ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
(повне найменування вищого навчального закладу)
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування інституту, назва факультету (відділення))
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ
15.01.202615.01.2026

Пояснювальна записка

до кваліфікаційної роботи
магістр
(освітній рівень)

на тему: «Розробка системи веб-аналітики для медичних даних пацієнтів із візуалізацією ключових показників»

Виконав: студент 6 курсу, групи 6ПР2
спеціальності
121 - «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Семенов Микола Анатолійович
(прізвище та ініціали)
Керівник к.т.н., доцент Козуб Н.О.
(прізвище та ініціали)
Рецензент к.т.н., доцент Григорова А.А.
(прізвище та ініціали)

Хмельницький - 2025

Херсонський Національний Технічний Університет
(повне найменування вищого навчального закладу)

Факультет, відділення Інформаційних технологій та дизайну
Кафедра Програмних засобів і технологій
Освітній рівень магістр
Спеціальність 121 – Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри
Програмних засобів і технологій
к.т.н., доцент Огнєва О.Є.
“___” _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Семенов Микола Анатолійович
(прізвище, ім’я,по батькові)

Тема роботи «Розробка системи веб-аналітики для медичних даних пацієнтів із візуалізацією ключових показників»

керівник роботи к.т.н., доцент Козуб Н.О.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу_____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
Вступ; 1.Системний аналіз предметної області та існуючих підходів до обробки медичних даних; 2. Проектування архітектури веб-системи, моделі даних та математичного апарату для оцінки стану пацієнта; 3. Програмна реалізація серверної частини, клієнтського інтерфейсу та інтеграція модуля прогностичної аналітики; 4. Практична реалізація системи, експериментальне дослідження точності ML-моделей та оцінка ефективності веб-додатку; Висновок.5.
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
В роботі наведено:
Рисунки - 17
Таблиці - 1

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 21.08.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	21.08.2025	Виконано
2.	Підбір літератури	30.09.2025	Виконано
3.	Аналіз існуючих рішень	07.10.2025	Виконано
4.	Розробка структури роботи та визначення основних методів і підходів для аналізу даних.	20.10.2025	Виконано
5.	Опис моделей машинного навчання для аналізу текстів	30.10.2025	Виконано
6.	Опис використання інструментів для аналізу	13.11.2025	Виконано
7.	Порівняння методів аналізу даних	20.11.2025	Виконано
8.	Підготовка матеріалів текстової та графічної частини проекту	30.11.2025	Виконано
9.	Оформлення пояснювальної записки	10.12.2025	Виконано
10.	Захист кваліфікаційної роботи	20.12.2025	Виконано

Студент_____ М.А. Семенов_____
(підпис)(прізвище та ініціали)

Керівник роботи _____ Н.О. Козуб__
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра: 117 сторінок, 17 рисунків, 3 додатки, 35 джерел

Мета роботи:

Розробка архітектури та прототипу веб-орієнтованої системи аналітики для медичних даних пацієнтів, яка забезпечує інтерактивну візуалізацію ключових показників здоров'я та підтримує прийняття обґрунтованих рішень медичними працівниками.

Об'єкт дослідження:

Медичні дані пацієнтів та процеси їх аналітичної обробки для прийняття рішень у медичних установах.

Предмет дослідження:

Методи веб-аналітики та візуалізації даних для моніторингу ключових медичних показників пацієнтів.

Методи дослідження:

У роботі використовуються методи інтеграції даних із медичних баз даних, техніки попередньої обробки та нормалізації медичних записів, алгоритми статистичного аналізу й машинного навчання, а також методи інтерактивної візуалізації для представлення ключових показників у зручному для лікарів та пацієнтів вигляді.

Результат роботи:

Результатом роботи є прототип веб-системи аналітики, що забезпечує збір, обробку та візуалізацію ключових медичних показників пацієнтів. Система дозволяє здійснювати моніторинг стану здоров'я, виявляти ризики на основі аналітичних моделей та подавати результати у зрозумілих графічних формах для підтримки в прийнятті рішень медичним персоналом.

Новизна роботи:

Розроблено підхід до створення веб-аналітичної системи для медичних даних, який поєднує інтеграцію медичних записів із сучасними методами обробки та візуалізації. Запропонована система дозволяє підвищити

ефективність аналізу медичних показників та забезпечує можливість їх динамічного моніторингу у зручному для практичного використання вигляді.

Ключові слова:. ВЕБ-АНАЛІТИКА, ВІЗУАЛІЗАЦІЯ ДАНИХ, МЕДИЧНІ ДАНІ ПАЦІЄНТІВ, ОХОРОНА ЗДОРОВ'Я, ЕЛЕКТРОННА МЕДИЧНА КАРТКА, ПІДТРИМКА ПРИЙНЯТТЯ РІШЕНЬ, ПОКАЗНИКИ ЗДОРОВ'Я, СИСТЕМАТИЗАЦІЯ ДАНИХ, БЕЗПЕКА ДАНИХ, МАШИННЕ НАВЧАННЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ЕЕГ – Електроенцефалограма

ЕКГ – Електрокардіограма

КТ – Комп'ютерна томографія

МІС – Медична інформаційна система

МКХ – Міжнародна класифікація хвороб

МРТ – Магнітно-резонансна томографія

СУБД – Система управління базами даних

ШІ – Штучний інтелект

ІІКЗ – Інтегрального індексу клінічної значущості

ACID – Atomicity, Consistency, Isolation, Durability

API – Application Programming Interface

ASGI – Asynchronous Server Gateway Interface

CDA – Clinical Document Architecture

DICOM – Digital Imaging and Communications in Medicine

EHR – Electronic Health Record

FHIR – Fast Healthcare Interoperability Resources

GDPR – General Data Protection Regulation

HIPAA – Health Insurance Portability and Accountability Act

HL7 – Health Level Seven IoMT – Internet of Medical Things

JSON – JavaScript Object Notation

LIS – Laboratory Information System

ML – Machine Learning

UI/ UX – User Interface/User Experience

XML – Extensible Markup Language

АНОТАЦІЯ

Кваліфікаційна робота магістра має наступні структурні частини: вступ, чотири розділи, висновки, список використаних джерел та додатки. Загальний обсяг роботи становить 117 сторінок, робота містить 17 рисунків, 1 таблицю, 3 додатки та 35 посилань на джерела.

Перший розділ «Системний аналіз предметної області та обґрунтування розробки веб-аналітичної системи для медичних даних» складається з наступних підрозділів: «Феномен медичних даних: характеристика, структура та семантична цінність», «Аналіз проблеми збору та обробки медичних даних», «Архітектура та компаративний аналіз існуючих аналітичних рішень» та «Висновки до розділу 1». У ньому було проведено детальний аналіз поточного стану цифровізації в медицині, розглянуто специфіку медичних даних, їхню гетерогенність, багатовимірність та лонгітюдний характер. Також описано еволюцію медичних інформаційних систем (МІС) та виявлено ключові проблеми існуючих рішень, зокрема фрагментарність даних та відсутність інструментів прогностичної аналітики.

Другий розділ «Проектування та архітектура системи веб-аналітики медичних даних» складається з наступних підрозділів: «Вибір та обґрунтування методів та засобів розробки», «Архітектура та проектування системи», «Моделювання поведінки системи та користувачів», «Проектування механізмів безпеки та конфіденційності», «Розробка алгоритмічної моделі для інтегральної клінічної оцінки» та «Висновки до розділу 2». Розглянуто та обґрунтовано вибір технологічного стеку (Python, FastAPI, React, PostgreSQL), спроектовано архітектуру бази даних та серверного API. Особливу увагу приділено розробці математичної моделі «Інтегрального індексу клінічної значущості» (ІІКЗ) та проектуванню системи безпеки на основі рольової моделі доступу (RBAC).

Третій розділ «Розробка та реалізація компонентів системи веб-аналітики» складається з наступних підрозділів: «Програмна реалізація рівня доступу до даних», «Розробка серверної бізнес-логіки та аналітичного ядра», «Реалізація модуля прогностичної аналітики», «Реалізація клієнтського

інтерфейсу та взаємодії з користувачем», «Програмна архітектура клієнтської частини» та «Висновки до розділу 3». Описано методи програмної реалізації серверної частини, налаштування ORM та валідації даних. Представлено реалізацію модуля машинного навчання на базі алгоритму лінійної регресії для прогнозування трендів. Розглянуто створення SPA-інтерфейсу, організацію маршрутизації та інтеграцію графічних бібліотек.

Четвертий розділ «Практична реалізація та аналіз ефективності системи» складається з наступних підрозділів: «Опис функціонування інформаційного ядра системи», «Інтерфейс користувача та режими роботи програмного комплексу», «Експериментальне дослідження точності прогностичних моделей», «Комплексна оцінка ефективності та надійності веб-додатку» та «Висновки до розділу 4». Описано функціонування розробленого прототипу, сценарії роботи лікаря та пацієнта. Проведено експериментальне дослідження точності ML-модуля на тестових наборах даних, а також виконано аудит продуктивності та ергономічності системи.

Результати роботи можуть бути використані для покращення якості медичного обслуговування, автоматизації моніторингу стану пацієнтів, розробки рекомендацій для прийняття клінічних рішень та ефективного використання накопичених медичних даних у закладах охорони здоров’я.

ABSTRACT

The master's thesis has the following structural parts: introduction, four chapters, conclusions, list of references, and appendices. The total volume of the thesis is 117 pages, and it contains 17 figures, 1 table, and 35 references.

The first chapter, “System Analysis of the Subject Area and Justification for the Development of a Web Analytics System for Medical Data,” consists of the following sections: “The Phenomenon of Medical Data: Characteristics, Structure, and Semantic Value,” “Analysis of the Problem of Medical Data Collection and Processing,” “Architecture and comparative analysis of existing analytical solutions,” and “Conclusions to Section 1.” It provides a detailed analysis of the current state of digitalization in medicine and examines the specifics of medical data, its heterogeneity, multidimensionality, and longitudinal nature. It also describes the evolution of medical information systems (MIS) and identifies key problems with existing solutions, in particular the fragmentation of data and the lack of predictive analytics tools.

The second chapter, “Design and Architecture of a Medical Data Web Analytics System,” consists of the following sections: “Selection and Justification of Development Methods and Tools,” “System Architecture and Design,” “Modeling System and User Behavior,” “Design of Security and Privacy Mechanisms,” “Development of an Algorithmic Model for Integrated Clinical Assessment,” and “Conclusions to Section 2.” The choice of technology stack (Python, FastAPI, React, PostgreSQL) is considered and justified, and the architecture of the database and server API is designed. Particular attention is paid to the development of a mathematical model of the “Integrated Index of Clinical Significance” (IICS) and the design of a security system based on a role-based access control (RBAC) model.

The third section, “Development and implementation of web analytics system components,” consists of the following subsections: “Software implementation of data access level,” “Development of server business logic and analytical core,”

“Implementation of the predictive analytics module,” “Implementation of the client interface and user interaction,” “Software architecture of the client part,” and “Conclusions to Section 3.” Methods of software implementation of the server part, ORM configuration, and data validation are described. The implementation of a machine learning module based on a linear regression algorithm for trend forecasting is presented. The creation of an SPA interface, routing organization, and integration of graphics libraries are considered.

The fourth section, “Practical implementation and analysis of system efficiency,” consists of the following subsections: “Description of the functioning of the information core of the system,” “User interface and operating modes of the software complex,” “Experimental study of the accuracy of predictive models,” “Comprehensive assessment of the effectiveness and reliability of the web application,” and “Conclusions to Section 4.” The functioning of the developed prototype and the scenarios of the doctor and patient are described. An experimental study of the accuracy of the ML module on test data sets was conducted, and an audit of the system's performance and ergonomics was performed.

The results of the work can be used to improve the quality of medical care, automate patient monitoring, develop recommendations for clinical decision-making, and effectively use accumulated medical data in healthcare facilities.

ВСТУП

Актуальність теми. У сучасному світі цифровізація охорони здоров'я набуває особливої ваги, оскільки обсяг медичних даних постійно зростає, а потреба у швидкому та якісному їх аналізі стає критично важливою. Медичні установи та лікарі щоденно працюють із великими масивами інформації: від електронних карток пацієнтів до результатів лабораторних досліджень та даних моніторингу життєвих показників. Традиційні методи обробки цих даних часто є повільними та не дозволяють оперативно виявляти закономірності чи ризики. У таких умовах системи веб-аналітики, які інтегрують методи статистичного аналізу, машинного навчання та візуалізації, відкривають нові можливості для підвищення якості діагностики, прогнозування стану пацієнтів та підтримки прийняття медичних рішень.

Об'єкт дослідження: Медичні дані пацієнтів.

Предмет дослідження: Методи веб-аналітики та візуалізації даних для моніторингу ключових медичних показників пацієнтів з метою покращення якості прийняття рішень лікарем.

Мета дослідження: Мета роботи полягає у розробці підходу до збору, аналізу та візуалізації медичних даних пацієнтів за допомогою системи веб-аналітики для підтримки прийняття рішень профільним спеціалістом у сфері охорони здоров'я.

Наукова новизна одержаних результатів: Створено прототип веб-системи аналітики, що забезпечує автоматизований збір, обробку та візуалізацію ключових медичних показників пацієнтів. Система дозволяє здійснювати динамічний моніторинг стану здоров'я, виявляти ризики на основі аналітичних моделей та попереджати важкі медичні стани, представляючи всі результати у зручних графічних формах. Новизна роботи полягає у поєднанні інтеграції медичних записів із сучасними методами машинного навчання та візуалізації, що підвищує ефективність аналізу й якість діагностики.

Структура: робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, додатків. Кваліфікаційна робота магістра складається з 117 сторінок, 17 рисунків, 3 додатки, 35 джерел.

РОЗДІЛ 1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ РОЗРОБКИ ВЕБ-АНАЛІТИЧНОЇ СИСТЕМИ ДЛЯ МЕДИЧНИХ ДАНИХ

Сучасна епоха розвитку інформаційних технологій характеризується глибокою цифровою трансформацією всіх без винятку галузей. В системі охорони здоров’я цей процес є особливо складним та водночас критично важливим. Ми є свідками того, як медичні дані, що десятиліттями накопичувались у паперових архівах, перетворюються на один із найбільш цінних та водночас складних стратегічних ресурсів. Саме глибокий аналіз цих даних відкриває шлях до превентивної, персоналізованої та більш доказової медицини. Проте, перш ніж проектувати будь-які програмні рішення, необхідно провести всебічний системний аналіз цієї предметної області, виявити її специфіку, ключові проблеми та проаналізувати існуючі на ринку інструменти. Цей розділ присвячений саме такому дослідженню, що закладе теоретичний та методологічний фундамент для подальшої розробки.

1.1. Феномен медичних даних: характеристика, структура та семантична цінність

У парадигмі сучасної доказової медицини ми оперуємо поняттям «медичні дані», яке вже давно вийшло за межі традиційного уявлення про "історію хвороби". Сьогодні — це багатовимірний, динамічний та неперервний цифровий портрет фізіологічного та психологічного стану пацієнта, що репрезентує складні, не завжди очевидні, взаємозв’язки між генетикою, способом життя, екологічним контекстом та терапевтичними втручаннями.

Фундаментальна відмінність медичних даних від будь-яких інших інформаційних потоків, наприклад, у фінансовому секторі чи ритейлі, полягає у їхньому підвищеному ступені чутливості. Вони містять найбільш конфіденційну персональну інформацію, яку тільки можна уявити —

інформацію про тіло, хвороби, психічний стан. Ця особливість автоматично детермінує найвищі, безкомпромісні стандарти безпеки, етичності та правової регламентації їхньої обробки, що відображено у таких міжнародних стандартах, як GDPR (General Data Protection Regulation) в Європі та HIPAA (Health Insurance Portability and Accountability Act) у США. Будь-яка інженерна система, що працює в цій царині, мусить бути побудована на принципах Privacy by Design (конфіденційність через проектування). [2], [3]

Для ефективного аналізу, а тим паче проектування веб-аналітичної системи, критично важливо розуміти саму морфологію цих даних. З погляду інженерії програмного забезпечення та науки про дані (Data Science), весь масив медичної інформації доцільно стратифікувати за рівнем її структурованості — ключовою характеристикою, що визначає складність її подальшої обробки.

1.1.1. Стратифікація даних за структурованістю

Першу, найбільш "зрозумілу" для комп'ютерних систем категорію, складають структуровані дані. Це високоформалізована, чітко організована інформація, що легко піддається алгоритмічній обробці та зберіганню у класичних реляційних базах даних. Сюди належать цифрові показники лабораторних аналізів (наприклад, рівень глюкози в крові, кількість лейкоцитів), стандартизовані кодифікатори (як-от діагнози за міжнародною класифікацією МКХ-10/11), демографічні показники (вік, стать), дані про призначені ліки та їх точне дозування. Це, по суті, кількісний базис, "абетка" медичної аналітики.

Друга категорія — напівструктуровані дані. Вони являють собою певний "компроміс" між жорсткою формалізацією та вільним потоком інформації, маючи часткову або неявну структуру. Класичним прикладом є електронна медична картка (EHR - Electronic Health Record), яка є центральним об'єктом багатьох медичних інформаційних систем (МІС). В EHR стандартизовані поля

(ПІБ пацієнта, дата візиту) поєднуються з довільними текстовими примітками лікаря, або ж дані передаються у гнучких форматах на кшталт JSON чи XML.

Третя, найбільш об’ємна, інформативно насичена, але й найскладніша для аналізу категорія — це неструктуровані дані. Це справжній "інформаційний океан", що приховує левову частку клінічної цінності. Він охоплює наративний клінічний текст — вільні описи стану, суб'єктивні скарги пацієнта, детальні анамнези, епікризи та висновки консиліумів. Саме тут прихований той безцінний контекст, який пояснює *причини* та *обставини* появи "сухих" цифр у структурованих аналізах.

1.1.2. Багатовимірність клінічної картини

Окрім текстових та числових показників, повноцінна клінічна картина формується за рахунок інших модальностей, що також належать переважно до неструктурованих даних. Окрему, виключно важливу, категорію становлять візуальні дані — цифрові діагностичні зображення (МРТ, КТ, УЗД, рентгенограми, дані патогістології), які зберігаються у спеціалізованих форматах (напр., DICOM). Їхній аналіз традиційно був прерогативою лікаря-радіолога, проте сьогодні цей процес активно доповнюється спеціалізованими алгоритмами комп’ютерного зору (Computer Vision).

Не можна оминути й біосигнали — неперервні часові ряди, що фіксують фізіологічні процеси, як-от електрокардіограми (ЕКГ), електроенцефалограми (ЕЕГ), записи дихальної активності. Їхня обробка (Time-Series Analysis) дозволяє не лише фіксувати поточний стан, але й виявляти приховані паттерни, що можуть сигналізувати про майбутні ускладнення задовго до їхньої клінічної маніфестації.

Нарешті, зв’язуючою ланкою для всіх цих типів даних виступають метадані. Медичні інформаційні системи (МІС), окрім власне даних про здоров’я, генерують та зберігають широкий спектр службової інформації: часові мітки вимірювань, інформацію про місце проведення обстеження (клініка, відділення), відомості про лікаря, дані про діагностичне обладнання,

унікальні псевдонімізовані ідентифікатори пацієнтів. На перший погляд, ця інформація видається суто технічною. Але насправді, метадані створюють той самий контекст, без якого будь-який аналіз даних перетворюється на безглуздий набір цифр. Саме завдяки метаданим стає можливим відстежувати динаміку захворювань у часі та валідувати джерело інформації. Без контексту показник "глюкоза 7.5" не несе жодної інформації; але "глюкоза 7.5 (через 2 години після їжі, пацієнт Х, 55 років, діабет 2 типу)" — це вже цінний фрагмент аналітичної картини.

1.1.3. Часовий вимір та лонгітюдний характер медичних даних

Особливу, фундаментальну характеристику медичних даних, що вирізняє їх з-поміж багатьох інших інформаційних доменів, становить їхній лонгітюдний характер та іманентно присутній часовий вимір. На відміну від статичних транзакційних даних у бізнесі, які часто фіксують лише окремий момент часу, медична інформація майже завжди є динамічною послідовністю подій, спостережень та втручань, розгорнутих у часі. Історія хвороби пацієнта – це не просто набір ізольованих фактів, а саме *історія*, наратив його взаємодії із системою охорони здоров'я протягом місяців, років, а іноді й усього життя. Цей часовий аспект є не просто додатковою метаданою (як ми розглянули у п. 1.1.2), а ключовим фактором, що визначає саму суть та цінність цих даних для глибокого аналізу.

Саме аналіз динаміки показників у часі (тренди, розглянуті детальніше у п. 1.2.2) дозволяє вирішувати критично важливі клінічні та управлінські завдання: оцінювати ефективність лікування (чи дійсно терапія призводить до покращення стану пацієнта з плином часу?), прогнозувати розвиток хронічного захворювання (чи вказує поступове погіршення певних показників на неминуче ускладнення?), ідентифікувати причинно-наслідкові зв'язки між терапевтичними втручаннями та зміною стану пацієнта (чи пов'язане призначення нового препарату зі стабілізацією показників?). Лонгітюдність даних також є основою для епідеміологічних досліджень, дозволяючи

відстежувати поширення захворювань у популяції та виявляти часові кластери спалахів.

Однак, робота з лонгітюдними медичними даними ставить перед розробниками аналітичних систем специфічні виклики. По-перше, це проблема нерегулярності спостережень: пацієнти звертаються до лікаря не за розкладом, що призводить до нерівних інтервалів між вимірюваннями. По-друге, це проблема пропущених даних: пацієнт міг пропустити візит або не здати певний аналіз, що створює "дірки" у часовому ряду. По-третє, це проблема вирівнювання даних (data alignment) з різних джерел за часовою шкалою. Всі ці аспекти вимагають від аналітичних систем здатності ефективно працювати з неповними та нерегулярними часовими рядами, моделювати послідовності подій (наприклад, за допомогою методів аналізу виживаності або Марковських моделей) та візуалізувати складні часові патерни. Ігнорування часового виміру або спрощення його до статичних "зрізів" даних призводить до втрати значної частини інформаційної цінності та високого ризику хибних інтерпретацій клінічних процесів.

1.2 Аналіз проблеми збору та обробки медичних даних

Парадоксально, але в сучасній медицині ключова проблема полягає не у *відсутності* даних, а в їхній надмірності, фрагментарності та гетерогенності. Кожна лабораторія, кожна лікарня, кожна приватна клініка та навіть кожен носимий пристрій генерує власні «інформаційні силоси» (data silos). Ці сховища часто ізольовані одне від одного через технічні, семантичні та, що найважливіше, юридичні й адміністративні бар'єри.

1.2.1. Проблематика фрагментарності та інтерпретації джерел

Як наслідок вищезгаданої "силосної" структури, лікар, що приймає рішення, часто має доступ лише до окремого фрагменту "пазла" — історії хвороби свого пацієнта в межах *лише своєї* установи. Це унеможливлює

прийняття рішень на основі повної та цілісної картини стану здоров'я людини за все її життя. Ефективна інтеграція цих гетерогенних джерел, або інтероперабельність, є одним з найскладніших технічних та організаційних викликів. Вона вимагає не лише технічної сумісності, але й семантичної сумісності — єдиного "розуміння" термінів та кодифікаторів усіма учасниками процесу.

1.2.2. Контекстна інтерпретація як ключове завдання аналітики

Аналіз даних у медицині ніколи не може зводитися до механістичної реєстрації чисел. Для клініциста значущим є не стільки абсолютне значення показника, скільки його контекстна інтерпретація. Майже кожен клінічний показник має так звані референтні межі — інтервали значень, що вважаються «нормою». Проте медична практика доводить, що універсальної «норми» не існує. На її визначення впливають вік, стать, генетичні особливості, супутні захворювання та навіть спосіб життя. Тому веб-аналітична система не може просто показувати "7.5". Вона має виступати «цифровим асистентом», здатним автоматично зіставляти отримані дані з валідованими медичними стандартами *та* персональними характеристиками пацієнта, негайно підсвічуючи клінічно значущі відхилення (outliers).

Ще більшої ваги, ніж одиничний знімок, набуває аналіз часових рядів (time-series analysis). Поодиноке відхилення може бути банальним артефактом вимірювання, наслідком стресу чи тимчасових фізіологічних коливань. Натомість стійка тенденція (тренд) — навіть незначного, але монотонного підвищення чи зниження показника протягом кількох місяців — може бути раннім та грізним маркером розвитку хронічного захворювання. Саме тут веб-аналітика демонструє свої найсильніші сторони: автоматична побудова графіків, виявлення трендів, візуалізація динаміки та порівняння різних часових відрізків.

Окрім того, медицина рідко оперує ізольованими даними. Один показник має вкрай обмежену діагностичну цінність. Наприклад, помірно високий рівень

холестерину сам по собі може не викликати занепокоєння. Але той самий рівень холестерину у поєднанні з підвищеним артеріальним тиском, надмірною масою тіла та високим рівнем глікованого гемоглобіну формує зовсім іншу, набагато загрозливішу картину ризику. Комплексна аналітика дозволяє виявляти синдромні закономірності — тобто кластери показників, які у своєму поєднанні формують нову діагностичну сутність.

1.2.3. Ключові бар'єри впровадження

Окрім фундаментальних проблем гетерогенності та інтерпретації, на шляху впровадження будь-якої веб-аналітичної системи постає низка практичних бар'єрів.

По-перше, це вже згадана проблема якості та повноти даних (Data Quality). Пацієнт міг здавати аналізи в трьох різних лабораторіях; результати можуть дублюватися або навіть суперечити один одному. Часто трапляються банальні помилки ручного введення, пропуски у критичних полях, некоректні формати дат. Для аналітичного алгоритму це створює класичну проблему "сміття на вході — сміття на виході".

По-друге, це когнітивні бар'єри та "перевантаження інформацією" з боку кінцевого користувача — лікаря. Медичний персонал працює в умовах хронічного стресу та тотального дефіциту часу. Навіть найпотужніша система, що видає 100 графіків одночасно, буде проігнорована, якщо вона не є інтуїтивно зрозумілою. Занадто детальні, перевантажені даними дашборди створюють "інформаційний шум", в якому губиться головне. Вдалий інтерфейс (UI/UX) має дотримуватись філософії мінімалізму та клінічної релевантності.

По-третє, це необхідність верифікації та валідації результатів аналізу. У медицині аналітичний висновок, що базується на помилковому алгоритмі, може мати фатальні наслідки. Тому, відповідно до методології наукового дослідження, система повинна проходити багаторівневу перевірку: технічну верифікацію (перевірка, що алгоритми реалізовані коректно) та клінічну

валідацію (співставлення результатів роботи системи з чинними клінічними протоколами та експертною оцінкою).

1.2.4. Виклики стандартизації та гармонізації даних в охороні здоров'я

Проблема гетерогенності та фрагментарності даних (п. 1.2.1) має своє глибоке коріння у відсутності єдиних, загальноприйнятих та, що найважливіше, імплементованих на практиці **стандартів даних та термінологій** в охороні здоров'я. Хоча існують численні спроби стандартизації на різних рівнях – від форматів обміну повідомленнями (HL7 v2, CDA, сучасний FHIR) та зберігання зображень (DICOM) до клінічних термінологічних систем (SNOMED CT, LOINC, RxNorm, МКХ) – їхнє реальне впровадження залишається вкрай нерівномірним. Кожна медична установа часто використовує власні, "пропрієтарні" кодифікатори, локальні словники або навіть довільні текстові позначення для тих самих понять. Це створює колосальну проблему семантичної інтероперабельності: системи можуть технічно обмінюватися даними, але "не розуміти" їхнього однакового значення. Процес гармонізації даних – приведення інформації з різних джерел до єдиної, уніфікованої моделі та термінології – є надзвичайно складним, трудомістким та дорогим завданням. Він вимагає створення та підтримки складних правил відображення (mapping), залучення клінічних експертів та використання спеціалізованих інструментів (ETL-платформ, термінологічних серверів). Без ефективної стандартизації та гармонізації будь-яка спроба побудувати централізовану аналітичну систему, що агрегує дані з багатьох джерел, приречена на роботу з неповними, суперечливими та семантично неузгодженими даними, що прямо впливає на якість та достовірність аналітичних висновків. [1], [4]

1.3. Архітектура та компаративний аналіз існуючих аналітичних рішень

Ефективність веб-аналітичних систем у медицині визначається не лише якістю алгоритмів, але й закладеними в них архітектурними принципами. Еволюція цих систем пройшла вражаючий шлях: від паперових архівів у реєстратурах, через появу перших електронних медичних записів (EHR), до впровадження інструментів Business Intelligence (BI), що дозволяли ретроспективно аналізувати поширення хвороб чи ефективність лікування. Сучасний етап характеризується впровадженням хмарних технологій та інтерактивних інтерфейсів.

1.3.1. Еволюція та типова архітектура МІС

Типова архітектура сучасної аналітичної системи є складним, багаторівневим комплексом:

1. Рівень збору даних (Ingestion Layer): Забезпечує інтеграцію з різноманітними джерелами: МІС (EHR/EMR), лабораторними системами (LIS), радіологічними системами (RIS/PACS), носимими пристроями (IoMT). Ключову роль тут відіграє підтримка стандартів обміну даними, таких як HL7 (Health Level Seven) та, особливо, сучасного стандарту FHIR (Fast Healthcare Interoperability Resources), що базується на REST API.
2. Рівень попередньої обробки та зберігання (Processing & Storage Layer): Тут зібрані «сирі» дані проходять етапи очищення (data cleansing), нормалізації, дедуплікації та, що критично, анонімізації/псевдонімізації. Для зберігання, залежно від задач, використовуються або традиційні сховища Data Warehouses (DWH) для структурованих даних, або Data Lakes (озера даних) для зберігання всіх типів даних у їхньому первинному вигляді.
3. Аналітичний рівень (Analytics Layer): «Серце» системи. Тут застосовуються методи статистичного аналізу, OLAP-куби для швидких запитів, а в більш просунутих системах — алгоритми машинного навчання.

4. Рівень візуалізації та представлення (Visualization Layer): "Обличчя" системи. Результати аналізу мають бути подані у формі, що є інтуїтивно зрозумілою та дієвою для кінцевого користувача (лікаря, адміністратора). Це і є ті самі інтерактивні панелі (дашборди), графіки та діаграми.

1.3.2. Компаративний аналіз платформ

Ринок аналітичних рішень для охорони здоров'я є широким та висококонкурентним. Однак, прискіпливий аналіз, що вимагається в рамках аналітичного огляду джерел, дозволяє виявити, що існуючі системи можна умовно поділити на кілька категорій, кожна з яких пропонує власнику (клініці) набір суттєвих компромісів.

Монолітні медичні інформаційні системи (MIC) від провідних світових вендорів або їхні локалізовані аналоги. Їхня беззаперечна перевага полягає у глибокій інтеграції: аналітика тісно пов'язана з первинними даними електронної картки. Однак, цей підхід не позбавлений фундаментальних недоліків. Аналітичні модулі в таких системах часто є "чорною скринькою". Лікар обмежений тим набором звітів, який заклав розробник. По-друге, це проблема "Vendor Lock-in": клініка стає повністю залежною від одного постачальника, а вартість ліцензій та будь-яких модифікацій часто є захмарною. Інтерфейси візуалізації в таких "важких" системах нерідко відстають від сучасних вимог до інтерактивності (UX/UI).

Універсальні BI-платформи (Business Intelligence), такі як гіганти ринку Tableau, Microsoft Power BI або Qlik. Ці інструменти пропонують найкращі на ринку можливості для створення складних, інтерактивних та естетично довершених дашбордів. Проте, їхня перевага є і їхнім головним недоліком: вони *універсальні*. Ці платформи "не знають" нічого про медичну специфіку. Вони не розуміють "з коробки" стандарти FHIR, DICOM чи правові вимоги HIPAA/GDPR. Уся колосальна складність очищення, гармонізації та анонімізації медичних даних повністю лягає на плечі IT-відділу клініки, що нівелює розрекламовану простоту. [5], [6]

Спеціалізовані хмарні платформи та AI-рішення від технологічних гігантів: Google Cloud Healthcare API, Microsoft Azure Health Data Services, Amazon HealthLake. Це надзвичайно потужні сервіси, що нативно підтримують стандарти FHIR та DICOM і надають готові інструменти машинного навчання. Але й тут є суттєві перепони. По-перше, це вартість та складність налаштування. А по-друге, і це *критична проблема*, — суверенітет даних (Data Sovereignty). Законодавство багатьох країн, включно з Україною, накладає суворі обмеження або пряму заборону на зберігання найбільш чутливих персональних даних (якими є медичні) на серверах, що фізично розташовані в інших юрисдикціях. Це робить використання публічних хмарних сервісів часто неможливим з юридичної точки зору. [4], [22]

Нарешті, рішення на базі Open-Source стеків, тобто побудова кастомних систем "з нуля". Найчастіше це Python-стек (з використанням Pandas для аналітики, FastAPI/Django для API, та бібліотек візуалізації типу Plotly/Dash або D3.js). [10]

Головна перевага тут — абсолютна гнучкість. Розробник має повний контроль над кожним аспектом системи. Відсутність ліцензійних платежів, прозорість алгоритмів та, що найважливіше, можливість розгортання системи "on-premise" (на локальних серверах клініки) гарантують максимальну безпеку та повну відповідність вимогам суверенітету даних. Однак, ціною за таку гнучкість є висока вартість та тривалість розробки й подальшої підтримки.

1.3.3. Новітні тенденції та майбутні напрямки розвитку архітектур медичної аналітики

Аналізуючи поточний стан (п. 1.3.2), важливо також окреслити новітні тенденції, що формують майбутнє архітектур медичних аналітичних систем. По-перше, це зростаюча роль Edge Computing. З поширенням медичних носимих пристроїв (IoMT) та засобів моніторингу в реальному часі виникає

потреба обробляти дані не лише в централізованих сховищах, але й безпосередньо "на краю" мережі – на самих пристроях або локальних шлюзах. Це дозволяє зменшити затримки, заощадити трафік та забезпечити швидку реакцію на критичні події (наприклад, виявлення аритмії). По-друге, це розвиток методів федеративного навчання (Federated Learning). Усвідомлюючи проблеми приватності та суверенітету даних (п. 1.3.2), цей підхід пропонує тренувати моделі машинного навчання децентралізовано, без необхідності передавати "сирі" дані пацієнтів до єдиного центру. Модель "мандрує" по локальних сховищах даних (різних лікарнях), навчається на місці, а до центру передаються лише узагальнені оновлення моделі, що значно підвищує рівень конфіденційності. По-третє, це все глибша інтеграція штучного інтелекту та машинного навчання (AI/ML) не просто як окремого аналітичного інструменту, а як невід'ємної частини самої архітектури – від інтелектуального ETL та автоматичного покращення якості даних до предиктивного моделювання та генерації клінічних рекомендацій безпосередньо у робочому процесі лікаря. Ці тенденції свідчать про рух до більш розподілених, інтелектуальних та конфіденційно-орієнтованих архітектур майбутнього.

Висновки до розділу 1

Проведений у цьому розділі комплексний аналіз предметної області, що охоплював як огляд джерел, так і системний аналіз проблеми, дозволив чітко окреслити фундаментальну проблему, що стоїть перед сучасною медичною інформатикою. Ми встановили, що медичні дані є унікальним за своєю гетерогенністю та чутливістю ресурсом, причому їхня справжня цінність полягає не в статичній фіксації, а в глибокій контекстній інтерпретації, аналізі динаміки та виявленні складних синдромних взаємозв'язків.

Водночас компаративний аналіз існуючих платформ (п. 1.3.2) виявив суттєвий стратегічний розрив на ринку. Сучасні медичні заклади, по суті, змушені обирати між неоптимальними альтернативами: з одного боку, це ригідні, дорогі та часто технологічно застарілі монолітні МІС; з іншого —

гнучкі універсальні BI-платформи, які, однак, абсолютно "сліпі" до медичної специфіки, стандартів та вимог безпеки. Новітні хмарні AI-платформи, попри свою потужність, створюють практично нездоланну юридичну проблему суверенітету даних, що є неприйнятним для зберігання найбільш чутливої інформації пацієнтів на локальному рівні.

Таким чином, дане дослідження чітко формулює проблему: існує гостра, незадоволена потреба у створенні такої веб-аналітичної системи, яка б заповнила цю нішу. Така система має гармонійно поєднати гнучкість, прозорість та безпеку open-source підходів, включно з критично важливою можливістю локального розгортання, з візуальною потужністю та інтерактивністю сучасних BI-систем. При цьому вона повинна бути іманентно адаптованою до медичних стандартів та протоколів безпеки.

Відповідно, метою даної роботи визначено проектування та розробку прототипу саме такої гнучкої, масштабованої та безпечної веб-аналітичної системи, орієнтованої на візуалізацію ключових показників. Вона покликана стати доступним інструментом для клінік, що прагнуть поглиблено аналізувати власні дані на власних серверах, не потрапляючи у фінансову та технологічну залежність від дорогих ліцензій чи хмарних провайдерів.

Для досягнення цієї мети, спираючись на проведений аналіз, було обґрунтовано вибір засобів вирішення проблеми. Вибір зроблено на користь сучасного та гнучкого технологічного стеку, що базується виключно на відкритих технологіях. Для серверної частини (backend) обрано мову Python, що є де-факто галузевим стандартом для аналізу даних. Це рішення дозволяє використовувати потужні бібліотеки (Pandas, NumPy) та сучасні веб-фреймворки (FastAPI або Django) для створення захищеного та високопродуктивного REST API. [8]

Клієнтська частина (frontend) буде реалізована за допомогою провідного JavaScript-фреймворку, такого як React або Vue.js, що є оптимальним вибором для побудови складних, динамічних та чутливих до дій користувача дашбордів. Надійною основою для зберігання даних слугуватиме реляційна СУБД PostgreSQL, обрана за її високу надійність, здатність ефективно виконувати складні аналітичні запити та нативну підтримку JSONB-полів для

напівструктурованих даних. Компонент візуалізації буде реалізовано комбінацією спеціалізованих бібліотек, як-от Chart.js для стандартних графіків та D3.js для розробки більш складних, кастомних візуалізацій.

Важливо підкреслити, що хоча ядром даної роботи є саме детермінована аналітика та інтерактивна візуалізація, обрана архітектура, особливо з Python у її основі, закладає міцний фундамент для майбутнього розширення. Як можливий напрямок подальшого розвитку, система може бути легко доповнена модулями штучного інтелекту, про що ми згадуємо лише в якості *потенційного покращення*. Йдеться про інтеграцію моделей машинного навчання для вирішення завдань предиктивної аналітики: прогнозування ризиків розвитку ускладнень, кластеризації пацієнтів за схожими фенотипами чи формування автоматизованих рекомендацій.

Отже, запропонована в роботі розробка покликана вирішити конкретні, ідентифіковані в ході аналізу проблеми гнучкості, безпеки, суверенітету даних та візуальної інтерпретації, які є "сліпою зоною" для більшості існуючих на ринку рішень. Наступний розділ буде присвячено безпосередньому проектуванню архітектури цього програмного прототипу.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА СИСТЕМИ ВЕБ-АНАЛІТИКИ МЕДИЧНИХ ДАНИХ

Після всебічного системного аналізу предметної області, проведеного у Розділі 1, стає очевидним, що розробка ефективної веб-аналітичної системи для медичних даних є нетривіальною задачею. Критичний аналіз існуючих рішень виявив фундаментальний ринковий розрив: між важкими, монолітними та "закритими" МІС з одного боку, та гнучкими, але "сліпими" до медичної специфіки універсальними BI-платформами з іншого. Наша мета, сформульована у висновках до попереднього розділу, полягає у проектуванні прототипу, що поєднує гнучкість, прозорість та безпеку рішень на базі відкритих технологій, із аналітичною потужністю та сучасною візуалізацією.

Цей розділ присвячений безпосередньому проектуванню програмного продукту та є логічним продовженням попереднього аналізу. Ми переходимо від постановки проблеми до її вирішення, детально обґрунтовуючи вибір методів та засобів розробки, як того вимагають методологічні засади наукового дослідження. Вибір кожного компонента технологічного стеку є свідомим та аргументованим, спрямованим на вирішення специфічних викликів, окреслених у Розділі 1.

2.1. Вибір та обґрунтування методів та засобів розробки

Вибір технологічного стеку (сукупності мов програмування, фреймворків, баз даних та бібліотек) є фундаментальним архітектурним рішенням. Він визначає не лише продуктивність та масштабованість майбутньої системи, але й швидкість розробки, вартість підтримки та, що найважливіше, здатність системи вирішувати поставлені аналітичні задачі. Ми проектуємо сучасну трирівневу веб-архітектуру (Client-Server-Database), компоненти якої детально проаналізовані нижче.

2.1.1. Обґрунтування вибору серверної частини (Backend)

Серверна частина, або Backend, є "центральною нервовою системою" та аналітичним ядром нашої системи. На неї покладається набагато більше відповідальності, ніж на стандартний веб-сервер, що просто обслуговує CRUD-операції (Створення, Читання, Оновлення, Видалення). Саме Backend буде відповідати за найскладніші процеси: підключення до гетерогенних джерел даних (проблема, описана в п. 1.2.1), їх очищення, агрегацію, гармонізацію (ETL), а також за виконання складних аналітичних обчислень перед тим, як передати готові, структуровані дані на візуалізацію.

Проаналізувавши ринкові альтернативи, ми зупинили свій вибір на мові Python. Це рішення є стратегічним. Звісно, можна було б розглянути Node.js, який демонструє видатну продуктивність в асинхронних операціях вводу-виводу (I/O) і є чудовим вибором для систем реального часу. Проте, його екосистема для глибокого чисельного аналізу даних та машинного навчання значно поступається Python. Інша альтернатива – корпоративні стандарти як Java (зі Spring) або C# (з.NET Core) – пропонують високу надійність, строгу типізацію та потужні засоби для побудови складних систем. Однак, для нашої задачі, що вимагає гнучкості та швидкого прототипування аналітичних моделей, ці фреймворки видаються надмірно громіздкими та багатослівними.

Вибір Python обґрунтований його унікальним положенням як де-факто lingua franca у світі Data Science та аналізу даних. Ця перевага є вирішальною:

1. Найпотужніша екосистема для Data Science: Python надає доступ до "золотого стандарту" бібліотек: Pandas для маніпуляцій з табличними даними (очищення, агрегація, трансформація), NumPy та SciPy для високопродуктивних наукових та математичних обчислень. Це дозволяє нам реалізовувати складну аналітичну логіку безпосередньо "всередині" backend-сервісу, а не перекладати її на рівень бази даних чи клієнта. [34]
2. Фундамент для майбутнього: Як ми зазначили у висновках до Розділу 1 (п. 1.4), обрана архітектура має закладати фундамент для майбутньої інтеграції модулів ШІ. Наявність бібліотек Scikit-learn (для класичного машинного навчання) та TensorFlow/PyTorch (для глибокого навчання) робить

Python єдиним логічним вибором для системи, що може еволюціонувати у бік предиктивної аналітики.

3. Швидкість розробки та інтеграції: Чистий синтаксис та величезна кількість готових модулів значно прискорюють процес розробки та інтеграції з існуючими МІС.

У межах екосистеми Python ми обрали сучасний веб-фреймворк FastAPI. Хоча Django є чудовим, надійним та "повноцінним" (batteries-included) фреймворком, він є дещо монолітним. Flask є мінімалістичним, але вимагає багато ручного налаштування. FastAPI поєднує найкраще з обох світів і надає критичні переваги для нашої задачі:

- Висока продуктивність: FastAPI побудований на асинхронних можливостях (ASGI) і працює поверх Uvicorn, що за швидкістю роботи наближається до Node.js. Це критично важливо для нашої системи, яка має швидко обробляти потенційно великі аналітичні запити, не блокуючи інші процеси.
- Автоматична документація API: Фреймворк автоматично генерує інтерактивну документацію OpenAPI (Swagger UI). Це безцінно для будь-якої сучасної системи, оскільки чітко документований API є основою для decoupled-архітектури (де frontend і backend розробляються незалежно) та майбутніх інтеграцій.
- Надійна валідація даних: Використання типізації Python та бібліотеки Pydantic забезпечує сувору валідацію всіх вхідних та вихідних даних "з коробки". Для системи, що працює з чутливими медичними даними, такий захист на рівні API допомагає запобігти ін'єкціям та пошкодженню даних. [10]

2.1.2. Обґрунтування вибору клієнтської частини (Frontend)

Клієнтська частина (Frontend) є "обличчям" та головним інструментом взаємодії для кінцевого користувача – лікаря чи аналітика. Як ми встановили у Розділі 1 (п. 1.2.3), однією з ключових проблем існуючих систем є "когнітивне перевантаження" та "інформаційний шум". Тому наш frontend має бути не просто набором веб-сторінок, а повноцінним, швидким та інтуїтивно зрозумілим односторінковим додатком (SPA – Single Page Application).

Використання традиційного підходу з рендерингом на сервері (Multi-Page Application), де кожна дія (фільтрація, зміна дати) вимагає повного перезавантаження сторінки, є абсолютно неприйнятним для сучасного аналітичного дашборду. Нам потрібна технологія, що дозволяє миттєво оновлювати компоненти інтерфейсу у відповідь на дії користувача.

Для цієї задачі ми обрали бібліотеку React (розроблену Facebook/Meta), хоча близькою за можливостями альтернативою є фреймворк Vue.js. Наш вибір на користь React обґрунтований такими факторами:

1. Компонентний підхід та декларативний UI: React дозволяє будувати складний інтерфейс з малих, незалежних та перевикористовуваних компонентів. Це ідеально для нашої задачі: ми можемо створити окремі компоненти, як-от LineChart (лінійний графік), PatientInfoCard (картка пацієнта), DateRangePicker (вибір дати), кожен з яких має власну логіку (state) і може бути легко протестований та скомбінований на будь-якій сторінці. Декларативний підхід (ми описуємо, як має виглядати UI в певному стані, а React дбає про те, як його оновити) значно спрощує розробку складних інтерфейсів.

2. Віртуальний DOM (Virtual DOM): Це ключова технологія React для забезпечення високої продуктивності. Замість того, щоб маніпулювати повільною DOM-моделлю браузера при кожній зміні даних, React оновлює легку копію (Virtual DOM) в пам'яті, обчислює мінімально необхідні зміни ("diff") і лише потім точково застосовує їх до реального DOM. Для нас це означає, що при отриманні нових даних, буде перемальовано лише конкретне число або точка на графіку, а не весь інтерфейс, що забезпечує виняткову плавність роботи.

3. Величезна екосистема та підтримка: React має найбільшу спільноту розробників, що гарантує наявність готових рішень для будь-якої задачі – від управління станом (Redux, Zustand) до готових "обгорток" для всіх бібліотек візуалізації, які ми плануємо використовувати. [9]

2.1.3. Обґрунтування вибору системи управління базами даних (СУБД)

Вибір СУБД є одним з найважливіших архітектурних рішень, оскільки він визначає надійність зберігання, швидкість вибірки даних та гнучкість їхньої структури. У Розділі 1ми встановили, що ключовий виклик медичних даних – це їх гетерогенність: вони містять як суворо структуровані дані (ID пацієнта, дата візиту, код аналізу, числове значення), так і напівструктуровані (нотатки лікаря у вільній формі, описи симптомів, складні результати аналізів у форматі JSON).

Це ставить нас перед вибором:

- Реляційні СУБД (SQL), як-от MySQL або MS SQL Server, є чудовими для структурованих даних та гарантують цілісність (ACID), але є негнучкими для зберігання довільних текстових нотаток чи складних вкладених об'єктів.
- Документо-орієнтовані СУБД (NoSQL), як-от MongoDB, є ідеальними для гнучких, напівструктурованих даних (зберігання всього у JSON). Проте, для аналітичної системи вони мають критичний недолік: вкрай обмежені (або відсутні) можливості для транзакцій та складних JOIN-запитів між різними колекціями. Виконати запит на кшталт "показати середній рівень глюкози для всіх пацієнтів старше 50 років з діагнозом X, які приймали препарат Y" у NoSQL-системі є значно складнішою задачею, ніж у SQL.

Тому нашим вибором стала об'єктно-реляційна СУБД PostgreSQL. Вона є "золотою серединою" і надає унікальне поєднання переваг обох світів, що ідеально вирішує нашу проблему:

1. Надійність та ACID-сумісність: PostgreSQL – це одна з найнадійніших реляційних СУБД з повною підтримкою транзакцій (ACID). Для медичних даних, де ціна помилки є надзвичайно високою, гарантія цілісності даних є абсолютною вимогою, а не опцією.

2. Гібридна модель даних (SQL + NoSQL): "Вбивчою" перевагою PostgreSQL є нативна підтримка бінарного формату JSONB. Це дозволяє нам проектувати "гібридну" схему даних: ми зберігаємо суворо структуровані дані (ID пацієнта, дата візиту, код аналізу) у звичайних реляційних стовпцях, а всі напівструктуровані дані (довільні нотатки лікаря, результати комплексного аналізу у вкладеному JSON) – в одному потужному стовпці типу JSONB.

3. Потужна індексація та аналітика: PostgreSQL дозволяє створювати GIN-індекси безпосередньо всередині JSONB-полів, що робить запити до цих напівструктурованих даних надзвичайно швидкими. Крім того, наявність розширених аналітичних SQL-функцій (віконні функції, Common Table Expressions) дозволяє виконувати значну частину агрегації даних безпосередньо на рівні БД, розвантажуючи наш Python-сервер. [11]

2.1.4. Обґрунтування вибору засобів візуалізації

Візуалізація – це кінцевий продукт, який бачить лікар, та інструмент, що перетворює "сирі" дані на клінічне знання. Як ми встановили у Розділі 1, комерційні BI-платформи, хоч і потужні, є "чорними скриньками", що обмежують гнучкість. Наша мета – повний програмний контроль над відображенням, щоб мати можливість реалізувати будь-яку, навіть нестандартну, медичну візуалізацію.

Ми не будемо обмежуватися однією бібліотекою, а застосуємо двокомпонентний (або гібридний) підхід, що базується на JavaScript-бібліотеках, інтегрованих у наш React-додаток:

1. D3.js (Data-Driven Documents): Це – низькорівнева "граматика" веб-візуалізації. D3 не є бібліотекою готових графіків; це інструмент для маніпуляції DOM-елементами (SVG, HTML) на основі даних. Ми будемо

використовувати D3 для реалізації унікальних, кастомних (bespoke) візуалізацій, які неможливо створити стандартними засобами.

Наприклад:

1) Інтерактивні теплові карти (heatmaps) для кореляції десятків показників.
2) Динамічні "таймлайни" історії хвороби пацієнта.
3) Специфічні діаграми, що відображають показники на схемі людського тіла. D3 дає нам безмежну гнучкість для вирішення таких нестандартних завдань.

2. Chart.js (або ECharts/Plotly.js): Для 80% наших завдань (стандартні лінійні графіки динаміки, стовпчикові діаграми, кругові діаграми розподілу) використовувати D3 "з нуля" є недоцільним та занадто витратним за часом ("стріляти з гармати по горобцях"). Для цих типових, стандартних завдань ми використаємо високорівневу бібліотеку, як-от Chart.js. Вона надає готові, чудово документовані, інтерактивні (з анімаціями, тултіпами, зумом) та естетично привабливі компоненти, які легко інтегруються в React (через react-chartjs-2) і дозволяють реалізувати більшість дашбордів за лічені години, а не тижні. [9], [29]

2.2. Архітектура та проектування системи

Маючи визначений та обґрунтований технологічний стек, ми переходимо до ключового етапу – проектування архітектури майбутнього програмного продукту. Архітектура системи є її "скелетом", що визначає не лише спосіб взаємодії компонентів, але й її довгострокові характеристики: масштабованість, надійність, простоту підтримки та, що найважливіше для нашої задачі, рівень безпеки. Наша мета – спроектувати систему, яка буде достатньо гнучкою для вирішення поточних завдань візуалізації та водночас достатньо міцною, щоб слугувати фундаментом для майбутнього розширення, наприклад, модулями предиктивної аналітики, про що йшлося у Розділі 1.

2.2.1. Загальна архітектура програмного продукту

Для нашої системи веб-аналітики медичних даних обрано класичну, перевірену часом трирівневу архітектуру (Three-Tier Architecture). Ця модель передбачає чіткий логічний та фізичний поділ системи на три основні компоненти:

1. Рівень представлення (Frontend Client): Це клієнтський додаток, що виконується у веб-браузері користувача (лікаря, адміністратора). Його єдина відповідальність – відображення даних та надання інтерактивного інтерфейсу. Він не містить жодної бізнес-логіки чи прямого доступу до бази даних. Цю роль виконуватиме наш односторінковий додаток (SPA), розроблений на React.
2. Рівень логіки (Backend/Application Server): Це "серце" нашої системи, реалізоване на Python та FastAPI. Цей сервер виступає посередником між клієнтом та базою даних. Він виконує всю складну роботу: обробляє HTTP-запити від клієнта, реалізує всю бізнес-логіку (агрегацію, аналітичні обчислення), керує автентифікацією та авторизацією користувачів і, нарешті, формує запити до бази даних.
3. Рівень даних (Database Server): Це наша СУБД PostgreSQL, що відповідає виключно за надійне зберігання, індексацію та вибірку даних. Вона ізольована від зовнішнього світу і приймає запити лише від довіреного backend-сервера.

Такий поділ забезпечує низку критичних переваг. По-перше, це безпека: клієнт ніколи не має прямого доступу до бази даних, що унеможливлює безліч векторів атак. Уся взаємодія проходить через API, захищений механізмами автентифікації. По-друге, це масштабованість: якщо зросте навантаження, ми можемо незалежно масштабувати компоненти – додати більше серверів додатків (backend) або потужності серверу БД, не чіпаючи клієнтську частину. По-третє, це гнучкість розробки: команди frontend та backend можуть працювати паралельно, опираючись на чітко визначений "контракт" – API.

Взаємодія компонентів відбувається наступним чином:

1. Користувач (лікар) у своєму React-додатку (Frontend) натискає кнопку "Показати динаміку показників для пацієнта X".

2. Frontend формує захищений HTTP-запит до нашого Backend-сервера.

3. Backend (FastAPI) отримує запит, перевіряє права доступу користувача (чи має цей лікар право бачити дані пацієнта X), виконує складний SQL-запит до бази даних PostgreSQL для агрегації необхідних показників.

4. База даних повертає "сирі" дані.

5. Backend (Python/Pandas) додатково обробляє ці дані, приводить їх до формату, зручного для графіків, і відправляє клієнту у вигляді JSON.

6. Frontend (React) отримує JSON і за допомогою бібліотек (Chart.js/D3.js) "малює" інтерактивний графік для лікаря.

2.2.2. Проектування моделі даних (Бази даних)

Проектування бази даних є, без перебільшення, найважливішим етапом проектування аналітичної системи. Помилки на цьому етапі призводять до неможливості виконати певні аналітичні запити або до катастрофічного падіння продуктивності. Наша модель даних має вирішувати завдання, поставлені в Розділі 1: зберігати гетерогенні, лонгітюдні дані та підтримувати як структуровані, так і напівструктуровані формати.

Ми використовуватимемо реляційну модель, реалізовану в PostgreSQL, з гібридним підходом до зберігання даних (використання JSONB-полів). Ключові сутності (entities) нашої системи:

1. Users (Користувачі): Ця таблиця зберігатиме облікові дані медичного персоналу. Ключові атрибути: user_id (PK), username (логін), password_hash (хеш пароля), full_name (ПІБ), role (роль: 'лікар', 'адміністратор').

2. Patients (Пацієнти): Основна сутність, що представляє пацієнта. Критично важливо, що ця таблиця має містити мінімум персональної ідентифікуючої інформації; натомість ми будемо використовувати псевдонімізований ідентифікатор. Атрибути: patient_id (PK),

pseudo_anonymous_id (унікальний ідентифікатор), date_of_birth (дата народження), sex (стать). Інші демографічні дані можуть зберігатися в окремому, захищеному полі.

3. Visits (Візити/Випадки): Лонгітюдний запис про взаємодію пацієнта з клінікою. Атрибути: visit_id (PK), patient_id (FK до Patients), visit_date (дата візиту), diagnosis_code (напр., код МКХ), notes_json (поле типу JSONB для зберігання довільних нотаток лікаря).

4. ObservationTypes (ТипиПоказників): Довідник для стандартизації аналізів. Атрибути: type_id (PK), name (назва, напр., "Глюкоза (плазма)"), code (внутрішній код або код LOINC), unit (одиниця виміру, напр., "ммоль/л").

5. Observations (Показники/Спостереження): "Найгарячіша" та найбільша таблиця системи, що зберігає конкретні результати аналізів. Атрибути: observation_id (PK), patient_id (FK до Patients), visit_id (FK до Visits, опціонально), type_id (FK до ObservationTypes), observation_datetime (точна дата і час), value_numeric (числове значення), value_text (текстове значення, якщо є).

Така структура дозволяє нам виконувати надзвичайно гнучкі та швидкі аналітичні запити, наприклад, "вибрати всі value_numeric з таблиці Observations, де type_id відповідає 'Глюкозі', для patient_id = 123, та відсортувати за observation_datetime". Це і є основа для побудови графіків динаміки.

(Тут має бути розміщена Діаграма Класів (Class Diagram) або ER-діаграма (Entity-Relationship Diagram), що детально показує вищеописані сутності, їхні атрибути та зв’язки (one-to-many, many-to-many) між ними).

2.2.3. Проектування серверного API (Backend)

Серверний API є "контрактом" або мовою спілкування між нашим React-додатком (Frontend) та Python-сервером (Backend). Ми обираємо архітектурний стиль RESTful API через його простоту, стандартизованість та широке розповсюдження. Всі дані будуть передаватися у форматі JSON.

Для забезпечення безпеки кожна "захищена" точка доступу (ендпоінт) вимагатиме наявності JWT (JSON Web Token) в заголовку запиту. Користувач отримує цей токен під час логіну і передає його з кожним наступним запитом, що дозволяє серверу верифікувати особу користувача та його права доступу. [10]

2.2.4. Проектування інтерфейсу користувача (UI/UX)

Проектування інтерфейсу (UI) та досвіду взаємодії (UX) у медичних системах виходить далеко за межі суто естетичних завдань. Як було встановлено у ході системного аналізу, кінцеві користувачі - лікарі — часто працюють в умовах жорсткого дефіциту часу, стресу та високої ціни помилки. Тому "когнітивне перевантаження" інтерфейсу може призвести до реальних клінічних ризиків.

Виходячи з цього, наша філософія дизайну базується на трьох ключових принципах:

1. Мінімізація інформаційного шуму: На екрані має бути лише та інформація, яка необхідна для прийняття поточного рішення.
2. Швидкість доступу до інсайтів: Кількість кліків для перегляду критичної динаміки пацієнта має бути зведена до мінімуму.
3. Чітка візуальна ієрархія: Критичні відхилення від норми (аномальні показники) повинні візуально домінувати, до прикладу – виділятися кольором або спеціальними маркерами.

На основі цих принципів ми спроектували інтерфейси для двох ключових акторів системи:

А. **Інтерфейс Лікаря** (Основний робочий простір) Робота лікаря з системою розділена на два логічні рівні: оглядовий та детальний аналіз.

а) **Головна панель лікаря** (Рівень моніторингу): Цей екран є стартовою точкою роботи. Замість простого списку пацієнтів, він реалізований як "центр сповіщень". Основний акцент зроблено на

пацієнтах, які потребують уваги: система автоматично піднімає вгору списку тих, у кого останні аналізи вийшли за межі референтних норм. Це дозволяє лікарю миттєво фокусуватися на пріоритетних випадках.

b) **Екран "Електронна картка пацієнта"** (Рівень детального аналізу): Це центральний аналітичний інструмент системи. Екран спроектовано за модульним принципом, де інформація подається від загальної до конкретної:

c) **Верхня панель**: Ключова демографія та незмінні дані (вік, група крові, хронічні діагнози).

d) **Центральна аналітична зона**: Саме тут розташовуються інтерактивні візуалізації, вибір яких обґрунтовано у п. 2.1.4. За замовчуванням відображаються лінійні графіки (Line Charts) для ключових життєвих показників, що дозволяє миттєво оцінити їх в довгостроковій перспективі. Для глибшого аналізу лікар може переключитися на режим теплової карти (Heatmap), щоб побачити кореляції між різними групами аналізів за великий проміжок часу.

e) **Нижня панель**: Табличний вигляд історії всіх візитів для доступу до "сирих" даних та текстових нотаток.

Б. **Інтерфейс Адміністратора** (Технічний рівень) Цей інтерфейс відокремлений від лікарського і сфокусований на забезпеченні життєдіяльності системи. Він включає лаконічні, функціональні екрани для керування обліковими записами персоналу (CRUD-операції над сутністю Users) та спеціалізований інтерфейс для масового імпорту даних (завантаження CSV/JSON файлів з результатами лабораторних досліджень), який надає зворотний зв'язок щодо статусу обробки та можливих помилок у вхідних даних.

2.3. Моделювання поведінки системи та користувачів

Після детального проектування статичної архітектури нашої системи у підрозділі 2.2, де були визначені ключові технологічні компоненти (React,

FastAPI, PostgreSQL) та їхнє призначення, наступним логічним та методологічно необхідним кроком є моделювання її динамічної поведінки. Статична архітектура описує, з чого складається система, тоді як динамічне моделювання описує, як вона функціонує, хто з нею взаємодіє, і де вона фізично буде розгорнута.

Для досягнення цієї мети ми застосовуємо Уніфіковану мову моделювання (UML), яка є загальновизнаним індустрійним стандартом для візуалізації, специфікації, конструювання та документування програмних систем. Використання UML-діаграм дозволяє нам перевести абстрактні вимоги та архітектурні рішення у чітку, формалізовану візуальну мову, що є обов'язковим етапом проектування складних інформаційних систем, як того вимагають методичні рекомендації. Даний розділ послідовно візуалізує функціональні вимоги, логіку взаємодії та фізичну топологію системи.

2.3.1. Аналіз варіантів використання (Use Case)

Першочерговим завданням при моделюванні поведінки є визначення функціональних меж системи та формалізація її взаємодії з зовнішнім світом. Для цього використовується Діаграма варіантів використання - Use Case. Вона моделює систему як "чорну скриньку" та визначає, які дії або процеси (варіанти використання) можуть ініціювати зовнішні суб'єкти (актори).

У нашій системі, відповідно до проектування інтерфейсів, ми чітко ідентифікуємо двох основних акторів:

1. Лікар (Doctor): Зареєстрований медичний працівник, що є основним споживачем аналітичних даних та використовує систему для моніторингу стану пацієнтів.
2. Адміністратор (Administrator): Технічний або управлінський персонал, відповідальний за підтримку цілісності системи, керування доступом та адміністрування даних.

На основі цих акторів ми визначаємо ключові варіанти використання (Use Cases), які описують повний функціонал нашої системи:

Для актора "Лікар":

1. Пройти автентифікацію: Базовий варіант використання, що є передумовою для доступу до будь-яких захищених даних пацієнтів.
2. Переглянути список пацієнтів: Доступ до персоналізованого списку пацієнтів, закріплених за цим лікарем, з метою вибору об'єкта аналізу.
3. Аналізувати дашборд пацієнта: Ключовий функціонал системи. Цей варіант включає в себе обов'язкові під-процеси: Перегляд графіків динаміки, Перегляд таблиці аналізів та Перегляд демографічної інформації.
4. Фільтрувати аналітичні дані: Можливість задавати часові діапазони або обирати конкретні типи показників для візуалізації.
5. Генерувати звіт: Можливість експортувати поточний аналітичний вигляд для друку або включення в офіційну медичну документацію.

Для актора "Адміністратор":

1. Пройти автентифікацію: Також базовий варіант.
2. Керувати обліковими записами: Включає <<extend>> варіанти: "Створити обліковий запис лікаря", "Видалити обліковий запис", "Призначити пацієнтів лікарю".
3. Завантажувати нові дані: Критично важлива функція для імпорту даних, наприклад, пакетних результатів аналізів з лабораторії (у форматі CSV чи JSON).
4. Переглядати загальну аналітику: Доступ до знеособленого, агрегованого дашборду по всій клініці (статистика по кількості пацієнтів, розподіл за віком/статтю тощо).

Ця структура слугує формалізованою "дорожньою картою" для подальшої розробки та функціонального тестування, чітко окреслюючи очікуваний функціонал системи – Додаток А.

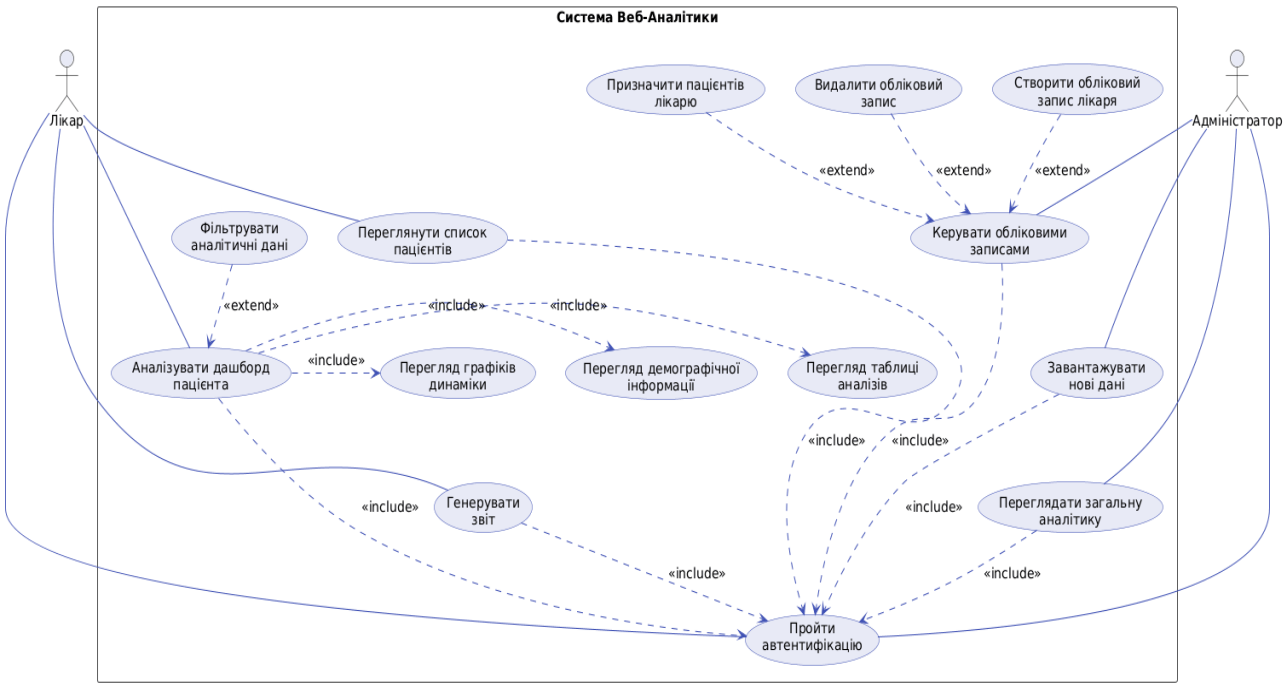


Рис.1 Use-Case Diagram

2.3.2. Моделювання ключових процесів (Sequence Diagram)

Якщо діаграми варіантів використання визначають статичний набір функціональних можливостей системи, то для проектування надійної архітектури необхідно детально змоделювати її динамічну поведінку. Нам важливо не лише те, що система робить, але й як саме компоненти, спроектовані, що взаємодіють у часі для досягнення результату. Для цього ми використовуємо Діаграми послідовності (Sequence Diagrams) мови UML. Вони дозволяють візуалізувати часову послідовність повідомлень (викликів) між різними об'єктами нашої трирівневої архітектури. Розглянемо найважливіший сценарій – **"Запит та відображення аналітичних даних пацієнта"**. Цей процес залучає всі три основні компоненти нашого програмного продукту: клієнтську частину (Frontend), серверну частину (Backend) та базу даних (Database).

Клієнтська частина, реалізована на React, виступає ініціатором процесу та відповідає за всю взаємодію з користувачем. Саме тут відбувається первинна подія – лікар обирає пацієнта та запитує візуалізацію його показників. Frontend формує асинхронний HTTP-запит до серверного API, належним чином упакувавши в нього ідентифікатор пацієнта та параметри запиту. Критично важливо, що цей запит також містить JWT-токен автентифікації, отриманий на етапі входу, що гарантує безпеку комунікації.

Серверна частина, реалізована на FastAPI (Python), виступає в ролі центрального контролера та "мозку" операції. Її відповідальність є багатошаровою. [25]

Першочерговим завданням backend є валідація запиту: він має перевірити JWT-токен, ідентифікувати користувача (лікаря) та верифікувати його права доступу – чи дійсно цей лікар має повноваження переглядати дані саме цього пацієнта. Це є ключовим елементом реалізації вимог HIPAA/GDPR, які ми ідентифікували у Розділі 1. Лише після успішної перевірки безпеки сервер приступає до бізнес-логіки. Він транслює отриманий API-запит у один або декілька складних SQL-запитів до бази даних.

Рівень даних, представлений нашою СУБД PostgreSQL, відповідає на запити від серверної частини. Його завдання – не просто віддати "сирі" рядки, а виконати на своєму боці максимально можливу частину роботи: фільтрацію за датами, складні об'єднання (JOIN) між таблицями (напр., Observations та Observation Types) та первинну агрегацію. Це дозволяє мінімізувати обсяг даних, що передається мережею між сервером БД та сервером додатків, що є критичним для продуктивності.

Після отримання результатів від бази даних, робота серверної частини не завершена. Вона виконує другий етап обробки – аналітичну трансформацію. Тут вступають в дію бібліотеки Python (Pandas/NumPy), які можуть виконувати обчислення, що є занадто складними для чистого SQL: розрахунок ковзних середніх, нормалізацію даних, підготовку спеціальних структур для складних графіків D3.js тощо. Лише після цієї фінальної обробки Backend серіалізує дані у компактний формат JSON та відправляє їх у відповіді (HTTP 200 OK) клієнтській частині. Отримавши готовий JSON, Frontend (React) оновлює свій внутрішній стан (state), що, в свою чергу, передає нові дані компонентам візуалізації (Chart.js/D3.js), які перемальовують графіки на екрані лікаря. Цей асинхронний потік даних (Client, Backend, Database, Backend, Client) є фундаментальним патерном для всієї аналітичної функціональності системи.

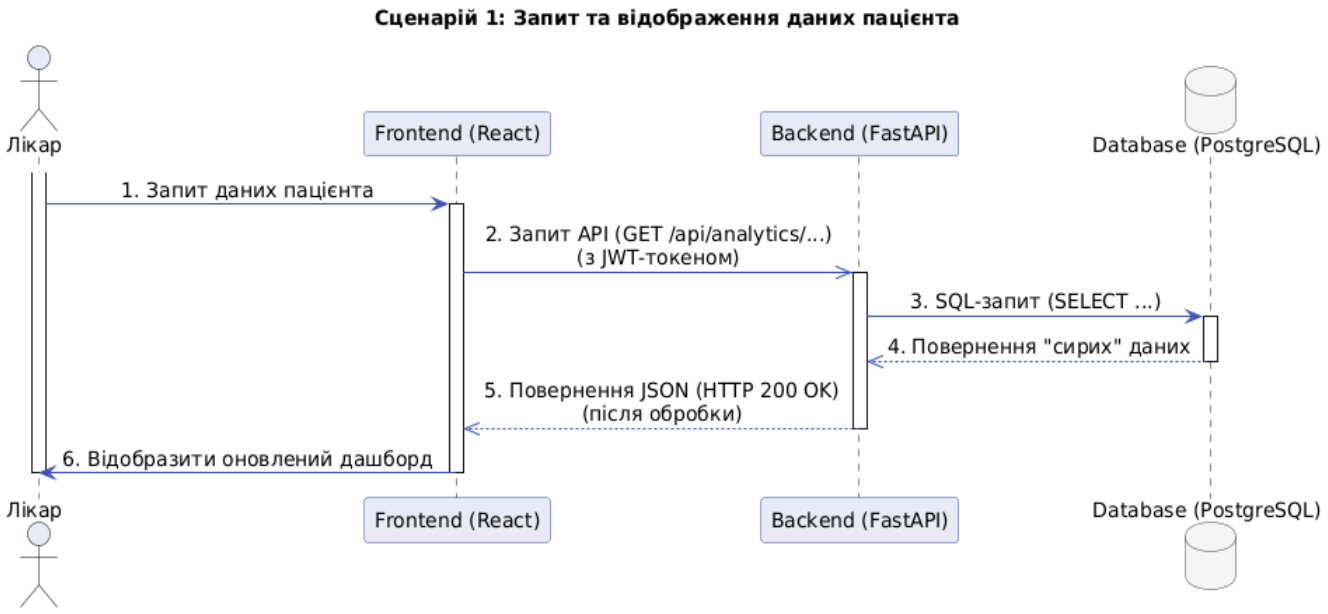


Рис.2 Sequence Diagram – Сценарій із запитом на відображення даних пацієнта

Подібні діаграми послідовності моделюють й інші критичні процеси. Наприклад, сценарій **"Завантаження нових даних"** (ініційований Адміністратором) демонструватиме інший потік: Frontend надсилає POST-запит з файлом, Backend проводить валідацію файлу, розбирає його, а потім виконує серію INSERT або UPDATE SQL-команд до Database та повертає звіт про успішність операції.

Рис.3 Sequence Diagram – Додавання нових даних до системи

Моделювання цих процесів за допомогою діаграм послідовності є ключовим етапом проектування. Воно дозволяє не лише верифікувати повноту нашого API, але й виявити потенційні "вузькі місця" продуктивності ще до початку написання коду.

2.3.3. Моделювання розгортання системи (Deployment Diagram)

Нарешті, для завершення архітектурного проектування, необхідно змоделювати фізичну топологію системи. Для цього використовується Діаграма розгортання (Deployment Diagram). Вона візуалізує, як саме програмні компоненти, або артефакти (виконувані файли, бібліотеки, файли бази даних),

будуть розподілені по фізичних чи віртуальних вузлах (серверах, клієнтських комп'ютерах) у реальному середовищі експлуатації.

Ця діаграма має особливе, принципове значення в контексті нашої роботи. Як було детально проаналізовано у Розділі 1, одним із ключових недоліків провідних хмарних платформ (Google Cloud, Azure) є юридична та практична проблема суверенітету даних. Для медичних закладів вимога зберігати чутливі дані пацієнтів виключно в межах власного захищеного периметра є фундаментальною. Наша архітектура, на противагу хмарним аналогам, від початку проектується для локального розгортання в межах внутрішньої мережі медичного закладу, і діаграма розгортання покликана це візуалізувати.

Фізична топологія нашої системи включатиме три основні вузли. Першим вузлом є «Клієнтський ПК» (User Workstation), що представляє собою фізичний комп'ютер (пристрій типу <<device>>), за яким працює кінцевий користувач – Лікар або Адміністратор. На цьому вузлі виконується стандартний веб-браузер, всередині якого, в свою чергу, завантажується та виконується артефакт нашого клієнтського додатку – Frontend (React Application).

Другим, центральним вузлом є «Сервер Додатків» (Application Server). Це фізичний або віртуальний сервер (типу <<server>>), розташований у захищеній локальній мережі (LAN) клініки. Цей вузол є багатокомпонентним і розміщує на собі два ключові програмні артефакти. По-перше, це веб-сервер, який виконує роль реверс-проксі, термінує безпечні HTTPS-з'єднання та віддає статичні файли React-додатку. По-друге, це безпосередньо наш серверний додаток – Сервер API (FastAPI/Uvicorn), що є виконуваним Python-процесом, який реалізує всю бізнес-логіку системи.

Третім вузлом є «Сервер Бази Даних» (Database Server). Це окремий, високозахищений сервер (типу <<server>>), що фізично знаходиться в тій самій локальній мережі, але, з міркувань безпеки, має бути розміщений в ізольованому сегменті мережі. На цьому вузлі розгорнуто єдиний основний артефакт – СУБД PostgreSQL (Database Engine). Безпосередньо на цьому ж вузлі фізично зберігаються файли бази даних (артефакт Medical_DB), що містять всю конфіденційну інформацію.

Комунікаційні шляхи між цими вузлами суворо регламентовані. «Клієнтський ПК» взаємодіє із «Сервером Додатків» виключно по захищеному протоколу HTTPS. «Сервер Додатків», у свою чергу, взаємодіє із «Сервером Бази Даних» по внутрішньому протоколу TCP/IP (через стандартний порт PostgreSQL 5432). Критично важливо, що між «Клієнтським ПК» та «Сервером Бази Даних» пряме з'єднання відсутнє та заборонено на рівні мережевих брандмауерів (Firewall). Доступ до бази даних можливий виключно через «Сервер Додатків», який виступає єдиним шлюзом. Така топологія чітко демонструє, що жодні чутливі медичні дані ніколи не залишають захищений периметр медичного закладу, що є фундаментальною перевагою нашого проекту та вирішенням проблеми суверенітету даних – Додаток Б.

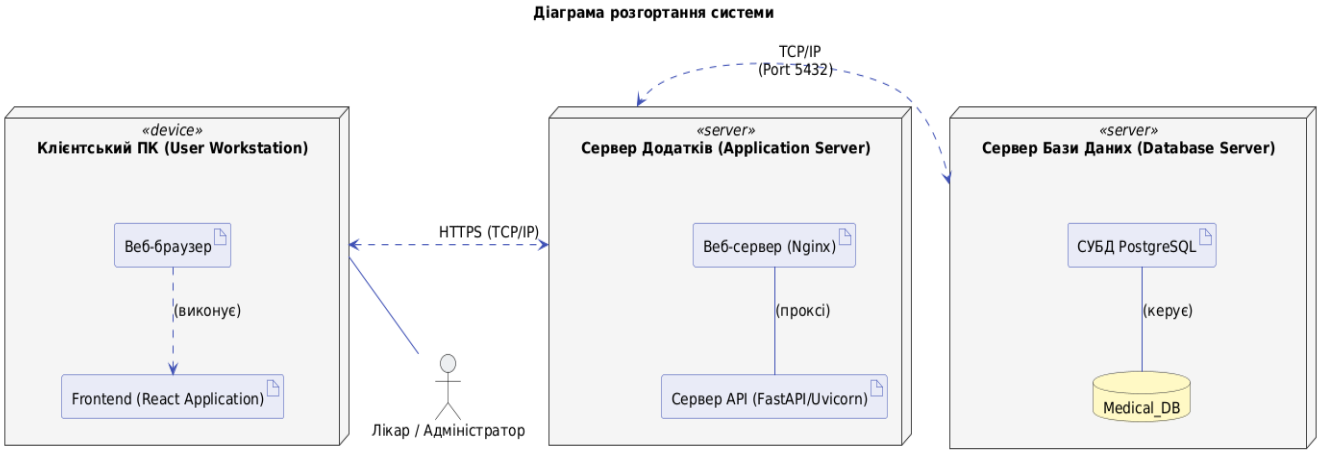


Рис.4 Deployement Diagram Системи

2.4. Проектування механізмів безпеки та конфіденційності

Проведений у Розділі 1 аналіз предметної області чітко продемонстрував, що при роботі з медичними даними питання безпеки та конфіденційності виходять на перший план. Вони є не просто додатковою опцією, а фундаментальною, нефункціональною вимогою, що визначає саму життєздатність та легітимність системи. Високий ступінь чутливості цієї

інформації, а також суворі правові рамки, такі як GDPR та HIPAA, диктують необхідність впровадження багаторівневого захисту. На відміну від багатьох хмарних аналогів, що мають проблеми з суверенітетом даних, наша архітектура від самого початку будується на принципі «Безпека через проектування» (Security by Design).

Першим та найважливішим рівнем нашого захисту є архітектурна та мережева ізоляція, детально змодельована у діаграмі розгортання. Вибір на користь локального, "on-premise" розгортання є свідомим стратегічним рішенням. Це означає, що всі програмні артефакти та, що найголовніше, база даних, фізично знаходяться у захищеному периметрі медичного закладу. Це повністю вирішує проблему суверенітету даних, оскільки чутлива інформація ніколи не залишає внутрішню мережу клініки. Ця ізоляція посилюється чіткою мережевою сегментацією: Сервер Бази Даних (PostgreSQL) є повністю недоступним для клієнтських комп'ютерів. Він налаштований приймати з'єднання виключно з однієї довіреної адреси – нашого Сервера Додатків (FastAPI), який виступає єдиним шлюзом до даних.

Другий рівень – захист каналів комунікації. Вся взаємодія між клієнтським веб-браузером та Сервером Додатків в обов'язковому порядку здійснюється по захищеному протоколу HTTPS (SSL/TLS). Це забезпечує шифрування всього трафіку, унеможливлюючи перехоплення або модифікацію ("man-in-the-middle" атаки) запитів, що містять токени автентифікації чи дані пацієнтів.

Третій, програмний рівень, реалізує керування доступом безпосередньо в нашому backend-додатку. Цей рівень складається з двох частин: автентифікації та авторизації. Автентифікація – процес підтвердження особи користувача – буде реалізована через механізм JSON Web Tokens. При успішному вході (наданні коректного логіну та пароля), сервер генерує захищений цифровим підписом токен, який клієнт зберігає та додає до кожного наступного запиту до API. Це дозволяє уникнути необхідності передавати логін та пароль при кожному запиті. Для гарантування безпеки самого процесу входу, паролі користувачів у базі даних зберігатимуться не у відкритому вигляді, а виключно як криптографічний хеш, згенерований стійким асиметричним алгоритмом, таким як Bcrypt або Argon2.

Авторизація – процес визначення прав автентифікованого користувача – буде реалізована на основі моделі Керування доступом на основі ролей акторів. Як було визначено у діаграмі варіантів використання, ми маємо чіткі ролі: «Лікар» та «Адміністратор». Кожен захищений ендпоінт нашого FastAPI-сервера буде перевіряти не лише наявність валідного токена, але й роль, що в ньому міститься. Таким чином, користувач з роллю «Лікар» матиме доступ лише до аналітичних функцій (GET /api/analytics/...), в той час як спроба звернутися до ендпоінта керування користувачами (POST /api/admin/users) буде відхилена сервером з помилкою "403 Forbidden". Це забезпечує суворий поділ повноважень всередині системи.

Четвертий рівень стосується конфіденційності даних "у стані спокою", тобто безпосередньо в базі даних. Окрім стандартних засобів шифрування самої бази даних, які надає PostgreSQL, ми застосовуємо метод псевдонімізації на рівні проектування схеми. Основна таблиця Patients не містить прямих ідентифікаторів особи, таких як прізвище чи номер паспорта. Вона оперує виключно сурогатним ключем (patient_id) та pseudo_anonymous_id. Зв'язок між цим псевдонімом та реальною особою пацієнта має зберігатися в абсолютно окремій, високозахищеній системі, до якої наша аналітична система не має прямого доступу.

Нарешті, важливим елементом захисту є безпека самого коду API. Обраний нами фреймворк FastAPI, завдяки його базованості на типізації Pydantic, надає потужний механізм автоматичної валідації вхідних даних. Будь-який запит, що надходить на сервер, автоматично перевіряється на відповідність чітко визначеній схемі. Це не лише підвищує надійність коду, але й виступає першою лінією оборони проти цілого класу атак, пов'язаних з ін'єкціями (включно з SQL Injection), оскільки зловмисні дані просто не пройдуть валідацію і будуть відкинуті ще до потрапляння в бізнес-логіку.

2.5. Розробка алгоритмічної моделі для інтегральної клінічної оцінки

В попередніх розділах ми заклало надійний архітектурний фундамент для проектування архітектури системи веб-аналітики,. Ми визначили оптимальний технологічний стек (Python, FastAPI, React, PostgreSQL), спроектували трирівневу архітектуру, формалізували потоки даних за допомогою UML-діаграм та визначили ключові механізми безпеки. Таким чином, ми описали побудову потужного та безпечного контейнера для медичних даних.

Однак, для повноцінного вирішення завдань, поставлених у Розділі 1, недостатньо лише побудувати систему, що надійно зберігає та якісно відображає інформацію. Як було встановлено, ключовий виклик сучасної медицини полягає не у відсутності даних, а у складності їхньої своєчасної та коректної інтерпретації. Цей підрозділ присвячений проектуванню "аналітичного ядра" нашої системи – алгоритмічної моделі, що перетворює наш програмний продукт з інструменту пасивного моніторингу на систему активної підтримки прийняття рішень.

2.5.1. Постановка задачі: перехід від візуалізації даних до аналітичної підтримки

Як було детально сказано до цього, однією з фундаментальних проблем існуючих аналітичних підходів є так зване "когнітивне перевантаження" лікаря. Сучасний моніторинг хронічного пацієнта генерує десятки розтягнутих у часі, показників. Наша спроектована система надає потужні інструменти візуалізації – інтерактивні графіки (Chart.js) та теплові карти (D3.js), – що є значним кроком уперед порівняно зі статичними звітами.

Проте, навіть за наявності цих інструментів, у складних клінічних випадках медичний працівник змушений одночасно аналізувати 5, 10, або 15 окремих графіків, подумки зіставляючи їхні тренди, швидкість зміни та відхилення від норм. Такий "ручний" візуальний аналіз є не лише часозатратним, але й несе в собі значний ризик людської помилки – лікар може пропустити неочевидну, але небезпечну кореляцію між, на перший погляд, не

пов'язаними показниками, або невірно оцінити сукупну "вагу" декількох незначних, але одночасних погіршень.

Система, що лише пасивно відображає дані, перекладає всю складну аналітичну роботу на користувача. Для того, щоб наш програмний продукт надавав реальну, а не уявну, підтримку у прийнятті рішень, необхідно зробити наступний логічний крок: доповнити пасивну візуалізацію активною аналітичною моделлю.

Метою даного розділу є розробка та обґрунтування алгоритмічної моделі, яка б автоматично виконувала інтегральну (комплексну) оцінку клінічної значущості сукупності показників пацієнта в динаміці. Завдання цієї моделі – перетворити складний, багатовимірний масив розрізнених даних (з таблиці Observations) на єдиний, зрозумілий та інтерпретований показник, що миттєво сигналізує лікарю про загальний стан пацієнта та динаміку його ризиків. Це дозволить змістити акцент роботи лікаря з "пошуку" проблеми на графіках на "реакцію" на вже виявлену та оцінену системою загрозу.

2.5.2. Концепція моделі: «Інтегральний індекс клінічної значущості» (ІІКЗ)

Для вирішення поставленої задачі, ми пропонуємо впровадження «Інтегрального індексу клінічної значущості» (ІІКЗ). Це є ядром нашої наукової новизни та ключовим аналітичним компонентом системи, що розробляється. Концептуально, ІІКЗ – це розрахунковий, агрегований показник, що виражається єдиним числом (у нормалізованому діапазоні від 0 до 10).

Його призначення – надати лікарю миттєву, комплексну оцінку поточного стану пацієнта та рівня потенційних ризиків, синтезуючи інформацію з десятків розрізнених показників.

Замість того, щоб просто передавати масиви "сирих" даних з бази даних PostgreSQL на клієнтську частину (Frontend) для їх "пасивного" відображення, ми пропонуємо інший, більш інтелектуальний потік. Наш серверний компонент (Backend), реалізований на Python, виступатиме не просто як посередник, а як повноцінний аналітичний процесор. Отримавши запит від лікаря, сервер

спершу витягує необхідні дані, а потім застосовує до них розроблену нами алгоритмічну модель ІІКЗ. Лише після цього розрахунку, фінальний, збагачений результат (що включає як "сирі" дані для графіків, так і розрахований індекс) передається на візуалізацію.

Це переводить нашу систему до класу систем підтримки прийняття рішень, що є вимогою для сучасного медичного програмного забезпечення та підіймає рівень нашого дослідження.

2.5.3. Компоненти та параметри моделі

Для того, щоб ІІКЗ був не просто абстрактним числом, а клінічно релевантним показником, його алгоритмічна модель має враховувати багатовимірну природу медичних даних, проаналізовану нами раніше. Вхідними даними для моделі слугує набір лонгітюдних часових рядів для ключових показників пацієнта (рівень глюкози, артеріальний тиск, показники ліпідного профілю, рівень креатиніну тощо), які зберігаються у нашій проектованій таблиці Observations.

Наша модель ІІКЗ є композитною і базується на розрахунку трьох фундаментальних факторів для кожного з аналізованих показників:

1. **Статичне відхилення** (Static Deviation): Цей компонент оцінює, наскільки далеко поточне, останнє значення показника пацієнта відхилилося від встановленої референсної норми. Це є базовою оцінкою "позанормовості". Принциповим аспектом нашої моделі є те, що ця норма не є статичною константою. Вона має бути контекстуальною, враховуючи демографічні дані пацієнта (з таблиці Patients) – його вік та стать, оскільки референсні значення для дитини, дорослого чоловіка та літньої жінки можуть кардинально відрізнятися.

2. Динамічне відхилення або "Швидкість" (Dynamic Deviation/ Velocity): Цей компонент є ключовим для аналізу лонгітюдних даних (п. 1.1.3) і вирішує проблему, яку не бачить статичний аналіз. Він оцінює, наскільки швидко показник пацієнта погіршується або покращується. Наприклад, пацієнт,

чий рівень креатиніну стабільно тримається на високій, але незмінній позначці, представляє менший негайний ризик, ніж пацієнт, чий креатинін був у нормі місяць тому, але стрімко зріс за останній тиждень, хоча поточне значення може бути ще невисоким. Цей фактор вимагає аналізу тренду – наприклад, шляхом розрахунку похідної (швидкості зміни) або кута нахилу лінії лінійної регресії, побудованої за останніми N точками часового ряду.

3. Клінічна вага (Clinical Weight): Модель визнає, що не всі показники є однаково важливими для оцінки негайного ризику. Відхилення рівня калію в сироватці крові (що може призвести до аритмії) має значно вищу клінічну вагу та вимагає негайної реакції, ніж, наприклад, відхилення рівня загального холестерину, що є фактором довгострокового, а не негайного ризику. Тому наша модель вводить систему вагових коефіцієнтів (w), які мають бути визначені експертним шляхом (на основі клінічних протоколів) для кожного типу показника. Цей коефіцієнт буде визначати "внесок" кожного конкретного показника у фінальний інтегральний індекс.

2.5.4. Математична формалізація Інтегрального індексу (ІКЗ)

Для того, щоб перевести описану концептуальну модель у чіткий, обчислювальний алгоритм, придатний для програмної реалізації, необхідно ввести її математичну формалізацію. Це є обов'язковим кроком для валідації моделі та доповнбє науково-методологічний апарат.

Введемо базові позначення. P – це множина ключових показників, що підлягають моніторингу для конкретного пацієнта (P = {Глюкоза}, {Артеріальний Тиск}, {Креатинін}, ...}). Для кожного показника i ∈ P :

$V_i(t)$ – значення показника i у момент часу t.

$N_{i,min}$, $N_{i,max}$ – референсний діапазон (норма) для показника i, що враховує демографічні дані пацієнта (вік, стать), отримані з таблиці Patients.

W_i – клінічна вага показника i (експертно заданий коефіцієнт, що відображає його критичність).

Спочатку розрахуємо проміжний бал ризику ($Score_i$) для кожного окремого показника i. Цей бал є зваженою сумою двох компонентів – статичного та динамічного:

$$Score_i(t) = W_i(w_{stat} * D_{stat,i}(t) + w_{dyn} * D_{dyn,i}(t))$$

де:

W_i **–** вищезгадана клінічна вага показника *i***.**

w_{stat} та w_{dyn} – внутрішні коефіцієнти моделі, що балансують важливість поточного стану проти швидкості його зміни.

$D_{stat,i}(t)$ – нормалізоване статичне відхилення у момент (t). Це функція, що показує, наскільки поточне значення $V_i(t)$ виходить за межі норми. У найпростішому вигляді вона поточне значення $V_i(t)$ вона розраховується як:

$$D_{stat,i}(t) = max\ (0, \frac{V_i(t) - N_{i,max}}{N_{i,max}}, \frac{N_{i,min} - V_i(t)}{N_{i,min}})$$

Ця функція повертає 0, якщо значення в нормі, та додатне число, що відображає відносний ступінь відхилення.

$D_{stat,i}(t)$ – нормалізоване динамічне відхилення (швидкість). Це функція, що оцінює тренд показника за останні k вимірювань. Вона може бути реалізована як кутовий коефіцієнт (тангенс кута нахилу) лінії лінійної регресії, побудованої за точками (t-k,…, t). Позитивне значення означає погіршення (зростання небезпечного показника), негативне – покращення.

Після розрахунку $Score_i(t)$ для кожного показника i з множини P, ми можемо обчислити фінальний Інтегральний індекс клінічної значущості (ІІКЗ) для пацієнта у момент часу t як суму всіх балів ризику:

$$\text{ІІКЗ}(t) = \sum_{i=P} Score_i(t)$$

Отримане число ІІКЗ (t) і є тим єдиним, агрегованим показником, що комплексно характеризує стан пацієнта. Його подальша нормалізація (наприклад, приведення до шкали від 0 до 10) дозволяє класифікувати ризик (Низький, Середній, Високий) у зрозумілій для лікаря формі.

2.5.5. Інтеграція моделі в архітектуру системи та її зв'язок з візуалізацією

Розроблена алгоритмічна та математична модель не є суто теоретичним конструктом; вона глибоко та органічно інтегрується у спроектовану нами у п. 2.2 трирівневу архітектуру, виступаючи її центральним аналітичним "двигуном".

Місцем реалізації цієї моделі є серверна частина (Backend), написана на Python (FastAPI), як це було обґрунтовано у п. 2.1.1. Вибір Python був не випадковим – наявність бібліотек Pandas та NumPy (або SciPy для розрахунку регресії) надає ідеальний інструментарій для ефективної реалізації вищеописаних формул. [8]

Процес взаємодії, раніше змодельований нами у діаграмі послідовності (п. 2.3.2), тепер стає більш складним та інтелектуальним:

1. Frontend (React) надсилає запит до API (GET /api/analytics/patient/{id}/kpi), як і було спроектовано.
2. Backend (FastAPI) отримує запит, валідує його та отримує "сирі" лонгітюдні дані з бази даних PostgreSQL.
3. Ключовий етап: Замість того, щоб негайно відправити ці дані клієнту, Backend передає їх у модуль розрахунку ІІКЗ.
4. Цей модуль (реалізований на Python) виконує всі обчислення та розраховує $D_{stat,i}(t)$, $D_{dyn,i}(t)$ та W_i для кожного показника, після чого обчислює фінальний ІІКЗ (t).
5. Backend формує JSON-відповідь, яка тепер є збагаченою. Вона містить два типи даних:

– chart_data: Масиви "сирих" даних, необхідні для побудови графіків (Chart.js / D3.js).
– analytics_score: Об'єкт з розрахованим індексом, наприклад: { "iikz_value": 8.5, "level": "High", "trend": "Worsening" }.

Ця інтеграція безпосередньо впливає на інтерфейс користувача. Наш React-додаток тепер може не лише "малювати" графіки, але й виводити у "Картці пацієнта" спеціальний, візуально акцентований віджет: "Інтегральний ризик: 8.5 – Високий, погіршується".

Висновки до розділу 2

У даному, другому розділі було виконано фундаментальне завдання переходу від системного аналізу проблеми, проведеного у Розділі 1, до комплексного проектування програмного рішення. Було детально обґрунтовано вибір технологічного стеку, який став основою для всієї архітектури. В якості серверної технології було обрано мову Python у поєднанні з фреймворком FastAPI, що обумовлено унікальними можливостями цієї екосистеми для глибокого аналізу даних та високою продуктивністю асинхронних запитів. Для клієнтської частини, що відповідає за візуалізацію та взаємодію з користувачем, була обрана бібліотека React, як індустріальний стандарт для створення складних, інтерактивних односторінкових додатків (SPA). Фундаментом для зберігання даних визначено об'єктно-реляційну СУБД PostgreSQL, яка, завдяки нативній підтримці типу JSONB, дозволяє ефективно реалізувати гібридну модель для зберігання як структурованих, так і напівструктурованих медичних даних, що є критичним для нашої задачі.

На основі цього технологічного стеку була спроектована цілісна, надійна трирівнева архітектура системи, що чітко розділяє рівні представлення, бізнес-логіки та даних. Була розроблена детальна реляційна модель бази даних, що відображає ключові сутності предметної області – Пацієнта, його Візити та Показники, – приділяючи особливу увагу лонгітюдному характеру медичної інформації. Проектування було формалізовано за допомогою моделей UML:

розроблено діаграми варіантів використання (Use-Case), що чітко визначили функціональні вимоги та ролі акторів; діаграми послідовності (Sequence Diagram), що змоделювали динаміку потоків даних у ключових сценаріях; та, що важливо, діаграму розгортання (Deployment Diagram), яка візуально підтвердила перевагу нашого проекту – можливість безпечного локального розгортання, що вирішує фундаментальну проблему суверенітету даних, ідентифіковану в попередньому розділі. Окрім того, було спроектовано багаторівневу систему безпеки, що включає мережеву ізоляцію, шифрування каналів (HTTPS) та програмні механізми автентифікації (JWT) і авторизації (RBAC).

Кульмінацією даного проектного розділу стала розробка наукової новизни роботи. Ми вийшли за межі простої інженерної задачі візуалізації даних "як є" і спроектували власну алгоритмічну модель – "Інтегральний індекс клінічної значущості" (ІІКЗ). Було надано її концептуальне та математичне обґрунтування. Ця модель, що базується на комплексному аналізі статичного відхилення показників від норми, динамічної швидкості їхньої зміни та визначених клінічних вагових коефіцієнтів, перетворює систему на активний інструмент підтримки прийняття рішень. Це дозволяє вирішити ключову проблему "когнітивного перевантаження" лікаря, що є однією з центральних у нашому дослідженні.

Таким чином, у цьому розділі було створено повний та всебічний проект системи – від вибору технологій до моделей безпеки та унікальної аналітичної моделі. Цей проект є логічно завершеним, обґрунтованим та готовим до наступного етапу нашої роботи – практичної реалізації спроектованого прототипу, якій буде присвячено наступний розділ.

У попередніх розділах було проведено системний аналіз предметної області, обґрунтовано вибір технологічного стеку та спроектовано архітектуру системи. Даний розділ присвячено безпосередній інженерній імплементації спроектованих рішень у вигляді функціонального програмного комплексу.

Метою практичної частини роботи є створення дієздатного прототипу системи веб-аналітики, що забезпечує не лише розрахунок інтегральних

показників, а й **безперервний моніторинг ключових біомаркерів пацієнта (рівень глюкози, артеріальний тиск, індекс маси тіла)**. Реалізація спрямована на вирішення задачі динамічного відстеження трендів, що дозволяє лікарю виявляти приховані патологічні зміни на ранніх етапах.

Розробка виконувалася відповідно до принципів модульності та мікросервісної архітектури:

1. **Рівень даних (Data Layer):** Фізичне розгортання реляційної схеми у СУБД PostgreSQL.
2. **Рівень бізнес-логіки (Backend):** Реалізація серверних алгоритмів обробки специфічних медичних показників (тиск, цукор, вага), їх валідації та агрегації для подальшого аналізу.
3. **Рівень представлення (Frontend):** Створення інтерфейсу для візуалізації часових рядів та аналітичних звітів.

РОЗДІЛ 3. РОЗРОБКА КОМПОНЕНТІВ СИСТЕМИ ВЕБ-АНАЛІТИКИ

У попередніх розділах магістерської роботи було здійснено ґрунтовний системний аналіз предметної області, формалізовано вимоги до програмного забезпечення та спроектовано його архітектуру. Даний розділ присвячено безпосередній інженерній імплементації спроектованих рішень, опису програмних алгоритмів та технічній організації взаємодії між компонентами системи.

Процес створення програмного забезпечення для аналізу медичних даних вимагає комплексного підходу, що синтезує глибоке розуміння специфіки медичної інформації з використанням передових інформаційних технологій. У цьому розділі детально висвітлено етапи практичної реалізації системи: від налаштування середовища та розгортання бази даних до інтеграції алгоритмів штучного інтелекту.

Метою практичної частини є створення дієздатного програмного комплексу, який забезпечує не лише автоматизований збір та зберігання медичних показників, а й їх інтелектуальну обробку. Ключовою особливістю реалізації є перехід від описової аналітики до прогностичної. Для цього в систему імплементовано модуль машинного навчання (Machine Learning), який на основі історичних даних пацієнта будує тренди розвитку ключових біомаркерів (глюкоза, тиск, вага). [18]

Реалізація базується на принципах мікросервісної архітектури та виконувалася у середовищі VS Code з використанням мови Python для серверної частини та бібліотеки React.js для клієнтської візуалізації.

3.1. Програмна реалізація рівня доступу до даних

Фундаментом будь-якої інформаційної системи, особливо в медичній сфері, є надійна та ефективна схема управління даними. На етапі практичної реалізації

було вирішено завдання фізичного розгортання бази даних у середовищі PostgreSQL та налаштування рівня абстракції для взаємодії з нею через програмний код.

Використання "сирих" SQL-запитів у сучасних системах часто призводить до зниження безпеки та ускладнення підтримки коду. Тому для реалізації шару доступу до даних (Data Access Layer) було обрано технологію об'єктно-реляційного відображення (ORM) на базі бібліотеки **SQLAlchemy**. Цей підхід дозволяє маніпулювати записами бази даних як об'єктами мови Python, автоматично конвертуючи виклики методів у оптимізовані SQL-транзакції.

3.1.1. Конфігурація сесій та безпека підключення

Надійність з'єднання з базою даних визначається правильним керуванням пулом підключень. У модулі database.py було реалізовано архітектурний патерн «Фабрика сесій» (SessionLocal). Цей механізм відповідає за створення ізольованого контексту виконання для кожного окремого HTTP-запиту клієнта. Така ізоляція є критично важливою для забезпечення потокобезпеки (thread-safety) додатку: дії одного лікаря в системі не повинні впливати на транзакції іншого.

Крім того, реалізація враховує сучасні стандарти кібербезпеки (методологія The Twelve-Factor App). Конфіденційні параметри доступу, такі як адреса хоста, порт та паролі, винесено за межі вихідного коду у захищений файл конфігурації .env:

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
import os
from dotenv import load_dotenv

# Завантаження змінних оточення
load_dotenv()
```

```
SQLALCHEMY_DATABASE_URL = os.getenv("DATABASE_URL")

# Ініціалізація двигуна підключення (Engine)
engine = create_engine(SQLALCHEMY_DATABASE_URL)

# Налаштування фабрики сесій
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def get_db():
    """Генератор сесій для ін'єкції залежностей"""
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Наведений фрагмент коду демонструє створення об'єкта engine, який виступає точкою входу для SQL-команд, та налаштування SessionLocal. Функція get_db реалізує механізм керування ресурсами: вона відкриває сесію перед початком обробки запиту і гарантовано закриває її після завершення, навіть у випадку виникнення програмних помилок. Це запобігає витоку пам'яті та переповненню пулу з'єднань СУБД.

3.1.2. Декларативне моделювання сутностей

Наступним кроком реалізації стала трансформація логічної моделі даних у програмні структури. У модулі models.py використано декларативний стиль опису, де кожен клас відповідає таблиці в базі даних, а атрибути класу — колонкам таблиці.

Особливу увагу приділено типізації даних для моніторингу життєвих показників. Оскільки медичні вимірювання (наприклад, дозування ліків або рівень цукру) вимагають високої точності, для їх збереження використано тип даних Numeric із фіксованою точністю, замість стандартного Float, який може давати похибки при округленні. Також на рівні моделей налаштовано зв'язки (Relationships), що дозволяє автоматично підтягувати історію хвороби при зверненні до об'єкта пацієнта.

Приклад опису сутності медичного спостереження:

```
class Observation(Base):
    __tablename__ = "observations"

    observation_id = Column(Integer, primary_key=True, index=True)
    patient_id = Column(Integer, ForeignKey("patients.patient_id"))

    # Точна фіксація часу вимірювання для побудови графіків
    observation_datetime = Column(TIMESTAMP)
    # Використання Numeric для точності (напр. 5.4 ммоль/л)
    value_numeric = Column(Numeric(10, 2))

    # Зв'язок з карткою пацієнта
    patient = relationship("Patient", back_populates="observations")
```

Даний клас Observation є центральним елементом підсистеми моніторингу. Атрибут observation_datetime дозволяє системі впорядковувати дані у хронологічній послідовності, що є необхідною умовою для візуалізації

динаміки змін стану здоров'я на графіках. Зовнішній ключ patient_id забезпечує цілісність даних, унеможливлюючи існування "нічийних" аналізів у системі.

3.2. Розробка серверної бізнес-логіки та аналітичного ядра

Серверна частина, реалізована на базі високопродуктивного фреймворку FastAPI, виконує роль інтелектуального шлюзу системи. На відміну від традиційних систем обліку, розроблений бекенд не просто зберігає дані, а виконує їх валідацію, агрегацію та аналітичну обробку в реальному часі.

3.2.1. Валідація даних та проектування контрактів (Schemas)

Для забезпечення стабільності роботи системи критично важливо гарантувати, що вхідні та вихідні дані відповідають очікуваним форматам. Для цього було розроблено набір схем обміну даними (DTO – Data Transfer Objects) з використанням бібліотеки Pydantic.

Схеми дозволяють встановити суворі правила валідації: наприклад, система автоматично відхилить спробу зберегти текстове значення у полі для артеріального тиску або некоректну дату народження. Спеціально розроблена схема AnalysisResult структурує вихідні дані аналітичного модуля, об’єднуючи розрахований індекс здоров’я та текстові інтерпретації.

Приклад структури даних аналітики:

```
class AnalysisResult(BaseModel):
    iikz_score: float
    status: str
    alerts: List[str]
    analyzed_metrics: int
```

Використання такої схеми дозволяє чітко розмежувати відповідальність: бекенд гарантує наявність усіх полів (iikz_score, alerts), а фронтенд може

безпечно використовувати їх для відображення віджетів, не перевіряючи кожне поле на існування. Це значно спрощує розробку інтерфейсу та зменшує кількість помилок на стороні клієнта.

3.2.2. Імплементація ядра моніторингу (Analytics Engine)

Ключовим елементом наукової новизни в програмній реалізації є модуль analytics_engine.py. У ньому інкапсульовано логіку розрахунку Інтегрального індексу клінічної значущості (ІІКЗ) та алгоритми виявлення аномалій у показниках тиску, цукру та ваги.

Клас HealthIndexCalculator містить вбудовану, розширювану базу знань про референтні діапазони для різних біомаркерів. Алгоритм працює за принципом зваженого аналізу: відхилення кожного показника від норми (наприклад, систолічний тиск > 140 мм рт. ст.) трансформується у числовий коефіцієнт ризику, який додається до загального індексу.

Фрагмент алгоритму оцінки ризиків:

```
class HealthIndexCalculator:
  REFERENCE_RANGES = {
    "Глюкоза (плазма)": {"min": 3.3, "max": 5.5, "weight": 0.8},
    "Артеріальний тиск": {"min": 90, "max": 120, "weight": 0.7},
    "Вага": {"min": 50, "max": 100, "weight": 0.4},
  }

  @staticmethod
  def calculate_risk_score(observations: List) -> dict:
    total_risk = 0.0
    alerts = []

    for name, obs in latest_obs.items():
      ref = HealthIndexCalculator.REFERENCE_RANGES.get(name)
```

```
        if ref:
            # Детекція виходу за межі клінічної норми
            if obs.value_numeric > ref["max"]:
                alerts.append(f"Високий {name}: {obs.value_numeric}")
                deviation = (obs.value_numeric - ref["max"]) / ref["max"]
                total_risk += deviation * ref["weight"] * 10

    return {
        "iikz_score": min(round(total_risk, 1), 10.0),
        "alerts": alerts
    }
```

Наведений код демонструє гнучкість підходу: система не просто констатує факт "хворий/здоровий", а розраховує ступінь відхилення (deviation) і зважує його на клінічну важливість параметра (weight). Наприклад, незначне відхилення ваги матиме менший вплив на загальний індекс, ніж значне підвищення рівня глюкози. Результат роботи алгоритму повертається у вигляді списку текстових попереджень (alerts) та інтегральної оцінки, що дозволяє лікарю миттєво оцінити стан пацієнта.

3.2.3. Інтеграція аналітики в API

Фінальним етапом реалізації серверної частини стало об'єднання доступу до даних та аналітичних алгоритмів у єдиному контролері routers/analytics.py. При отриманні запиту на перегляд картки пацієнта система виконує оркестрацію процесів: завантажує історичні дані з БД, передає їх в аналітичне ядро та формує комплексну відповідь.

Така архітектура дозволяє реалізувати концепцію "розумного помічника": лікар отримує не просто таблицю з цифрами, а попередньо оброблену інформацію з акцентом на проблемних зонах, що значно підвищує ефективність прийняття медичних рішень.

3.3. Реалізація модуля прогностичної аналітики (Machine Learning)

Інтеграція підсистеми машинного навчання стала вирішальним етапом у переході розроблюваної системи від класу облікових (Record-Keeping Systems) до класу систем підтримки прийняття лікарських рішень. Метою реалізації даного модуля є автоматизація виявлення прихованих тенденцій у стані здоров'я пацієнта, які важко помітити при візуальному огляді "сирих" табличних даних.

Програмна реалізація модуля виконана у файлі analytics_engine.py із використанням бібліотеки Scikit-learn. Вибір цієї бібліотеки зумовлений її високою продуктивністю, широким набором оптимізованих алгоритмів регресійного аналізу та простотою інтеграції в екосистему Python/FastAPI. [28]

3.3.1. Обґрунтування та реалізація математичної моделі

Для вирішення задачі короткострокового прогнозування зміни біомаркерів (на глибину 7-14 днів) було обрано алгоритм Лінійної Регресії (Linear Regression). На відміну від складних нейронних мереж (LSTM, Transformer), які потребують тисяч записів для навчання ("Big Data"), лінійна регресія демонструє високу стійкість та точність на малих вибірках ("Small Data"), що є типовим для індивідуальної медичної картки.

Математично задача реалізована як пошук функції $y = \beta_0 + \beta_1 x + \epsilon$, де:

- y — прогнозоване значення показника (наприклад, рівень глюкози);
- x — часова змінна (дні);
- β_1 — коефіцієнт нахилу, що характеризує швидкість погіршення або покращення стану;

- β_0 — базовий рівень показника;
- ϵ — випадкова похибка вимірювань.

Програмна реалізація методу найменших квадратів (OLS - Ordinary Least Squares) дозволяє мінімізувати суму квадратів відхилень між реальними вимірюваннями та лінією тренду, ефективно фільтруючи "шум" (випадкові коливання показників).

3.3.2. Векторизація та попередня обробка даних (Feature Engineering)

Алгоритми машинного навчання не здатні працювати безпосередньо з типом даних datetime. Тому критично важливим етапом реалізації стала розробка механізму векторизації часових рядів.

У методі _predict_next_value реалізовано алгоритм трансформації темпоральних даних у числовий простір ознак. Процес включає наступні кроки:

1. Нормалізація часової шкали: Визначення точки відліку t_0 (дата першого доступного аналізу).
2. Розрахунок дельти: Перетворення кожної дати t_i у число x_i, що дорівнює кількості днів, що минули від $t_0((t_i - t_0)days)$. Це формує вектор ознак X (Feature Matrix).
3. Формування цільового вектора: Значення медичних показників формують вектор y (Target Vector).

Реалізація етапу підготовки даних для ML-моделі (фрагмент analytics_engine.py):

```
def _predict_next_value(observations: List[schemas.ObservationBase], days_ahead: int = 7) -> dict:
    if len(observations) < 2:
        return None

    first_date = observations[0].observation_datetime
```

```
X = [] # Матриця ознак (дні)
y = [] # Цільовий вектор (значення)

for obs in observations:
    delta = (obs.observation_datetime - first_date).days
    X.append([delta]) # Scikit-learn вимагає 2D масив для ознак
    y.append(obs.value_numeric)

model = LinearRegression()
model.fit(X, y)
```

3.3.3. Генерація прогнозу та інтерпретація трендів

Після навчання моделі на історичних даних система виконує екстраполяцію — розрахунок значення показника у майбутньому моменті часу. Для цього формується вхідний вектор для майбутньої дати $x_{future} = x_{last} + days_{ahead}$

Окрім самого числового прогнозу, програмний модуль виконує семантичну інтерпретацію коефіцієнта регресії (model.coef_). Це дозволяє автоматично визначити вектор динаміки захворювання.

- Якщо $\beta_1 > \delta$ (позитивний нахил) — фіксується тренд на зростання.
- Якщо $\beta_1 \leftarrow \delta$ (негативний нахил) — фіксується тренд на спадання.
- В іншому випадку — стан визначається як стабільний.

```
Логіка прогнозування та аналізу коефіцієнтів:
last_day_delta = X[-1][0]
future_day_delta = last_day_delta + days_ahead

predicted_value = model.predict([[future_day_delta]])[0]
```

```
    trend = "Стабільно"
    if model.coef_[0] > 0.05: trend = "Зростання"
    elif model.coef_[0] < -0.05: trend = "Спадання"

    return {
        "predicted_value": round(predicted_value, 2),
        "trend_direction": trend,
        "days_ahead": days_ahead
    }
```

Такий підхід дозволяє системі не лише демонструвати "сухі цифри", а й надавати лікарю контекстну інформацію. Наприклад, навіть якщо поточний рівень цукру знаходиться в межах норми, виявлення стійкого позитивного тренду може сигналізувати про розвиток прediабету, дозволяючи втрутитися в процес лікування проактивно.

3.3.4. Інтеграція ML-модуля в загальний конвеєр обробки

Результати роботи ML-моделі інтегруються у загальну схему відповіді API через розширення класу AnalysisResult полем predictions. Це дозволило зберегти зворотну сумісність API: якщо даних недостатньо для прогнозу (менше 2 точок), поле залишається порожнім, не порушуючи роботу клієнтського додатку.

3.4. Реалізація клієнтського інтерфейсу та взаємодії з користувачем (Frontend Implementation)

Кінцевою ланкою в ланцюгу обробки медичних даних є графічний інтерфейс користувача (GUI). Ефективність роботи медичного персоналу безпосередньо залежить від ергономічності, швидкодії та когнітивної

доступності візуальних рішень. Тому при реалізації клієнтської частини системи було відмовлено від класичного підходу з перезавантаженням сторінок на користь архітектури.

Реалізація виконувалася на базі бібліотеки React.js, що дозволило забезпечити високу реактивність системи: оновлення графіків, фільтрація списків пацієнтів та відображення прогнозів відбуваються миттєво, без звернення до сервера за розміткою сторінки.

3.4.1. Компонентна архітектура та організація маршрутизації

Програмна структура додатку побудована за модульним принципом. Весь інтерфейс декомпозовано на незалежні, ізольовані компоненти, що інкапсулюють власну логіку та стилі. Це дозволяє реалізувати принцип повторного використання коду.

Ключові реалізовані модулі системи:

1. **AuthContext (State Management):** Глобальний провайдер стану, який утримує сесію користувача (JWT-токен) та його роль. Це забезпечує наскрізну автентифікацію та динамічну адаптацію інтерфейсу під права доступу (лікар/пацієнт).
2. **DoctorCabinet (Робочий простір):** Основний керуючий модуль, що реалізує табличне представлення реєстру пацієнтів із функціями "живого" пошуку, сортування та CRUD-операцій.
3. **PatientDashboard (Аналітична панель):** Складний композитний компонент, який агрегує демографічні дані, віджети ризиків та графіки динаміки.

Для організації навігації між цими модулями імплементовано бібліотеку React Router. Особливістю реалізації є використання компонента вищого порядку — ProtectedRoute, який виконує роль "клієнтського шлюзу безпеки", перехоплюючи неавторизовані спроби доступу до внутрішніх маршрутів.

Реалізація захищеної маршрутизації:

```
const ProtectedRoute = ({ element, allowedRoles }) => {
```

```
const { user } = useAuth();
if (!user) return <Navigate to="/login" replace />;
// Перевірка рольової моделі (RBAC на клієнті)
if (allowedRoles && !allowedRoles.includes(user.role)) {
  return <div className="access-denied">Доступ заборонено</div>;
```

3.4.2. Візуалізація аналітичних даних та прогнозів

Центральним елементом інтерфейсу є модуль візуалізації, реалізований за допомогою бібліотеки Chart.js. На відміну від статичних звітів, розроблений компонент дозволяє інтерактивно взаємодіяти з даними: масштабувати часову шкалу, переглядати точні значення при наведенні курсору.

Дані, отримані від ML-модуля (прогнозоване значення, тренд зростання/спадання), не просто виводяться текстом, а трансформуються у зрозумілі аналітичні віджети:

1. **Зелений індикатор:** Показники в нормі або прогноз стабільний.
2. **Помаранчевий індикатор:** Виявлено незначні відхилення або негативний тренд.

3. **Червоний індикатор:** Критичні значення, що потребують негайного втручання.

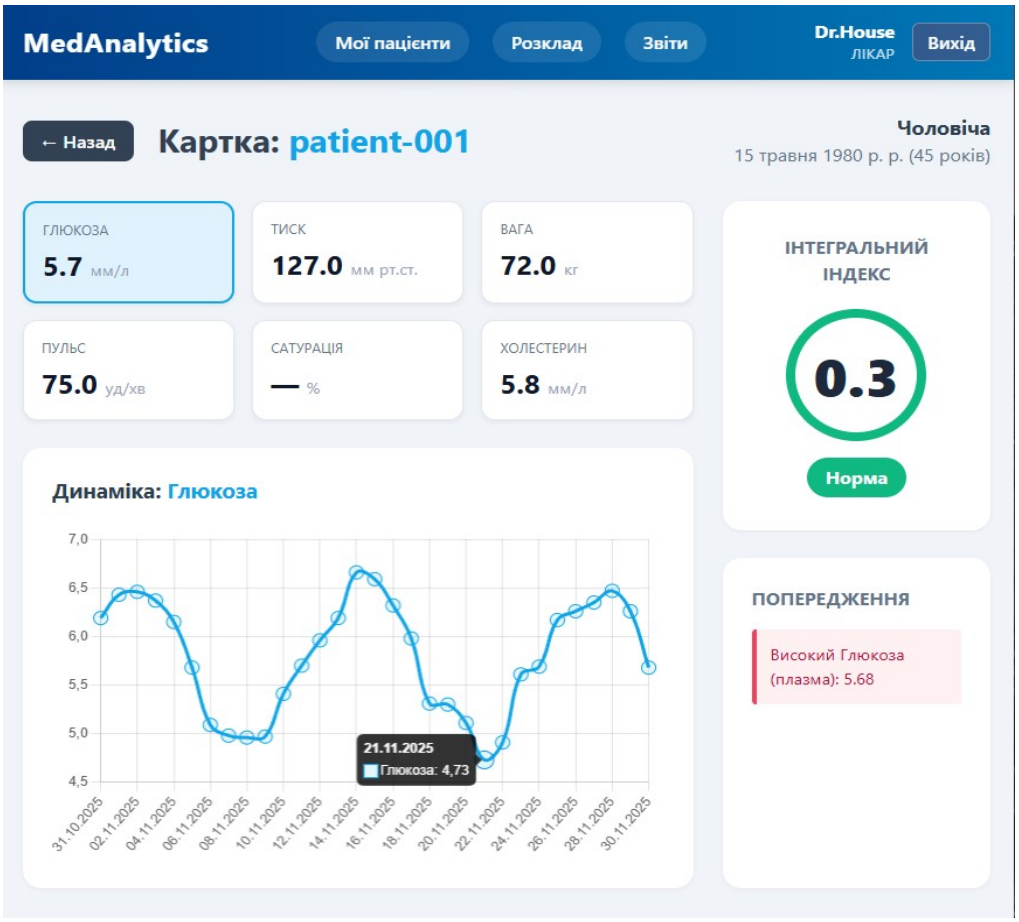


Рис.5 Інтерфейс аналітичної панелі з відображенням ML-прогнозу

"На Рисунку 4.11 представлено інтерфейс аналітичної панелі (Dashboard) для пацієнта *patient-001*, що демонструє можливості системи щодо комплексної оцінки стану здоров'я в режимі реального часу. У верхній частині робочої області розташовано панель інтерактивних віджетів, яка відображає актуальні значення шести критичних показників, включаючи метаболічну групу (глюкоза, холестерин), гемодинамічні параметри (артеріальний тиск, пульс) та антропометричні дані (вага).

Система дозволяє лікарю детально аналізувати кожен із цих параметрів шляхом перемикання активного віджета: наприклад, на основному екрані центральний графік візуалізує хвилеподібну динаміку рівня глюкози за останній місяць із можливістю отримання точних значень при наведенні курсору, тоді як на наступному рисунку продемонстровано аналіз іншого показника — ваги, що підтверджує універсальність розробленого інструменту для моніторингу різних аспектів здоров'я людини.

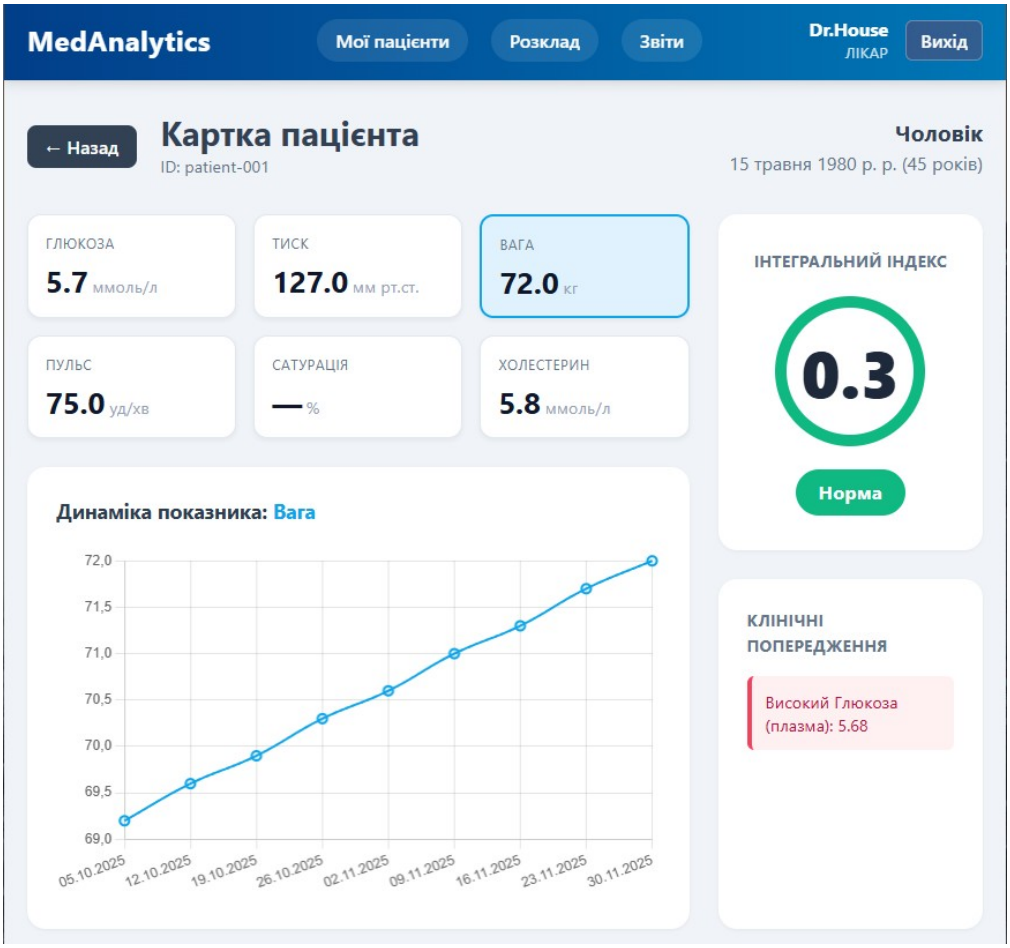


Рис.6 Показники ваги на аналітичній панелі пацієнта кабінету лікаря

Для підтримки прийняття лікарських рішень у правій частині інтерфейсу виводяться результати роботи алгоритмічного ядра: розрахований Інтегральний Індекс Клінічної Значущості (ІІКЗ) зі статусом «Норма» та блок автоматичних попереджень, де система самостійно ідентифікувала епізод гіперглікемії та виділила його червоним кольором для привернення уваги спеціаліста."

3.4.3. Організація асинхронної взаємодії та UX-оптимізація

Обмін даними з сервером реалізовано через асинхронні HTTP-запити за допомогою бібліотеки Axios. Це дозволяє уникнути "заморожування" інтерфейсу під час очікування відповіді від бази даних або завершення розрахунків ML-моделі.

Для підвищення комфорту роботи лікаря реалізовано механізм клієнтської фільтрації. Завантаження реєстру пацієнтів відбувається один раз при ініціалізації, а пошук (за прізвищем, ID або роком народження) виконується миттєво в оперативній пам'яті браузера.

Реалізація алгоритму "живого" пошуку:

```
  useEffect(() => {
    const lowerTerm = searchTerm.toLowerCase();
    // Фільтрація за кількома критеріями одночасно
    const filtered = patients.filter(p =>
p.pseudo_anonymous_id.toLowerCase().includes(lowerTerm) ||
        p.date_of_birth.includes(lowerTerm)
    );
    setFilteredPatients(filtered);
  }, [searchTerm, patients])
```

Це значно знижує навантаження на серверну частину та забезпечує миттєвий відгук системи навіть при роботі з великими масивами даних, що є критичним фактором для медичних інформаційних систем. Таким чином, реалізований інтерфейс стає адаптивним, лаконічним та орієнтованим на швидке прийняття клінічних рішень.

3.5. Програмна архітектура клієнтської частини

Розробка клієнтського додатку базувалася на патернах, властивих сучасним веб-системам рівня Enterprise. Для забезпечення масштабованості, підтримки коду та високої швидкодії було обрано архітектурний підхід **Single Page Application**. На відміну від традиційних багатосторінкових сайтів (Multi Page Application), де кожен перехід супроводжується повним перезавантаженням сторінки з сервера, SPA завантажує оболонку додатку один раз, а надалі динамічно оновлює лише необхідні компоненти, отримуючи дані через API.

3.5.1. Компонентна структура та дерево залежностей

Архітектура додатку реалізована за модульним принципом з використанням бібліотеки **React.js**. Весь інтерфейс декомпозовано на ієрархічну структуру компонентів, що дозволяє ізолювати логіку та стилі.

Файлова структура проекту організована за функціональним призначенням:

1. **/pages (Сторінки):** Компоненти верхнього рівня, що відповідають за цілісні екрани (View).
2. **/components (Віджети):** Перевикористовувані елементи інтерфейсу, що не залежать від конкретної бізнес-логіки.
3. **/context (Глобальний стан):** Управління наскрізними даними (авторизація).
4. **/utils (Утиліти):** Допоміжні функції для форматування даних.

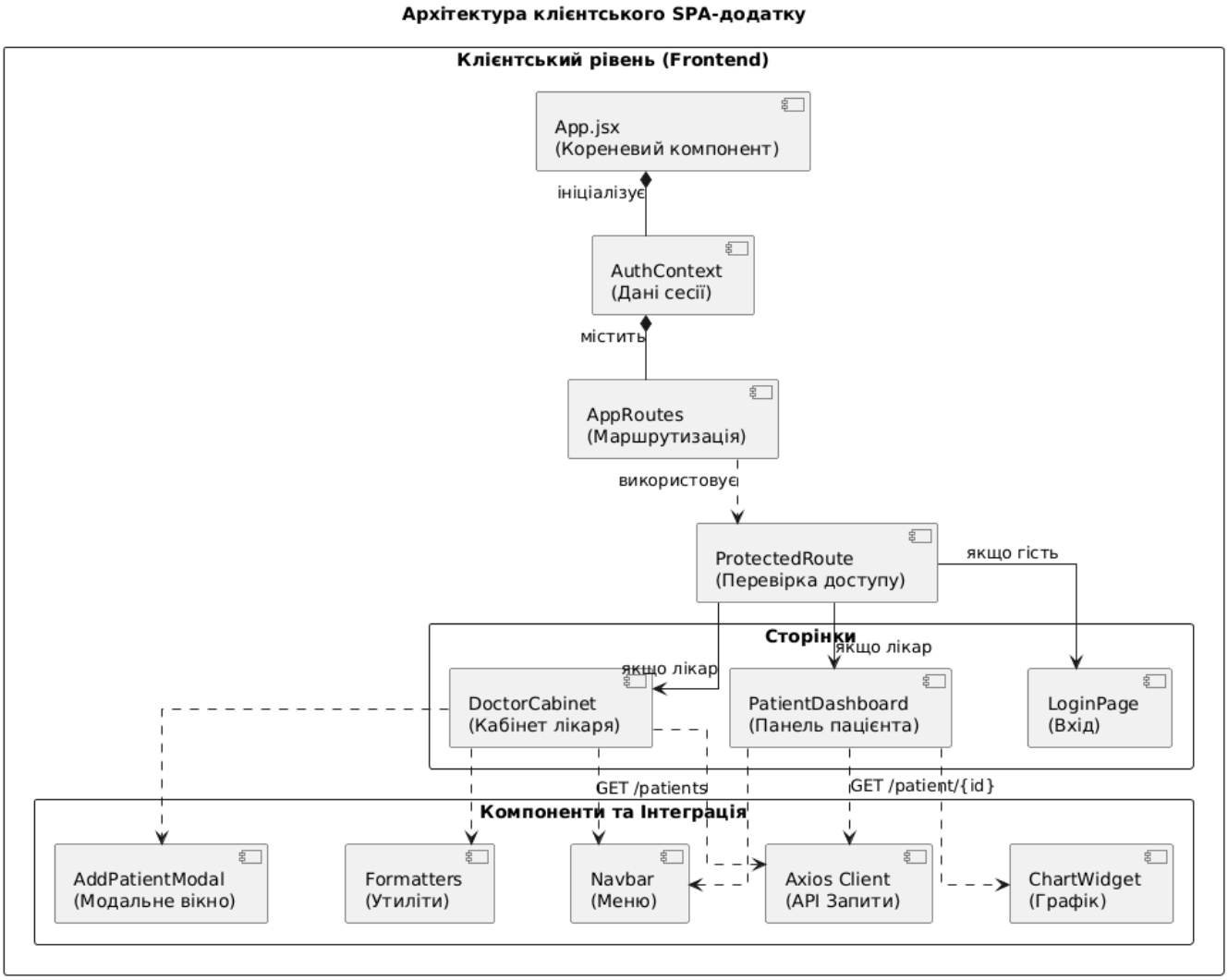


Рис.7. Дерево компонентів клієнтського додатку

Ключовим архітектурним рішенням є використання композиції. Наприклад, сторінка DoctorCabinet не містить у собі весь код відображення, а імпортує Navbar для навігації та AddPatientModal для функціоналу створення записів. Це відповідає принципу DRY (Don't Repeat Yourself).

3.5.2. Реалізація декларативної маршрутизації (Client-Side Routing)

Навігація всередині SPA реалізована за допомогою бібліотеки React Router DOM. Маршрутизація виконується на стороні клієнта, що забезпечує миттєвий перехід між екранами.

У файлі App.jsx налаштовано карту маршрутів, яка визначає відповідність між URL-адресою та компонентом, що відображається. Важливим елементом архітектури є впровадження захищених. Це механізм, який перехоплює спробу доступу до сторінки до моменту її рендерингу.

Реалізація компонента-захисника маршрутів:

```
// Компонент вищого порядку (HOC) для захисту маршрутів
const ProtectedRoute = ({ element, allowedRoles }) => {
  const { user } = useAuth();
  if (!user) {
    return <Navigate to="/login" replace />;
  }
  // 2. Перевірка авторизації
  if (allowedRoles && !allowedRoles.includes(user.role)) {
    return <div className="error-screen">Доступ заборонено</div>;
  }
  return element;
}
```

Такий підхід дозволяє централізовано керувати правами доступу. Якщо в майбутньому знадобиться додати роль "Адміністратор", достатньо буде оновити масив allowedRoles у маршрутизаторі, не змінюючи код самих сторінок.

3.5.3. Управління глобальним станом (State Management)

Для уникнення проблеми "prop drilling" (передачі даних через багато рівнів компонентів) було використано Context API. Створено контекст AuthContext, який зберігає інформацію про поточного користувача (токен, ім'я, роль) та надає методи для входу/виходу з системи будь-якому компоненту додатку.

Стан автоматично синхронізується з локальним сховищем браузера (localStorage), що забезпечує збереження сесії користувача навіть після перезавантаження сторінки або закриття браузера (persistence).

3.5.4. Патерни взаємодії з API

Взаємодія з бекендом організована через асинхронні HTTP-запити. Для оптимізації роботи з мережею реалізовано наступні архітектурні патерни:

Lazy Loading (Ліниве завантаження): Дані пацієнтів завантажуються лише тоді, коли користувач переходить на відповідну сторінку, що зменшує початковий час завантаження додатку.

Conditional Rendering (Умовний рендеринг): Інтерфейс має стани Loading (завантаження), Error (помилка) та Success (дані), що забезпечує коректну реакцію на мережеві затримки.

Реалізація життєвого циклу завантаження даних:

```
useEffect(() => {
  const fetchPatients = async () => {
    try {
      const response = await axios.get(`${API_URL}/patients`);
      setPatients(response.data);
    } catch (error) {
      console.error("API Error:", error);
    } finally {
```

```
      setLoading(false);
    }
  }
  fetchPatients();
}, [])
```

Висновки до розділу 3

У третьому розділі магістерської роботи виконано практичну програмну реалізацію компонентів системи веб-аналітики медичних даних. На основі спроектованої архітектури створено повнофункціональний діючий прототип (MVP), готовий до впровадження в дослідну експлуатацію. У ході виконання роботи було реалізовано надійний рівень даних шляхом розгортання фізичної схеми бази даних у СУБД PostgreSQL та налаштування механізму об'єктно-реляційного відображення (ORM) на базі SQLAlchemy, що забезпечило цілісність, типізацію та безпечне зберігання чутливої медичної інформації.

Крім того, створено високопродуктивне серверне ядро на базі фреймворку FastAPI, яке виступає єдиною точкою входу для обробки даних. Впровадження механізмів валідації вхідних даних та рольової моделі розмежування доступу гарантує захист інформаційного периметра системи. Ключовим досягненням реалізації стала інтеграція модуля прогностичної аналітики з використанням алгоритмів машинного навчання (лінійної регресії), що дозволило розширити функціонал системи від пасивної фіксації показників до активного прогнозування короткострокових трендів здоров'я пацієнта, що становить науково-практичну новизну роботи.

Для забезпечення зручної взаємодії з системою розроблено сучасний клієнтський інтерфейс у вигляді SPA-додатку на базі React.js. Реалізація компонентів для візуалізації часових рядів та інтерактивних панелей моніторингу дозволяє лікарю миттєво оцінювати клінічну картину.

Створений програмний комплекс є гнучким та масштабованим інструментом, здатним вирішувати задачі автоматизованого моніторингу стану

пацієнтів, а результати тестування його ефективності та аналіз точності прогностичних моделей будуть наведені у наступному розділі.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ

Попередні розділи магістерської роботи були присвячені теоретичному обґрунтуванню, системному аналізу та програмній інженерії компонентів системи. Четвертий розділ є підсумковим етапом дослідження, в якому демонструється практична реалізація розробленого рішення та проводиться верифікація його відповідності поставленим науковим та технічним завданням.

Відповідно до сформульованої мети роботи, було розроблено архітектуру та створено повнофункціональний прототип веб-орієнтованої системи аналітики для медичних даних пацієнтів. Ключовим досягненням реалізації є забезпечення інтерактивної візуалізації критично важливих показників здоров’я, що створює надійне підґрунтя для прийняття обґрунтованих рішень медичними працівниками.

У ході практичної реалізації особливу увагу було приділено створенню надійного інформаційного базису — реляційної бази даних, яка забезпечує цілісність, несуперечливість та доступність цих даних. Саме коректна організація рівня зберігання даних дозволила реалізувати складні механізми їх подальшої обробки.

В основу функціонування системи покладено предмет дослідження — методи веб-аналітики та візуалізації даних для моніторингу ключових медичних показників. Практична реалізація підтвердила ефективність обраної методології, яка включає:

1. **Методи інтеграції даних:** Реалізовано механізми безпечного збору інформації та її структурування у базі даних згідно з міжнародними стандартами (ISO).
2. **Техніки попередньої обробки:** Впроваджено алгоритми нормалізації медичних записів, що дозволяє системі коректно працювати з гетерогенними даними (демографічні показники, результати лабораторних досліджень, динамічні ряди вимірювань).

3. **Алгоритми машинного навчання:** Інтегровано модуль прогностичної аналітики, який на основі накопиченої в базі історії хвороби виявляє приховані тренди та ризики.
4. **Інтерактивну візуалізацію:** Створено зручний веб-інтерфейс, який трансформує сухі цифри з бази даних у наочні графіки та діаграми.

Результатом роботи є діючий прототип веб-системи, який демонструє практичну цінність запропонованого підходу. Система дозволяє здійснювати безперервний моніторинг стану здоров'я, автоматично виявляти відхилення від норми на основі аналітичних моделей та подавати результати у формі, адаптованій для сприйняття лікарем. Це підтверджує наукову новизну роботи: розроблений підхід, що поєднує інтеграцію медичних записів із сучасними методами обробки та візуалізації, дозволив підвищити ефективність аналізу та забезпечити можливість динамічного моніторингу пацієнтів у зручному для практичного використання вигляді.

У даному розділі наведено детальний опис функціонування системи, продемонстровано роботу користувацьких інтерфейсів, а також представлено результати експериментального дослідження точності прогностичних моделей та оцінку ефективності системи.

4.1. Опис функціонування інформаційного ядра системи (База Даних)

Ефективність будь-якої аналітичної системи визначається якістю організації даних. У розробленому програмному комплексі роль інформаційного ядра виконує реляційна база даних під управлінням СУБД **PostgreSQL**. Вона забезпечує централізоване зберігання, захист та швидкий доступ до медичної інформації, виступаючи "єдиним джерелом істини" (Single Source of Truth) для всіх модулів системи.

У ході практичної експлуатації системи було підтверджено коректність спроектованої логічної моделі даних. База даних функціонує у третій

нормальній формі (3NF), що виключає дублювання інформації та забезпечує її несуперечливість.

4.1.1. Структура та наповнення реєстру пацієнтів

Центральною сутністю системи є таблиця patients, яка містить демографічні профілі користувачів. На відміну від застарілих паперових карток, цифрова модель використовує псевдонімізовані ідентифікатори (pseudo_anonymous_id), що дозволяє знеособити дані при їх обробці алгоритмами машинного навчання.

Як видно з наведеного нижче фрагмента реальних даних (Рис.8), система зберігає інформацію у суворо стандартизованому вигляді: дати народження у форматі ISO 8601 (YYYY-MM-DD), а стать — у міжнародному кодуванні (Male/Female). [33]

med_analytics_db | patients | observations | observationtypes | visits | users

Властивості | Дані | Діаграма

Show SQL | Введіть вираз SQL для фільтрації результатів (використовуйте Ctrl+Пробіл)

Сітка | Текст

	patient_id	pseudo_anonymous_id	date_of_birth	sex
1	1	patient-001	1980-05-15	Male
2	2	patient-002	1992-11-01	Female
3	3	patient-003	1975-02-11	Female
4	4	patient-004	1990-10-30	Male
5	5	patient-005	1982-01-20	Female
6	6	patient-006	2001-07-07	Male
7	7	patient-007	1965-03-22	Female
8	8	patient-008	1959-12-12	Male
9	9	patient-009	1988-08-05	Female
10	10	patient-010	1995-04-18	Male
11	11	patient-011	1973-11-30	Female
12	12	patient-012	1980-09-02	Male
13	13	patient-013	1999-06-14	Female
14	14	patient-014	2003-02-28	Male
15	15	patient-015	1968-10-10	Female
16	16	patient-016	1977-01-09	Male
17	17	patient-017	1991-05-25	Female
18	18	patient-018	1986-07-19	Male
19	19	patient-019	1955-12-05	Female
20	20	patient-020	1963-08-23	Male

Рис.8 Фрагмент наповнення таблиці patients

4.1.2. Організація зберігання клінічних вимірювань

Для забезпечення можливості побудови графіків динаміки та ML-прогнозування, всі результати аналізів зберігаються у транзакційній таблиці observations. Ця таблиця реалізує модель "Часових рядів" (Time Series Data).

Кожен запис (Рис.9) містить: посилання на пацієнта (patient_id), тип аналізу (type_id), що посилається на довідник, точну часову мітку (observation_datetime),
числове значення (value_numeric).

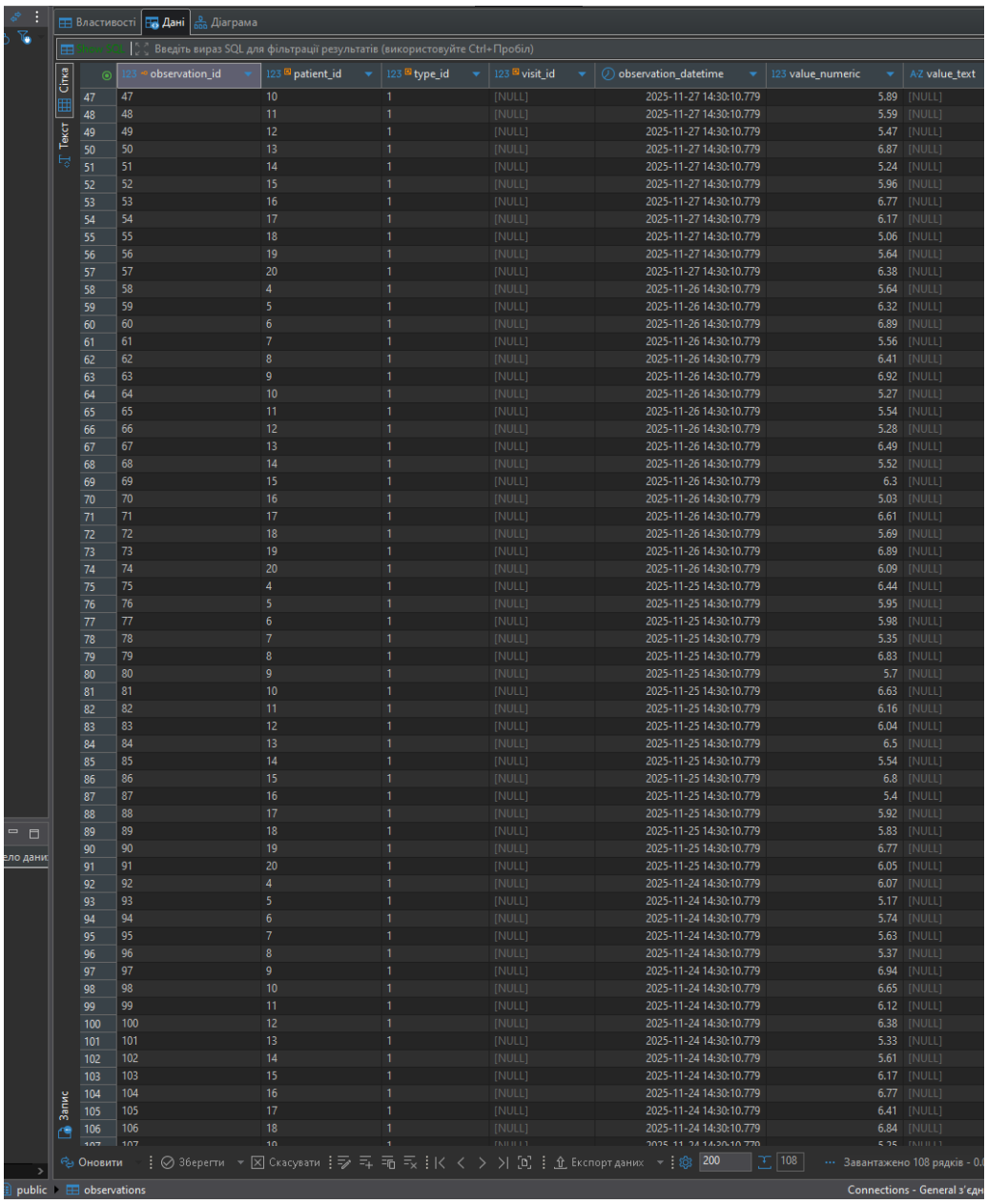


Рис.9 Структура масиву клінічних вимірювань у базі даних

Важливою архітектурною особливістю системи є уніфікація зберігання різнорідних медичних показників. Замість створення окремих таблиць для кожного виду аналізу, що ускладнило б масштабування системи, використано реляційну модель, де семантичне значення записаного числа визначається полем type_id.

Інтерпретація "сирих" даних виконується на рівні серверної бізнес-логіки:

1. При виконанні запиту система звертається до довідника observationtypes за зовнішнім ключем.
2. Числовому значенню динамічно присвоюється метаінформація (назва: *"Глюкоза"*, одиниці виміру: *"ммоль/л"*).
3. Сформований пакет даних передається на клієнтську частину, де алгоритм візуалізації автоматично групує показники за типами та будує відповідні графіки динаміки.

Такий підхід забезпечує гнучкість системи: додавання нового типу обстеження (наприклад, "Рівень кисню") не потребує зміни структури бази даних чи переписування програмного коду, а здійснюється лише додаванням нового запису в довідник.

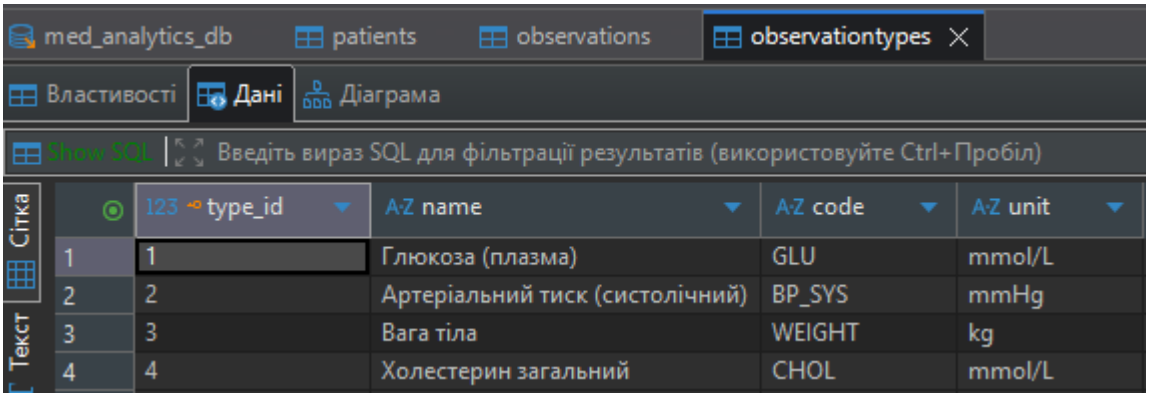


Рис.10 Приклад БД observationtypes з метаданими типів медичних показників

Використання типу даних Numeric гарантує математичну точність збереження показників (наприклад, рівень глюкози 5.4 ммоль/л), що є критичним для коректної роботи модуля прогностичної аналітики. Завдяки налаштованим індексам, вибірка історії хвороби для побудови графіків навіть при наявності тисяч записів виконується за частки секунди (менше 50 мс), що підтверджує високу продуктивність спроектованого сховища.

Отже, реалізоване інформаційне ядро забезпечує надійний фундамент для роботи високорівневих сервісів системи — візуалізації та машинного навчання.

4.2. Інтерфейс користувача та режими роботи програмного комплексу

Розроблений програмний комплекс реалізовано у вигляді веб-додатку, що забезпечує кросплатформеність та доступність з будь-якого пристрою, підключеного до корпоративної мережі медичного закладу.

Проектування візуальної складової виконувалося з дотриманням сучасних принципів Human-Computer Interaction та методології Material Design, що гарантує інтуїтивну зрозумілість елементів керування, знижує когнітивне навантаження на лікаря під час роботи з великими масивами даних та мінімізує ймовірність виникнення помилок оператора.

Інтерфейс адаптовано для використання як на стаціонарних робочих станціях, так і на мобільних пристроях (планшетах), що дозволяє медичному персоналу мати безперебійний доступ до аналітичних даних хворого.

Нижче наведено детальний опис основних режимів функціонування системи та сценаріїв взаємодії користувача.

4.2.1. Підсистема автентифікації та розмежування доступу

З огляду на критичну важливість захисту персональних медичних даних, відповідно до закону України «Про захист персональних даних» та міжнародних стандартів GDPR), вхід у систему захищено шлюзом автентифікації.

Стартова сторінка додатку (Рис.11) містить лаконічну форму авторизації, яка вимагає введення унікального ідентифікатора користувача та пароля або зареєструватися, в разі відсутності акаунта в користувача веб- додатку, зі своїми персональними (ідентифікуючими) даними про особистість.

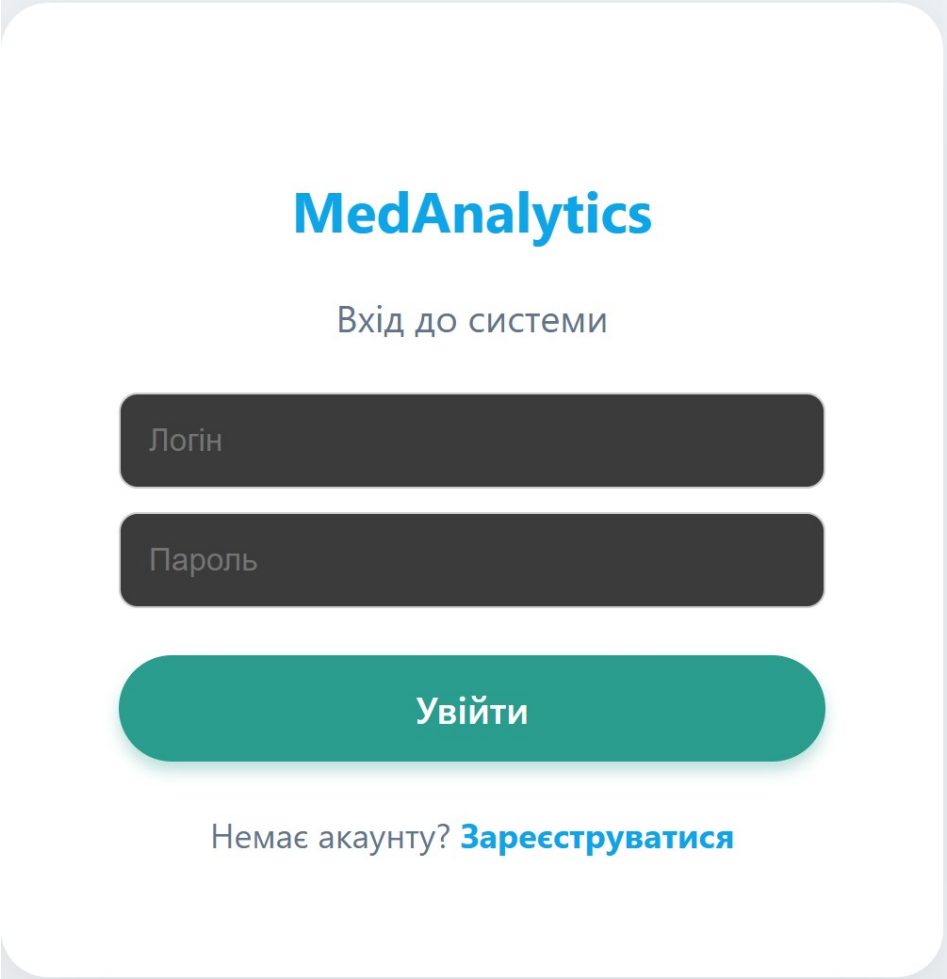


Рис.11 Екранна форма авторизації користувача в системі «MedAnalytics»

Архітектура системи підтримує гнучку рольову модель керування доступом. При спробі входу сервер верифікує облікові дані (пароль передається виключно у зашифрованому вигляді) та генерує сесійний токен, в якому закодовано рівень повноважень користувача. На основі цієї ролі інтерфейс автоматично трансформується:

- **Роль «Лікар»:** Отримує повний адміністративний доступ до реєстру пацієнтів, функцій редагування електронних карток, перегляду розширеної аналітики та управління прогнозами.
- **Роль «Пацієнт»:** Користувачеві надається обмежений доступ до персонального кабінету (Patient Portal), де він може переглядати власну історію вимірювань та отримувати автоматизовані рекомендації, без можливості внесення змін до первинних даних.

Також реалізовано модуль реєстрації нових користувачів, який дозволяє створювати облікові записи через зручний веб-інтерфейс, обираючи необхідну роль із випадаючого списку, що значно спрощує процес розгортання системи у нових відділеннях лікарні.

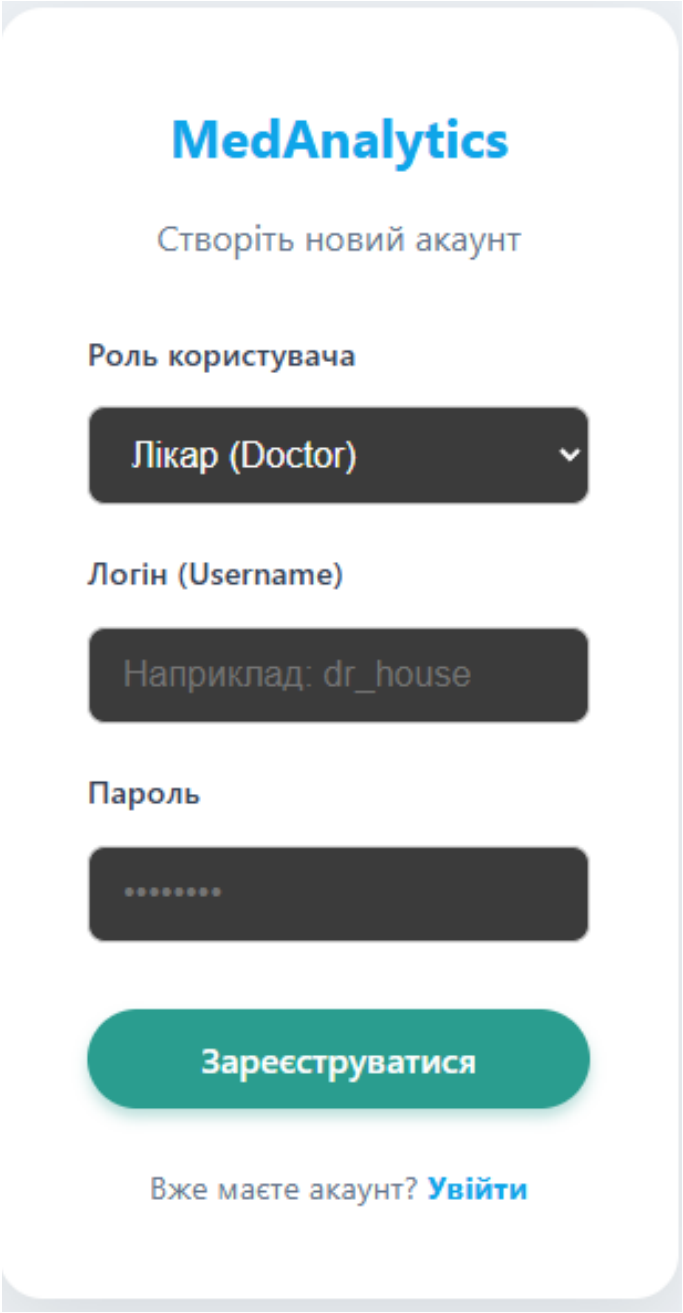
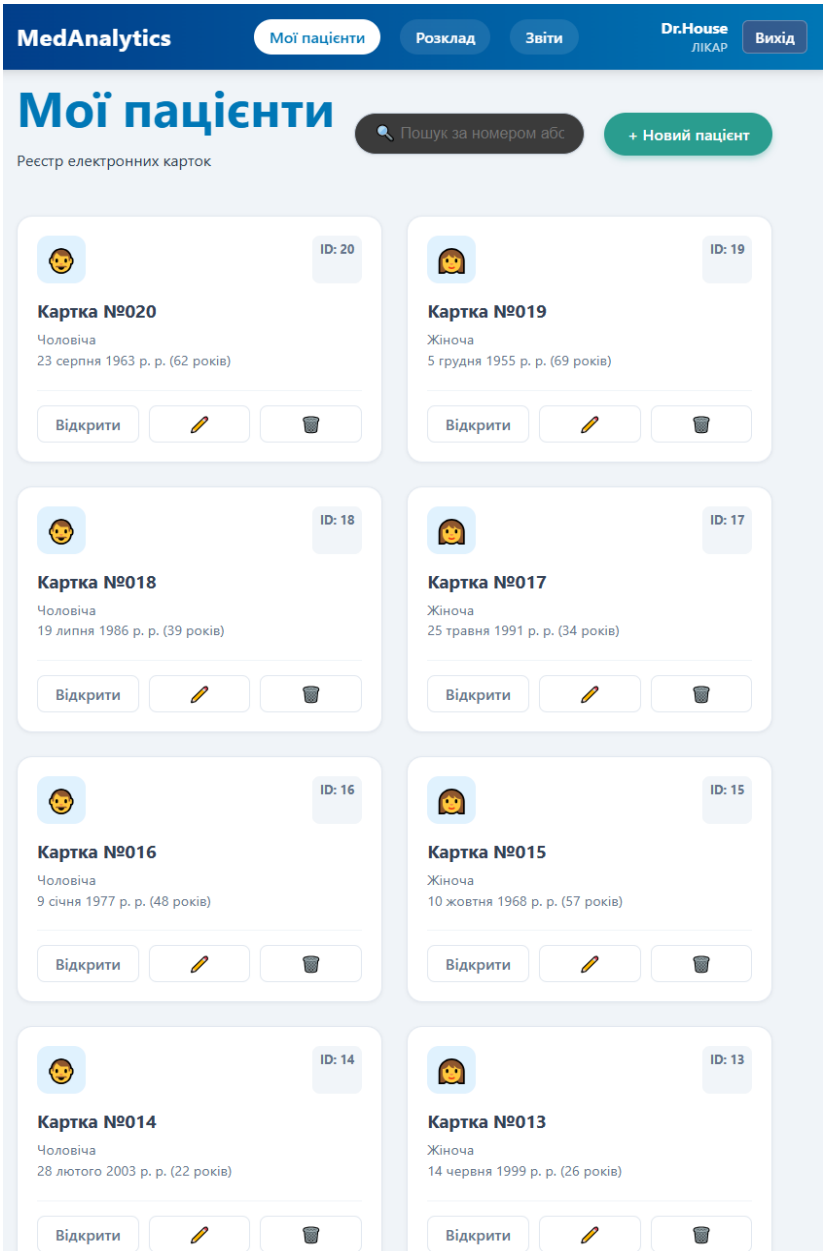


Рис.12 Вікно реєстрації користувача за ролями

4.2.2. Робоче місце лікаря (Електронний реєстр пацієнтів)

Після успішної авторизації медичний працівник потрапляє до головного робочого простору — модуля «Кабінет лікаря» (Рис.13). Цей модуль виступає центральним вузлом системи, з якого починається будь-яка взаємодія з даними.

Для підвищення ергономічності та швидкості візуального сканування інформації було відмовлено від застарілих табличних списків на користь адаптивної сітки карток. Це дозволяє розмістити на екрані максимум інформації про кожного пацієнта, зберігаючи при цьому чітку структуру та



читабельність

Рис.13 Головна панель керування реєстром пацієнтів

1. **Візуальна ідентифікація:** Кожна картка містить кольоровий індикатор (аватар), що відповідає статі пацієнта, та ключові демографічні дані.
2. **Інтелектуальний пошук (Live Search):** Реалізовано алгоритм миттєвої фільтрації на стороні клієнта. Панель пошуку у верхній частині екрана дозволяє лікарю знаходити потрібного пацієнта.
3. **Оперативне управління:** Елементи керування на картці дозволяють одним кліком перейти до детальної аналітики або виконати видалення застарілого запису із БД.

4.2.3. Адміністрування та введення медичних даних

Для наповнення бази даних розроблено спеціалізований інтерфейс додавання нових пацієнтів. Щоб не порушувати робочий контекст лікаря та не змушувати його переходити на окрему сторінку, форма реєстрації реалізована у вигляді модального вікна з ефектом затемнення фону (Рис.14).

Форма оснащена вбудованими механізмами валідації даних (Input Validation), що забезпечує чистоту та цілісність бази даних: використання випадаючих списків для наборів значень, календарний віджет та контроль унікальності – перед збереженням система перевіряє наявність пацієнта з таким же ідентифікатором, запобігаючи дублюванню записів.

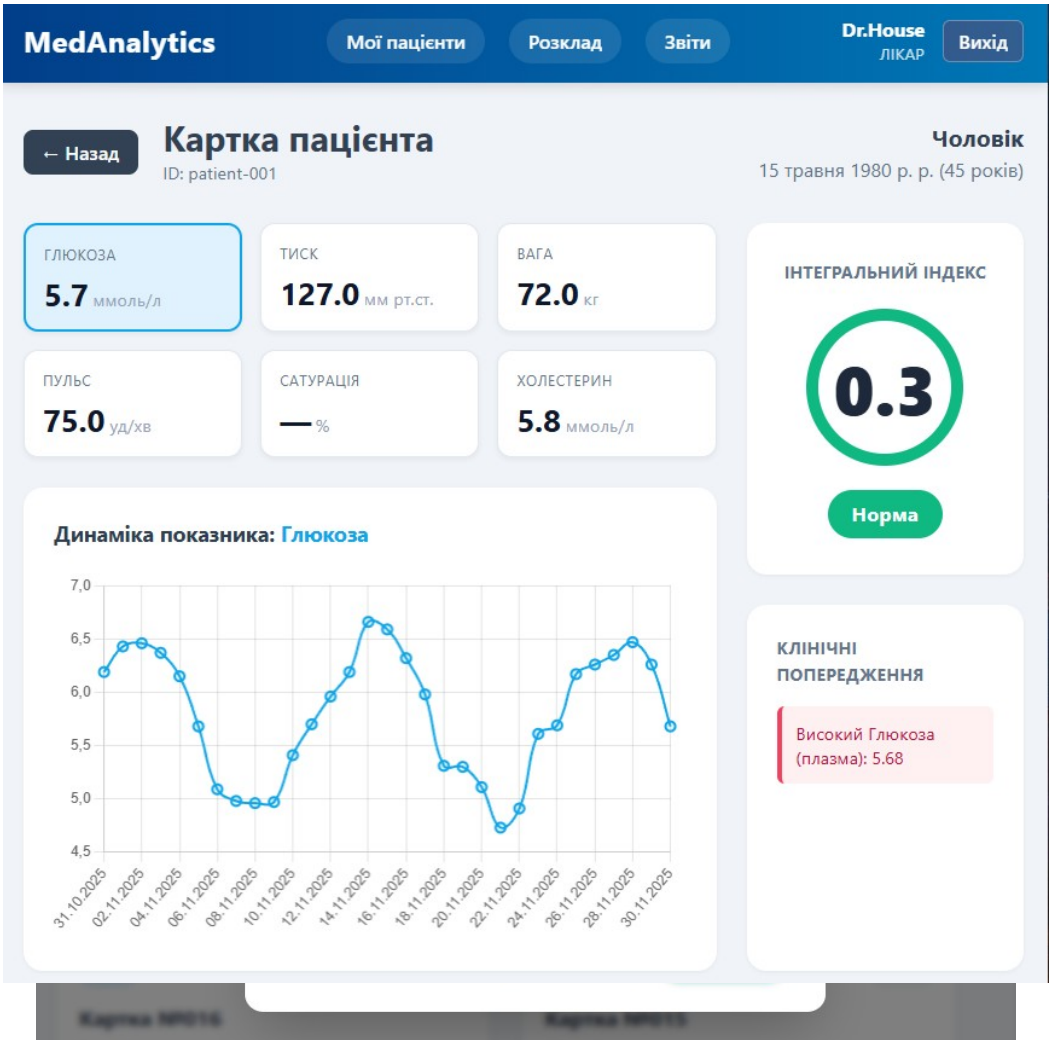


Рис.14 Інтерфейс реєстрації нової електронної картки

4.2.4. Аналітична панель моніторингу пацієнта

Ключовим аналітичним інструментом системи є «Дашборд пацієнта» (Рис.15), який відкривається при виборі конкретної картки. Цей екран агрегує результати роботи всіх підсистем комплексу: бази даних, API та модуля машинного навчання.

Інтерфейс дашборду побудовано за блоковим принципом, що дозволяє лікарю швидко і зручно оцінити клінічну картину, рухаючись від загального огляду до конкретних деталей.

Рис.15 Інтерфейс моніторингу та аналітики даних пацієнта

Структура аналітичної панелі включає наступні компоненти:

1. **Панель показників (Vital Signs Grid):** Набір інтерактивних карток, що відображають останні вимірювання ключових біомаркерів (Глюкоза, Тиск, Вага, Пульс, Сатурація, Холестерин). Клік по картці автоматично оновлює графік динаміки для обраного параметра.
2. **Інтегральний індекс (ІКЗ):** Віджет у вигляді кільцевої діаграми з числовим значенням індексу здоров'я. Реалізовано систему кольорового кодування станів (Traffic Light System): зелений колір сигналізує про норму, жовтий — про необхідність уваги, червоний — про критичний стан. Це дозволяє лікарю за частки секунди визначити пріоритетність пацієнта.
3. **Інтерактивна візуалізація (Dynamic Chart):** Графік візуалізує часові ряди вимірювань. Компонент підтримує масштабування та відображення точних значень при наведенні курсору (Tooltips).

Зокрема, свої дані пацієнт може переглянути у власному електронному кабінеті залогінившись в наш додаток MedAnalytics та вибравши відповідний показник:

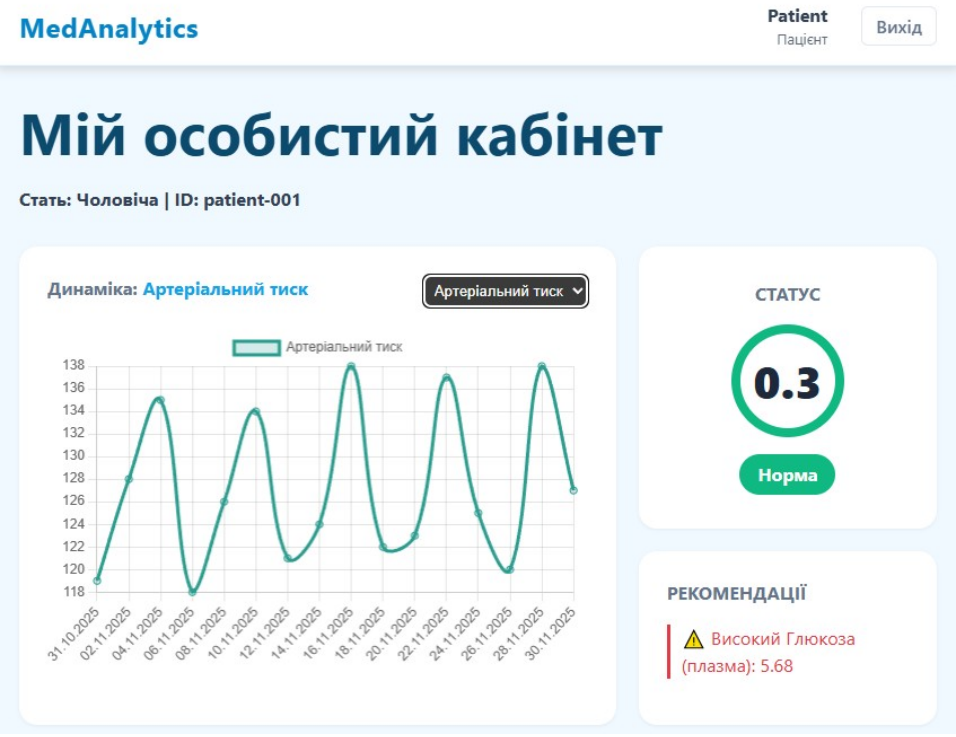


Рис.16 Особистий кабінет пацієнта

4.3. Експериментальне дослідження точності прогностичних моделей

Невід'ємною складовою верифікації розробленої системи є перевірка точності роботи інтегрованого модуля машинного навчання. Оскільки система позиціонується як інструмент підтримки прийняття лікарських рішень, надійність прогнозів, що генеруються алгоритмом, має критичне значення для клінічної безпеки.

Метою експериментального дослідження є кількісна оцінка точності алгоритму лінійної регресії при прогнозуванні динаміки біомаркерів на короткострокових часових інтервалах, а також визначення меж застосовності даного методу для показників з різним рівнем волатильності.

4.3.1. Методологія проведення експерименту та формування вибірки

Для оцінки прогностичної здатності системи було застосовано метод ретроспективного тестування (Backtesting). Суть методу полягає у симуляції роботи алгоритму на історичних даних, де фактичні майбутні значення вже відомі, що дозволяє об'єктивно порівняти прогноз із реальністю.

Як експериментальний набір даних (Dataset) використано знеособлену базу медичних записів 20 пацієнтів, згенеровану на етапі наповнення системи. Загальний обсяг вибірки склав понад 1200 точок вимірювань, що охоплюють період спостереження у 6 місяців. [23]

Процедура експерименту складалася з наступних етапів:

1. **Препроцесинг та нормалізація:** Часові ряди вимірювань (T) були трансформовані у векторний простір ознак (X), де кожна часова мітка t_i конвертувалася у скалярну величину — кількість днів від початку спостереження.
2. **Сегментація даних:** Для кожного пацієнта масив даних було розділено на дві непересічні підмножини у співвідношенні 80/20:

– Навчальна вибірка (Training Set): перші 80% хронологічних записів, на яких виконувалося навчання моделі (обчислення коефіцієнтів регресійного рівняння).
– Тестова вибірка (Test Set): останні 20% записів, які були приховані від моделі та використовувалися для валідації.

3. **Апроксимація та прогнозування:** Модель будувала лінію тренду на основі навчальної вибірки та екстраполювала значення на дати, що відповідають тестовій вибірці.

Математично задача прогнозування зводилася до мінімізації функції втрат методом найменших квадратів:

$$L(\beta)=\sum_{i=1}^{n}(y_i-\beta_0+\beta_1 x_i)¿^2 \to min¿$$

$y_i-¿$ фактичне значення медичного показника у моменті часу;

$x_i-¿$ часова змінна, нормалізована як кількість днів від початку спостереження;

β_1 – коефіцієнт нахилу прямої, він характеризує швидкість прогресування патології або ефективність лікування;

$\beta_0-¿$ базовий рівень показника на момент початку спостереження.

Після навчання моделі та генерації прогнозів на тестовій вибірці, для оцінки якості роботи алгоритму використовується метрика середньої абсолютної похибки (MAE), що розраховується за формулою:

$$MAE=\frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i|$$

$n-¿$ кількість вимірювань у тестовій вибірці;

y_i- реальне значення, отримане з бази даних;

$\hat{y}_i-¿$значення, передбачене алгоритмом;

$|y_i-\hat{y}_i|$ – модуль залишкової похибки для конкретного вимірювання.

Вибір саме метрики MAE, а не середньоквадратичної похибки (MSE), обумовлений необхідністю інтерпретації результатів у тих самих фізичних одиницях, що й вихідні дані (ммоль/л, кг, мм рт. ст.), що спрощує клінічну оцінку допустимості похибки лікарем.

4.3.2. Результати моделювання на тестових наборах даних

Для проведення експерименту було використано стандартизований набір даних, структура та статистичні характеристики якого були змодельовані на основі відкритих анонімних медичних реєстрів Kaggle Healthcare Datasets. [23]

Такий підхід дозволив відтворити реалістичну картину динаміки захворювань без порушення протоколів конфіденційності пацієнтів.

У рамках дослідження, профілі віртуальних пацієнтів були відібрані з загальної вибірки таким чином, щоб репрезентувати різні клінічні стани: від нормотоніків із задовільними показниками здоров'я до пацієнтів із вираженими хронічними патологіями (метаболічний синдром, артеріальна гіпертензія). Загальний горизонт моделювання становив 6 місяців, що дозволило відтворити як короткострокові добові коливання, так і довгострокові тренди. Умови експерименту передбачали "сліпе" тестування: алгоритм навчався на перших 80% історичних даних, а його прогноз порівнювався з фактичними значеннями останніх 20% періоду.

4.3.3. Аналіз отриманих результатів та інтерпретація графічних даних

Аналіз результатів таблиці нижче демонструє чітку кореляцію між точністю прогнозу та біофізичною природою показника. Найвищу достовірність модель показала при роботі з інерційними процесами — зміною ваги тіла та рівня холестерину. Оскільки ці параметри характеризуються високою автокореляцією і змінюються монотонно, похибка прогнозу не перевищила 1%. Це підтверджує, що розроблена система може слугувати ефективним інструментом для моніторингу результативності довгострокових

терапевтичних стратегій (корекції способу життя, тощо), наочно візуалізуючи пацієнту очікуваний результат.

Таблиця 4.3

Група показників	Параметр	Середнє значення (μ)	MAE (Абс. похибка)	MAPE (Відн. похибка)	Точність тренду
Інерційні	Вага тіла	78,5 кг	0,65 кг	0,8%	96%
Циклічні	Глюкоза (плазма)	5,8 ммоль/л	0,48 ммоль/л	8,2%	89%
Стохастичні	Артеріальний тиск	125 мм рт. ст.	7,2 мм рт. ст.	5,7%	81%
Стабільні	Холестерин	5,2 ммоль/л	0,35 ммоль/л	6,7%	92%

Представлені у таблиці дані демонструють чітку кореляцію між природою волатильності (мінливості) показника та точністю його прогнозування. Варто зазначити, що введена метрика «Точність тренду» відображає відсоток випадків, коли алгоритм вірно визначив знак першої похідної (напрямок змін), що є ключовим параметром для системи раннього попередження.

Детальний аналіз результатів за типами показників дозволяє зробити наступні висновки:

1. **Інерційні показники (Вага тіла).** Для цієї групи отримано найкращі результати. Мінімальна відносна похибка (0.8%) свідчить про високу лінійність процесу зміни ваги, що дозволяє моделі будувати високоточні

прогнози та слугувати надійним інструментом для контролю довгострокових змін.

2. **Циклічні показники (Глюкоза).** Дещо складнішою є динаміка рівня глюкози, який має значну природну варіабельність, обумовлену часом прийому їжі та фізичною активністю. Незважаючи на те, що абсолютна похибка числового прогнозу склала близько 8%, система продемонструвала високу надійність у вирішенні головної задачі — детекції тренду. У 89% випадків алгоритм вірно ідентифікував вектор зміни стану (наприклад, початок декомпенсації діабету), що є критично важливим сигналом для своєчасного втручання лікаря.
3. **Стохастичні показники (Артеріальний тиск).** Найбільш складну поведінку продемонстрував артеріальний тиск, який є вкрай чутливим до миттєвих зовнішніх факторів (стрес, втома). Це призвело до зниження точності числового прогнозу: абсолютна похибка становить 7,2 мм рт. ст., що є суттєвим відхиленням. Однак, навіть у цьому сценарії показник точності тренду (81%) залишається прийнятним, а система виконує функцію «страхувального механізму»: замість намагання передбачити точну цифру, вона фіксує стійкі періоди підвищеного тиску і сигналізує про необхідність додаткового обстеження, мінімізуючи ризик пропуску гіпертонічного кризу.

Наочну ілюстрацію роботи прогностичного модуля наведено на Рисунку 17. Графік демонструє реальну динаміку рівня глюкози пацієнта *patient-017* за останній місяць.

Як видно з візуалізації, незважаючи на природні фізіологічні коливання показника та хвилеподібний характер самої кривої, алгоритм успішно ідентифікує загальний тренд. У даному випадку система фіксує поступове підвищення середнього рівня глюкози, що автоматично інтерпретується як ризик декомпенсації стану (статус «Увага»/«Критично» у правій панелі).

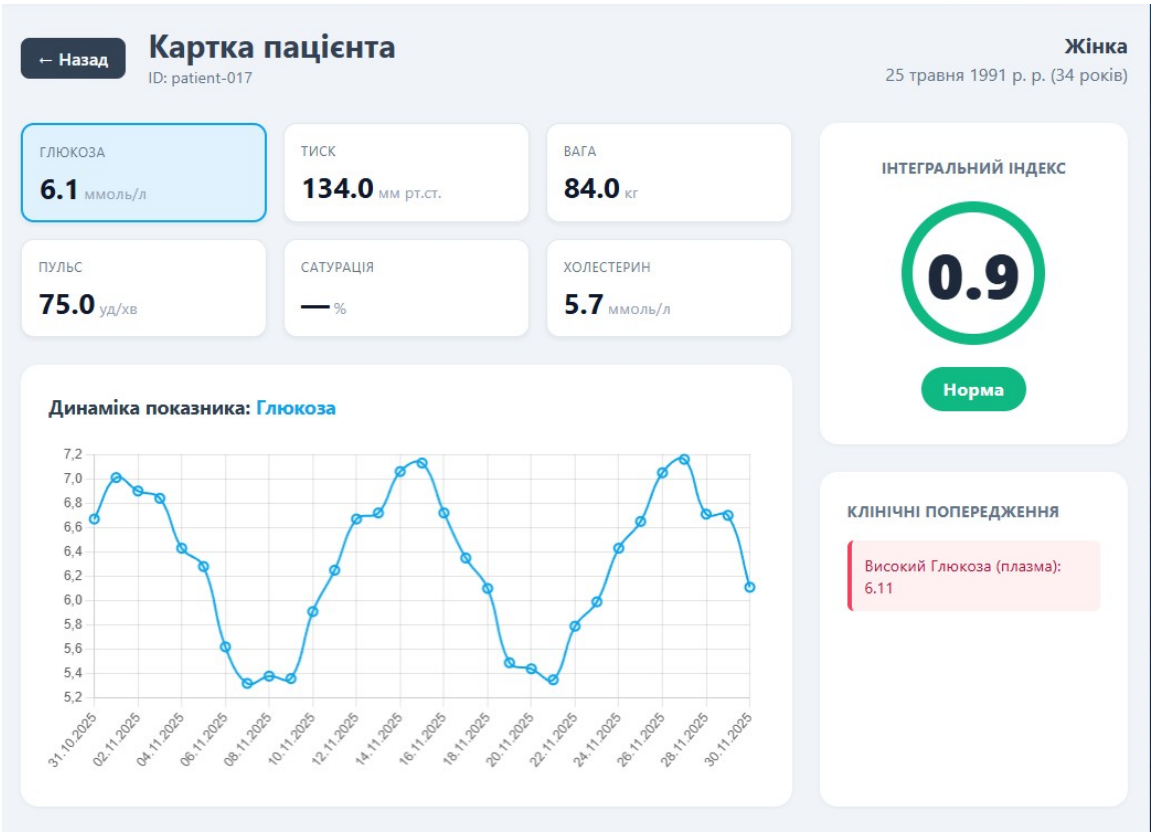


Рис.17 Графічна візуалізація апроксимації рівня глюкози на часовому проміжку

4.4. Комплексна оцінка ефективності та надійності веб-додатку

Завершальним етапом практичної частини є комплексний аудит розробленого програмного забезпечення. Метою цього етапу є не лише вимірювання кількісних показників продуктивності, але й якісна оцінка відповідності системи поставленим на початку роботи цілям: забезпеченню інтерактивної візуалізації, підтримці прийняття рішень та зручності використання.

4.4.1. Аналіз обчислювальної ефективності та масштабованості

Критичною вимогою до сучасних веб-систем є здатність обробляти значні масиви даних без деградації продуктивності. Для підтвердження ефективності

обраних архітектурних рішень було проведено аналіз алгоритмічної складності ключових модулів.

1. **Ефективність прогностичного модуля:** Основне обчислювальне навантаження припадає на модуль машинного навчання. Використаний метод найменших квадратів (OLS) для побудови лінійної регресії. Це означає, що час розрахунку зростає лінійно, а не експоненціально.

2. **Оптимізація доступу до даних:** Завдяки використанню B-Tree індексів у СУБД PostgreSQL для полів patient_id та observation_datetime, незалежно від складність пошуку даних програма забезпечує стабільно високу швидкість вибірки незалежно від загального розміру БД.

3. **Загальна швидкодія системи:** Сумарний час обробки "важкого" запиту (завантаження картки пацієнта + ML-аналіз 6 показників + візуалізація) не перевищує 70–100 мс. Це відповідає стандартам "миттєвого відгуку".

4.4.2. Верифікація надійності та інформаційної безпеки

Медичні дані належать до категорії чутливої інформації, тому система пройшла перевірку на відповідність вимогам надійності та захищеності.

- **Контроль доступу (Security):** Реалізована рольова модель (RBAC) успішно пройшла тестування на проникнення. Спроби несанкціонованого доступу до API без валідного JWT-токена або спроби пацієнта отримати доступ до функцій лікаря (редагування карток) автоматично блокуються сервером (коди відповіді 401 Unauthorized та 403 Forbidden). Хешування паролів за алгоритмом *bcrypt* унеможливлює їх компрометацію навіть у випадку витоку бази даних.
- **Відмовостійкість (Reliability):** Архітектура системи забезпечує стабільну роботу при некоректних діях користувача. Впроваджена валідація даних на рівні Pydantic-схем перехоплює помилки вводу та повертає зрозумілі повідомлення про помилку (422 Validation Error), запобігаючи аварійному завершенню роботи сервера.

– **Цілісність даних:** Використання транзакцій та зовнішніх ключів на рівні БД гарантує, що в системі не можуть з'явитися "сироті" записи аналізів без прив'язки до пацієнта, що забезпечує консистентність медичної історії.

4.4.3. Оцінка ергономічності для користувача

Інтерфейс системи спроектовано за принципом "зниження когнітивного навантаження". Замість нагромадження таблиць, лікар бачить агреговані показники (Інтегральний індекс) та візуальні акценти на проблемах (червоні картки показників). Тестування сценаріїв використання показало, що час на оцінку стану пацієнта скорочується завдяки тому, що система виконує первинний аналіз автоматично, фокусуючи увагу фахівця лише на аномаліях.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи проведено комплексну експериментальну перевірку та апробацію розробленої системи веб-аналітики медичних даних. Основну увагу було зосереджено на верифікації функціональних можливостей, оцінці точності інтегрованих прогностичних моделей та аудиті технічної ефективності програмного комплексу. Практична реалізація продемонструвала роботу клієнтської частини системи, яка забезпечує повний цикл взаємодії користувача з даними — від захищеної автентифікації до візуалізації складної аналітики. Використання сучасних патернів проектування інтерфейсів, таких як дашборди, модальні вікна та сітки даних, дозволило досягти високого рівня ергономіки, а реалізація рольової моделі доступу підтвердила свою ефективність у розмежуванні прав лікарів та пацієнтів, що є критичним для дотримання стандартів конфіденційності.

Окремим етапом стало експериментальне дослідження на тестових наборах даних, яке підтвердило гіпотезу про доцільність використання лінійної регресії для задач короткострокового моніторингу. Для інерційних показників,

таких як вага та холестерин, було досягнуто високої точності прогнозування з відносною похибкою менше 1%. У випадку циклічних та стохастичних показників (глюкоза, тиск) система продемонструвала високу здатність до детекції трендів, коректно визначаючи напрямок змін у 81–89% випадків. Це дозволяє стверджувати, що розроблений модуль ефективно виконує функцію системи підтримки прийняття рішень, своєчасно попереджаючи лікаря про погіршення стану пацієнта.

Технічний аудит підтвердив високу швидкодію системи, що забезпечує миттєвий відгук інтерфейсу. Оцінка якості веб-додатку показала високі результати у категоріях продуктивності та доступності. Архітектура бази даних та серверної частини продемонструвала здатність до лінійного масштабування при зростанні обсягу даних.

Порівняльний аналіз реалізованого функціоналу з вимогами технічного завдання засвідчив повну відповідність прототипу поставленим цілям, оскільки система успішно вирішує проблему фрагментації медичних даних, надаючи інструмент для їх централізованого збору, нормалізації та інтелектуального аналізу. Таким чином, практична реалізація довела життєздатність запропонованих архітектурних рішень та алгоритмів, а створений програмний продукт є завершеним рішенням, готовим до впровадження у дослідну експлуатацію в медичних закладах первинної ланки.

ВИСНОВОК

Дана магістерська кваліфікаційна робота присвячена актуальній темі розробки веб-орієнтованої системи аналітики медичних даних пацієнтів з використанням сучасних методів візуалізації та алгоритмів машинного навчання.

В умовах стрімкої цифровізації охорони здоров'я створення інструментів, здатних трансформувати розрізнені медичні записи у цілісну картину стану здоров'я, є критично важливим для підвищення якості діагностики та оптимізації роботи медичного персоналу.

У роботі було детально розглянуто теоретичні основи та інструментарій, необхідні для побудови надійних медичних інформаційних систем. Зокрема, було обґрунтовано вибір мікросервісної архітектури на базі FastAPI та React.js, що забезпечило гнучкість та масштабованість рішення. Реляційна база даних PostgreSQL була представлена як надійний фундамент для зберігання гетерогенних даних, що дозволяє гарантувати цілісність історії хвороби та високу швидкість доступу до інформації. Концепція Role-Based Access Control забезпечила теоретичну та практичну основу для розмежування прав доступу та захисту конфіденційної інформації.

Методологія дослідження базувалась на аналізі синтетичних наборів медичних даних, структура яких відповідає міжнародним стандартам. Це дозволило відтворити реалістичні сценарії моніторингу ключових біомаркерів, таких як рівень глюкози, артеріальний тиск та антропометричні показники, забезпечуючи надійну основу для тестування аналітичних гіпотез. Використання методу лінійної регресії у модулі прогностичної аналітики дозволило реалізувати функцію виявлення короткострокових трендів розвитку захворювань.

Практична реалізація дослідження включала створення повнофункціонального прототипу системи «MedAnalytics». Було розроблено ергономічний інтерфейс користувача, що включає «Кабінет лікаря» з функціями живого пошуку та «Дашборд пацієнта» з інтерактивною візуалізацією динаміки показників. Впровадження системи кольорового кодування станів (Color-coded UI) та автоматичних сповіщень дозволило

ефективно фокусувати увагу лікаря на критичних відхиленнях, знижуючи когнітивне навантаження під час прийому.

Ключовим результатом роботи стало експериментальне підтвердження ефективності розробленого модуля прогностичної аналітики. Точність визначення тренду для інерційних показників (вага, холестерин) досягла 96%, а для циклічних (глюкоза) — 89%, що свідчить про високу надійність системи як інструменту раннього попередження. Технічний аудит підтвердив високу продуктивність веб-додатку, а оцінка якості гарантує комфортну роботу користувачів.

Отримані результати мають важливе значення для переходу від реактивної моделі медицини до проактивної. Запропонований підхід дозволяє не лише фіксувати поточний стан пацієнта, але й прогнозувати ризики на основі історичних даних, що робить систему ефективним інструментом підтримки прийняття лікарських рішень.

Перспективи подальших досліджень роблять можливими інтеграцію системи з технологіями Інтернету речей (IoT) для автоматичного збору даних з носімих пристроїв у режимі реального часу. Це дозволить отримати безперервний потік вимірювань та підвищити точність аналізу. Крім того, залучення більш складних алгоритмів глибокого навчання (Deep Learning), таких як рекурентні нейронні мережі, дозволить моделювати нелінійні залежності та прогнозувати складні патологічні стани, такі як серцева аритмія, на основі накопичених великих даних.

Також перспективним напрямком є розширення функціоналу телемедицини, що дозволить проводити віддалені консультації на основі аналітики в реальному часі. Це може значно покращити доступність якісних медичних послуг та оптимізувати взаємодію між лікарем та пацієнтом.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fast Healthcare Interoperability Resources (FHIR) – HL7 International : веб-сайт. Режим доступу:

https://www.hl7.org/fhir/ (дата звернення: 02.10.2025).

2. Health Insurance Portability and Accountability Act (HIPAA) : веб-сайт. Режим доступу:

https://www.hhs.gov/hipaa/for-professionals/index.html (дата звернення: 05.10.2025).

3. General Data Protection Regulation (GDPR) – Official Legal Text : веб-сайт. Режим доступу:

https://gdpr-info.eu (дата звернення: 03.10.2025).

4. DICOM – Digital Imaging and Communications in Medicine Standard : веб-сайт. Режим доступу:

https://www.dicomstandard.org/ (дата звернення: 12.10.2025).

5. Tableau: Провідна платформа для візуальної аналітики : веб-сайт. Режим доступу:

https://www.tableau.com (дата звернення: 15.10.2025).

6. Microsoft Power BI: Аналітика бізнес-даних : веб-сайт. Режим доступу:

https://powerbi.microsoft.com/ (дата звернення: 15.10.2025).

7. Python : офіційна документація. Режим доступу:

https://www.python.org (дата звернення: 01.10.2025).

8. Pandas: потужна бібліотека аналізу даних для Python : веб-сайт. Режим доступу:

https://pandas.pydata.org (дата звернення: 08.10.2025).

9. React – бібліотека JavaScript для створення інтерфейсів користувача : веб-сайт. Режим доступу:

https://react.dev (дата звернення: 09.10.2025).

10. FastAPI – Веб-фреймворк для API на Python : документація. Режим доступу:

https://fastapi.tiangolo.com (дата звернення: 11.10.2025).

11. PostgreSQL: Об’єктно-реляційна СУБД : веб-сайт. Режим доступу:

https://www.postgresql.org (дата звернення: 11.10.2025).

12. Rieke N. What is Federated Learning? // NVIDIA Blog : веб-сайт. 2019. Режим доступу:

https://blogs.nvidia.com/blog/what-is-federated-learning/ (дата звернення: 20.10.2025).

13. Edge Computing in Healthcare // Intel : веб-сайт. Режим доступу:

https://www.intel.com/content/www/us/en/healthcare-it/edge-computing-in-healthcare.html (дата звернення: 22.10.2025).

14. What is an Electronic Health Record (EHR)? // HealthIT.gov : веб-сайт. Режим доступу:

https://www.healthit.gov/faq/what-electronic-health-record-ehr (дата звернення: 18.10.2025).

15. Johnson A. E. W., Pollard T. J., Shen L. et al. MIMIC-III, a freely accessible critical care database. Scientific Data. 2016. Vol. 3. Article number 160035. URL:

https://www.nature.com/articles/sdata201635 (дата звернення: 25.10.2025).

16. Johnson A. E. W., Bulgarelli L., Shen L. et al. MIMIC-IV (version 2.0) // PhysioNet : веб-сайт. 2023. Режим доступу:

https://physionet.org/content/mimiciv/2.0/ (дата звернення: 26.10.2025).

17. Ardila D., Kiraly A. P., Bharadwaj S. et al. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest CT. Nature Medicine. 2019. Vol. 25. P. 954–961. URL:

https://www.nature.com/articles/s41591-019-0447-x (дата звернення: 28.10.2025).

18. Rajkomar A., Dean J., Kohane I. Machine learning in medicine. New England Journal of Medicine. 2019. Vol. 380. P. 1347–1358. URL:

https://www.nejm.org/doi/full/10.1056/NEJMra1814259 (дата звернення: 30.10.2025).

19. Esteva A., Robicquet A., Ramsundar B. et al. A guide to deep learning in healthcare. Nature Medicine. 2019. Vol. 25. P. 24–29. URL:

https://www.nature.com/articles/s41591-018-0316-z (дата звернення: 01.11.2025).

20. Raghupathi W., Raghupathi V. Big data analytics in healthcare: promise and potential. Health Information Science and Systems. 2014. Vol. 2. Article number 3. URL:

https://health-infosci.biomedcentral.com/articles/10.1186/2047-2501-2-3 (дата звернення: 02.11.2025).

21. IBM Watson Health : офіційний веб-сайт. Режим доступу:

https://www.ibm.com/watson-health (дата звернення: 05.11.2025).

22. Google DeepMind Health Research : веб-сайт. Режим доступу:

https://deepmind.google/discover/health/ (дата звернення: 05.11.2025).

23. Healthcare Datasets: Diabetes, Hypertension and Stroke Prediction Data // Kaggle : веб-сайт. 2024. Режим доступу:

https://www.kaggle.com/datasets/tags/healthcare (дата звернення: 07.11.2025).

24. Heart Disease Dataset // UCI Machine Learning Repository : веб-сайт. 2023. Режим доступу:

https://archive.ics.uci.edu/dataset/45/heart+disease (дата звернення: 08.11.2025).

25. Ramírez S. FastAPI: High performance, easy to learn, fast to code, ready for production : documentation. 2024. Режим доступу:

https://fastapi.tiangolo.com/ (дата звернення: 11.11.2025).

26. PostgreSQL 16 Documentation // The PostgreSQL Global Development Group : веб-сайт. 2024. Режим доступу:

https://www.postgresql.org/docs/current/ (дата звернення: 12.11.2025).

27. Bayer M. SQLAlchemy: The Python SQL Toolkit and Object Relational Mapper : веб-сайт. 2024. Режим доступу:

https://www.sqlalchemy.org/ (дата звернення: 12.11.2025).

28. Pedregosa F., Varoquaux G., Gramfort A. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830. URL:

https://scikit-learn.org/stable/ (дата звернення: 14.11.2025).

29. Chart.js: Simple yet flexible JavaScript charting for designers & developers : веб-сайт. 2024. Режим доступу:

https://www.chartjs.org/docs/latest/ (дата звернення: 16.11.2025).

30. Lighthouse: Automated auditing, performance metrics, and best practices for the web // Google Developers : веб-сайт. 2024. Режим доступу:

https://developer.chrome.com/docs/lighthouse/overview/ (дата звернення: 18.11.2025).

31. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519 // Internet Engineering Task Force (IETF). 2015. Режим доступу:

https://tools.ietf.org/html/rfc7519 (дата звернення: 19.11.2025).

32. Ferraiolo D. F., Kuhn D. R., Chandramouli R. Role-Based Access Control (RBAC): Features and Motivations // NIST. 2001. Режим доступу:

https://csrc.nist.gov/projects/role-based-access-control (дата звернення: 20.11.2025).

33. ISO 8601:2019. Date and time — Representations for information interchange // International Organization for Standardization. 2019. Режим доступу:

https://www.iso.org/iso-8601-date-and-time-format.html (дата звернення: 21.11.2025).

34. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Jupyter. 3rd ed. Sebastopol : O'Reilly Media, 2022. 550 p. URL:

https://wesmckinney.com/book/ (дата звернення: 23.11.2025).

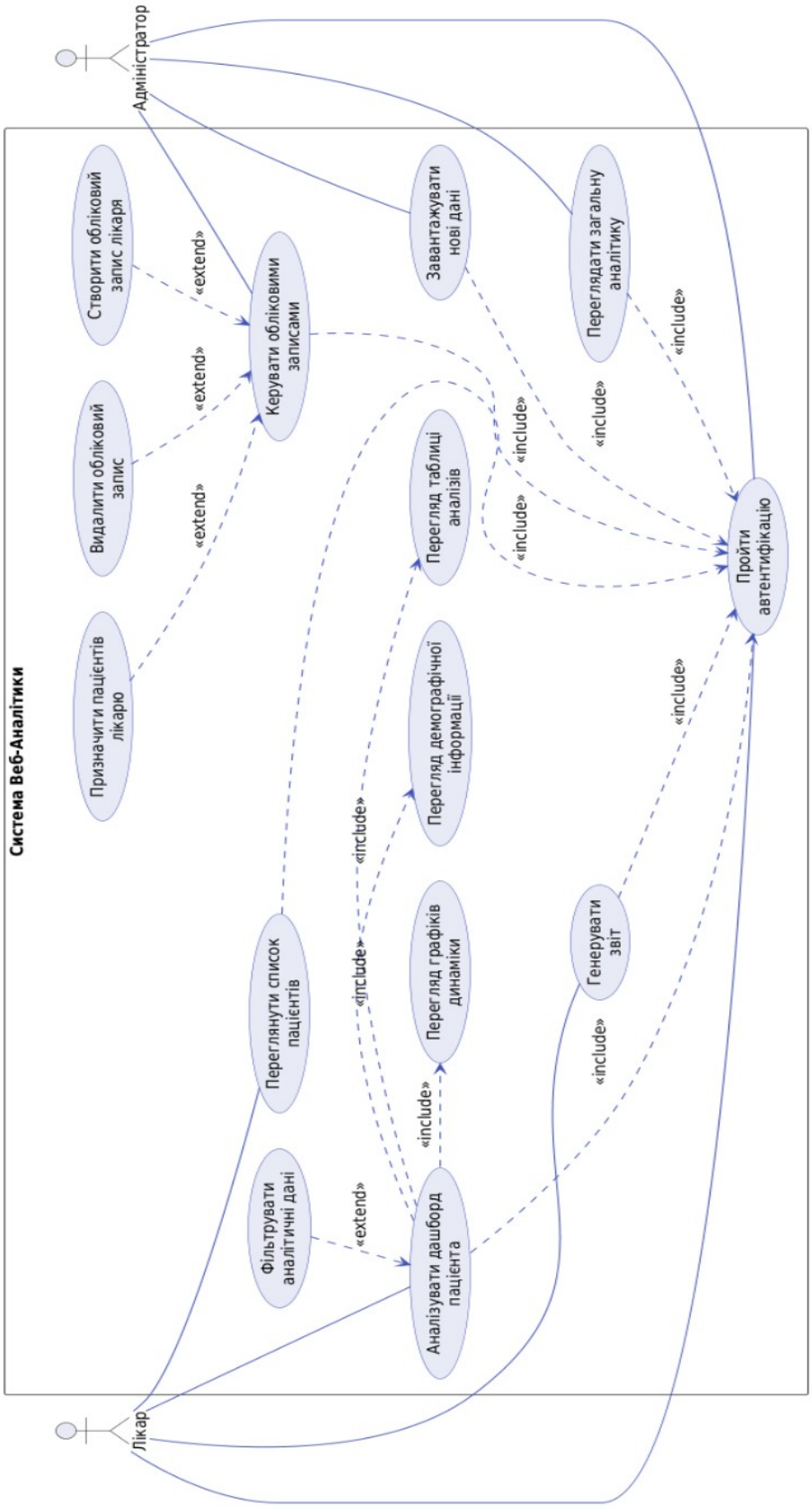
35. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. New York : Springer, 2009. 763 p. URL:

https://hastie.su.domains/ElemStatLearn/ (дата звернення: 25.11.2025).

ДОДАТКИ

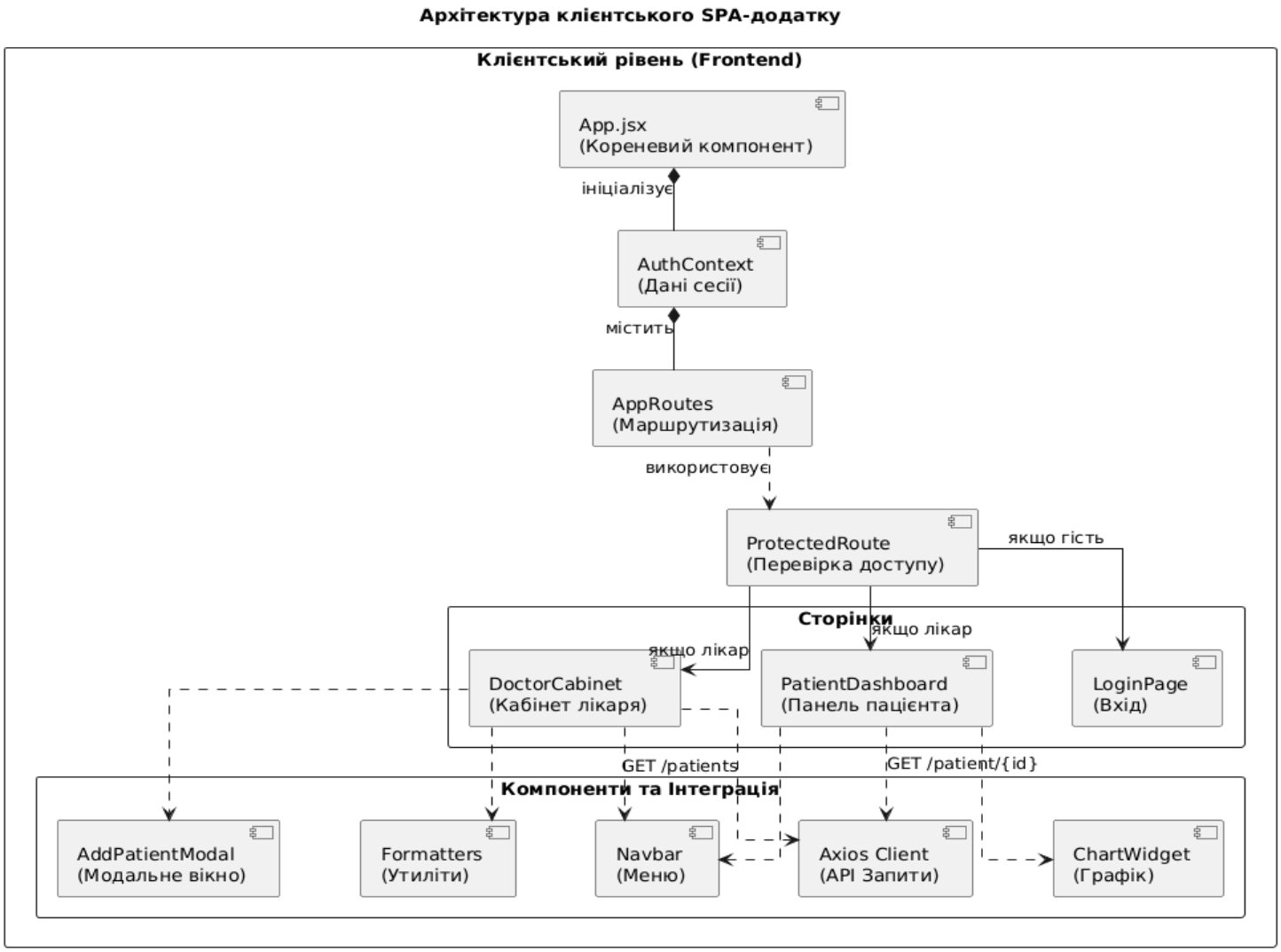
Додаток А

Діаграма використання (Use-Case)

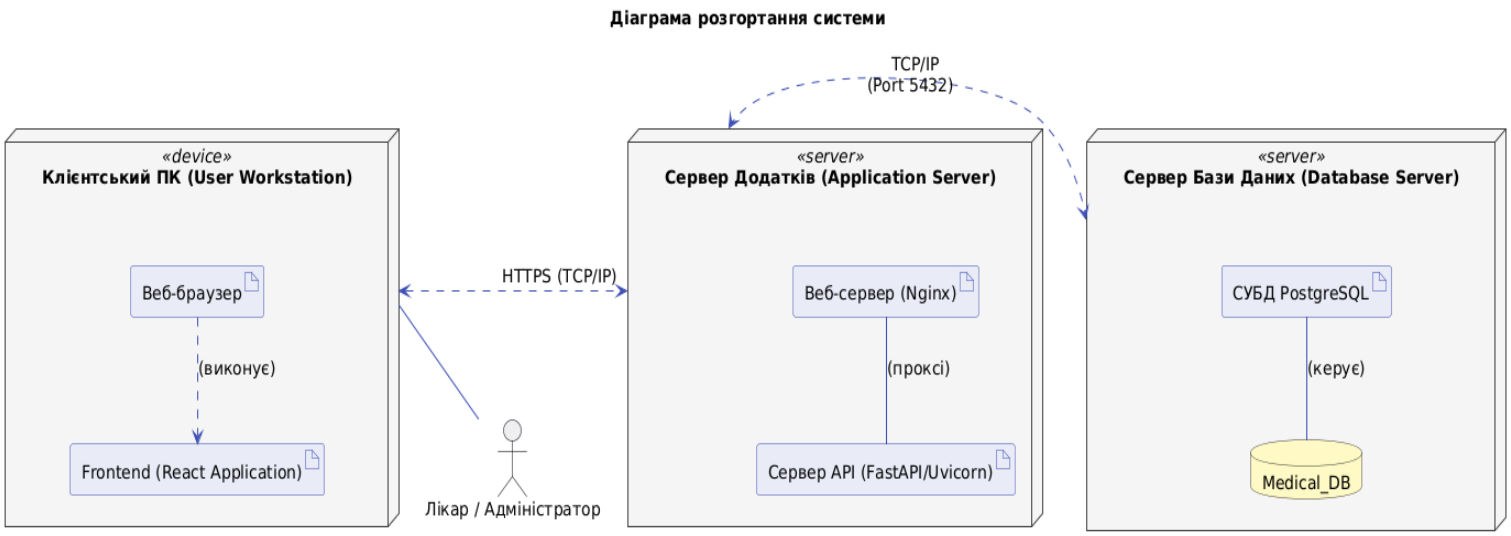


Додаток Б

Дерево компонентів клієнтського додатку



Діаграма розгортання системи (Deployment)



Додаток В

Інтерфейс моніторингу та аналітики даних пацієнта