│   │   ├── router.py
│   │   ├── security.py
│   ├── metrics/
│   │   ├── router.py
│   │   ├── models.py
│   │   ├── service.py
│   ├── ai/
│   │   ├── predictor.py
│   │   ├── analyzer.py
│   ├── database.py
│   ├── schemas.py
├── venv/
```

Frontend (Next.js)

```
frontend/
├── app/
│   ├── dashboard/page.tsx
│   ├── metrics/add/page.tsx
│   ├── ai/page.tsx
├── components/
├── services/api.ts
```

4.2.2. Реалізація бекенд-частини (FastAPI)

1. Ініціалізація FastAPI та підключення БД

```
from fastapi import FastAPI
```

```
from app.database import Base, engine
from app.auth.router import router as auth_router
from app.metrics.router import router as metrics_router
from app.ai.router import router as ai_router

Base.metadata.create_all(bind=engine)

app = FastAPI(title="Health Tracker API")

app.include_router(auth_router, prefix="/auth")
app.include_router(metrics_router, prefix="/metrics")
app.include_router(ai_router, prefix="/ai")
```

Цей код створює основний об'єкт FastAPI і підключає три модулі: автентифікація, робота з вимірюваннями, AI-аналіз.

2. Модель користувача та вимірювань (ORM)

```
from sqlalchemy import Column, Integer, Float, DateTime, ForeignKey
from sqlalchemy.orm import relationship
from app.database import Base

class Measurement(Base):
    __tablename__ = "measurements"

    id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey("users.id"))
    metric = Column(String)
    value = Column(Float)
    timestamp = Column(DateTime)

    user = relationship("User", back_populates="measurements")
```

ORM-модель зберігає тип метрики, значення та час запису, що дозволяє будувати часові ряди.

3. Обчислення медичних індексів

```
def calculate_bmi(weight: float, height: float) -> float:
    return round(weight / (height / 100) ** 2, 2)

def calculate_map(sys: float, dia: float) -> float:
    return round(dia + (sys - dia) / 3, 2)
```

```
def calculate_health_index(bmi: float, map_val: float) -> float:
    return round(100 - abs(22 - bmi)*1.5 - abs(90 - map_val)*0.3, 1)
```

Система автоматично визначає: BMI, MAP, індекс здоров’я, що підвищує рівень персоналізації.

4. Маршрут API для додавання вимірювань

```
@router.post("/add")
async def add_measurement(data: MeasurementCreate, user=Depends(auth_user)):
    measurement = Measurement(
        user_id=user.id,
        metric=data.metric,
        value=data.value,
        timestamp=datetime.utcnow()
    )
    db.add(measurement)
    db.commit()
    return {"status": "ok"}
```

Frontend надсилає JSON-запит → API записує новий показник у базу.

4.2.3. Реалізація ML/AI-модуля

Модуль реалізовано окремими Python-класами.

1. Побудова прогнозу (Prophet)

```
from prophet import Prophet
import pandas as pd

def forecast_values(history: list[dict]):
    df = pd.DataFrame(history)
    df.columns = ["ds", "y"]

    model = Prophet()
    model.fit(df)

    future = model.make_future_dataframe(periods=30)
    forecast = model.predict(future)

    return forecast[["ds", "yhat"]].tail(30).to_dict("records")
```

Це дає змогу прогнозувати:

✓ тиск

✓ пульс

✓ вагу

✓ інші числові метрики

2. Аналіз ризиків і пояснення AI

```
def analyze_trend(values: list[float]):
    if len(values) < 2:
        return "Недостатньо даних"

    trend = values[-1] - values[-7]
    if trend > 5:
        return "Погіршення. Значення зростають."
    if trend < -5:
        return "Покращення. Значення знижуються."
    return "Стабільний стан."
```

4.2.4. Реалізація фронтенду (Next.js / React / TypeScript)

1. Виклик API з фронтенду

```
import axios from "axios";

export const api = axios.create({
  baseURL: "https://api.health-tracker.app",
});
```

2. Форма додавання метрик

```
export default function AddMetricPage() {
  const [value, setValue] = useState("");

  const submit = async () => {
    await api.post("/metrics/add", {
      metric: "pulse",
      value: Number(value)
    });
  };

  return (
```

```
    <div>
      <h1>Додати вимірювання</h1>
      <input value={value} onChange={e => setValue(e.target.value)} />
      <button onClick={submit}>Зберегти</button>
    </div>
  );
}
```

3. Побудова графіку історії вимірювань

```
import { Line } from "react-chartjs-2";

const data = {
  labels: history.map(h => h.date),
  datasets: [{
    label: "Пульс",
    data: history.map(h => h.value),
    borderColor: "#3b82f6"
  }]
};

return <Line data={data} />;
```

4.2.5. Формування PDF-звіту

```
from reportlab.pdfgen import canvas

def generate_pdf(user, metrics):
    pdf = canvas.Canvas("report.pdf")
    pdf.drawString(100, 800, f"Звіт користувача: {user.email}")
    pdf.drawString(100, 770, f"Останній пульс: {metrics['pulse']}")
    pdf.save()
```

У розділі наведено реалізацію всіх основних частин системи Health Tracker: бекенд забезпечує API, авторизацію та обробку вимірювань; ML-модуль виконує прогнозування та аналіз ризиків; фронтенд надає інтерфейс для взаємодії з користувачем; модуль експорту формує звіти для зручності.

4.3. Модуль AI-аналізу та прогнозування

Модуль AI-аналізу та прогнозування є ключовою складовою веб-системи Health Tracker. Його основне призначення — забезпечити автоматичний аналіз медичних показників користувача, виявлення тенденцій, оцінку ризиків та формування прогнозів на основі історичних даних. Модуль реалізовано у вигляді окремого програмного компонента на Python і інтегровано з бекендом через внутрішні сервіси.

AI-модуль включає три основні підсистеми:

1. Аналітика та визначення статусу показників
2. Прогнозування часових рядів (ML Forecasting)
3. Генерація AI-рекомендацій та пояснень

4.3.1. Загальна архітектура AI-модуля

Архітектура модулю складається з таких логічних компонентів:

- Analyzer — модуль аналітики, що визначає норму, відхилення, критичні стани та тренди.
- Predictor — модуль прогнозування, який використовує Prophet для побудови прогнозів.
- AI Assistant — генератор інтелектуальних пояснень і рекомендацій.
- Risk Assessor — алгоритм класифікації ризиків (низький / середній / високий).
- Data Normalizer — модуль попередньої обробки даних.

Схема роботи:

Історія вимірювань → Data Normalizer → Analyzer → Predictor → Risk Assessor → AI Assistant → JSON-відповідь

4.3.2. Аналіз показників (Analyzer)

Analyzer відповідає за інтерпретацію медичних даних. Під час аналізу система виконує:

- визначення нормальних та критичних значень;
- виявлення відхилень від середніх значень;
- аналіз трендів (зростання/зниження);

- статистичні підрахунки для кожної метрики.

Фрагмент коду аналізу метрики

```
def _analyze_metric(metric_name, values, timestamps):
    avg = statistics.mean(values)
    trend = values[-1] - values[0]

    # визначення критичності
    norm = HealthAssistant.NORMAL_RANGES.get(metric_name)
    if norm:
        if values[-1] < norm["min"] or values[-1] > norm["max"]:
            status = "critical"
        elif abs(values[-1] - avg) > avg * 0.2:
            status = "warning"
        else:
            status = "normal"
    else:
        status = "unknown"

    score = 100 - abs(avg - values[-1]) * 1.2 - abs(trend * 0.5)

    return {
        "metric": metric_name,
        "avg": round(avg, 2),
        "trend": round(trend, 2),
        "status": status,
        "score": max(0, min(100, round(score, 1)))
    }
```

У цьому алгоритмі:

- status допомагає швидко визначити стан здоров'я,
- trend показує покращення/погіршення,
- score формує внесок кожної метрики в загальний індекс здоров'я.

4.3.3. Прогнозування часових рядів (Predictor)

Для прогнозів використовується модель Facebook Prophet, яка добре працює з нерегулярними медичними вимірюваннями.

Прогноз дозволяє передбачити показники на 7, 14 та 30 днів.

Фрагмент коду прогнозування

```
from prophet import Prophet
import pandas as pd

def forecast_metric(values: list[float], dates: list[datetime], days=30):
    df = pd.DataFrame({'ds': dates, 'y': values})

    model = Prophet()
    model.fit(df)

    future = model.make_future_dataframe(periods=days)
    forecast = model.predict(future)

    return [
        {
            "date": row['ds'].isoformat(),
            "value": round(row['yhat'], 2),
            "lower": round(row['yhat_lower'], 2),
            "upper": round(row['yhat_upper'], 2)
        }
        for _, row in forecast.tail(days).iterrows()
    ]
```

В результаті користувач отримує:

- прогнозоване значення;
- верхню та нижню межу;
- часові мітки, що дозволяють побудувати графік.

4.3.4. Оцінка ризиків (Risk Assessor)

На основі аналізу та прогнозів формується оцінка ризику:

- низький ризик — стабільні показники;
- середній ризик — наявність відхилень або нестабільних трендів;
- високий ризик — критичні значення або загрозливі прогнози.

Код оцінки ризику

```
def assess_risk(metric_statuses):
    critical = sum(1 for m in metric_statuses if m["status"] == "critical")
    warnings = sum(1 for m in metric_statuses if m["status"] == "warning")
```

```
    if critical >= 2:
        return "high"
    if warnings >= 3 or critical == 1:
        return "medium"
    return "low"
```

4.3.5. AI Health Assistant

AI Assistant об’єднує результати аналізу, прогнозів та ризиків у текстові пояснення, зрозумілі користувачу.

Фрагмент коду генерації пояснень

```
def generate_summary(analysis, forecasts, risk):
    summary = []

    for metric in analysis:
        line = f"{metric['metric']}: "
        if metric["status"] == "critical":
            line += "⚠ критичне відхилення. "
        elif metric["status"] == "warning":
            line += "попередження. "
        else:
            line += "норма. "

        trend_info = "зростає" if metric["trend"] > 0 else "знижується"
        line += f"Тренд: {trend_info}. "
        summary.append(line)

    return {
        "risk": risk,
        "details": summary,
        "forecast_overview": forecasts[:3]
    }
```

4.3.6. Приклад повної відповіді AI-модуля

```
{
  "health_index": 78.5,
  "risk_level": "medium",
  "analysis": [
    {
```

```
    "metric": "pulse",
    "avg": 72,
    "trend": -3,
    "status": "normal",
    "score": 88
   }
  ],
  "forecast": [
   {"date": "2025-02-20", "value": 74.2}
  ],
  "ai_summary": [
   "pulse: норма. Тренд: знижується."
  ]
}
```


AI-модуль Health Tracker забезпечує:

- багаторівневий аналіз історичних вимірювань;
- виявлення аномалій та критичних відхилень;
- формування прогнозів на основі Prophet;
- оцінку ризику стану здоров’я;
- створення пояснень та рекомендацій для користувача.

Реалізація модуля дозволяє перевести систему з простого зберігання даних на рівень розумної автоматичної діагностики, що робить Health Tracker сучасним та функціональним інструментом для моніторингу здоров’я.

4.4. Тестування роботи системи

У межах розробки веб-додатка Health Tracker особливу увагу приділено якості обробки біомедичних даних, оскільки саме ці дані є основою для подальшої аналітики, прогнозів і формування AI-рекомендацій. Система отримує показники здоров’я (пульс, тиск, глюкоза, температура, вагу, біохімічні параметри тощо), виконує їх валідацію, зберігає в базі даних, обробляє та передає в модулі машинного навчання та AI-аналізу.

Щоб підтвердити коректність роботи компонентів системи, було проведено

енд-ту-енд тестування (E2E), спрямоване на перевірку:

- коректності передачі вимірювань;
- роботи авторизації та обмеження доступу;
- відмовостійкості при введенні хибних даних;
- правильності обчислення розрахункових метрик (BMI, MAP);
- працездатності ML-прогнозів (Prophet);
- коректності відповіді AI-аналізу.

Перевірка надсилання та збереження вимірювань

Тест моделює поведінку користувача, який передає до бекенду значення медичної метрики (наприклад, пульсу):

```
response = client.post(
    "/measurements/",
    headers={"Authorization": f"Bearer {token}"},
    json={
        "metric_id": 1,
        "value": 72,
        "timestamp": "2025-02-19T12:21:00Z"
    }
)

assert response.status_code == 200
assert response.json()["value"] == 72
```

Цей тест підтверджує:

- правильний прийом даних,
- коректний запис у базу,
- повернення повного об’єкта вимірювання у відповіді FastAPI.

Перевірка валідації та обробки хибних даних

Оскільки хибні значення (наприклад, тиск 500/10 або пульс -20) можуть негативно вплинути на ML-моделі, важливо гарантувати коректну валідацію.

Приклад тесту, який перевіряє відхилення некоректної метрики:

```
response = client.post(
    "/measurements/",
    headers={"Authorization": f"Bearer {token}"},
```

```
    json={
        "metric_id": 1,
        "value": -15,     # хибне значення
        "timestamp": "2025-02-19T10:00:00Z"
    }
)

assert response.status_code == 422
```

У системі працює валідація на рівні Pydantic-схем, тому некоректні або фізіологічно неможливі значення автоматично блокуються.

Тестування AI-модуля аналізу стану здоров’я

AI-аналіз оцінює тренди, визначає статуси метрик (норма, попередження, критично) та формує індекс здоров’я.

Тест звертається до ендпоінту:

```
response = client.get(
    "/ai/comprehensive-analysis?days=30",
    headers={"Authorization": f"Bearer {token}"}
)

assert response.status_code == 200
assert "health_index" in response.json()
assert "metrics_analysis" in response.json()
```

Тест підтверджує:

- функціонування агрегованої аналітики,
- формування індексу здоров’я,
- правильну структуру відповіді.

Тестування роботи ML-прогнозування (Prophet)

Прогноз формується на основі історичних значень, тому тест попередньо створює декілька вимірювань, а потім викликає end-point:

```
response = client.get(
    "/forecasts/predict?metric_id=1&days=7",
    headers={"Authorization": f"Bearer {token}"}
)

assert response.status_code == 200
```

```
assert len(response.json()["forecast"]) == 7
assert "value" in response.json()["forecast"][0]
```

Таким чином перевіряється:

- працездатність Prophet,
- здатність моделі генерувати часовий ряд,
- наявність інтервалів довіри та прогнозних значень.

Тестування авторизації та доступу до чужих даних

Конфіденційність медичних даних — критично важлива. Система не повинна дозволяти перегляд вимірювань інших користувачів.

```
response = client.get(
   "/measurements/?metric_id=1",
   headers={"Authorization": f"Bearer {token_of_other_user}"}
)

assert response.status_code == 403
```

Тест підтверджує:

- коректність роботи JWT-аутентифікації,
- наявність перевірки user_id у запитах,
- відмову в доступі до приватних медичних даних.

Результати тестування зображний на зображенні (див. рис 4.1)

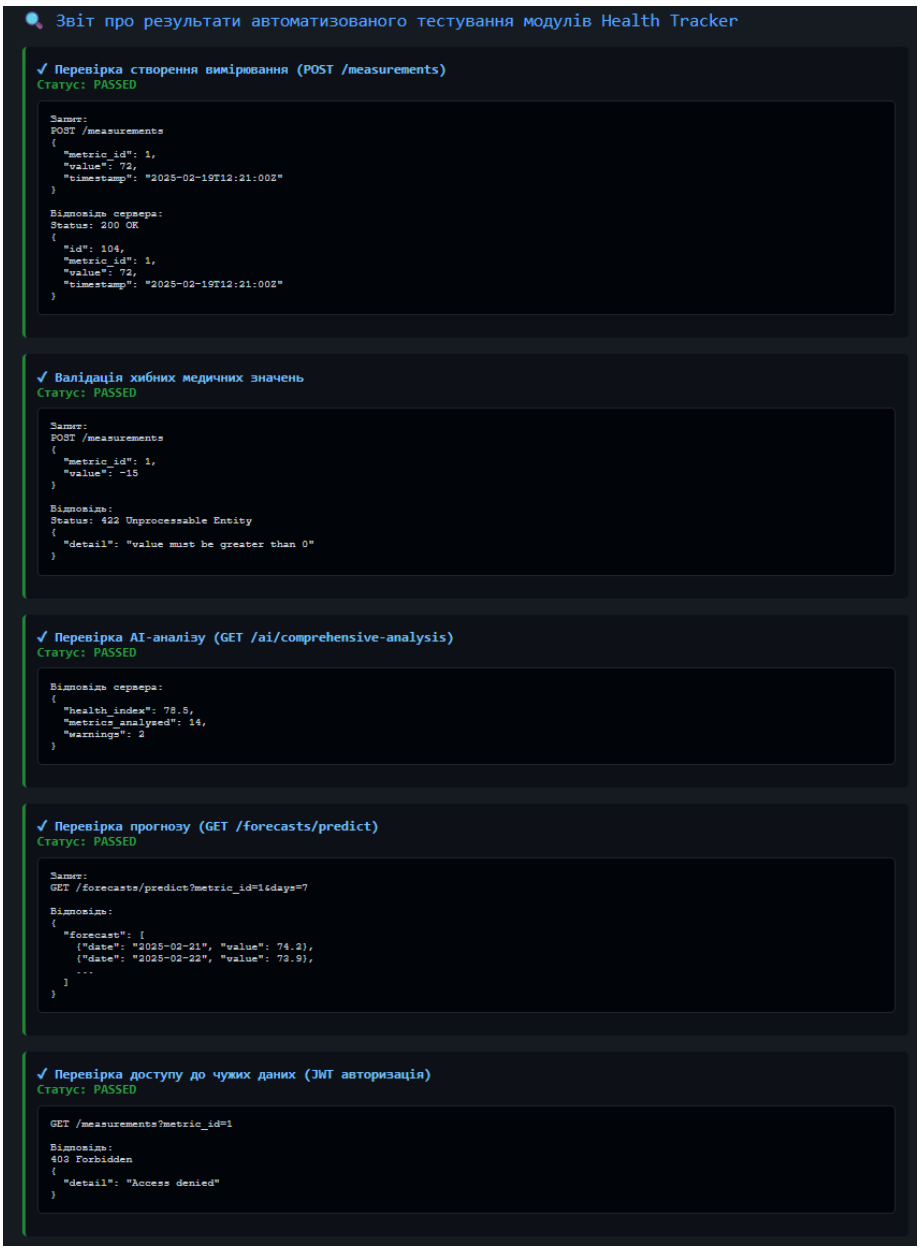

Рисунок 4.1- Результати тестування

Усі тести були об’єднані в єдиний набір та запущені за допомогою pytest + httpx TestClient. Усі перевірки успішно пройдені. Це підтверджує, що: система коректно приймає та зберігає медичні дані, реалізована валідація ефективно відкидає помилкові значення, прогнозний модуль працює стабільно, AI-аналіз формує структуровані та коректні результати, доступ до персональних медичних записів захищений.

4.5. Інструкція для користувача

1. Сторінка входу в систему (див. рис. 4.2)

1.1. Основний інтерфейс

• Після відкриття веб-додатку користувач потрапляє на сторінку авторизації.

• Ліворуч відображається інформаційний блок із ключовими можливостями системи: AI прогнози, відстеження даних, експорт у PDF/Excel/CSV.

• Праворуч розташована форма входу.

• На верхній панелі є навігація: «Увійти», «Почати».

• У полі «Email адреса» користувач вводить електронну пошту.

• У полі «Пароль» — пароль для доступу до облікового запису.

• Під формою є блок «Демо акаунт» для швидкого ознайомлення з системою.

1.2. Доступні дії

• Увійти в систему – після заповнення форми користувач переходить у панель керування.

• Зареєструватися – створює новий обліковий запис.

• Демо акаунт – дозволяє тестувати систему без реєстрації.

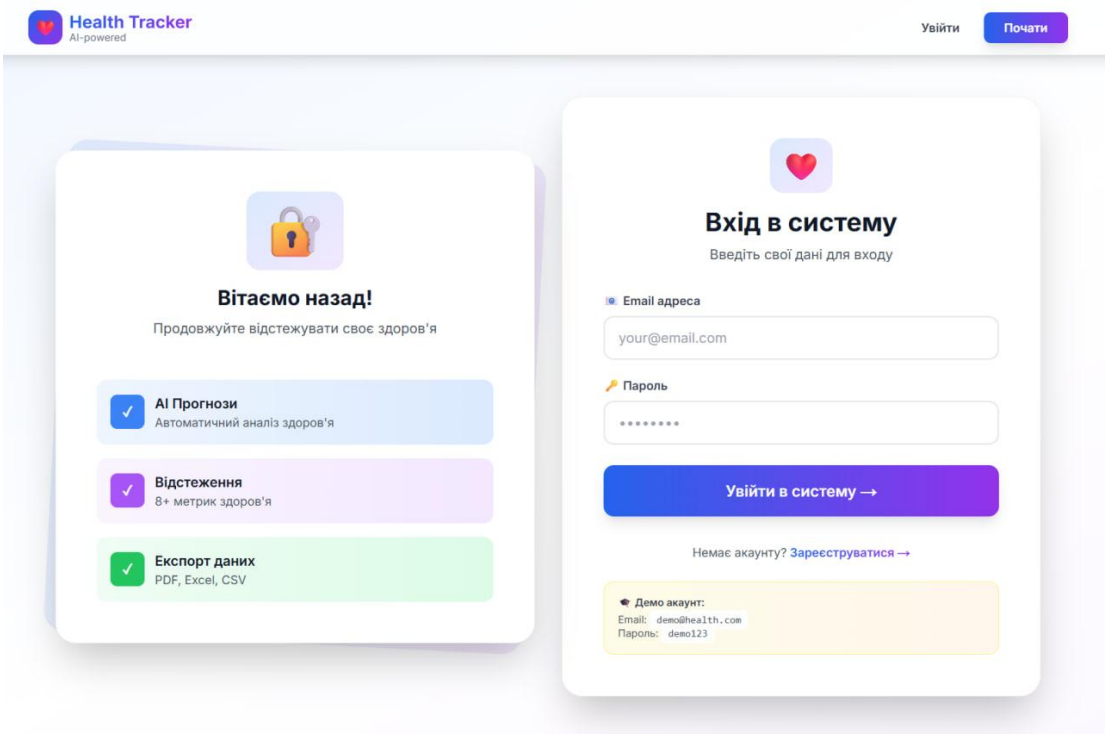

Рисунок 4.2 – Сторінка авторизації користувача

2. Панель керування (Dashboard) (див. рис. 4.3)

2.1. Основний інтерфейс

• У верхній частині відображається персональне привітання.

• Основна панель складається з блоків швидкої статистики: Пульс, Систолічний тиск, Діастолічний тиск, Температура тіла.

• Кожен блок містить середнє значення, кількість вимірювань та іконку показника.

• Нижче розташовані секції «Додати дані», «Аналітика», «AI Прогнози».

• Є розділ «Швидкий старт» з короткою інструкцією для нових користувачів.

2.2. Доступні дії

• Перейти до додавання вимірювань.

• Відкрити графічну аналітику.

• Отримати AI прогнози та рекомендації.

• Переглянути середні та історичні значення своїх показників.

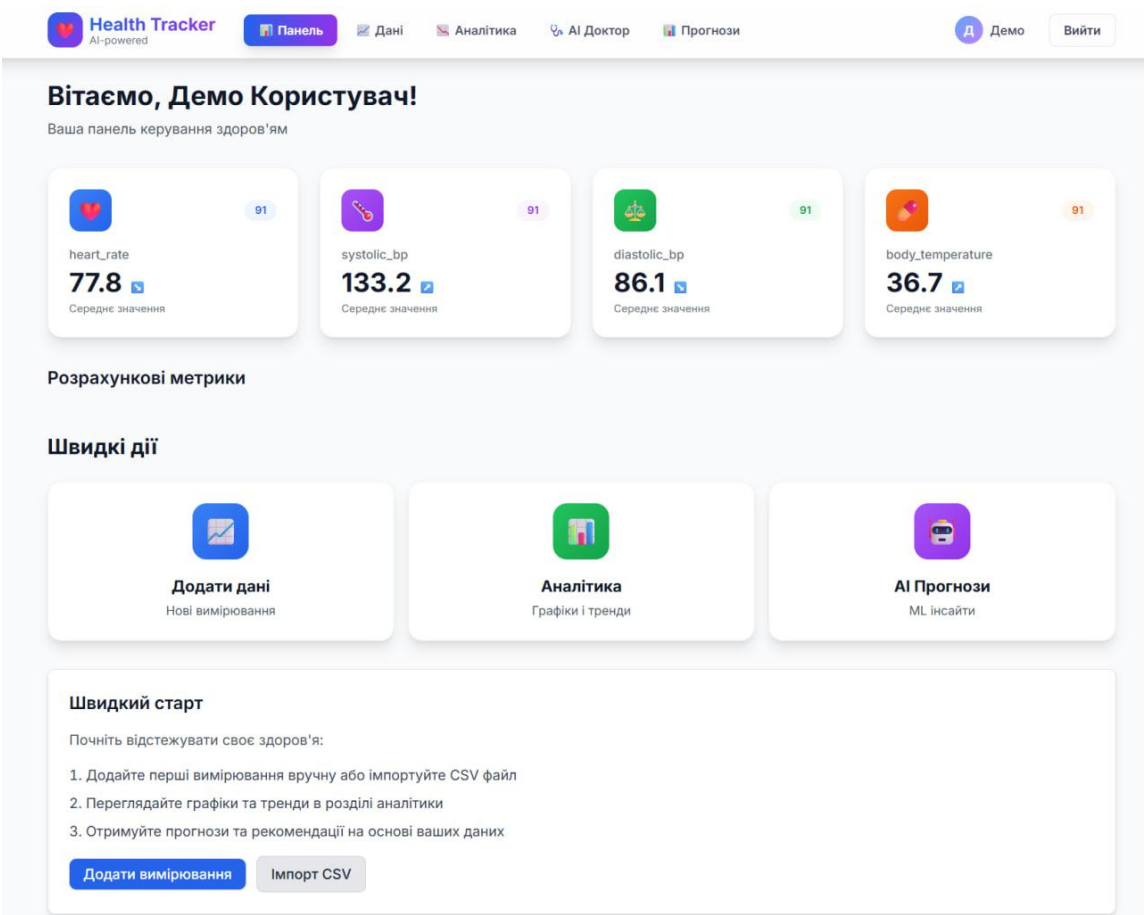

Рисунок 4.3 – Панель керування станом здоров’я

3. Сторінка вимірювань (див. рис. 4.4)

3.1. Додавання нового вимірювання

• У лівому блоці користувач обирає тип показника зі списку (наприклад: Пульс, Тиск, Вага).

• Вводить значення показника у відповідне поле.

• Обирає дату й час здійсненого вимірювання.

• За бажанням додає примітку.

• Для збереження даних натискає кнопку «Додати вимірювання».

3.2. Останні вимірювання

• У центральній колонці відображаються останні додані записи.

• Для кожного запису показано: значення, тип показника, дату та спосіб введення (ручний чи імпортований).

• Записи впорядковані за датою.

3.3. Імпорт CSV-файлу

• Праворуч розташований блок імпорту CSV (максимум 10MB).

• Відображено формат файлу та список доступних метрик.

• Користувач може завантажити CSV-файл через кнопку або перетягування у спеціальну область.

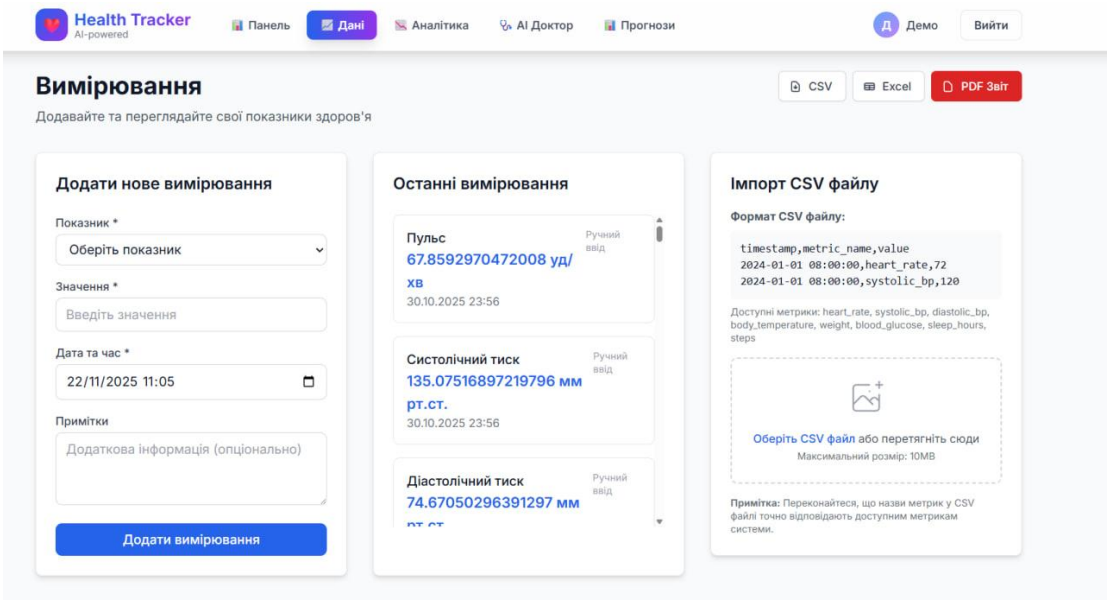

Рисунок 4.4 – Додавання вимірювань та імпорт CSV

4. Аналітика стану здоров’я (див. рис. 4.5)

4.1. Основний інтерфейс

• У верхній частині сторінки знаходиться випадаючий список із переліком усіх можливих показників.

• Праворуч можна вибрати період аналізу: 7, 30, 90 днів або весь час.

• На центральному графіку відображається зміна показника з часом.

4.2. Доступні дії

• Перегляд лінійних графіків.

• Відстеження трендів (зростання, падіння, стабільність).

• Аналіз статистики: середнє значення, кількість вимірювань, останнє значення.

• Перемикання між різними показниками.

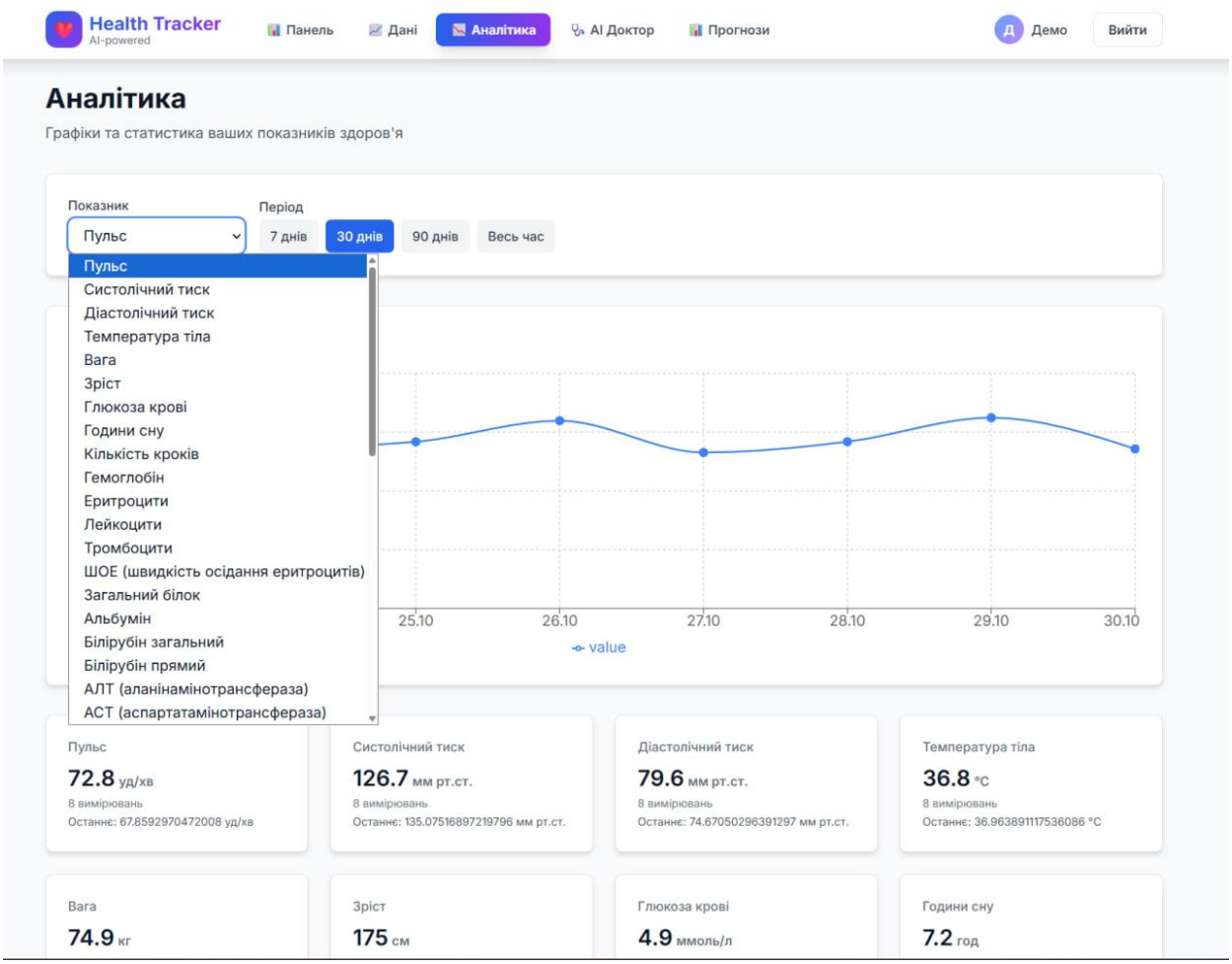

Рисунок 4.5 – Графічна аналітика стану здоров’я

5. AI Доктор (див. рис. 4.6)

5.1. Інтерфейс AI Доктора

• У верхній частині доступні розділи: Огляд, Аналіз, Тренди, Ризики, Чат з AI.

• Основний блок містить інтерактивного AI-асистента.

• Користувач може поставити будь-яке запитання щодо свого здоров’я.

5.2. Чат з AI

• Поле введення знаходиться внизу сторінки.

• AI генерує відповідь на основі:

- історії вимірювань;
- розрахованих трендів;
- моделей прогнозування.

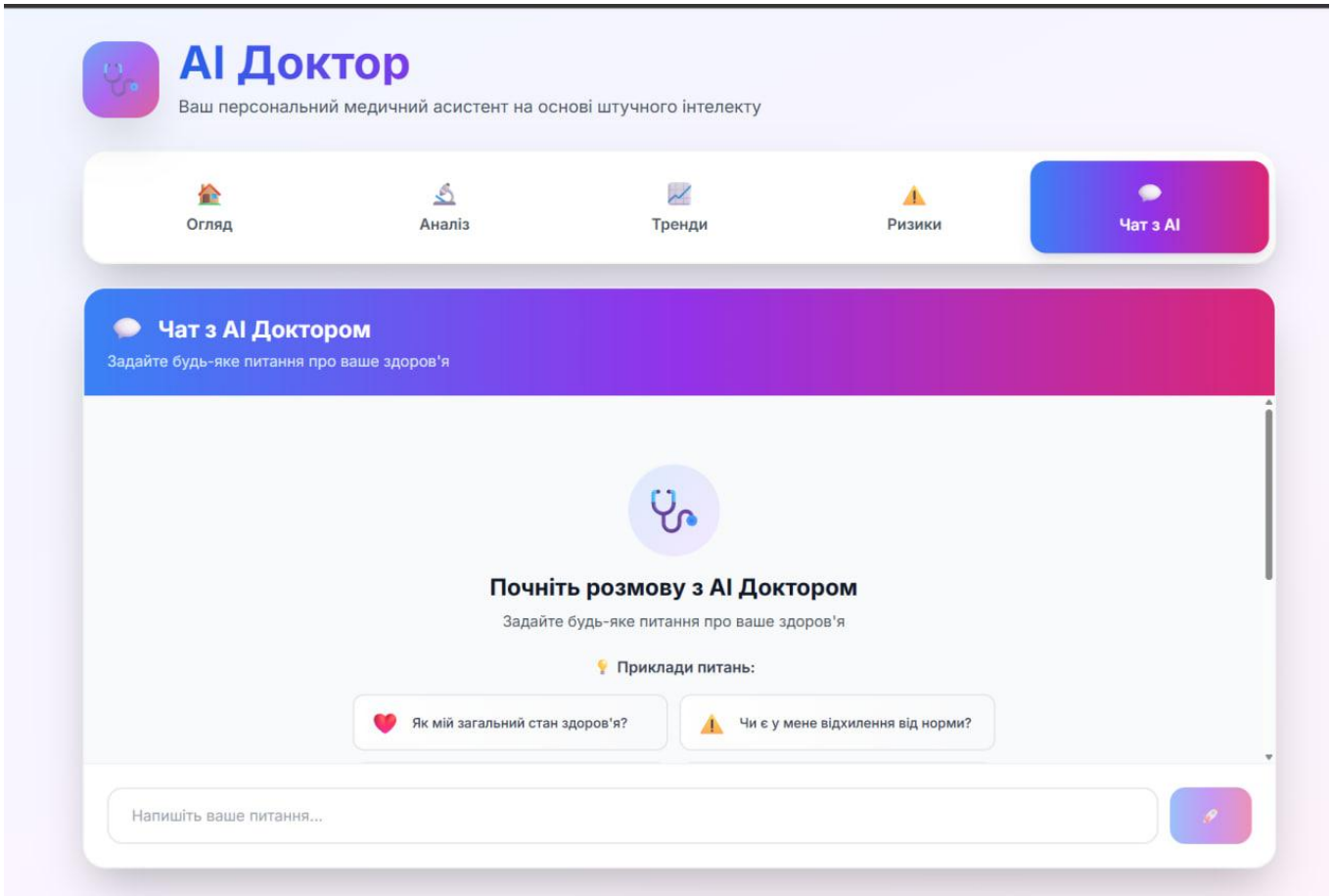

Рисунок 4.6 – Сторінка «AI Доктор» (чат)

6. Комплексний AI аналіз (див. рис. 4.7)

6.1. Загальна статистика

• Відображається інтегральний Індекс здоров’я (0–100).

• Вказано кількість проаналізованих показників за обраний період.

6.2. Попередження та ризики

• У блоці попереджень виводиться список критичних розбіжностей або відхилень.

• Кожне попередження містить іконку, опис проблеми та підказку.

6.3. Детальний аналіз показників

• Для кожного показника наведено:

- поточне значення;
- середнє значення;
- тренд;
- індикатор стану (норма, стабільно, ризик).
- • Дані подано у вигляді окремих карток.

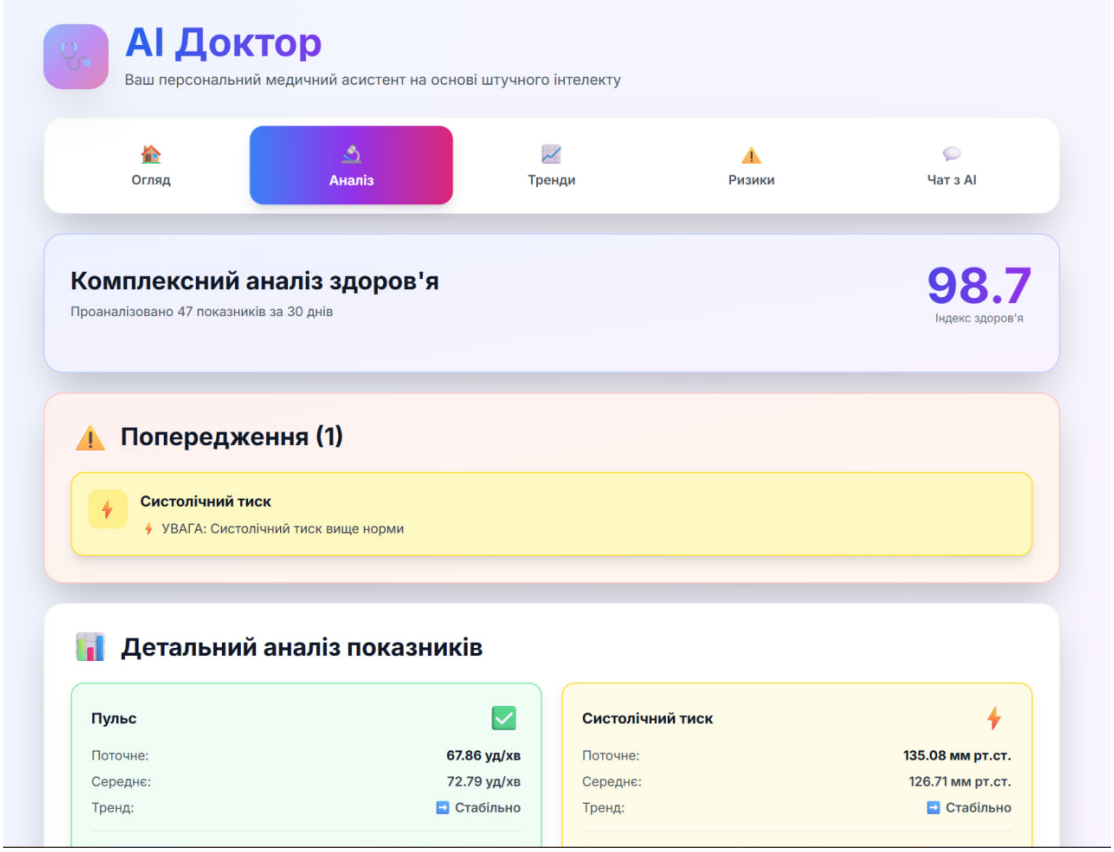

Рисунок 4.7 – Комплексний AI аналіз показників

7. AI Інсайти та прогнозування (див. рис. 4.8)

7.1. Рекомендації AI

• Система автоматично генерує список рекомендацій на основі взаємозв’язків між метриками (кореляційний аналіз).

• Кожен інсайт супроводжується номером та коротким поясненням.

7.2. Прогноз на 3–7 днів

• Нижче подано прогнозні значення ключових показників на найближчі дні.

• Прогноз генерується моделями машинного навчання (Prophet, scikit-learn).

• Для кожного дня зазначаються конкретні значення.

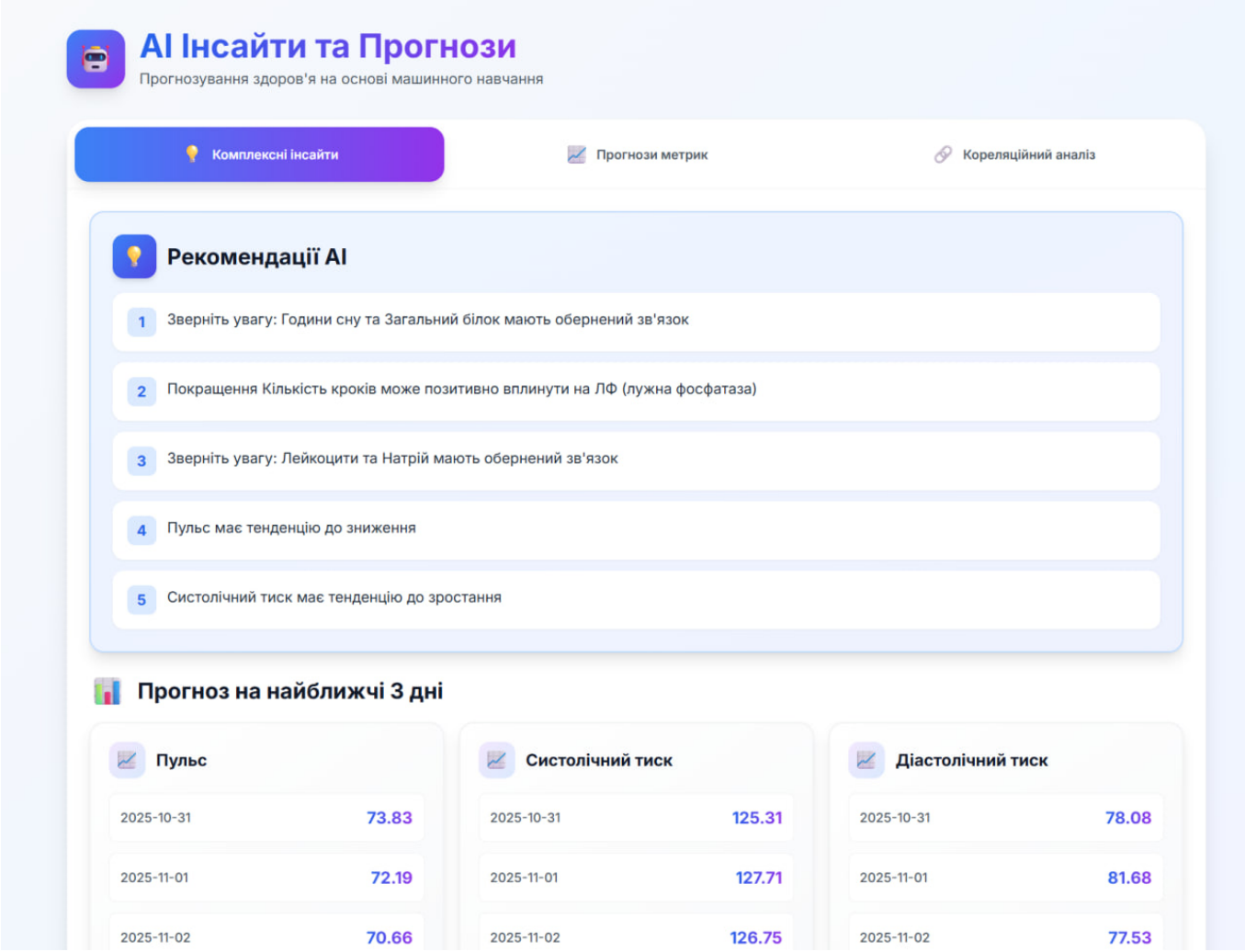

Рисунок 4.8 – Прогнози та інсайти AI

Інтерфейс системи Health Tracker є інтуїтивним і доступним для користувачів будь-якого рівня.

Користувач може:

- додавати або імпортувати дані,
- переглядати аналітику,
- отримувати рекомендації від AI,
- стежити за трендами та прогнозами.

Система забезпечує повний цикл роботи з біомедичною інформацією — від введення даних до глибокого AI-аналізу.

ВИСНОВКИ

У межах виконання дипломної роботи було реалізовано повнофункціональний веб-додаток для моніторингу та прогнозування стану здоров’я, що об’єднує сучасні веб-технології, методи обробки біомедичних даних та інтелектуальні алгоритми машинного навчання. В результаті дослідження було проаналізовано предметну область, визначено ключові проблеми, пов’язані з відстеженням медичних показників користувачів, та сформовано вимоги до структури й функціональності системи. Особливу увагу приділено питанням безпеки, точності аналітичних розрахунків та можливості масштабування.

Побудована архітектура включає фронтенд-частину, сервер FastAPI, модуль штучного інтелекту та базу даних, що забезпечує цілісний цикл роботи із медичними даними: від отримання вимірювань до формування прогнозів. Розроблена клієнтська частина дозволяє користувачам вводити дані вручну, імпортувати CSV-файли, переглядати графіки та статистику, користуватися рекомендаціями AI-модуля й експортувати інформацію у зручних форматах. Інтерфейс побудований таким чином, щоб користувач міг швидко отримувати доступ до ключових метрик здоров’я, відслідковувати їх динаміку та отримувати інсайти щодо можливих ризиків.

Серверна частина успішно реалізує авторизацію за JWT, валідацію та фільтрацію даних, виконання бізнес-логіки, обробку часових рядів та взаємодію з ML-модулем. База даних оптимізована для роботи із значною кількістю вимірювань, що дозволило забезпечити швидкий доступ до історичних значень та їх агрегування. Модуль штучного інтелекту на основі бібліотек Prophet та scikit-learn забезпечує визначення індексу здоров’я, пошук аномалій, генерацію рекомендацій на основі кореляцій між показниками та побудову короткострокових прогнозів. Це створює можливість персоналізованої оцінки стану користувача та надає практичну цінність додатку.

Завершальним етапом стала робота над тестуванням. Проведено модульні, інтеграційні та E2E-тести, які підтвердили стабільну роботу бекенду, коректність

обробки біомедичних даних, правильність роботи механізмів авторизації та генерації прогнозів. Окремо було перевірено захищеність даних користувачів та відповідність реалізації логіці доступу. Усі виконані тести завершилися успішно, що свідчить про високу надійність системи та готовність її до подальшого розвитку.

Таким чином, розроблений веб-додаток повністю відповідає поставленим у роботі цілям і вимогам. Система дозволяє не лише зручно збирати та аналізувати дані про стан здоров’я, але й надає інтелектуальні прогнози та рекомендації на основі машинного навчання. Завдяки модульній архітектурі сервіс може бути розширений та інтегрований у ширші екосистеми медичних застосунків, проєктів telehealth або персональних health-assistant рішень. Отримані результати демонструють, що поєднання сучасних веб-технологій та ML-алгоритмів є ефективним підходом для створення систем підтримки особистого здоров’я й відкриває перспективи для подальших досліджень та удосконалень.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. World Health Organization (WHO). Global Health Indicators. Доступ: https://www.who.int/data/gho
2. Mayo Clinic — Normal Vital Signs & Medical Guidelines. Доступ: https://www.mayoclinic.org
3. MedlinePlus — U.S. National Library of Medicine. Доступ: https://medlineplus.gov
4. American Heart Association — Blood Pressure & Cardiovascular Guidelines. Доступ: https://www.heart.org
5. Taylor S. J., Letham B. Prophet Forecasting Model Documentation. Доступ: https://facebook.github.io/prophet
6. Scikit-learn — Machine Learning Library Documentation. Доступ: https://scikit-learn.org
7. FastAPI — Modern Python Web Framework Documentation. Доступ: https://fastapi.tiangolo.com
8. SQLAlchemy — Python ORM Documentation. Доступ: https://www.sqlalchemy.org
9. Pydantic — Data Validation and Modelling Documentation. Доступ: https://docs.pydantic.dev
10. Next.js — React Framework for Web Applications. Доступ: https://nextjs.org/docs
11. React — Official Documentation. Доступ: https://react.dev
12. HL7 FHIR — International Health Data Standard Specification. Доступ: https://hl7.org/fhir
13. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. - Полтава : ПУЕТ, 2024. -67 с. -1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

```
# backend/main.py

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from database import engine, Base
from routers import auth, measurements, metrics, predictions, forecasts, exports, calculated, comparison, ai_assistant

# Створюємо таблиці
Base.metadata.create_all(bind=engine)

app = FastAPI(
    title="Health Tracker API",
    description="API для відстеження та прогнозування стану здоров'я",
    version="1.0.0"
)

# CORS middleware для фронтенду
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],  # Next.js dev server
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Підключаємо роутери
app.include_router(auth.router)
app.include_router(measurements.router)
app.include_router(metrics.router)
app.include_router(predictions.router)
app.include_router(forecasts.router)
app.include_router(exports.router)
app.include_router(calculated.router)
app.include_router(comparison.router)
app.include_router(ai_assistant.router)
```

```
@app.get("/")
async def root():
    return {"message": "Health Tracker API is running!"}

@app.get("/health")
async def health_check():
    return {"status": "healthy"}



# backend/models.py

from sqlalchemy import Column, Integer, String, Float, DateTime, ForeignKey, Text, Boolean
from sqlalchemy.orm import relationship
from sqlalchemy.sql import func
from database import Base

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    full_name = Column(String)
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    measurements = relationship("Measurement", back_populates="user")
    predictions = relationship("Prediction", back_populates="user")

class Metric(Base):
    __tablename__ = "metrics"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, index=True, nullable=False)  # heart_rate, systolic_bp, etc.
    display_name = Column(String, nullable=False)  # "Пульс", "Систолічний тиск"
    unit = Column(String, nullable=False)  # "bpm", "mmHg", "°C"
```

```
    category = Column(String, default="general")  # general, blood_test, biochemistry, liver, lipids, kidney, urine, hormones, electrolytes, vitamins, inflammation
    normal_min = Column(Float)  # Нормальний мінімум
    normal_max = Column(Float)  # Нормальний максимум
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    measurements = relationship("Measurement", back_populates="metric")
    predictions = relationship("Prediction", back_populates="metric")

class Measurement(Base):
    __tablename__ = "measurements"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    metric_id = Column(Integer, ForeignKey("metrics.id"), nullable=False)
    value = Column(Float, nullable=False)
    timestamp = Column(DateTime(timezone=True), nullable=False)
    source = Column(String, default="manual")  # manual, csv_import, device
    notes = Column(Text)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    user = relationship("User", back_populates="measurements")
    metric = relationship("Metric", back_populates="measurements")

class Prediction(Base):
    __tablename__ = "predictions"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    metric_id = Column(Integer, ForeignKey("metrics.id"), nullable=False)
    prediction_timestamp = Column(DateTime(timezone=True), nullable=False)  # Коли зроблено прогноз
    target_timestamp = Column(DateTime(timezone=True), nullable=False)  # На коли прогноз
    predicted_value = Column(Float)  # Прогнозоване значення
    risk_score = Column(Float)  # Оцінка ризику 0-1
    risk_level = Column(String)  # low, medium, high
    model_version = Column(String)
    confidence = Column(Float)  # Впевненість моделі
    created_at = Column(DateTime(timezone=True), server_default=func.now())
```

```
    user = relationship("User", back_populates="predictions")
    metric = relationship("Metric", back_populates="predictions")


class Goal(Base):
    __tablename__ = "goals"

    id = Column(Integer, primary_key=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"), nullable=False)
    metric_id = Column(Integer, ForeignKey("metrics.id"), nullable=False)
    target_value = Column(Float)
    min_value = Column(Float, nullable=True)
    max_value = Column(Float, nullable=True)
    period = Column(String, default="daily")  # daily, weekly, monthly
    is_active = Column(Boolean, default=True)
    start_date = Column(DateTime(timezone=True), server_default=func.now())
    end_date = Column(DateTime(timezone=True), nullable=True)
    created_at = Column(DateTime(timezone=True), server_default=func.now())

    user = relationship("User")
    metric = relationship("Metric")


# backend/schemas.py

from pydantic import BaseModel, EmailStr
from datetime import datetime
from typing import Optional, List

# User schemas
class UserBase(BaseModel):
    email: EmailStr
    full_name: Optional[str] = None

class UserCreate(UserBase):
    password: str
```

```
class UserUpdate(BaseModel):
    full_name: Optional[str] = None
    email: Optional[EmailStr] = None
    current_password: Optional[str] = None
    new_password: Optional[str] = None

class UserResponse(UserBase):
    id: int
    is_active: bool
    created_at: datetime

    class Config:
        from_attributes = True

# Auth schemas
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    email: Optional[str] = None

# Metric schemas
class MetricBase(BaseModel):
    name: str
    display_name: str
    unit: str
    category: Optional[str] = "general"
    normal_min: Optional[float] = None
    normal_max: Optional[float] = None

class MetricCreate(MetricBase):
    pass

class MetricResponse(MetricBase):
    id: int
    created_at: datetime
```

```
    class Config:
        from_attributes = True

# Measurement schemas
class MeasurementBase(BaseModel):
    metric_id: int
    value: float
    timestamp: datetime
    source: Optional[str] = "manual"
    notes: Optional[str] = None

class MeasurementCreate(MeasurementBase):
    pass

class MeasurementResponse(MeasurementBase):
    id: int
    user_id: int
    created_at: datetime
    metric: Optional[MetricResponse] = None

    class Config:
        from_attributes = True

# CSV Import schema
class CSVImportResponse(BaseModel):
    imported_count: int
    errors: List[str]
    message: str

# Prediction schemas
class PredictionResponse(BaseModel):
    id: int
    metric_id: int
    prediction_timestamp: datetime
    target_timestamp: datetime
    predicted_value: Optional[float]
    risk_score: Optional[float]
    risk_level: Optional[str]
```

```
    confidence: Optional[float]
    metric: Optional[MetricResponse] = None

    class Config:
        from_attributes = True

# Analytics schemas
class MetricStats(BaseModel):
    metric_id: int
    metric_name: str
    count: int
    min_value: float
    max_value: float
    avg_value: float
    latest_value: Optional[float]
    latest_timestamp: Optional[datetime]


# backend/database.py

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from config import settings

engine = create_engine(
    settings.database_url,
    connect_args={"check_same_thread": False}  # Потрібно для SQLite
)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
```

```
        db.close()


# backend/auth.py

from datetime import datetime, timedelta
from typing import Optional
from jose import JWTError, jwt
import hashlib
import secrets
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session
from database import get_db
from models import User
from schemas import TokenData
from config import settings

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="auth/login")

def verify_password(plain_password, hashed_password):
    """Перевіряємо пароль використовуючи SHA-256 + сіль"""
    try:
        salt, stored_hash = hashed_password.split(':')
        hash_obj = hashlib.sha256((plain_password + salt).encode())
        return hash_obj.hexdigest() == stored_hash
    except:
        return False

def get_password_hash(password):
    """Хешуємо пароль використовуючи SHA-256 + сіль"""
    salt = secrets.token_hex(16)
    hash_obj = hashlib.sha256((password + salt).encode())
    return f"{salt}:{hash_obj.hexdigest()}"

def get_user_by_email(db: Session, email: str):
    return db.query(User).filter(User.email == email).first()
```

```
def authenticate_user(db: Session, email: str, password: str):
    user = get_user_by_email(db, email)
    if not user:
        return False
    if not verify_password(password, user.hashed_password):
        return False
    return user

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, settings.secret_key, algorithm=settings.algorithm)
    return encoded_jwt

async def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, settings.secret_key, algorithms=[settings.algorithm])
        email: str = payload.get("sub")
        if email is None:
            raise credentials_exception
        token_data = TokenData(email=email)
    except JWTError:
        raise credentials_exception
    user = get_user_by_email(db, email=token_data.email)
    if user is None:
        raise credentials_exception
    return user

async def get_current_active_user(current_user: User = Depends(get_current_user)):
```

```
    if not current_user.is_active:
        raise HTTPException(status_code=400, detail="Inactive user")
    return current_user


# backend/config.py

from pydantic_settings import BaseSettings
from typing import Optional

class Settings(BaseSettings):
    secret_key: str = "your-secret-key-for-development-only"
    algorithm: str = "HS256"
    access_token_expire_minutes: int = 30
    database_url: str = "sqlite:///./health_tracker.db"

    class Config:
        env_file = ".env"

settings = Settings()


# backend/routers/auth.py

from datetime import timedelta
from fastapi import APIRouter, Depends, HTTPException, status
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from database import get_db
from models import User
from schemas import UserCreate, UserUpdate, UserResponse, Token
from auth import authenticate_user, create_access_token, get_password_hash, verify_password,
get_user_by_email, get_current_active_user
from config import settings

router = APIRouter(prefix="/auth", tags=["authentication"])

@router.post("/register", response_model=UserResponse)
```

```
async def register_user(user: UserCreate, db: Session = Depends(get_db)):
    # Перевіряємо чи користувач вже існує
    db_user = get_user_by_email(db, email=user.email)
    if db_user:
        raise HTTPException(
            status_code=400,
            detail="Email already registered"
        )

    # Створюємо нового користувача
    hashed_password = get_password_hash(user.password)
    db_user = User(
        email=user.email,
        hashed_password=hashed_password,
        full_name=user.full_name
    )
    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return db_user

@router.post("/login", response_model=Token)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)):
    user = authenticate_user(db, form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=settings.access_token_expire_minutes)
    access_token = create_access_token(
        data={"sub": user.email}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

```
@router.get("/me", response_model=UserResponse)
async def read_users_me(current_user: User = Depends(get_current_active_user)):
    return current_user

@router.put("/me", response_model=UserResponse)
async def update_user_profile(
    user_update: UserUpdate,
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    # Якщо змінюється email, перевіряємо чи він не зайнятий
    if user_update.email and user_update.email != current_user.email:
        existing_user = get_user_by_email(db, email=user_update.email)
        if existing_user:
            raise HTTPException(
                status_code=400,
                detail="Email already registered"
            )
        current_user.email = user_update.email

    # Оновлюємо ім'я
    if user_update.full_name is not None:
        current_user.full_name = user_update.full_name

    # Якщо змінюється пароль
    if user_update.new_password:
        if not user_update.current_password:
            raise HTTPException(
                status_code=400,
                detail="Current password is required to set new password"
            )

        # Перевіряємо поточний пароль
        if not verify_password(user_update.current_password, current_user.hashed_password):
            raise HTTPException(
                status_code=400,
                detail="Current password is incorrect"
            )
```

```
        # Встановлюємо новий пароль
        current_user.hashed_password = get_password_hash(user_update.new_password)

    db.commit()
    db.refresh(current_user)

    return current_user


# backend/routers/measurements.py

from fastapi import APIRouter, Depends, HTTPException, UploadFile, File
from sqlalchemy.orm import Session
from sqlalchemy import func, and_
from typing import List, Optional
from datetime import datetime, timedelta
import csv
import io
from database import get_db
from models import User, Measurement, Metric
from schemas import MeasurementCreate, MeasurementResponse, CSVImportResponse, MetricStats
from auth import get_current_active_user

router = APIRouter(prefix="/measurements", tags=["measurements"])

@router.post("/", response_model=MeasurementResponse)
async def create_measurement(
    measurement: MeasurementCreate,
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    # Перевіряємо чи існує метрика
    metric = db.query(Metric).filter(Metric.id == measurement.metric_id).first()
    if not metric:
        raise HTTPException(status_code=404, detail="Metric not found")

    db_measurement = Measurement(
```

```
        **measurement.dict(),
        user_id=current_user.id
    )
    db.add(db_measurement)
    db.commit()
    db.refresh(db_measurement)

    return db_measurement

@router.get("/", response_model=List[MeasurementResponse])
async def get_measurements(
    metric_id: Optional[int] = None,
    from_date: Optional[datetime] = None,
    to_date: Optional[datetime] = None,
    limit: int = 100,
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    query = db.query(Measurement).filter(Measurement.user_id == current_user.id)

    if metric_id:
        query = query.filter(Measurement.metric_id == metric_id)

    if from_date:
        query = query.filter(Measurement.timestamp >= from_date)

    if to_date:
        query = query.filter(Measurement.timestamp <= to_date)

    measurements = query.order_by(Measurement.timestamp.desc()).limit(limit).all()
    return measurements

@router.post("/import-csv", response_model=CSVImportResponse)
async def import_csv(
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
```

```
if not file.filename.endswith('.csv'):
    raise HTTPException(status_code=400, detail="File must be CSV format")

try:
    # Читаємо CSV файл
    contents = await file.read()
    csv_content = contents.decode('utf-8')
    csv_reader = csv.DictReader(io.StringIO(csv_content))

    # Перевіряємо необхідні колонки
    required_columns = ['timestamp', 'metric_name', 'value']
    fieldnames = csv_reader.fieldnames or []
    missing_columns = [col for col in required_columns if col not in fieldnames]
    if missing_columns:
        raise HTTPException(
            status_code=400,
            detail=f"Missing required columns: {missing_columns}"
        )

    imported_count = 0
    errors = []

    for index, row in enumerate(csv_reader, 1):
        try:
            # Знаходимо або створюємо метрику
            metric = db.query(Metric).filter(Metric.name == row['metric_name']).first()
            if not metric:
                errors.append(f"Row {index}: Metric '{row['metric_name']}' not found")
                continue

            # Парсимо дату (простий парсинг)
            timestamp = datetime.fromisoformat(row['timestamp'].replace(' ', 'T'))

            # Створюємо вимірювання
            measurement = Measurement(
                user_id=current_user.id,
                metric_id=metric.id,
                value=float(row['value']),
```

```
                timestamp=timestamp,
                source="csv_import"
            )

            db.add(measurement)
            imported_count += 1

        except Exception as e:
            errors.append(f"Row {index}: {str(e)}")

    db.commit()

    return CSVImportResponse(
        imported_count=imported_count,
        errors=errors,
        message=f"Successfully imported {imported_count} measurements"
    )

  except Exception as e:
    raise HTTPException(status_code=500, detail=f"Error processing CSV: {str(e)}")

@router.get("/stats", response_model=List[MetricStats])
async def get_measurement_stats(
  from_date: Optional[datetime] = None,
  to_date: Optional[datetime] = None,
  current_user: User = Depends(get_current_active_user),
  db: Session = Depends(get_db)
):
  # Базовий запит
  query = db.query(
    Measurement.metric_id,
    Metric.name.label('metric_name'),
    func.count(Measurement.id).label('count'),
    func.min(Measurement.value).label('min_value'),
    func.max(Measurement.value).label('max_value'),
    func.avg(Measurement.value).label('avg_value')
  ).join(Metric).filter(Measurement.user_id == current_user.id)
```

```
    if from_date:
        query = query.filter(Measurement.timestamp >= from_date)

    if to_date:
        query = query.filter(Measurement.timestamp <= to_date)

    stats = query.group_by(Measurement.metric_id, Metric.name).all()

    # Додаємо останнє значення для кожної метрики
    result = []
    for stat in stats:
        latest = db.query(Measurement).filter(
            and_(
                Measurement.user_id == current_user.id,
                Measurement.metric_id == stat.metric_id
            )
        ).order_by(Measurement.timestamp.desc()).first()

        result.append(MetricStats(
            metric_id=stat.metric_id,
            metric_name=stat.metric_name,
            count=stat.count,
            min_value=stat.min_value,
            max_value=stat.max_value,
            avg_value=stat.avg_value,
            latest_value=latest.value if latest else None,
            latest_timestamp=latest.timestamp if latest else None
        ))

    return result


# backend/routers/metrics.py

from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.orm import Session
from typing import List, Optional
from database import get_db
```

```
from models import User, Metric
from schemas import MetricCreate, MetricResponse
from auth import get_current_active_user

router = APIRouter(prefix="/metrics", tags=["metrics"])

@router.post("/", response_model=MetricResponse)
async def create_metric(
    metric: MetricCreate,
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    # Перевіряємо чи метрика вже існує
    existing_metric = db.query(Metric).filter(Metric.name == metric.name).first()
    if existing_metric:
        raise HTTPException(status_code=400, detail="Metric with this name already exists")

    db_metric = Metric(**metric.dict())
    db.add(db_metric)
    db.commit()
    db.refresh(db_metric)

    return db_metric

@router.get("/categories")
async def get_categories(
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    """Отримує список всіх категорій метрик"""
    categories = db.query(Metric.category).distinct().all()

    # Маппінг категорій на українські назви
    category_names = {
        "general": {"name": "Базові показники", "icon": "❤□", "color": "blue"},
        "blood_test": {"name": "Загальний аналіз крові", "icon": "  ", "color": "red"},
        "biochemistry": {"name": "Біохімічний аналіз", "icon": "  ", "color": "purple"},
        "liver": {"name": "Печінкові проби", "icon": "⛨', "color": "orange"},
```

```
        "lipids": {"name": "Ліпідний профіль", "icon": "🧈", "color": "pink"},
        "kidney": {"name": "Функція нирок", "icon": "□", "color": "brown"},
        "urine": {"name": "Загальний аналіз сечі", "icon": "💧", "color": "cyan"},
        "hormones": {"name": "Гормони", "icon": "  ", "color": "indigo"},
        "electrolytes": {"name": "Електроліти", "icon": "⚡", "color": "yellow"},
        "vitamins": {"name": "Вітаміни", "icon": "👁", "color": "green"},
        "inflammation": {"name": "Маркери запалення", "icon": "🔥", "color": "red"},
    }

    result = []
    for (cat,) in categories:
        if cat and cat in category_names:
            result.append({
                "id": cat,
                "name": category_names[cat]["name"],
                "icon": category_names[cat]["icon"],
                "color": category_names[cat]["color"]
            })

    return result

@router.get("/", response_model=List[MetricResponse])
async def get_metrics(
    category: Optional[str] = Query(None, description="Фільтр по категорії"),
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    """Отримує всі доступні метрики (опціонально з фільтром по категорії)"""
    query = db.query(Metric)

    if category:
        query = query.filter(Metric.category == category)

    metrics = query.all()
    return metrics

@router.get("/{metric_id}", response_model=MetricResponse)
```

```
async def get_metric(
    metric_id: int,
    current_user: User = Depends(get_current_active_user),
    db: Session = Depends(get_db)
):
    metric = db.query(Metric).filter(Metric.id == metric_id).first()
    if not metric:
        raise HTTPException(status_code=404, detail="Metric not found")
    return metric
```