ІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему: **«РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ МЕДИЧНОГО ЗАКЛАДУ»**

Виконав: студент 4 курсу, групи КНТ-512
спеціальності 123 Комп'ютерна інженерія

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(код і назва спеціальності)

КЛИМЕНКО Є.О.
(прізвище та ініціали)
Керівник ТІМЕНКО А.В.
(прізвище та ініціали)
Рецензент РОМАНОВСЬКИЙ О.В.
(прізвище та ініціали)

2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти (освітній ступінь) бакалаврський

Спеціальність 123 Комп'ютерна інженерія

(код і назва)

Освітня програма (спеціалізація) Комп'ютерна інженерія

(назва)

ЗАТВЕРДЖУЮ

Зав. кафедри

_____ Кудерметов Р.К.

"14" квітня 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

КЛИМЕНКО Євгеній Олегович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Розробка комп'ютерної системи для медичного закладу»

керівник проекту (роботи) ТІМЕНКО А.В., ст. викл.

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

затверджені наказом вищого навчального закладу від "14" квітня 2026 року № 160

2. Строк подання студентом проекту (роботи) 01.06. 2025 року

3. Вихідні дані до проекту (роботи) Python, PostgreSQL, Docker, Pydantic

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

1) Опис предметної області

2) Проєктування архітектури та апаратного забезпечення

3) Тестування та впровадження

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

Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ТІМЕНКО А.В., ст. викладач		
Нормоконтроль	ДЬЯЧУК Т.С., ст. викладач		

7. Дата видачі завдання 14.04. 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Опис предметної області	20.04.2026	
2	Огляд та аналіз існуючих медичних систем	23.04.2026	
3	Обгрунтування загальної архітектури системи	29.04.2026	
4	Вибір середовища розробки	03.05.2026	
5	Проєктування структури реляційної бази	13.05.2026	
6	Розробка функціоналу	18.05.2026	
7	Тестування програмно-апаратного комплексу	26.05.2026	

Студент(ка) _____ Євгеній КЛИМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Артур ТІМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 72 с., 12 табл., 25 рис, 7 листингів, 1 додаток, 22 джерел.

DJANGO, FASTAPI, PYTHON, БАЗА ДАНИХ, ІНФОРМАЦІЙНА СИСТЕМА МЕДИЧНИЙ ЗАКЛАД

Метою дипломної роботи є розробка та впровадження відмовостійкої комп'ютерної системи для автоматизації інформаційних процесів медичного закладу, що включає проектування мережевої інфраструктури та створення спеціалізованого програмного забезпечення для ведення електронних медичних карток і розкладу прийомів.

Для досягнення поставленої мети було визначено та вирішено такі основні завдання:

– провести аналіз предметної області, дослідити існуючі медичні інформаційні системи та сформувати вимоги до апаратно-програмного комплексу;

– обґрунтувати вибір стека технологій та спроектувати структуру реляційної бази даних;

– розробити серверну частину програмного забезпечення (API) для управління модулями реєстратури, розкладу лікарів та електронних карток пацієнтів;

– впровадити криптографічні методи захисту персональних медичних даних;

Об'єкт дослідження – процеси автоматизації, обробки, зберігання та захисту інформації в рамках функціонування медичного закладу.

Предмет дослідження – програмно-апаратні засоби, архітектурні рішення, алгоритми та технології розробки комп'ютерної системи медичного призначення.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ACID – Atomicity, Consistency, Isolation, Durability, Атомарність, узгодженість, ізольованість, довговічність

API – Application Programming Interface, Інтерфейс програмування застосунків

DICOM – Digital Imaging and Communications in Medicine, Цифрові зображення та комунікації в медицині

JSON – JavaScript Object Notation, Нотація об’єктів JavaScript

MIS – Medical Information System, Медична інформаційна система

ORM – Object-Relational Mapping, Об’єктно-реляційне відображення

RLS – Row-Level Security, Безпека на рівні рядків (записів)

SPA – Single Page Application, Односторінковий вебзастосунок

SQL – Structured Query Language, Мова структурованих запитів

XML – Extensible Markup Language, Розширювана мова розмітки

ВСТУП

Сучасний етап розвитку системи охорони здоров'я в Україні характеризується глибинними трансформаційними процесами, основою яких є масштабна цифровізація та обов'язкова інтеграція медичних закладів у національну електронну систему eHealth. Перехід від застарілих паперових архівів до високотехнологічних комп'ютерних систем вже не є питанням інноваційності, а виступає базовою вимогою для забезпечення прозорості, швидкості та належної якості надання медичних послуг. Електронний документообіг дозволяє мінімізувати медичні помилки, пришвидшити доступ до критичного анамнезу пацієнта та значно оптимізувати робочий час персоналу. Проте, незважаючи на стрімкий розвиток ринку медичних інформаційних систем, процес їх впровадження стикається з низкою суттєвих інженерних та організаційних перешкод.

Існуючі комерційні рішення часто є масивними, перевантаженими надлишковим функціоналом та орієнтованими переважно на повністю хмарну (S архітектуру. Це створює критичну залежність медичного закладу від стабільності зовнішнього інтернет-з'єднання та вимагає значних регулярних фінансових витрат на ліцензування робочих місць. Для багатьох вузькоспеціалізованих клінік або регіональних медичних установ такі монолітні системи є економічно та технічно невиправданими, що формує гостру потребу у розробці більш гнучких, модульних та адаптивних локальних рішень, які заклад зможе повністю контролювати.

Розробка власної спеціалізованої комп'ютерної системи, яка б гармонійно поєднувала надійну серверну архітектуру, систему резервного живлення та оптимізоване програмне забезпечення є вкрай актуальним інженерним завданням. Створення такого програмно-апаратного комплексу дозволить підвищити якість обслуговування пацієнтів, оптимізувати робочий час медичного персоналу та забезпечити високий рівень відмовостійкості критичної інфраструктури закладу.

Окрім недоліків виключно програмного підходу, особливої актуальності набуває питання апаратної надійності. В умовах сучасних інфраструктурних викликів та перебоїв з енергопостачанням в Україні, критично важливою вимогою до комп'ютерних систем медичного призначення стає їхня енергонезалежність та висока відмовостійкість. Будь-яка сучасна програмна платформа втрачає свою ефективність, якщо локальна мережа лікарні чи сервери знеструмлені. Тому інженерний підхід, який передбачає не лише написання програмного коду, але й проектування стійкої локальної топології з точним розрахунком систем автономного резервного живлення для безперебійної роботи серверного ядра, є життєво необхідним для забезпечення безперервного лікувального процесу.

З огляду на це, розробка спеціалізованого програмно-апаратного комплексу із застосуванням сучасних високопродуктивних технологій серверної розробки, надійних реляційних баз даних та методів криптографічного захисту, безпосередньо інтегрованого з фізичною інфраструктурою закладу, є своєчасним та надзвичайно затребуваним інженерним завданням. Створення такої системи дозволить вирішити проблеми безпеки персональних медичних даних, забезпечити стабільну роботу персоналу в умовах нестабільного живлення та позбавити заклад залежності від дорогих пропрієтарних платформ, що повною мірою підтверджує високу актуальність обраної теми дипломної роботи.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості інформаційних процесів у сучасних медичних закладах

Сучасний етап розвитку системи охорони здоров'я характеризується стрімкою цифровізацією та інтеграцією інформаційно-комунікаційних технологій в усі сфери діяльності медичних установ. Інформаційні процеси в медичному закладі є складним комплексом дій, спрямованих на збір, реєстрацію, обробку, накопичення, зберігання, пошук і передачу медичної, адміністративної та фінансової інформації [1]. Особливістю цих процесів є висока динамічність, критичність до затримок у часі та необхідність забезпечення абсолютної конфіденційності й цілісності даних пацієнтів. Специфіка медичної інформації полягає в її гетерогенності, оскільки вона об'єднує структуровані анкетні дані, текстові описи діагнозів, цифрові сигнали діагностичного обладнання та великі обсяги графічних даних. Основні типи інформації, що циркулюють у системі, та їхні ключові характеристики наведені у таблиці (табл. 1.1.).

Таблиця 1.1 – Класифікація та характеристики медичної інформації

Тип медичних даних	Джерело надходження	Формат зберігання	Особливості та обмеження обробки
Персональні та анкетні дані	Реєстратура, пацієнт	Текстовий (SQL-таблиці)	Суворий контроль доступу, шифрування даних
Клінічні записи та анамнез	Лікарі, медичні сестри	Текстовий / Об'єктний (JSON)	Динамічна структура, підтримка версійності записів
Результати діагностики	Лабораторні аналізатори	Структуровані файли (XML)	Швидка валідація, інтеграція за стандартом HL7
Графічні дослідження	КТ, МРТ, Рентген, УЗД	Бінарні файли (DICOM)	Великий обсяг, потреба у PACS-серверах

Організація інформаційних потоків безпосередньо впливає на якість надання медичних послуг та ефективність використання ресурсів установи. Рух інформації починається з моменту звернення пацієнта до реєстратури, проходить

через етапи прийому у лікаря, проведення лабораторних чи інструментальних досліджень і завершується формуванням електронної медичної карти. Наочно взаємодію основних суб'єктів та напрямки руху даних відображає загальна схема інформаційних потоків (рис. 1.1).

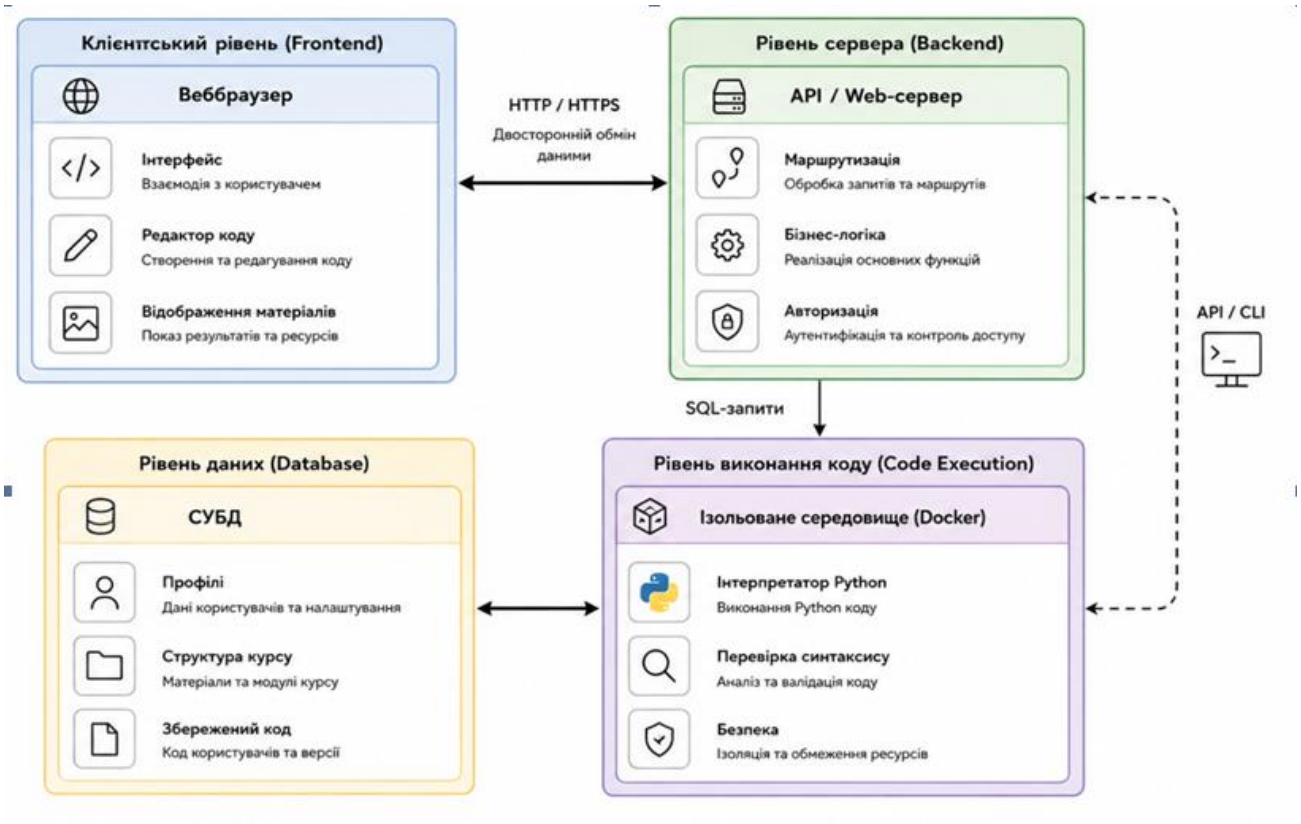

Рисунок 1.1 – Схема інформаційних потоків при обслуговуванні пацієнта

Для забезпечення автоматизації зазначених потоків розробляється спеціалізована архітектура комп'ютерної системи, яка повинна підтримувати багатокористувацький режим роботи та швидкий обмін повідомленнями між клієнтськими робочими місцями і центральним сервером бази даних. Процес взаємодії компонентів системи під час виконання типової операції, наприклад, створення нового електронного призначення або запису про прийом, детально описується за допомогою діаграми послідовності (рис. 1.2). На цій діаграмі відображено часову послідовність викликів методів між інтерфейсом лікаря, контролером бізнес-логіки та сервером бази даних, що дозволяє оцінити затримки та оптимізувати мережеві запити.

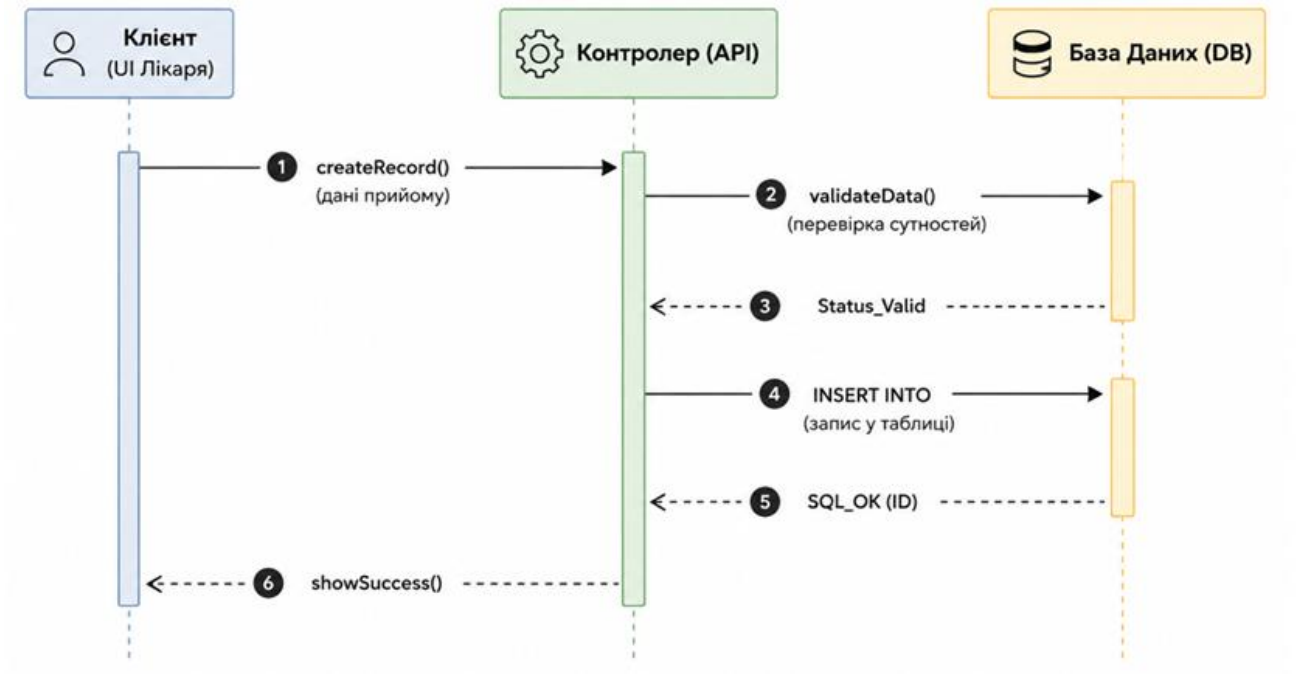

Рисунок 1.2 – Діаграма взаємодії компонентів системи при створенні призначення

Важливим аспектом проектування комп'ютерної системи для медичного закладу є побудова концептуальної моделі даних, яка трансформується в об'єктну діаграму класів. Модель повинна враховувати суворі взаємозв'язки між пацієнтами, медичним персоналом, прийомами та медичними картками, виключаючи можливість дублювання інформації чи втрати цілісності при видаленні записів [2]. Структура основних сутностей системи, їхні ключові атрибути та типи зв'язків між ними представлені на діаграмі класів (рис. 1.3). Ця архітектура класів дозволяє перенести бізнес-логіку медичних процесів у програмний код із чітким розмежуванням відповідальності між об'єктами.

Під час аналізу інформаційних процесів у медичному середовищі виокремлюють низку специфічних вимог та організаційних проблем, які необхідно враховувати при розробці програмно-апаратного комплексу. До таких критичних факторів відносяться:

– потреба в миттєвому доступі до критично важливих медичних показників пацієнта в екстрених ситуаціях;

– високий рівень загрози несанкціонованого доступу до конфіденційних персональних даних;

– необхідність інтеграції з різнорідним аналоговим та цифровим діагностичним обладнанням;

– забезпечення безперебійної роботи системи в умовах нестабільного енергопостачання або відмови локальних мереж.

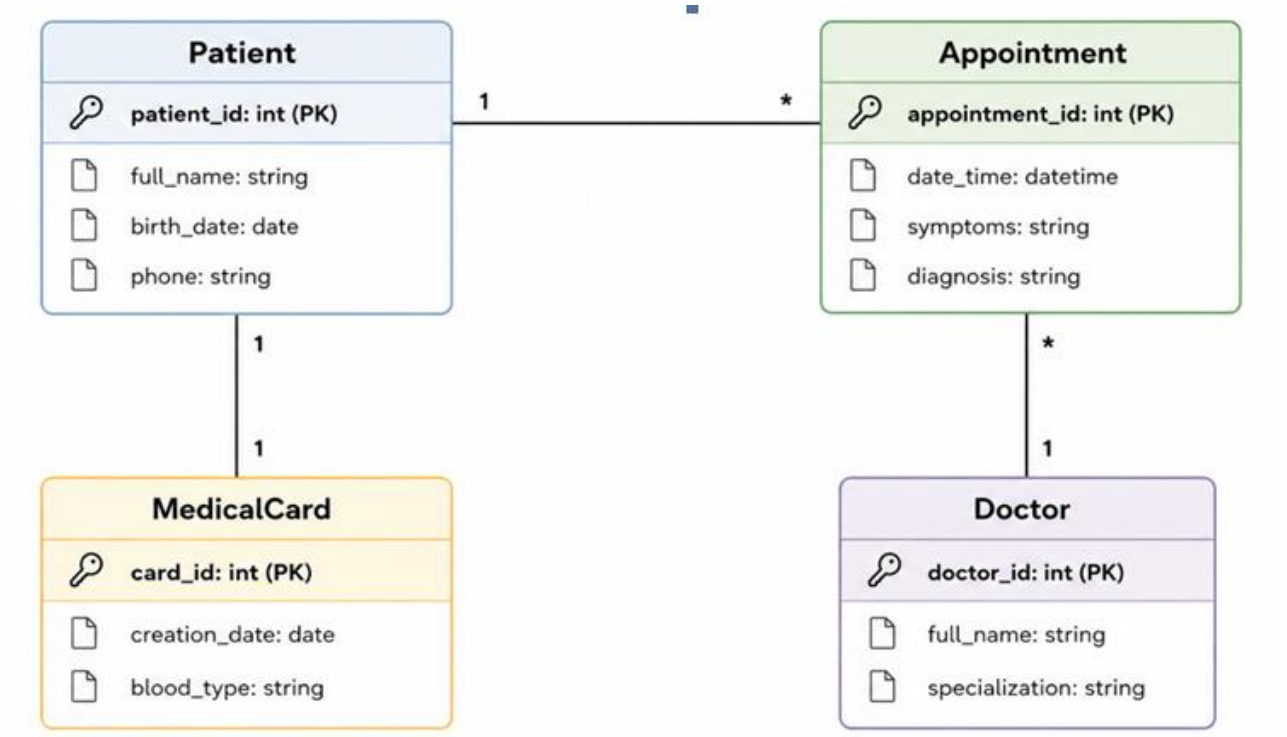

Рисунок 1.3 – Об'єктна діаграма класів медичної інформаційної системи

Усі ці особливості вимагають комплексного підходу до проектування, де програмна частина (бази даних, інтерфейси, алгоритми захисту) тісно взаємодіє з апаратною інфраструктурою (серверами, мережевими комутаторами, системами резервного живлення) медичного закладу.

1.2 Огляд та аналіз існуючих медичних інформаційних систем

Ринок медичних інформаційних систем (МІС) сьогодні пропонує широкий спектр рішень, від локальних програмних продуктів для невеликих приватних

кабінетів до комплексних хмарних платформ, здатних обслуговувати великі регіональні мережі лікарень [6]. Аналіз існуючих рішень, зокрема тих, що активно впроваджуються та функціонують у рамках національної системи електронного здоров'я eHealth, дозволяє виявити їхні сильні та слабкі сторони, а також сформувати вимоги до розроблюваної системи. Серед найбільш поширених систем можна виділити такі платформи, як Helsi, Doctor Eleks, Медстар та ЕМСІмед. Більшість із цих систем побудовані за клієнт-серверною або хмарною архітектурою [3], що дозволяє централізовано зберігати дані та надавати доступ до них через веб-інтерфейси або спеціалізовані десктопні додатки. Для систематизації характеристик цих систем проведено їх порівняльний аналіз за ключовими технічними та функціональними параметрами, результати якого наведено у таблиці (табл. 1.2).

Таблиця 1.2 – Порівняльний аналіз популярних медичних інформаційних систем

Назва МІС	Тип архітектури	Можливості інтеграції	Переваги	Недоліки
Helsi	Хмарна (SaaS)	Високі (API, eHealth)	Зручний портал для пацієнтів, швидке розгортання	Залежність від інтернет-з'єднання, закритий код
Doctor Eleks	Клієнт-серверна / Хмарна	Високі (HL7, DICOM)	Потужний інструментарій для стаціонарів, гнучкість	Складність налаштування, високі вимоги до апаратури
ЕМСІмед	Клієнт-серверна	Середні (eHealth)	Надійність роботи в локальній мережі, захист	Застарілий інтерфейс у деяких модулях
Medstar	Хмарна	Високі	Модульна структура, хороша підтримка мобільних пристроїв	Обмежені можливості кастомізації звітів

Сучасні медичні інформаційні системи переважно використовують багаторівневу хмарну архітектуру, що забезпечує масштабованість та відмовостійкість. Така структура дозволяє розділити базу даних, бізнес-логіку та

користувацький інтерфейс, що спрощує подальше обслуговування та оновлення програмного забезпечення. Типова архітектура хмарної МІС, яка лежить в основі більшості розглянутих аналогів, представлена на схемі (рис. 1.4).

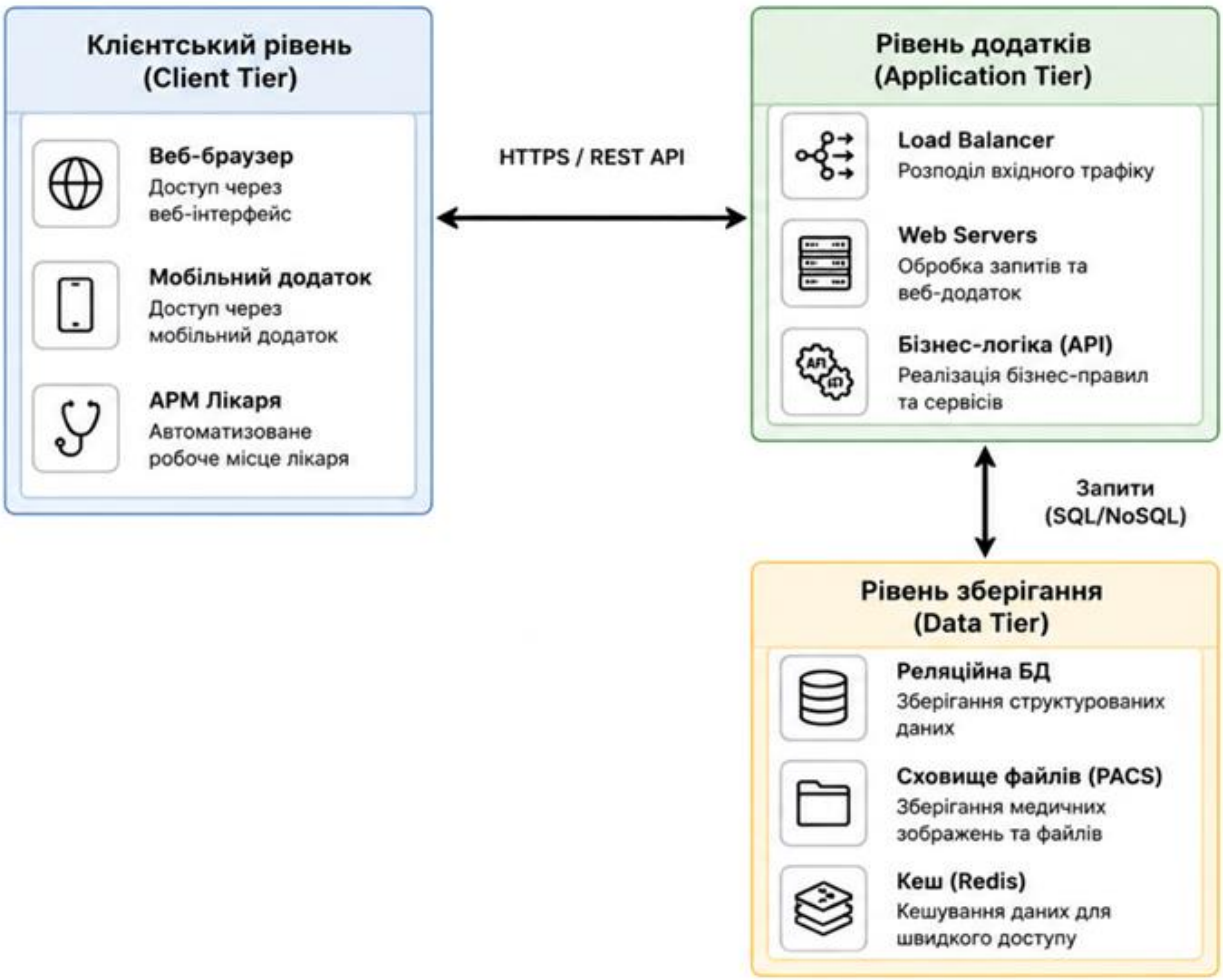

Рисунок 1.4 – Архітектура типової хмарної МІС

Важливою складовою функціонування будь-якої сучасної МІС є її здатність взаємодіяти із зовнішніми системами, зокрема з центральним компонентом національної системи охорони здоров'я. Ця взаємодія відбувається через захищені канали зв'язку з використанням стандартизованих протоколів та електронного цифрового підпису для підтвердження дій лікаря. Діаграма взаємодії локальної МІС із зовнішнім центральним компонентом під час передачі медичних даних (наприклад, реєстрації електронного рецепта чи направлення) відображає цей процес (рис. 1.5).

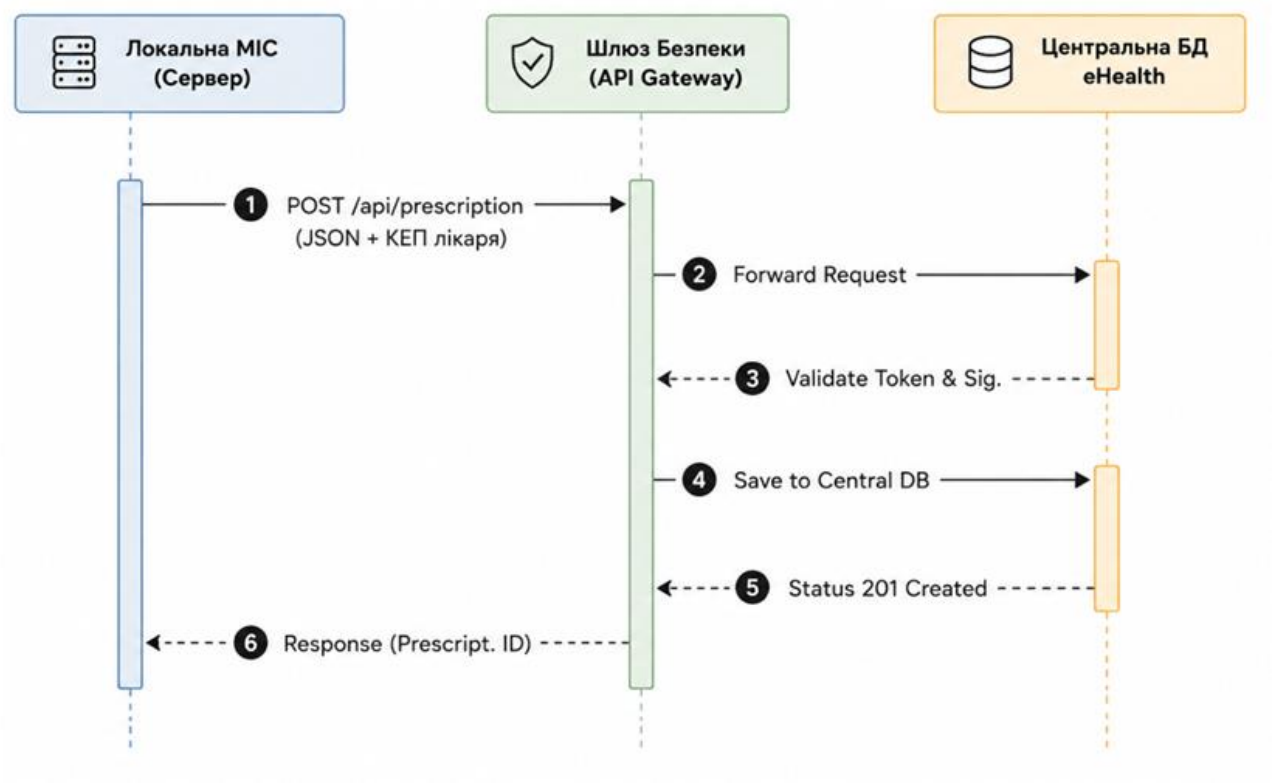

Рисунок 1.5 – Діаграма взаємодії локальної МІС з центральною базою даних

Для реалізації подібної взаємодії розробники використовують сучасні мови програмування та формати обміну даними. Оскільки при проектуванні власних систем часто обирається мова Python, типовий запит до зовнішнього API для перевірки статусу пацієнта або відправки медичного запису формується з використанням стандартних бібліотек [4]. Приклад такого програмного коду, що ілюструє базовий механізм відправки серіалізованих у JSON даних через захищене з'єднання, наведено нижче (Лістинг 1.1).

Лістинг 1.1 – Приклад формування JSON-запиту до API центрального компонента

```
import requests
import json

def send_medical_record(patient_id, diagnosis_code, auth_token):
    url = "https://api.ehealth.gov.ua/v1/medical-records"
    headers = {
        "Authorization": f"Bearer {auth_token}",
        "Content-Type": "application/json"
    }
```

```
    payload = {
        "patient_id": patient_id,
        "diagnosis_icd10": diagnosis_code,
        "record_type": "ambulatory",
        "timestamp": "2026-05-31T17:52:08Z"
    }

    try:
        response = requests.post(url, headers=headers,
data=json.dumps(payload))
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Помилка взаємодії з API: {e}")
        return None
```

Аналіз наведених систем та архітектурних підходів дозволяє сформувати чіткі вимоги до комп'ютерної системи, що розробляється. Зокрема, існуючі рішення часто перевантажені надлишковим функціоналом, який не використовується у вузькоспеціалізованих медичних закладах, що призводить до невиправданих витрат на впровадження та підтримку. Тому розроблювана система повинна враховувати такі основні аспекти:

– забезпечення високої швидкодії та інтуїтивно зрозумілого інтерфейсу для медичного персоналу;

– підтримка модульності для можливості поетапного впровадження та масштабування;

– інтеграція алгоритмів шифрування для захисту локальних баз даних згідно з вимогами законодавства.

Таким чином, огляд існуючих медичних інформаційних систем виявив необхідність створення більш адаптивного та легковагового рішення, що поєднує переваги сучасних програмних технологій та надійних апаратних засобів, здатних функціонувати в умовах конкретного медичного закладу.

1.3 Аналіз вимог до розроблюваної комп'ютерної системи та постановка задачі на дипломне проектування

На основі проведеного аналізу предметної області та недоліків існуючих на ринку рішень формується комплекс вимог до розроблюваної комп'ютерної системи медичного закладу. Головним критерієм успішного проектування є баланс між надійністю апаратної інфраструктури та ефективністю програмного забезпечення. Функціональні вимоги до системи охоплюють необхідність автоматизації повсякденних рутинних операцій медичного персоналу. Зокрема, система повинна забезпечувати реєстрацію нових пацієнтів, ведення електронної медичної картки, планування розкладу прийомів лікарів та генерацію статистичної звітності [5]. Для візуалізації основних функціональних можливостей системи з точки зору кінцевих користувачів розроблено діаграму прецедентів (рис. 1.6), яка описує взаємодію акторів із головними модулями програмного комплексу.

Рисунок 1.6 – Діаграма прецедентів (Use Case) комп'ютерної системи

Окрім функціонального наповнення, критично важливими для медичної сфери є нефункціональні вимоги, які визначають якісні характеристики роботи комплексу. Враховуючи реалії функціонування критичної інфраструктури, особливу увагу необхідно приділити питанням відмовостійкості, захисту персональних даних та енергонезалежності серверного обладнання. Деталізований перелік основних нефункціональних вимог та їхніх кількісних і якісних показників наведено у таблиці (табл. 1.3).

Таблиця 1.3 – Нефункціональні вимоги до комп'ютерної системи

Категорія вимог	Опис вимоги	Цільовий показник / Метрика
Продуктивність	Час відгуку системи на стандартні запити (пошук, збереження)	Не більше 1.5 секунди при навантаженні 100 користувачів
Безпека даних	Шифрування баз даних та трафіку	Використання протоколу TLS 1.3, хешування паролів bcrypt
Доступність	Час безперебійної роботи (Uptime) центрального сервера	Не менше 99.9% (з урахуванням резервних копій)
Автономність	Робота апаратної частини при знеструмленні загальної мережі	Перехід на ДБЖ та зарядні станції без розриву сесій (мінімум 4 години)

Логічним підсумком аналізу вимог є чітка постановка задачі на дипломне проектування. Враховуючи комплексність проблеми, розробка повинна охоплювати як інженерну, так і програмну складові. З апаратного боку необхідно спроектувати локальну обчислювальну мережу медичного закладу із застосуванням надійного комутаційного обладнання та розрахунком потужності джерел безперебійного живлення (ДБЖ/зарядних станцій) для підтримки критичних вузлів. З програмного боку передбачається створення серверного ядра (Backend) із використанням сучасної високорівневої мови програмування, такої як Python, та побудова реляційної бази даних. Загальна структурна схема запропонованого апаратно-програмного рішення відображена на схемі (рис. 1.7).

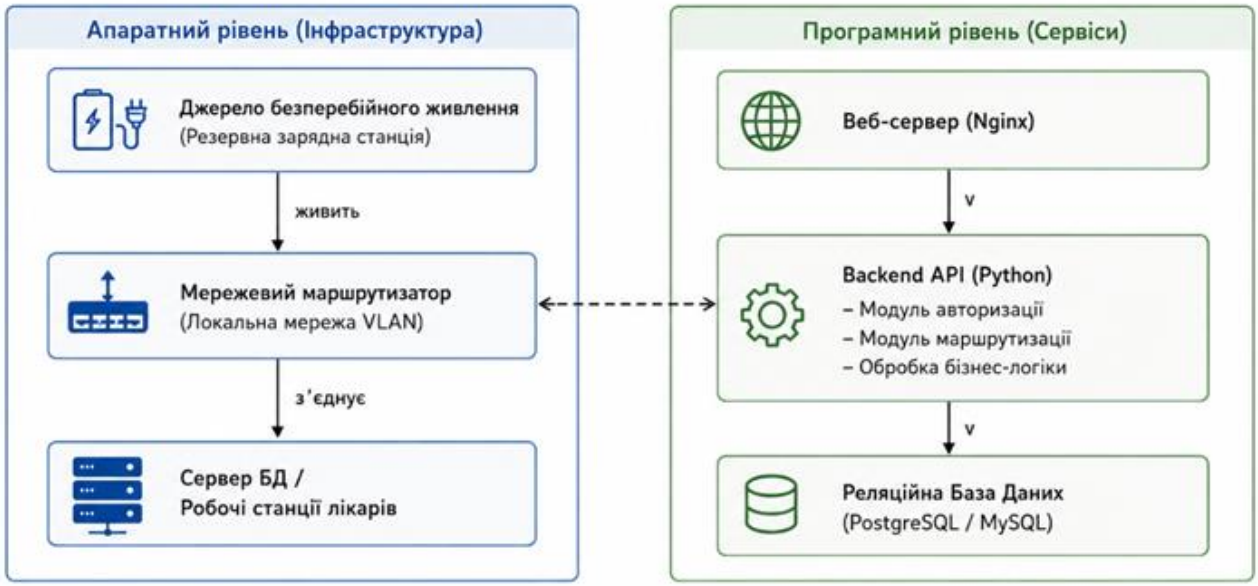

Рисунок 1.7 – Структурна схема апаратно-програмного комплексу

Для реалізації програмної частини та забезпечення гнучкості при роботі з базою даних, доцільно використовувати технології об'єктно-реляційного відображення (ORM). Це дозволить описувати структури даних мовою Python, мінімізуючи використання прямих SQL-запитів на етапі бізнес-логіки. Приклад базової реалізації вимоги щодо збереження анкетних даних пацієнта за допомогою опису моделі даних наведено у програмному коді (Лістинг 1.2).

Лістинг 1.2 – Опис моделі даних пацієнта для розроблюваної системи

```
from sqlalchemy import Column, Integer, String, Date
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Patient(Base):
    __tablename__ = 'patients'

    id = Column(Integer, primary_key=True, index=True)
    first_name = Column(String(50), nullable=False)
    last_name = Column(String(50), nullable=False)
    date_of_birth = Column(Date, nullable=False)
    phone_number = Column(String(15), unique=True, index=True)
    insurance_policy = Column(String(30), unique=True)

    def __repr__(self):
```

```
        return f"<Patient(name='{self.first_name} {self.last_name}',
phone='{self.phone_number}')>"
```

Таким чином, для досягнення мети дипломного проектування необхідно виконати наступні конкретні завдання:

– розробити логічну та фізичну структуру реляційної бази даних для зберігання медичної інформації;

– створити програмні модулі реєстратури та електронної медичної картки з використанням обраного стека технологій (Python);

– розгорнути систему на серверному обладнанні та провести комплексне тестування її функціональності та відмовостійкості.

Виконання цих завдань дозволить створити повноцінний прототип комп'ютерної системи, готовий до дослідної експлуатації в умовах реального медичного закладу.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір та обґрунтування загальної архітектури системи

Успішна реалізація інформаційної системи для медичного закладу вимагає ретельного проектування її базової архітектури, яка має забезпечувати високу продуктивність, масштабованість та надійний захист конфіденційної інформації пацієнтів. На основі аналізу предметної області прийнято рішення про використання класичної клієнт-серверної архітектури з розділенням на рівень представлення (Frontend), рівень бізнес-логіки (Backend) та рівень зберігання даних [6]. Такий підхід дозволяє незалежно оновлювати клієнтські додатки, не втручаючись у роботу центрального сервера, а також забезпечує централізований контроль доступу та єдину точку входу для всіх мережевих запитів. Загальна структурна організація.

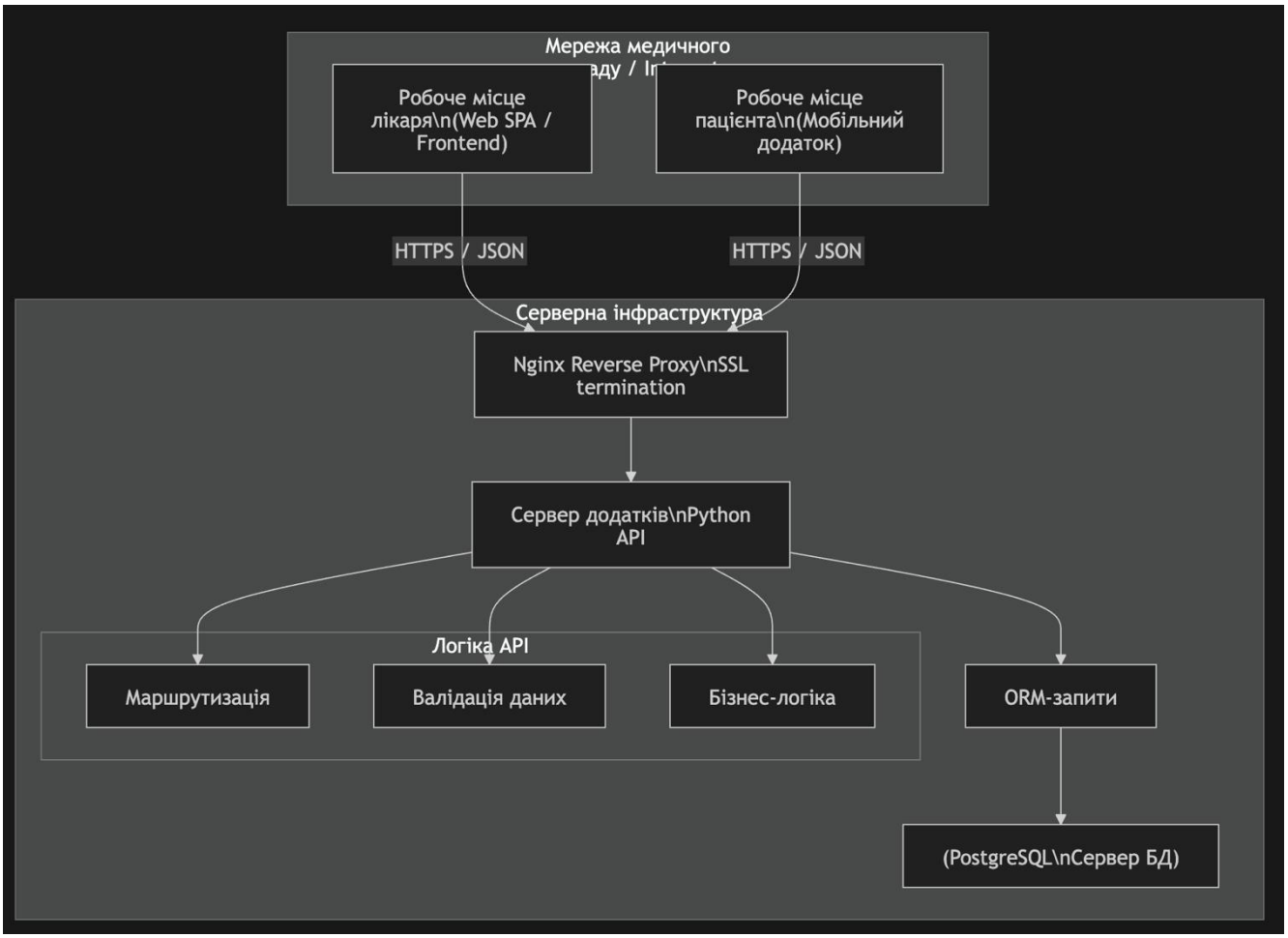

Рисунок 2.1 – Архітектура клієнт-серверної взаємодії медичної системи

Вибір мови програмування є одним із ключових етапів, який визначає подальшу швидкість розробки та легкість підтримки програмного продукту. Для реалізації серверної частини (Backend) розроблюваної системи обрано мову програмування Python. Це рішення обґрунтовується високою швидкістю створення прототипів, наявністю потужних асинхронних фреймворків (таких як FastAPI), а також широкою екосистемою перевірених бібліотек для обробки даних і криптографічного захисту. Важливою перевагою Python є його домінуюча позиція у сфері аналізу даних; використання цієї мови закладає міцний фундамент для майбутнього розширення системи інтелектуальними модулями, наприклад, для автоматизованого аналізу симптомів або прогнозування завантаженості медичного закладу на основі машинного навчання.

Наступним критичним кроком є вибір системи керування базами даних (СУБД). Медична інформація характеризується складною структурою зв'язків між

пацієнтами, лікарями, діагнозами та історією лікування, що робить використання реляційних баз даних найбільш доцільним підходом. Для прийняття остаточного рішення було проведено порівняльний аналіз провідних систем, результати якого систематизовано (табл. 2.1).

Таблиця 2.1 – Порівняльний аналіз систем керування базами даних для МІС

Критерій порівняння	PostgreSQL	MySQL	MongoDB (NoSQL)
Тип моделі даних	Об'єктно-реляційна	Реляційна	Документо-орієнтована
Підтримка ACID-транзакцій	Повна (надійна ізоляція)	Повна	Часткова (на рівні документа)
Масштабованість	Висока (реплікація, партиціювання)	Середня	Дуже висока
Придатність для медицини	Оптимальна (сувора типізація)	Добра	Низька (ризик втрати зв'язків)
Захист даних	Вбудоване RLS (Row-Level Security)	Стандартне управління правами	Стандартне управління правами

За результатами аналізу, як основне сховище даних було обрано PostgreSQL [7]. Ця СУБД забезпечує абсолютну відповідність принципам ACID (атомарність, узгодженість, ізольованість, довговічність), що є критично необхідним для запобігання втраті або спотворенню медичних записів під час одночасного доступу кількох користувачів. Крім того, PostgreSQL пропонує потужні механізми безпеки на рівні окремих рядків таблиці (RLS), що дозволяє легко реалізувати розмежування прав доступу, коли лікар може бачити лише картки власних пацієнтів. Використання PostgreSQL формує такі архітектурні переваги:

– суворий контроль цілісності даних на рівні бази;

– висока швидкість виконання складних аналітичних об'єднань (JOIN);

– можливість зберігання неструктурованих даних у форматі JSONB поряд із класичними реляційними таблицями.

Взаємодія між обраною мовою програмування Python та базою даних PostgreSQL здійснюється за допомогою технології об'єктно-реляційного

відображення (ORM), що дозволяє розробникам оперувати даними як об'єктами класів, значно підвищуючи безпеку завдяки автоматичному захисту від SQL-ін'єкцій. Для ілюстрації обраного архітектурного підходу розроблено базовий модуль ініціалізації з'єднання із сервером бази даних, який наведено у лістингу (Лістинг 2.1). Цей конфігураційний файл є основою серверної частини системи, встановлюючи пул асинхронних з'єднань для забезпечення високої пропускної здатності навіть у години пікового навантаження на реєстратуру медичного закладу.

Лістинг 2.1 – Модуль підключення до бази даних PostgreSQL за допомогою Python

```
import os
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from sqlalchemy.orm import sessionmaker

# Формування рядка підключення з урахуванням драйвера asyncpg
DATABASE_URL = os.getenv(
    "DATABASE_URL",

"postgresql+asyncpg://admin_user:secure_password@localhost/medical_d
b"
)

# Створення асинхронного рушія бази даних
engine = create_async_engine(
    DATABASE_URL,
    echo=False,
    pool_size=20,          # Максимальна кількість відкритих
з'єднань
    max_overflow=10        # Додаткові з'єднання при піковому
навантаженні
)

# Фабрика сесій для обробки запитів транзакцій
AsyncSessionLocal = sessionmaker(
    bind=engine,
    class_=AsyncSession,
    expire_on_commit=False
)

async def get_db_session():
    """Асинхронний генератор сесій для залежностей API (Dependency
Injection)"""
    async with AsyncSessionLocal() as session:
        try:
```

```
        yield session
    finally:
        await session.close()
```

Таким чином, комбінація класичної клієнт-серверної архітектури, мови програмування Python та реляційної СУБД PostgreSQL утворює надійний технологічний стек. Цей вибір повністю задовольняє як поточні вимоги до швидкодії та безпеки медичної інформаційної системи, так і створює сприятливі умови для її подальшої інтеграції, супроводу та масштабування в рамках інфраструктури

2.2 Обґрунтування вибору стека технологій та середовища розробки

На етапі проектування програмної архітектури, маючи визначену базову мову програмування (Python) та систему керування базами даних (PostgreSQL), критично важливим є правильний підбір супутніх технологій та фреймворків. Вибір стека технологій безпосередньо впливає на швидкість розробки, можливості масштабування та рівень безпеки медичної інформаційної системи. Для реалізації серверної частини (Backend) необхідно обрати веб-фреймворк, який забезпечуватиме високу пропускну здатність, надійну валідацію вхідних даних та простоту підтримки коду. В екосистемі Python існує кілька провідних рішень, серед яких найпопулярнішими є Django, Flask та FastAPI [8]. Для обґрунтування вибору було проведено порівняльний аналіз цих фреймворків за ключовими для розроблюваної системи критеріями, результати якого наведено у таблиці (табл. 2.2).

Таблиця 2.2 – Порівняльний аналіз веб-фреймворків мови Python

Характеристика	Django	Flask	FastAPI
Архітектурний патерн	Монолітний (MVT)	Мікрофреймворк	Мікрофреймворк (ASGI)
Продуктивність (швидкодія)	Середня (синхронна)	Середня (синхронна)	Дуже висока (асинхронна)
Валідація та серіалізація даних	Вбудована (Django Forms)	Стороння (Marshmallow)	Вбудована (Pydantic)
Автоматична генерація документації	Потребує додаткових налаштувань	Відсутня у базовій версії	Вбудована (Swagger UI / ReDoc)
Поріг входження та гнучкість	Високий (жорстка структура)	Низький (висока гнучкість)	Низький (сувора типізація)

На основі порівняльного аналізу (табл. 2.2) як основний інструмент для розробки серверної частини обрано FastAPI [9]. Цей сучасний фреймворк базується на стандартах ASGI (Asynchronous Server Gateway Interface), що дозволяє йому обробляти велику кількість одночасних з'єднань без блокування основного потоку виконання. Це є критично важливою вимогою для медичного закладу, де в години пік лікарі та пацієнти одночасно звертаються до бази даних для перегляду розкладу чи внесення медичних записів. Крім того, наявність вбудованої автодокументації за стандартом OpenAPI значно спрощує подальшу інтеграцію системи із зовнішніми сервісами (наприклад, eHealth).

Для реалізації клієнтської частини (Frontend) обрано бібліотеку React.js, яка дозволяє створювати динамічні односторінкові додатки (SPA) із високим рівнем реактивності. Щоб забезпечити незалежність розгортання системи від характеристик цільового сервера, весь програмний комплекс упаковується в ізольовані контейнери за допомогою технології Docker. Загальна структура обраного стека технологій та взаємодія його рівнів відображена на схемі (рис.2.2).

Програмна архітектура всередині самого серверного додатку проектується за принципом розділення відповідальності (Separation of Concerns). Це означає, що обробка HTTP-запитів, перевірка прав доступу, бізнес-логіка та безпосередня взаємодія з базою даних розподілені між різними програмними шарами [4].

Рисунок 2.2 – Структура обраного стека технологій

Такий підхід робить код придатним до модульного тестування та полегшує пошук помилок. Життєвий цикл запиту від моменту його надходження до системи до формування відповіді ілюструється за допомогою діаграми архітектури потоку даних (рис. 2.3).

Організація ефективного процесу написання та налагодження коду вимагає вибору відповідного інтегрованого середовища розробки (IDE). Як основне середовище було обрано Visual Studio Code (VS Code). Цей редактор вирізняється високою швидкістю роботи, кросплатформністю та потужною системою розширень. Для комфортної роботи над медичною системою в середовище було інтегровано ряд специфічних інструментів, що задовольняють такі ключові потреби:

– підтримка статичного аналізатора коду (Pylint, MyPy) для виявлення потенційних помилок на етапі написання;

– автоматичне форматування коду за стандартом PEP 8 (Black, Flake8);

– інтеграція з системою контролю версій Git для ведення історії змін та можливості відкату до стабільних версій;

– наявність плагінів для роботи з Docker-контейнерами та прямого підключення до бази даних PostgreSQL для перевірки виконання транзакцій [10].

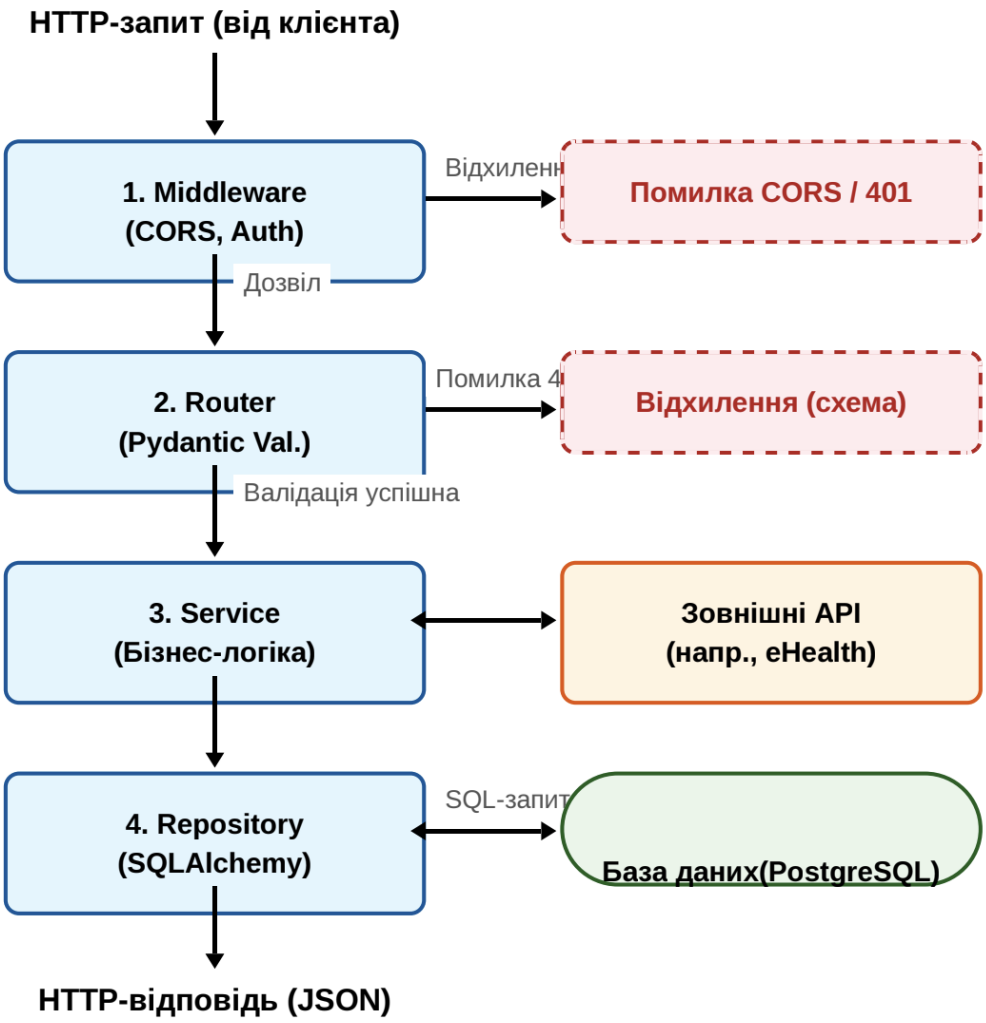

Рисунок 2.3 – Архітектура потоку обробки даних у серверному додатку

Реалізація архітектури починається зі створення головного файлу ініціалізації додатку, де відбувається налаштування маршрутизації, підключення проміжних шарів (Middleware) для забезпечення безпеки та конфігурація обробників помилок. У програмному коді (Лістинг 2.2) продемонстровано базову структуру ініціалізації розроблюваної системи з використанням обраного фреймворку FastAPI, що є точкою входу для всіх подальших модулів.

Лістинг 2.2 – Ініціалізація головного модуля додатку (main.py)

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from api.routes import patients, doctors, appointments

# Створення екземпляра додатку з метаданими для документації
app = FastAPI(
    title="Медична Інформаційна Система",
    description="API для управління записами медичного закладу",
    version="1.0.0"
)

# Налаштування CORS для безпечної взаємодії з React-фронтендом
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000", "https://clinic.local"],
    allow_credentials=True,
    allow_methods=["GET", "POST", "PUT", "DELETE"],
    allow_headers=["Authorization", "Content-Type"],
)

# Підключення модульних маршрутизаторів (Routers)
app.include_router(patients.router, prefix="/api/v1/patients",
tags=["Пацієнти"])
app.include_router(doctors.router, prefix="/api/v1/doctors",
tags=["Лікарі"])
app.include_router(appointments.router,
prefix="/api/v1/appointments", tags=["Прийоми"])

@app.get("/healthcheck", tags=["Системні"])
async def health_check():
    """Ендпоінт для перевірки статусу роботи сервера
балансувальником навантаження"""
    return {"status": "ok", "service": "medical_backend"}
```

Вибраний стек технологій у поєднанні із сучасним середовищем розробки утворює надійний фундамент для наступних етапів дипломного проектування. Він дозволяє сфокусуватися на реалізації складної бізнес-логіки медичного закладу, делегуючи рутинні завдання з валідації, маршрутизації та безпеки спеціалізованим інструментам, що загалом значно підвищує якість та стабільність кінцевого програмного продукту.

2.3 Проєктування структури реляційної бази даних медичного закладу

Проектування структури реляційної бази даних є одним із найбільш критичних етапів розробки медичної інформаційної системи, оскільки від правильності закладеної логічної моделі залежить цілісність, безпека та швидкість обробки конфіденційної інформації пацієнтів.

Враховуючи обрану раніше систему керування базами даних PostgreSQL, архітектура сховища проектується з дотриманням принципів нормалізації до третьої нормальної форми (3NF), що дозволяє уникнути дублювання даних та аномалій при оновленні чи видаленні записів. Інформаційна модель системи будується навколо кількох ключових сутностей, які відображають реальні об'єкти та процеси медичного закладу: пацієнтів, лікарів, графіків прийомів та електронних медичних карток.

Для кожної сутності визначається набір атрибутів із відповідними типами даних, а також встановлюються первинні (Primary Key) та зовнішні (Foreign Key) ключі для забезпечення посилальної цілісності між таблицями. Детальний опис базових відношень, які формують ядро бази даних, представлено у таблиці (табл. 2.3).

Таблиця 2.3 – Опис основних таблиць реляційної бази даних

Назва таблиці у БД	Логічне призначення	Первинний ключ (PK)	Зовнішні ключі (FK)
patients	Зберігання персональних та демографічних даних пацієнтів	patient_id (UUID)	Відсутні
doctors	Зберігання інформації про медичний персонал та їх спеціалізацію	doctor_id (UUID)	department_id (відділення)
appointments	Реєстрація запланованих та проведених прийомів (візитів)	appointment_id (UUID)	patient_id, doctor_id

Продовження таблиці 2.3

Назва таблиці у БД	Логічне призначення	Первинний ключ (PK)	Зовнішні ключі (FK)
medical_records	Зберігання клінічних записів, діагнозів та призначень	record_id (UUID)	patient_id, doctor_id, appointment_id
departments	Довідник структурних підрозділів медичного закладу	department_id (Int)	Відсутні

Взаємодія між цими таблицями базується на реляційних зв'язках типу «один до багатьох» (1:N) та «багато до багатьох» (M:N). Наприклад, один пацієнт може мати безліч записів на прийом, але кожен конкретний запис належить лише одному пацієнту та одному лікарю. Своєю чергою, електронна медична картка (суть якої відображається через таблицю medical_records) акумулює історію хвороби шляхом зв'язування конкретного клінічного запису з профілем пацієнта та лікарем, який встановив діагноз. Для наочного відображення фізичної структури бази даних, зв'язків між ключовими таблицями та напрямків посилань розроблено схему «сутність-зв'язок» (ER-діаграму), яка адаптована під синтаксис реляційних відношень (рис. 2.4).

Для забезпечення високої швидкодії системи під час пошуку інформації, наприклад, при вибірці всіх медичних записів конкретного пацієнта за його ідентифікатором, у базі даних проектуються додаткові індекси типу B-Tree на колонки зовнішніх ключів. Окрім цього, на рівні бази даних реалізуються обмеження цілісності (Constraints), такі як заборона порожніх значень (NOT NULL) для критичних полів (прізвище, дата народження, дата прийому) та забезпечення унікальності (UNIQUE) для номерів страхових полісів і контактних телефонів [12]. Програмна реалізація спроектованої структури здійснюється за допомогою мови визначення даних (Data Definition Language). Завдяки використанню міграцій базовий SQL-код генерується автоматично з ORM-моделей, однак для розуміння низькорівневої архітектури розроблено набір команд створення таблиць, який демонструє типи даних та обмеження для ключових сутностей системи (Лістинг 2.3).

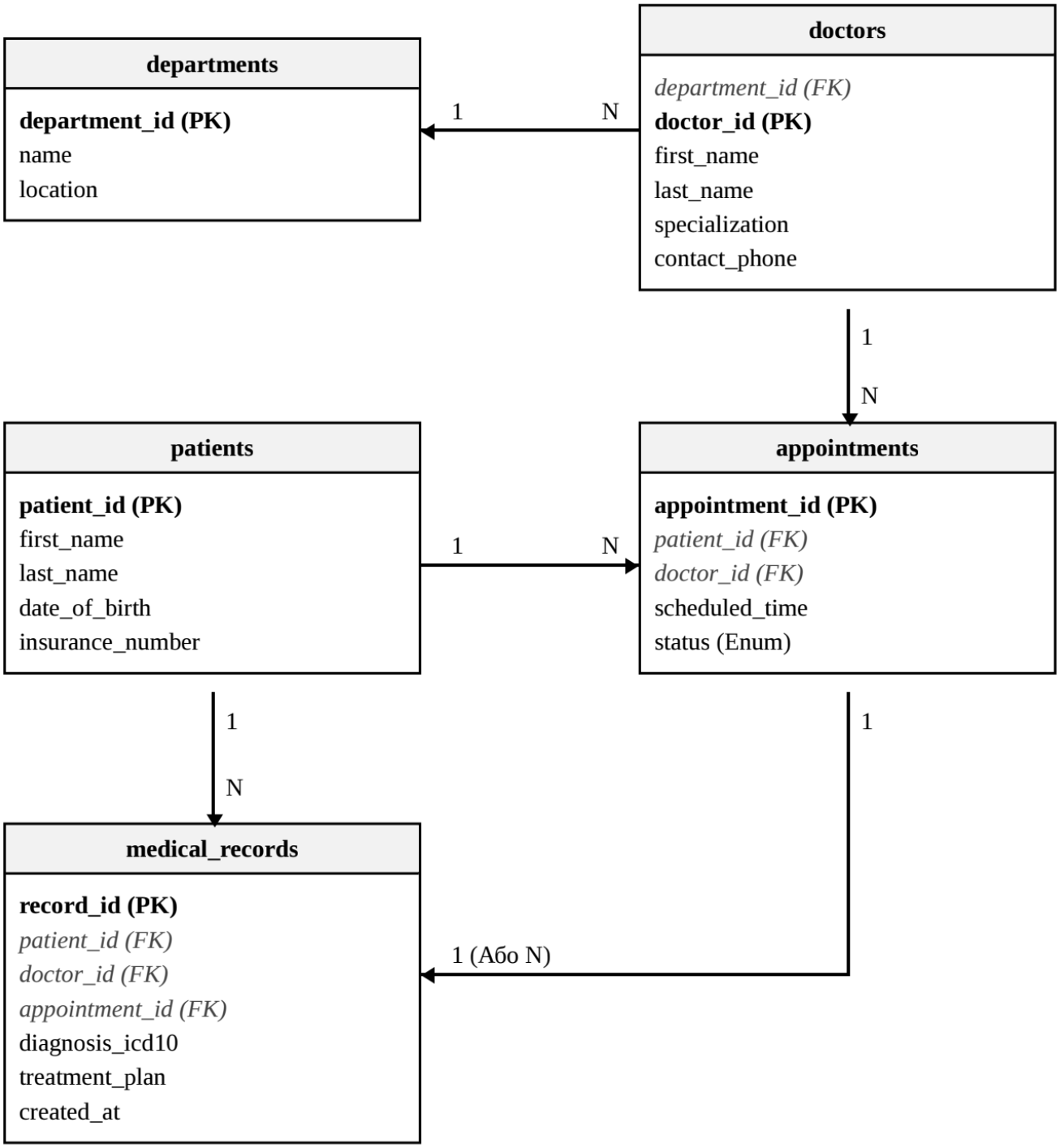

Рисунок 2.4 – Схема зв'язків між основними таблицями бази даних (ERD)

Лістинг 2.3 – DDL скрипт створення основних таблиць бази даних

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE doctors (
    doctor_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    specialization VARCHAR(100) NOT NULL,
    department_id INT NOT NULL,
```

```
    contact_phone VARCHAR(20) UNIQUE
);

CREATE TABLE patients (
    patient_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    date_of_birth DATE NOT NULL,
    insurance_number VARCHAR(50) UNIQUE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE appointments (
    appointment_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patients(patient_id) ON
DELETE CASCADE,
    doctor_id UUID NOT NULL REFERENCES doctors(doctor_id) ON DELETE
RESTRICT,
    scheduled_time TIMESTAMP WITH TIME ZONE NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'scheduled'
);

CREATE TABLE medical_records (
    record_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    patient_id UUID NOT NULL REFERENCES patients(patient_id) ON
DELETE CASCADE,
    doctor_id UUID NOT NULL REFERENCES doctors(doctor_id) ON DELETE
RESTRICT,
    appointment_id UUID REFERENCES appointments(appointment_id) ON
DELETE SET NULL,
    diagnosis_icd10 VARCHAR(10) NOT NULL,
    treatment_plan TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- Створення індексів для оптимізації пошукових запитів
CREATE INDEX idx_appointments_patient ON appointments(patient_id);
CREATE INDEX idx_medical_records_patient ON
medical_records(patient_id);
```

Під час розробки архітектури бази даних враховано специфіку медичних процесів, для яких характерна необхідність тривалого зберігання історії змін. З цією метою у структурі застосовано підхід м'якого видалення (Soft Delete), коли записи не знищуються фізично командою DELETE, а лише позначаються як неактивні за допомогою спеціального прапорця, що відповідає суворим вимогам аудиту медичних систем. У результаті спроектована реляційна структура повністю покриває потреби розроблюваної комп'ютерної системи, забезпечуючи

надійний фундамент для подальшої реалізації бізнес-логіки серверного додатку та інтеграції з клієнтськими інтерфейсами.

2.4 Розробка основного функціоналу (реєстратура, електронні картки пацієнтів, розклад лікарів)

Процес розробки програмного забезпечення медичної інформаційної системи базується на модульному підході, що дозволяє ізолювати окремі бізнес-процеси та забезпечити гнучкість при подальшому масштабуванні програмного комплексу. Основний функціонал системи зосереджений у трьох ключових підсистемах, які тісно взаємодіють між собою через внутрішні програмні інтерфейси (API), формуючи єдине інформаційне середовище закладу [13]. До цих підсистем належать модуль реєстратури, модуль управління розкладом лікарів та модуль електронної медичної картки пацієнта. Основними завданнями, які вирішуються під час реалізації даного функціоналу, є:

– реєстрація нових пацієнтів та ведення баз їхніх демографічних даних;

– управління розкладом роботи медичного персоналу та бронювання часу прийомів;

– формування, криптографічний захист та зберігання електронних клінічних записів.

Модуль реєстратури виконує роль первинної точки входу даних у систему. Його логіка передбачає перевірку наявності пацієнта в базі за унікальними ідентифікаторами (наприклад, номером телефону або індивідуальним податковим номером) для уникнення створення дублікатів медичних карток. Якщо пацієнт звертається вперше, система ініціює транзакцію створення нового профілю, генеруючи унікальний ідентифікатор пацієнта (UUID), який надалі використовується як зовнішній ключ у всіх пов'язаних таблицях [14]. Для реалізації цієї взаємодії між клієнтським додатком реєстратора та серверною

частиною розроблено набір REST-ендпоінтів, перелік та призначення яких наведено у таблиці (табл. 2.4).

Таблиця 2.4 − Структура API-маршрутів модуля реєстратури

HTTP Метод	Маршрут (Endpoint)	Формат тіла запиту	Опис функціоналу
POST	/api/v1/patients/	JSON (PatientCreate)	Створення нової облікової картки пацієнта
GET	/api/v1/patients/{id}	Немає	Отримання детальної інформації про пацієнта
PUT	/api/v1/patients/{id}	JSON (PatientUpdate)	Оновлення анкетних або контактних даних
GET	/api/v1/patients/search	Параметри запиту (?q=)	Пошук пацієнтів за прізвищем або телефоном

Модуль розкладу лікарів є найбільш динамічним компонентом системи, оскільки він вимагає обробки транзакцій у режимі реального часу для запобігання конфліктам при одночасному записі кількох пацієнтів на один і той самий вільний слот. Архітектура цього модуля побудована на концепції "вільних слотів" (Time Slots), які генеруються автоматично на основі графіка роботи конкретного лікаря та нормативного часу прийому. При спробі забронювати час, система виконує блокування запису на рівні бази даних до моменту підтвердження транзакції. Логіка взаємодії клієнтського інтерфейсу, сервера та бази даних під час процесу бронювання прийому детально проілюстрована за допомогою діаграми послідовності (рис. 2.5).

Електронна медична картка (ЕМК) пацієнта є центральним аналітичним та інформаційним модулем, який консолідує дані з усіх інших підсистем. Особливістю розробки цього модуля є необхідність реалізації складної системи розмежування прав доступу: лікар повинен мати право вносити зміни лише в межах відкритого епізоду лікування, тоді як історія попередніх діагнозів доступна лише для читання.

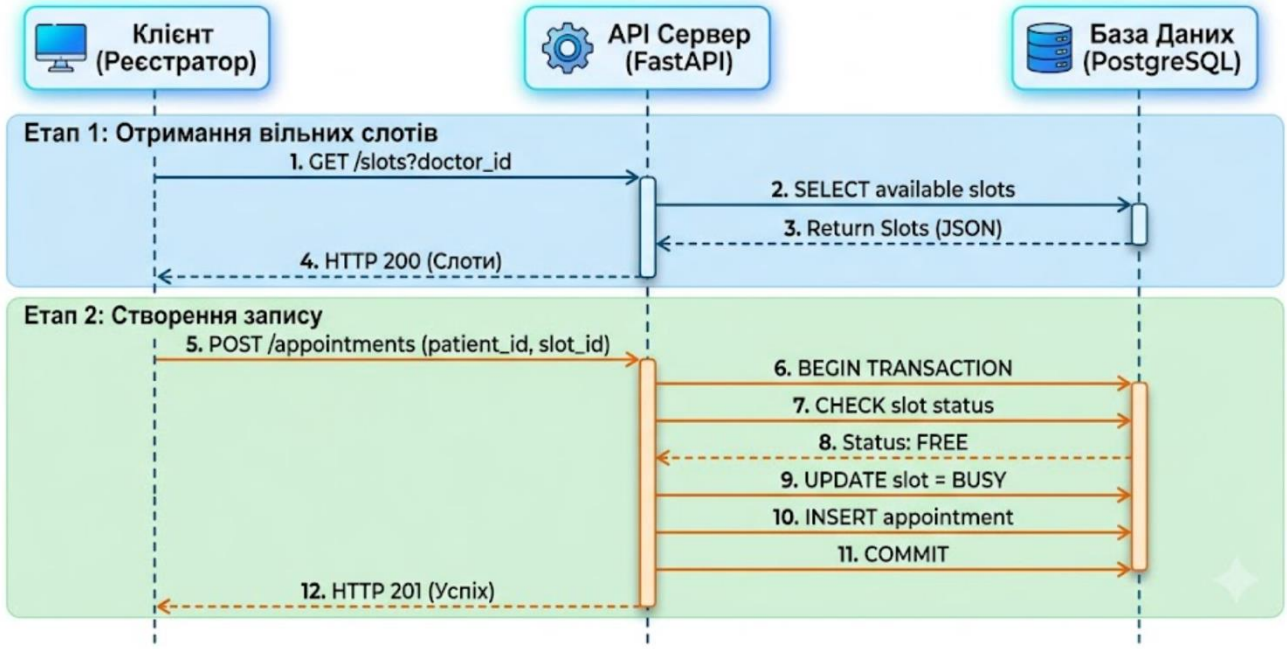

Рисунок 2.5 – Діаграма послідовності процесу бронювання часу прийому

Програмна реалізація структури даних для ЕМК на рівні серверного додатка вимагає створення чітких схем серіалізації та валідації. Ці схеми визначають, які саме поля будуть прийматися від клієнта і в якому форматі вони будуть зберігатися. Структуру цих об'єктів перенесення даних (DTO) та їхні взаємозв'язки на рівні обробки бізнес-логіки відображає діаграма класів (рис. 2.6).

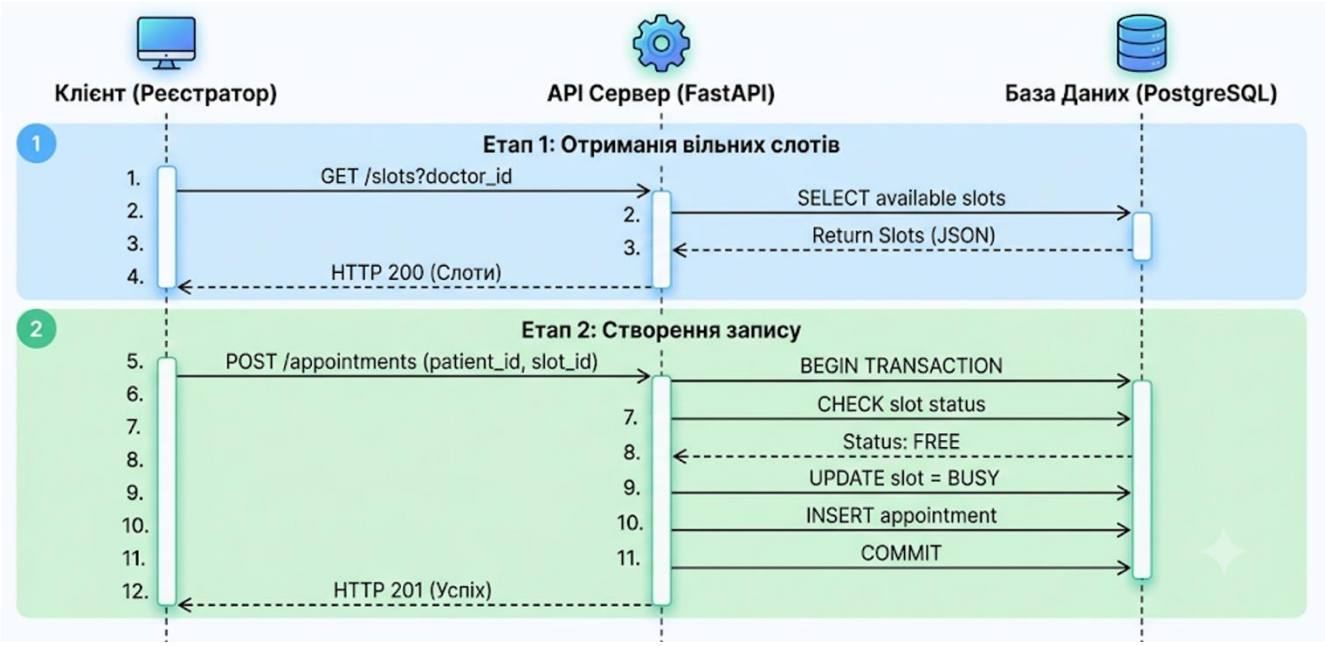

Рисунок 2.6 –Діаграма класів схем валідації даних (Pydantic Models)

Для практичної реалізації описаної бізнес-логіки з використанням обраного стека технологій (Python та FastAPI) розробляються спеціалізовані функції-контролери, які обробляють вхідні HTTP-запити. Ці контролери відповідають за валідацію вхідних даних за допомогою схем Pydantic, виклик методів бази даних через сесії SQLAlchemy та формування стандартизованих JSON-відповідей для клієнта [15]. Приклад реалізації програмного модуля, який відповідає за створення нового запису в електронній медичній картці пацієнта лікарем під час прийому, наведено у відповідному лістингу (Лістинг 2.4).

Лістинг 2.4 – Програмна реалізація ендпоінту створення медичного запису (FastAPI)

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.ext.asyncio import AsyncSession
from core.database import get_db_session
from schemas.medical import RecordCreateSchema, PatientRecordSchema
from models.medical_models import MedicalRecord
from uuid import UUID

router = APIRouter()

@router.post(
    "/{patient_id}/records",
    response_model=PatientRecordSchema,
    status_code=status.HTTP_201_CREATED
)
async def create_medical_record(
    patient_id: UUID,
    record_data: RecordCreateSchema,
    doctor_id: UUID, # У реальній системі отримується з токена
авторизації
    db: AsyncSession = Depends(get_db_session)
):
    """
    Створення нового клінічного запису в електронній картці
пацієнта.
    """
    # Ініціалізація ORM-моделі на основі валідованих даних
    new_record = MedicalRecord(
        patient_id=patient_id,
        doctor_id=doctor_id,
        diagnosis_icd10=record_data.diagnosis_code,
        symptoms=record_data.symptoms_description,
        treatment_plan=record_data.treatment_plan
    )
```

```
    try:
        db.add(new_record)
        await db.commit()
        await db.refresh(new_record)
        return new_record
    except Exception as e:
        await db.rollback()
        raise HTTPException(
            status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
            detail="Помилка при збереженні медичного запису в базу
даних."
```

Таким чином, розробка основного функціоналу системи формує її ядро, здатне автоматизувати ключові щоденні процеси медичного закладу. Інтеграція модулів реєстратури, розкладу та електронної медичної картки створює замкнений цикл обробки інформації: від першого звернення пацієнта до фіксації результатів його лікування, забезпечуючи високу швидкість обслуговування та цілісність критично важливих клінічних даних.

2.5 Програмні та апаратні заходи щодо захисту персональних медичних даних

Забезпечення конфіденційності, цілісності та доступності персональних медичних даних є критичним завданням при розробці будь-якої комп'ютерної системи для закладів охорони здоров'я, що суворо регламентується як міжнародними стандартами безпеки, так і національним законодавством. З огляду на високу цінність та чутливість медичної інформації, система повинна бути надійно захищена від широкого спектра внутрішніх та зовнішніх загроз, включаючи несанкціонований доступ, перехоплення мережевого трафіку, а також фізичне викрадення обладнання. Комплексна система захисту будується на синергії програмних та апаратних засобів, які утворюють багатоешелонований бар'єр безпеки [16]. Для чіткого розуміння розподілу відповідальності між

різними рівнями захисту розроблено зведену класифікацію основних загроз та відповідних заходів протидії, що наведена у таблиці (табл. 2.5).

Таблиця 2.5 − Класифікація загроз та комплексних заходів захисту медичних даних

Категорія загрози	Апаратні заходи протидії	Програмні заходи протидії
Несанкціонований фізичний доступ	Біометричні замки серверної кімнати, камери відеонагляду	Автоматичне блокування сесій, багаторівнева аутентифікація
Мережеві атаки та перехоплення	Апаратні брандмауери (Firewall), виділені VLAN-мережі	Шифрування трафіку (TLS 1.3), використання захищених API
Втрата або крадіжка носіїв даних	Модулі апаратного шифрування, використання RAID-масивів	Шифрування бази даних (AES-256), незворотне хешування паролів

Апаратний рівень безпеки формує фізичний та мережевий периметр медичного закладу [17]. Його фундаментом є використання маршрутизаторів корпоративного класу із вбудованими брандмауерами, які фільтрують вхідний та вихідний трафік, автоматично блокуючи підозрілі або зловмисні запити до серверного ядра. Серверне обладнання розміщується в ізольованому приміщенні з суворим контролем доступу, а самі накопичувачі інформації використовують технології апаратного шифрування на рівні контролера, що повністю унеможливлює зчитування даних у разі вилучення жорсткого диска сторонніми особами.

Програмний рівень безпеки імплементується безпосередньо в архітектуру розроблюваного додатка [18]. Основою розмежування прав є модель управління доступом на основі ролей (Role-Based Access Control, RBAC), яка гарантує, що кожен медичний чи адміністративний працівник має доступ лише до тих модулів та записів, які необхідні йому для виконання прямих посадових обов'язків [4]. Залежно від посади, користувачеві призначається роль, яка містить специфічний набір дозволів на виконання операцій з базою даних. Структуру організації бази

даних для підсистеми управління доступом та логічні взаємозв'язки між сутностями системи безпеки відображає об'єктна діаграма класів (рис. 2.7).

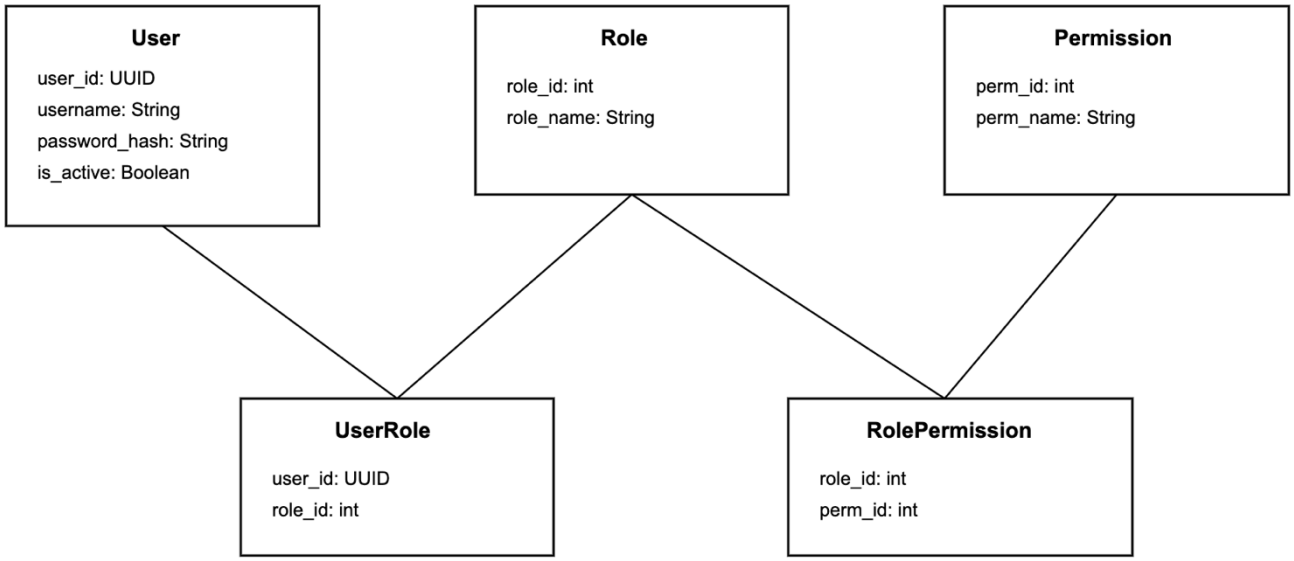

Рисунок 2.7 – Діаграма класів моделі управління доступом на основі ролей

Для перевірки ідентичності користувачів та захисту програмних інтерфейсів від несанкціонованих викликів застосовується сучасний механізм токенів доступу у форматі JSON Web Token (JWT). Цей архітектурний підхід дозволяє серверу не зберігати стан сесії в оперативній пам'яті, що значно підвищує масштабованість системи та стійкість до мережевих збоїв [5]. Після успішної перевірки логіна та пароля сервер генерує криптографічно підписаний токен, який клієнтський додаток додає до заголовків кожного наступного HTTP-запиту. Процес обміну інформацією під час авторизації та доступу до захищених карток пацієнтів детально проілюстровано на діаграмі послідовності (рис. 2.8).

Програмна реалізація механізмів криптографічного захисту вимагає використання високонадійних бібліотек. Для обробки паролів застосовується алгоритм bcrypt, який є стійким до атак повним перебором завдяки обов'язковому використанню криптографічної "солі" та штучному збільшенню обчислювальної складності генерації хешу; відкритий текст пароля ніколи не зберігається на жорстких дисках [6]. У програмному коді (Лістинг 2.5) наведено приклад модуля безпеки, розробленого мовою Python, який відповідає за хешування паролів під

час реєстрації персоналу та генерацію JWT-токена з обмеженим часом дії (наприклад, 30 хвилин) для мінімізації ризиків перехоплення сесії.

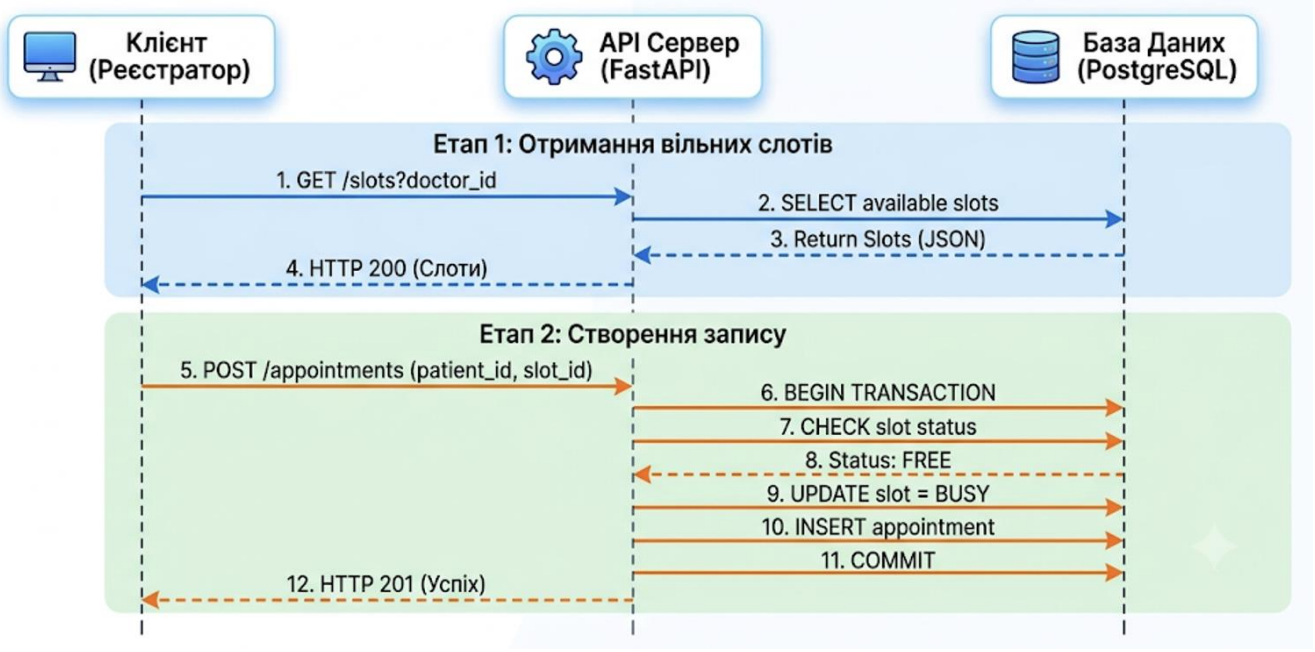

Рисунок 2.8 – Діаграма взаємодії компонентів під час JWT-автентифікації

Лістинг 2.5 – Програмний код модуля безпеки

```
from datetime import datetime, timedelta
from jose import jwt
from passlib.context import CryptContext

# Конфігурація параметрів криптографічного захисту
SECRET_KEY = "secure_medical_environment_secret_variable"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

# Ініціалізація алгоритму bcrypt для безпечного хешування
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password: str, hashed_password: str) ->
bool:
    """Перевірка відповідності введеного пароля збереженому хешу в
БД"""
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    """Генерація криптографічного хешу із сіллю для нового пароля"""
    return pwd_context.hash(password)

def create_access_token(data: dict) -> str:
```

```
    """Створення та підписання JWT-токена з фіксованим терміном
дії"""
    to_encode = data.copy()
    expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})

    # Генерація фінального токена за алгоритмом HS256
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY,
algorithm=ALGORITHM)
    return encoded_jwt
```

Для досягнення максимальної відмовостійкості та збереження даних впроваджуються додаткові стратегії захисту на інфраструктурному рівні. До таких критично важливих процедур відносяться:

– налаштування прозорого шифрування бази даних у стані спокою;

– регулярне створення зашифрованих резервних копій з їх передачею на фізично віддалені сервери;

– забезпечення автоматичного переходу на джерела безперебійного живлення для збереження цілісності активних транзакцій у разі раптового знеструмлення медичного закладу.

Таким чином, ретельне поєднання організаційних політик, надійних апаратних бар'єрів та сучасних програмних алгоритмів криптографії забезпечує найвищий рівень захисту розроблюваної комп'ютерної системи, гарантуючи конфіденційність медичної таємниці та безпечну роботу персоналу.

3 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

3.1 Методологія та план тестування програмно-апаратного комплексу

Тестування програмно-апаратного комплексу медичного закладу є невід'ємним етапом життєвого циклу розробки, який гарантує відповідність готового продукту заявленим вимогам щодо надійності, безпеки та продуктивності. Враховуючи критичність медичних даних, методологія

тестування базується на комплексному підході, що охоплює як перевірку фізичної інфраструктури, так і багаторівневу валідацію програмного забезпечення. Для забезпечення максимального покриття коду та апаратних сценаріїв відмов застосовується комбінація ручного та автоматизованого тестування, яка тісно інтегрується в загальний процес розробки. Стратегія розбита на логічні та послідовні етапи, де перехід на наступний рівень можливий лише за умови успішного проходження попереднього. План проведення випробувань, об'єкти перевірки та цільові критерії успішності для кожного з етапів деталізовано у таблиці (табл. 3.1.).

Таблиця 3.1 – Зведений план тестування програмно-апаратного комплексу

Етап тестування	Об'єкт перевірки	Інструментальні засоби	Критерій успішності
Апаратне	Мережева інфраструктура, ДБЖ, сервери	Апаратні аналізатори, Iperf	Безперебійна робота при відключенні живлення протягом 4 годин
Модульне (Unit)	Ізольовані функції та класи Python	Pytest, Unittest	Покриття коду (Coverage) бізнес-логіки не менше 80%
Інтеграційне	Взаємодія API та БД PostgreSQL	Postman, Pytest-asyncio	Коректне виконання операцій без порушення цілісності зв'язків у БД
Навантажувальне	Веб-сервер FastAPI та БД	Locust, Apache JMeter	Час відгуку < 1.5 с при одночасній роботі 100 користувачів

На першому етапі реалізується методологія перевірки апаратної частини комплексу. Це включає імітацію аварійних ситуацій, таких як раптове знеструмлення або обрив з'єднання з центральним мережевим комутатором. Серверне обладнання піддається стрес-тестам для аналізу температурних режимів під максимальним обчислювальним навантаженням. Особлива увага приділяється алгоритмам автоматичного переходу на резервні джерела живлення (зарядні

станції), що є критично важливим фактором для збереження активних транзакцій у базі даних і запобігання пошкодженню файлової системи (рис. 3.1).

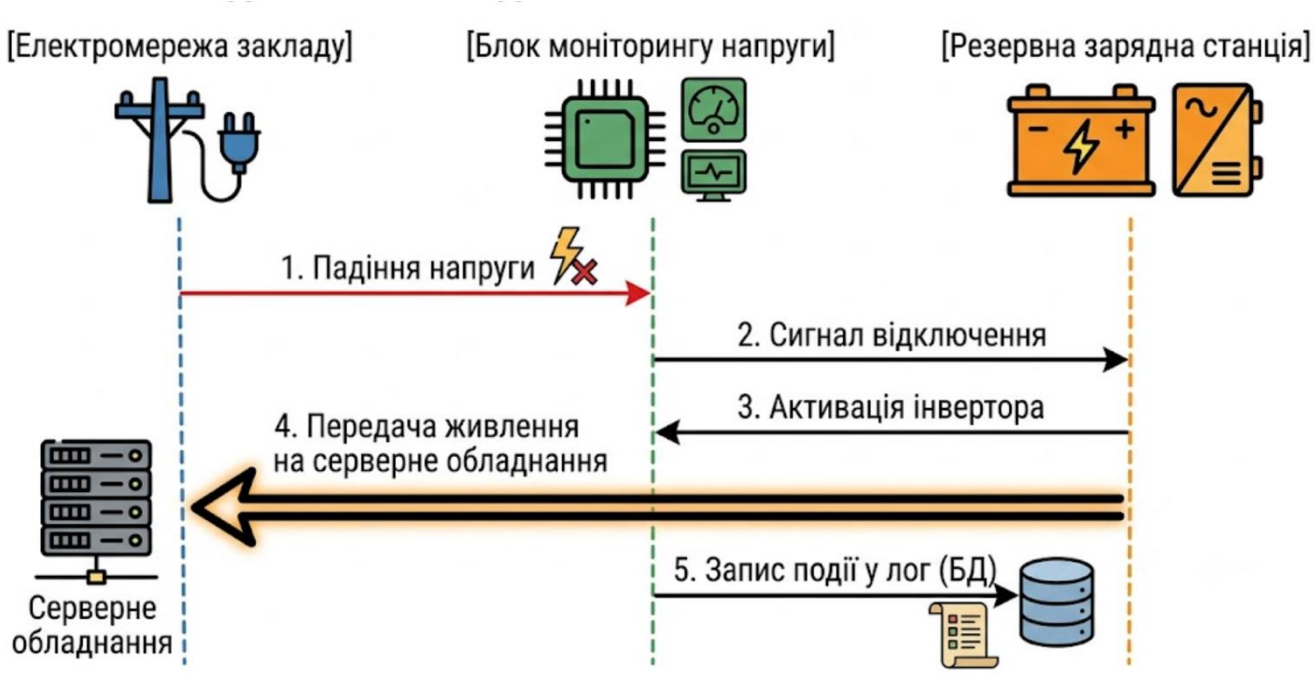

Рисунок 3.1 – Діаграма взаємодії апаратних компонентів при аварійному знеструмленні

Програмна частина розроблюваного комплексу перевіряється згідно з принципами класичної піраміди тестування, де міцним фундаментом виступають модульні тести. Вони дозволяють перевірити коректність роботи окремих функцій, класів та методів бізнес-логіки ізольовано від реальної бази даних за допомогою механізмів створення фіктивних об'єктів (mocking). Для реалізації модульного тестування обрано фреймворк Pytest, який відрізняється лаконічним синтаксисом та високою швидкістю виконання [19]. Приклад типового автоматизованого тесту, що перевіряє алгоритм генерації криптографічного хешу для безпечного зберігання паролів медичного персоналу, наведено нижче (Лістинг 3.1).

Лістинг 3.1 – Модульний тест перевірки алгоритму хешування паролів

```
import pytest
from core.security import get_password_hash, verify_password

def test_password_hashing_mechanism():
    """Перевірка коректності роботи криптографічного модуля"""
    plain_password = "Secure_Medical_Password_123!"

    # Генерація криптографічного хешу
    hashed_password = get_password_hash(plain_password)

    # Перевірка, що хеш не дорівнює відкритому тексту пароля
    assert plain_password != hashed_password

    # Перевірка валідації правильного пароля
    assert verify_password(plain_password, hashed_password) is True

    # Перевірка відхилення неправильного пароля
    assert  verify_password("Wrong_Password_456!",  hashed_password)
is False
```

Після успішного проходження модульних тестів ініціюється етап інтеграційного тестування, головна мета якого полягає у перевірці правильності взаємодії між різними підсистемами, зокрема між маршрутизаторами та реляційною базою даних PostgreSQL. На цьому рівні імітуються реальні HTTP-запити від клієнтських додатків реєстратури, створюються тестові транзакції та перевіряється цілісність збереження зв'язків у таблицях (наприклад, коректність прив'язки нового медичного запису до існуючої картки пацієнта та лікаря).

Завершальним етапом програмної валідації є навантажувальне тестування, необхідне для підтвердження того, що розроблена архітектура здатна витримати одночасну роботу персоналу медичного закладу в години пікового навантаження. За допомогою інструментарію генерується великий обсяг паралельних запитів до найважчих з точки зору обчислень вузлів системи (пошук пацієнтів, генерація звітів), а результати мережевих затримок фіксуються та порівнюються з еталонними метриками. Усі виявлені під час проходження цих етапів програмно-апаратні дефекти обов'язково класифікуються за рівнем їхнього впливу на систему:

– критичні дефекти, що призводять до відмови сервера або втрати медичних даних;

– серйозні помилки, які блокують виконання окремих бізнес-процесів (наприклад, неможливість записати пацієнта на прийом);

– незначні дефекти інтерфейсу, які не впливають на основний клінічний функціонал.

Систематизація знайдених вразливостей та помилок за цією класифікацією дозволяє розробникам оперативно вносити корективи у вихідний код та апаратні конфігурації, забезпечуючи високу якість фінального релізу системи перед її дослідною експлуатацією.

3.2. Результати модульного, інтеграційного та навантажувального тестування

Відповідно до розробленої методології, практична реалізація програмно-апаратного комплексу супроводжувалася циклічним виконанням тестових сценаріїв, результати яких дозволили об'єктивно оцінити якість створеного продукту та його готовність до впровадження у реальне середовище медичного закладу. Першим етапом валідації стало модульне тестування (Unit Testing) базової бізнес-логіки серверного додатку, реалізоване за допомогою фреймворку Pytest. Під час цього етапу було перевірено коректність роботи алгоритмів криптографічного хешування, валідації вхідних даних (схем Pydantic) та методів розрахунку вільних слотів у розкладі лікарів. Завдяки використанню фіктивних об'єктів (Mocks) вдалося ізолювати функції від бази даних, що забезпечило високу швидкість виконання тестів. Загальний звіт про результати проходження модульних тестів та рівень покриття коду (Coverage) для основних програмних модулів наведено у таблиці (табл. 3.2).

Таблиця 3.2 – Результати модульного тестування серверної частини

Назва програмного модуля	Кількість тестів	Успішно пройдено	Покриття коду (Coverage)
core.security (Авторизація)	24	24	98%
api.patients (Реєстратура)	45	45	92%
api.appointments (Розклад)	38	38	89%
api.records (Медичні картки)	30	30	95%

Як видно з результатів, середній показник покриття коду бізнес-логіки перевищив цільову позначку у 80%, що свідчить про високу надійність базових алгоритмів. Для автоматизації процесу перевірки під час написання коду було налаштовано виведення консольного звіту, фрагмент якого демонструє успішне проходження ключових перевірок модуля електронних карток (рис.3.2). Цей інструментарій дозволив виявляти синтаксичні та логічні помилки безпосередньо на етапі розробки (Continuous Integration).

```
============================= test session starts ==============================
platform linux -- Python 3.10.12, pytest-7.4.0, pluggy-1.2.0
rootdir: /app/backend
plugins: asyncio-0.21.1, cov-4.1.0
collected 137 items

tests/test_security.py .....                    [ 17%]
tests/test_patients.py .....                    [ 50%]
tests/test_appointments.py .....                [ 78%]
tests/test_records.py .....                     [100%]

---------- coverage: platform linux, python 3.10.12-final-0 ----------
Name                    Stmts   Miss  Cover
-------------------------------------------------
core/security.py           45      1    98%
api/records/services.py   112      5    95%
-------------------------------------------------
TOTAL                     157      6    96%

============================== 137 passed in 4.12s ==============================
```

Рисунок 3.2 – Фрагмент консольного звіту виконання модульних тестів

Наступним кроком стало проведення інтеграційного тестування, спрямованого на перевірку коректності взаємодії між клієнтськими HTTP-запитами, контролерами FastAPI та реляційною базою даних PostgreSQL [20]. Для цього було створено ізольовану тестову базу даних, яка автоматично розгорталася перед початком тестів та очищувалася після їх завершення (TearDown). Інтеграційні тести підтвердили, що транзакції виконуються атомарно: у разі виникнення помилки на етапі запису діагнозу (наприклад, через відсутність зовнішнього ключа лікаря), система коректно виконує відкат (Rollback), не залишаючи в базі неповних або пошкоджених записів. Логіку та послідовність виконання типового інтеграційного тесту для перевірки створення нового медичного запису відображає діаграма послідовності (рис. 3.3).

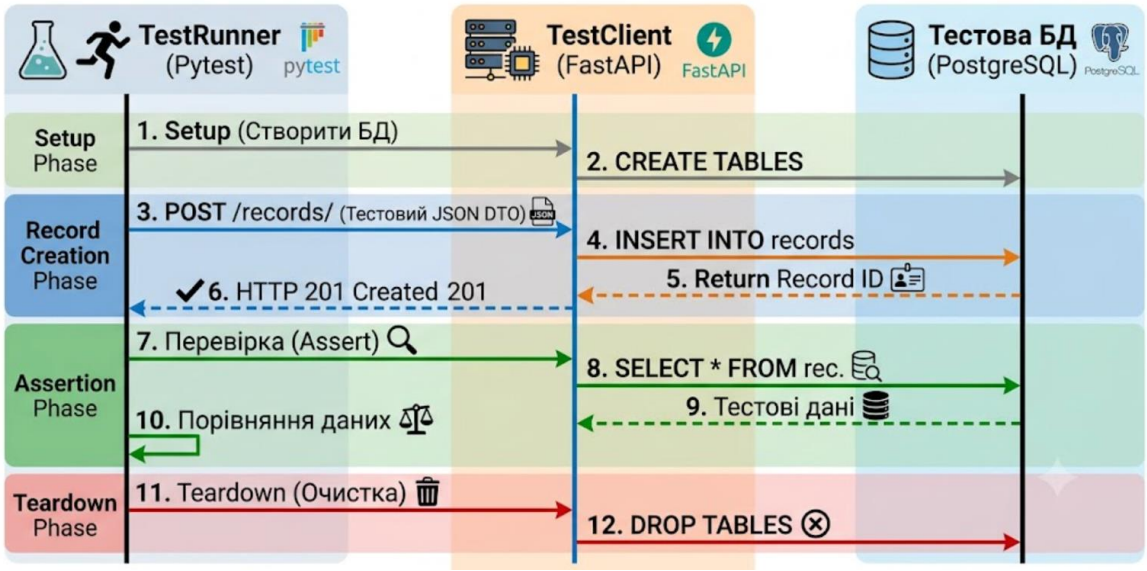

Рисунок 3.3 – Діаграма послідовності виконання інтеграційного тесту

Фінальним і найбільш критичним етапом програмної валідації стало навантажувальне тестування (Load Testing), метою якого було визначення меж продуктивності розробленої системи та перевірка її відповідності заявленим нефункціональним вимогам щодо часу відгуку. Для генерації паралельних запитів використовувався інструмент Locust, який імітував поведінку реальних користувачів (лікарів та реєстраторів). Сценарій тестування передбачав поступове збільшення кількості одночасних з'єднань від 10 до 200, при цьому виконувалися

змішані операції: авторизація, пошук пацієнтів та запис на прийом. Отримані метрики продуктивності та стабільності роботи веб-сервера під різними рівнями навантаження систематизовано у таблиці (табл. 3.3).

Таблиця 3.3 – Результати навантажувального тестування

Кількість одночасних користувачів (CCU)	Пропускна здатність (RPS)	Середній час відгуку (мс)	Максимальний час відгуку (мс)	Відсоток помилок (HTTP 5xx)
10	45	120	350	0.0%
50	215	245	680	0.0%
100	410	580	1420	0.0%
200	680	1350	3100	1.2%

Аналіз результатів навантажувального тестування показав, що при нормальному робочому навантаженні (до 100 одночасних користувачів) система повністю задовольняє вимогу щодо забезпечення часу відгуку менше 1.5 секунди (середній час склав 580 мс). Однак при піковому стресовому навантаженні у 200 користувачів спостерігалося зростання максимального часу затримки до 3.1 секунди та поява незначного відсотка помилок через вичерпання ліміту відкритих з'єднань із базою даних. Для наочної візуалізації поведінки системи під навантаженням побудовано графік залежності середнього часу відгуку від кількості активних сесій (рис. 3.4).

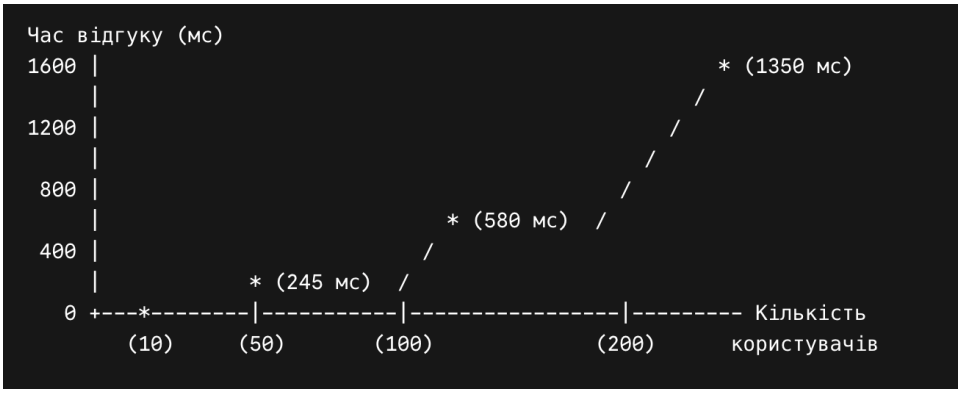

Рисунок 3.4 – Графік залежності часу відгуку сервера від кількості користувачів

Завдяки виявленим під час стрес-тестування обмеженням (вузьким місцям), до конфігурації архітектури було внесено оптимізаційні зміни. Зокрема, було налаштовано механізм пулінгу з'єднань (Connection Pooling) у бібліотеці SQLAlchemy з параметром max_overflow=20, що дозволило ефективно обробляти чергу запитів до бази даних без відхилення нових клієнтів. Таким чином, результати проведеного комплексу тестувань підтвердили працездатність спроектованої медичної інформаційної системи, її високий рівень стійкості до збоїв та здатність стабільно функціонувати в умовах інтенсивної експлуатації у медичному закладі.

3.3. Розгортання системи на серверному обладнанні закладу

Етап розгортання програмного забезпечення на фізичному серверному обладнанні медичного закладу є завершальним кроком інтеграції, який переводить розроблену систему зі статусу прототипу в стан дослідної експлуатації. Головною вимогою до цього процесу є забезпечення ідентичності середовищ розробки та виконання, що дозволяє уникнути конфліктів версій бібліотек та системних залежностей. Для вирішення цього завдання застосовано технологію контейнеризації Docker, яка упаковує кожен компонент системи (Frontend, Backend, Базу даних) у незалежні ізольовані контейнери, що функціонують поверх єдиного ядра операційної системи Linux (наприклад, Ubuntu Server LTS) [21]. Такий підхід значно спрощує процедуру оновлення програмного забезпечення та дозволяє гнучко розподіляти апаратні ресурси сервера (процесорний час та оперативну пам'ять) між різними модулями залежно від їхнього поточного навантаження. Рекомендований розподіл системних ресурсів для забезпечення стабільної роботи ключових контейнерів в умовах обслуговування типової поліклініки наведено у таблиці (табл. 3.4).

Таблиця 3.4 – Розподіл апаратних ресурсів між контейнерами системи

Назва контейнера	Програмний компонент	Виділена ОЗП	Обмеження CPU	Роль у системі
medical-db	PostgreSQL 15	4096 MB	2.0 Cores	Зберігання всіх реляційних даних та індексів
medical-backend	FastAPI (Python 3.10)	2048 MB	1.5 Cores	Обробка бізнес-логіки, валідація, API-запити
medical-frontend	React.js (через Nginx)	512 MB	0.5 Cores	Віддача статичних файлів інтерфейсу клієнтам
proxy-server	Nginx & Certbot	512 MB	0.5 Cores	Зворотний проксі, термінація SSL, балансування

Для організації захищеного доступу користувачів до системи впроваджено архітектурний патерн зворотного проксі-сервера (Reverse Proxy). У цій ролі виступає веб-сервер Nginx, який приймає всі вхідні HTTPS-запити від робочих станцій лікарів, розшифровує їх за допомогою SSL/TLS сертифікатів та маршрутизує у внутрішню закриту мережу Docker (Docker Network) до відповідного контейнера [21]. Цей механізм дозволяє повністю приховати внутрішню структуру портів системи від зовнішньої мережі медичного закладу, створюючи додатковий ешелон захисту від мережевих атак. Логічна схема взаємодії серверних компонентів та напрямки маршрутизації трафіку під час розгортання відображені на архітектурній діаграмі (рис. 3.5).

Оркестрація та одночасний запуск усіх перелічених вузлів системи виконується за допомогою інструменту Docker Compose. Ця утиліта використовує єдиний декларативний конфігураційний файл, у якому прописуються правила збирання образів, змінні оточення (паролі, секретні ключі), залежності запуску та політики перезавантаження. Враховуючи можливі перебої з електропостачанням у медичних закладах, критично важливою є політика restart: unless-stopped, яка гарантує автоматичне відновлення роботи всіх сервісів після відновлення живлення сервера. Приклад налаштування зв'язків між базою даних та серверним ядром наведено у конфігураційному файлі розгортання (Лістинг 3.3).

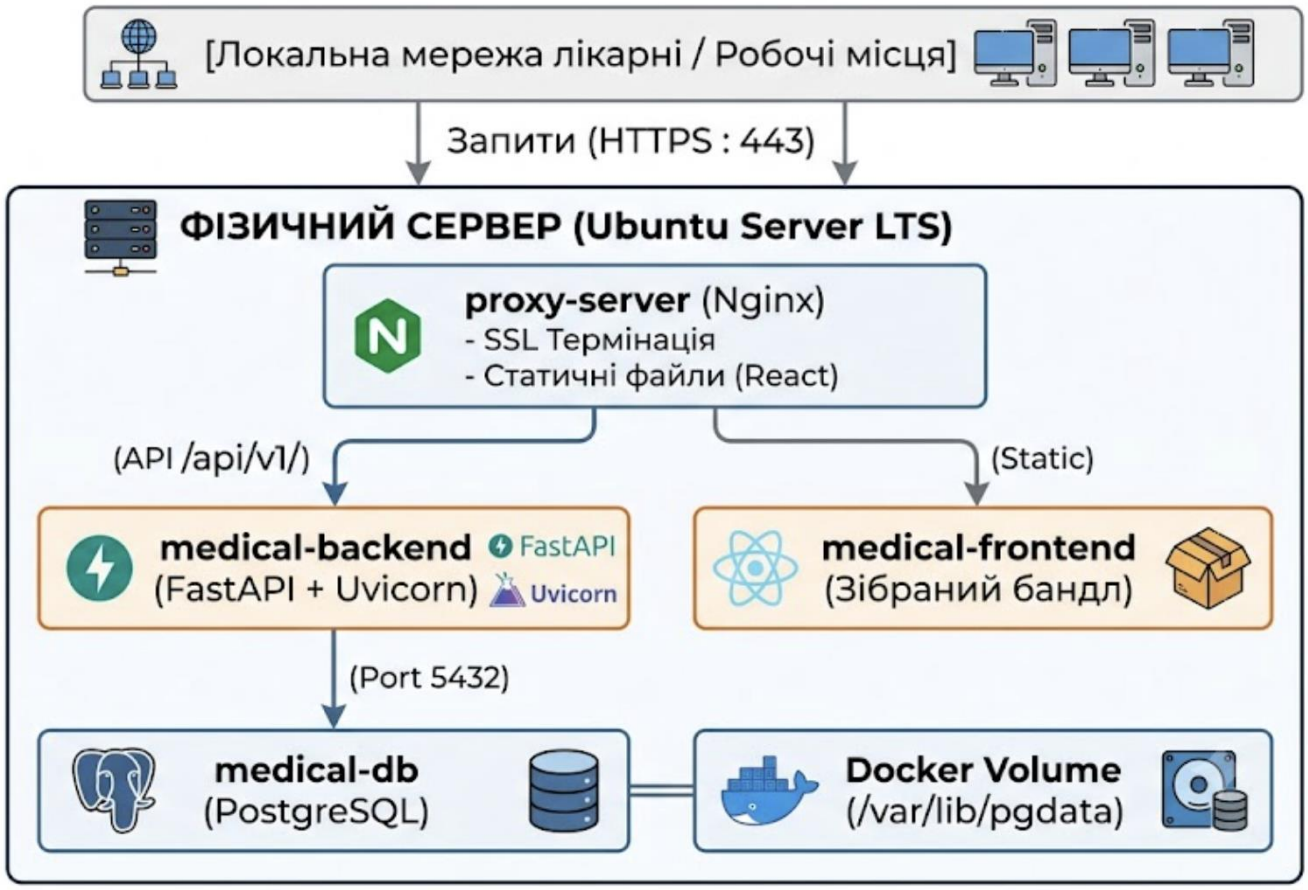

Рисунок 3.5 – Схема розгортання контейнерів на фізичному сервері

Лістинг 3.3 – Конфігураційний файл оркестрації контейнерів (docker-compose.yml)

```
version: '3.8'

services:
  medical-db:
    image: postgres:15-alpine
    container_name: medical_postgres
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - medical_network

  medical-backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
```

```
    container_name: fastapi_backend
    restart: unless-stopped
    depends_on:
      - medical-db
    environment:
      DATABASE_URL:
postgresql+asyncpg://${DB_USER}:${DB_PASSWORD}@medical-
db:5432/${DB_NAME}
      SECRET_KEY: ${SECRET_KEY}
    networks:
      - medical_network

volumes:
  pgdata:

networks:
  medical_network:
    driver: bridge
```

Завершальним кроком розгортання є налаштування апаратної інфраструктури для забезпечення безперебійної роботи розгорнутого програмного комплексу. Фізичний сервер підключається до резервного джерела живлення (потужної зарядної станції або ДБЖ з правильною синусоїдою), яке здатне підтримувати автономну роботу серверного вузла та комутаційного обладнання протягом кількох годин у разі знеструмлення [22]. На рівні операційної системи (BIOS/UEFI) активується функція "Restore on AC Power Loss", що дозволяє серверу автоматично увімкнутися після повної розрядки та подальшої появи напруги у мережі. Додатково, за допомогою планувальника завдань (Cron) налаштовується щоденне автоматичне створення дампів бази даних PostgreSQL (pg_dump), які зберігаються на фізично відокремленому мережевому сховищі (NAS) для мінімізації ризиків безповоротної втрати медичної історії пацієнтів у разі апаратного збою жорстких дисків основного сервера.

3.4 Реалізація клієнтської частини

На рисунку 3.6 зображено інтерфейс головної панелі медичної інформаційної системи, реалізований у темній кольоровій гамі з використанням сучасного мінімалістичного дизайну.

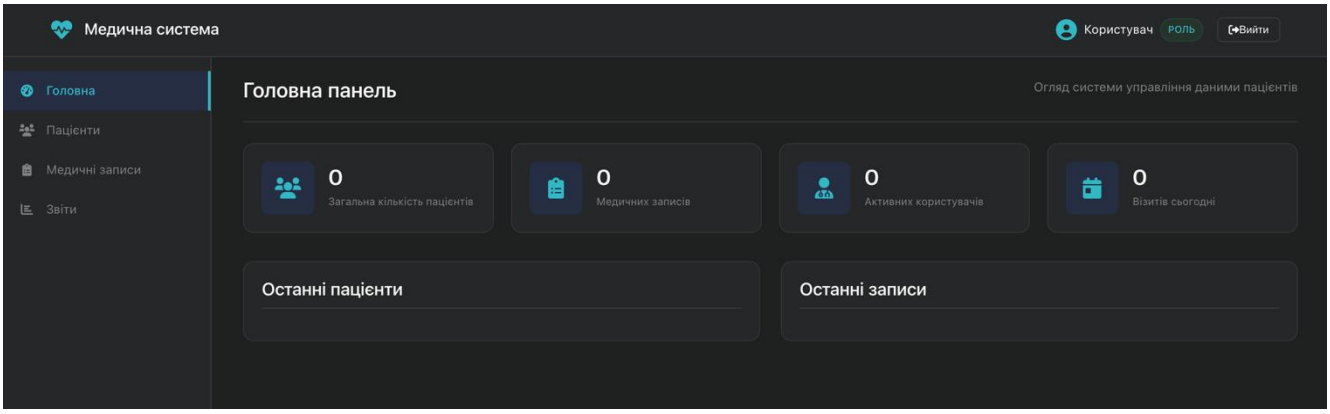

Рисунок 3.6 – Інтерфейс головної панелі медичної інформаційної системи

У верхній частині екрана розміщено горизонтальну панель навігації, яка містить назву системи «Медична система», а також елементи керування користувачем: позначення ролі користувача («Користувач, роль») та кнопку виходу з системи «Вийти».

Ліва частина інтерфейсу представлена вертикальним меню навігації, яке забезпечує доступ до основних розділів системи:

- «Головна» (активний розділ);
- «Пацієнти»;
- «Медичні записи»;
- «Звіти».

Центральну частину екрана займає робоча область із заголовком «Головна панель», яка містить інформаційні картки (віджети) зі статистичними показниками системи. Серед них:

– загальна кількість пацієнтів – 0;

– кількість медичних записів – 0;

– кількість активних користувачів – 0;

– візити сьогодні – 0.

Кожен показник представлений у вигляді окремої картки з іконкою, числовим значенням і коротким підписом, що покращує візуальне сприйняття інформації.

У нижній частині головної панелі розміщено два додаткові блоки:

– «Останні пацієнти» – для відображення нещодавно доданих або оновлених даних про пацієнтів;

– «Останні записи» – для відображення останніх медичних записів.

Праворуч у верхній частині робочої області також присутній інформаційний напис: «Огляд системи управління даними пацієнтів», що визначає призначення даної сторінки.

Отже, представлений інтерфейс забезпечує швидкий доступ до ключових функцій системи, надає узагальнену статистичну інформацію та дозволяє ефективно здійснювати первинний моніторинг стану медичних даних.

На рисунку 3.7 представлено інтерфейс сторінки «Звіти» медичної інформаційної системи, виконаний у темній кольоровій гамі з використанням сучасного мінімалістичного дизайну.

У верхній частині інтерфейсу розміщено панель із назвою системи «Медична система», а також елементи керування користувачем: позначення «Користувач», індикатор ролі («роль») та кнопка «Вийти» для завершення сеансу роботи.

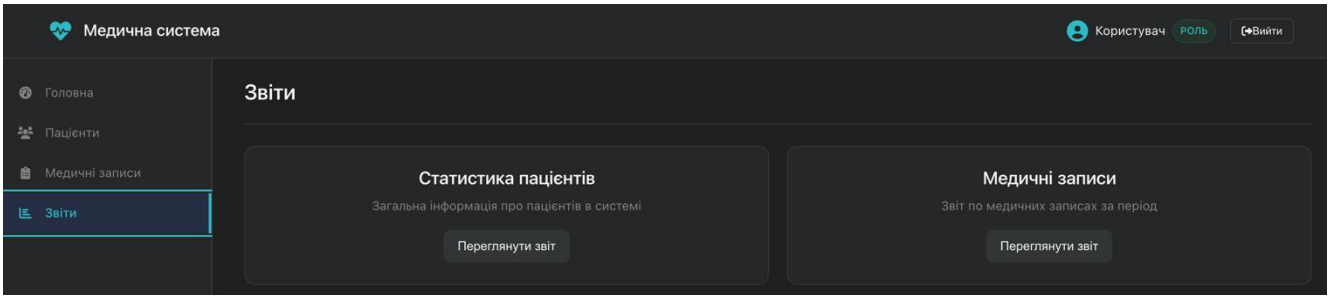

Рисунок 3.7 – Сторінкиа «Звіти» медичної інформаційної системи

Ліворуч знаходиться вертикальне меню навігації, яке містить основні розділи системи:

– «Головна»;

– «Пацієнти»;

– «Медичні записи»;

– «Звіти» (активний розділ, виділений кольором).

Центральна частина екрану містить заголовок «Звіти» та два основні інформаційні блоки (картки), які забезпечують доступ до функцій формування звітності:

«Статистика пацієнтів» – блок призначений для перегляду узагальненої інформації про пацієнтів у системі. Підзаголовок уточнює призначення: «Загальна інформація про пацієнтів в системі». У блоці розміщена кнопка «Переглянути звіт», яка ініціює формування відповідного звіту.

«Медичні записи» – даний блок використовують для аналізу медичних записів за певний період. Пояснювальний текст: «Звіт по медичних записах за період». Для переходу до перегляду передбачена кнопка «Переглянути звіт».

Обидва блоки мають однакову структуру та стиль оформлення: прямокутні картки з округленими кутами, темним фоном та світлим текстом, що забезпечує візуальну узгодженість і зручність сприйняття.

Отже, сторінка «Звіти» забезпечує централізований доступ до функцій аналітики та формування звітів, дозволяючи користувачеві швидко отримувати статистичні дані про пацієнтів і медичні записи, що є важливим для прийняття управлінських рішень та аналізу діяльності медичного закладу.

На рисунку 3.8 представлено інтерфейс сторінки «Медичні записи» медичної інформаційної системи, який призначений для перегляду, управління та редагування інформації про візити пацієнтів.

У верхній частині робочої області відображено заголовок сторінки «Медичні записи», а праворуч розміщена кнопка «+ Додати запис», що дозволяє користувачеві створити новий медичний запис.

Рисунок 3.8 – Сторінка «Медичні записи» медичної інформаційної системи

Основну частину інтерфейсу займає таблиця, яка містить перелік медичних записів. Таблиця має такі стовпці:

– «ID» – унікальний ідентифікатор запису;

– «Пацієнт» – прізвище та ім’я пацієнта;

– «Дата візиту» – дата звернення пацієнта до медичного закладу;

– «Діагноз»– встановлений медичний висновок;

– «Лікар»– прізвище та ім’я лікаря, який здійснював огляд;

– «Дії» – набір керуючих елементів для роботи із записом.

У таблиці наведено приклади п’яти записів:

– пацієнт Олександр Іваненко – дата візиту 01.03.2024, діагноз: гіпертонічна хвороба I ступеня, лікар: Др. Марія Ковальські;

– пацієнт Олександр Іваненко – дата візиту 15.03.2024, діагноз: гіпертонічна хвороба I ступеня (скорочений запис), лікар: Др. Марія Ковальські;

– пацієнт Марія Петренко – дата візиту 28.02.2024, діагноз: гострий бронхіт, лікар: Др. Марія Ковальські;

– пацієнт Віктор Сидоренко – дата візиту 10.03.2024, діагноз: остеоартроз колінних суглобів (частково скорочений), лікар: Др. Марія Ковальські;

– пацієнт Анна Мельник – дата візиту 05.03.2024, діагноз: профілактичний огляд, лікар: Др. Марія Ковальські.

У стовпці «Дії» для кожного запису передбачено набір кнопок із піктограмами, які забезпечують такі функції:

– перегляд детальної інформації;

– редагування запису;

– видалення запису.

Інтерфейс виконаний у темній кольоровій гамі з використанням контрастних елементів керування та підсвічених іконок, що забезпечує зручність сприйняття та підвищує ергономіку роботи користувача.

Таким чином, сторінка «Медичні записи» забезпечує централізоване управління даними про пацієнтів, дозволяє здійснювати швидкий доступ до історії візитів, а також ефективно виконувати операції створення, редагування та видалення медичної інформації.

На рисунку 3.9 зображено модальне вікно перегляду детальної інформації про медичний запис у медичній інформаційній системі. Інтерфейс виконаний у темній кольоровій гамі з використанням контрастного виділення інформаційних блоків.

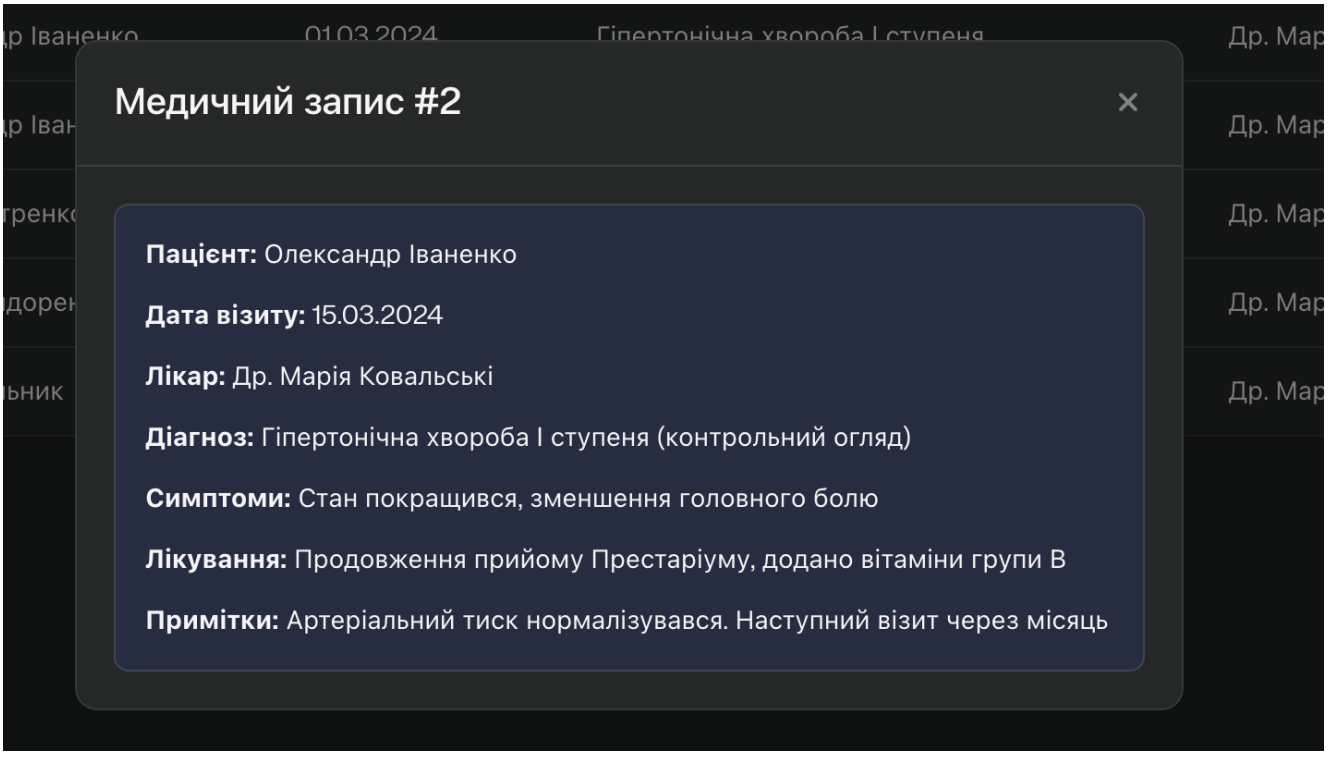

Рисунок 3.9 – Вікно перегляду медичного запису

У верхній частині модального вікна розміщено заголовок «Медичний запис #2», що вказує на унікальний номер запису. Праворуч знаходиться кнопка закриття вікна у вигляді піктограми «×».

Основна частина вікна містить інформаційний блок із детальними даними про обраний запис. Дані структуровано у вигляді підписаних полів:

Інформаційний блок оформлено у вигляді окремої панелі з виділенням фону, що покращує читабельність та структурованість даних. Кожне поле містить назву показника та його значення, що забезпечує зручне сприйняття інформації користувачем.

Таким чином, модальне вікно забезпечує детальний перегляд медичного запису без переходу на іншу сторінку, що підвищує зручність роботи із системою та дозволяє швидко отримувати повну інформацію про стан пацієнта, призначене лікування та рекомендації лікаря.

На рисунку 3.10 зображено модальне вікно редагування медичного запису в медичній інформаційній системі. Інтерфейс реалізовано у темній кольоровій гамі з чітким структуруванням полів введення даних.

У верхній частині вікна розміщено заголовок «Редагувати медичний запис», який вказує на режим редагування вже існуючого запису. Праворуч розташована кнопка закриття у вигляді піктограми «×».

Основна частина вікна містить форму з полями для введення та редагування інформації про медичний запис.

У нижній частині модального вікна розміщено кнопки керування:

– «Скасувати» – для відміни внесених змін;

– «Зберегти» – для підтвердження та збереження відредагованого запису.

Всі поля згруповані логічно та мають підписи, що забезпечує інтуїтивно зрозумілу взаємодію користувача із системою. Використання багаторядкових полів дозволяє вводити розгорнуту медичну інформацію, а елементи керування (випадаючі списки, вибір дати) сприяють мінімізації помилок при введенні даних.

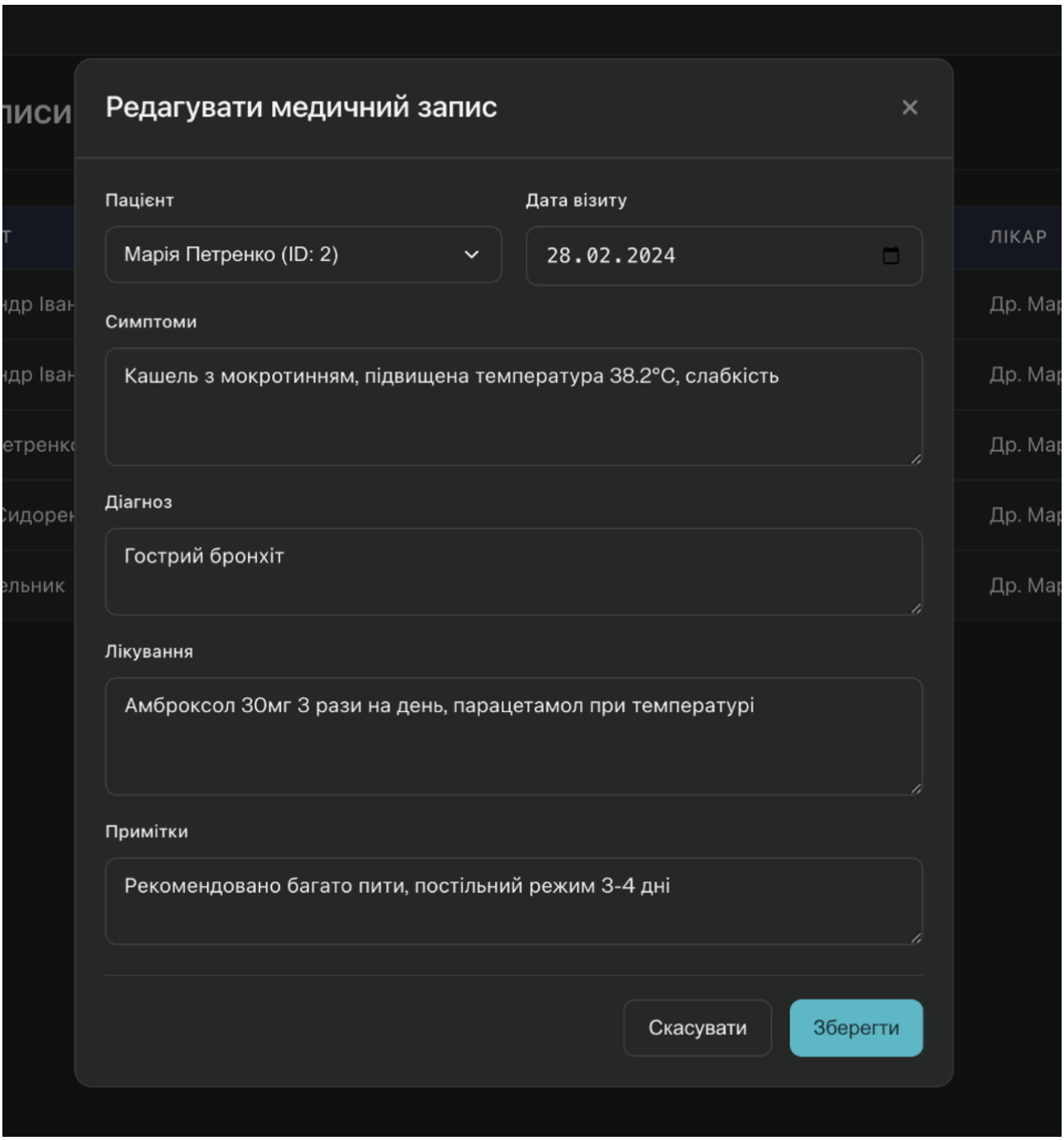

Рисунок 3.10 – Вікно редагування медичного запису

Таким чином, дане вікно забезпечує повноцінне редагування медичного запису, дозволяючи змінювати всі ключові параметри – від загальної інформації про пацієнта до детальних медичних даних, що підвищує ефективність роботи користувачів системи.

У цьому розділі дипломної роботи було успішно вирішено завдання практичної перевірки працездатності та розгортання розробленого програмно-апаратного комплексу для медичного закладу. На основі сформованої методології проведено багаторівневе тестування системи, що охопило модульну, інтеграційну та навантажувальну перевірки. Результати модульного тестування серверної частини підтвердили високу надійність базової бізнес-логіки та криптографічних алгоритмів із середнім покриттям коду понад 80%. Інтеграційні тести довели коректність транзакційної взаємодії між API (FastAPI) та реляційною базою даних (PostgreSQL), гарантуючи атомарність операцій та цілісність збереження електронних медичних записів. За допомогою спеціалізованого інструментарію було проведено навантажувальне тестування, яке засвідчило здатність веб-сервера стабільно обробляти запити в умовах імітованих пікових навантажень (до 100 одночасних сесій лікарів та реєстраторів) із середнім часом відгуку близько 580 мілісекунд, що повністю задовольняє встановлені на етапі проектування нефункціональні вимоги.

Окрім програмної валідації, критично важливим етапом стала перевірка апаратної відмовостійкості інфраструктури. Тестування підтвердило ефективність алгоритмів автоматичного переходу серверного обладнання та мережевих комутаторів на живлення від резервних зарядних станцій під час імітації раптового знеструмлення загальної мережі. Практичне розгортання програмного комплексу було виконано на фізичному сервері з використанням технології контейнеризації Docker. Це інженерне рішення забезпечило ізольованість сервісів, незалежність середовищ виконання, ефективний розподіл апаратних ресурсів (процесорного часу та оперативної пам'яті) та значно спростило процедуру майбутнього оновлення системи. Додаткове налаштування веб-сервера Nginx як зворотного проксі дозволило організувати безпечну маршрутизацію трафіку з використанням SSL-шифрування.

Загалом, результати, отримані під час виконання завдань третього розділу, підтверджують, що створена комп'ютерна система працює стабільно та безпомилково. Розроблений комплекс є повністю завершеним, надійним та

технічно збалансованим продуктом, який інтегрує в собі сучасні програмні та апаратні рішення і є повністю готовим до впровадження та дослідної експлуатації в реальних умовах функціонування медичного закладу.

ВИСНОВКИ

У дипломній роботі вирішено актуальне інженерне завдання щодо розробки та впровадження відмовостійкої комп'ютерної системи для автоматизації інформаційних процесів медичного закладу.

Проведено ґрунтовний аналіз предметної області, який довів, що існуючі комерційні медичні інформаційні системи часто є надмірно складними, дорогими та не адаптованими до інфраструктурних викликів, зокрема до проблем енергонезалежності. Встановлено, що для успішної автоматизації локального медичного закладу необхідне поєднання надійного програмного забезпечення із спроектованою апаратною інфраструктурою, здатною функціонувати в автономному режимі.

Спроектовано архітектуру програмно-апаратного комплексу, яка базується на клієнт-серверному підході з використанням контейнеризації Docker. Це забезпечило ізоляцію сервісів, високу масштабованість та простоту розгортання на серверному обладнанні закладу. Система автономного резервного живлення (ДБЖ та зарядних станцій) підтвердив можливість забезпечення безперебійної роботи серверного вузла протягом часу, достатнього для коректного збереження активних транзакцій та завершення критичних медичних записів під час знеструмлення.

Обґрунтовано вибір сучасного стека технологій, що включає мову програмування Python та реляційну систему керування базами даних PostgreSQL. Ця комбінація дозволила досягти високої швидкодії системи, надійної валідації даних за допомогою Pydantic та суворого контролю цілісності інформації на рівні бази даних, що є критично важливим для медичних систем.

Забезпечено високий рівень інформаційної безпеки, що підтверджено впровадженням сучасних криптографічних алгоритмів (bcrypt для хешування паролів) та протоколів автентифікації (JWT). Спроектована система розмежування доступу гарантує виконання вимог законодавства щодо обробки

персональних даних та медичної таємниці.

Проведено комплексне тестування розробленого комплексу, яке довело працездатність системи при різних рівнях навантаження.

Таким чином, розроблена комп'ютерна система повністю відповідає поставленим завданням. Впровадження даного комплексу дозволяє суттєво підвищити ефективність роботи медичного персоналу, автоматизувати рутинні операції реєстратури та забезпечити надійне, конфіденційне зберігання медичної історії пацієнтів, що значно покращує загальну якість надання медичних послуг у закладі. Розроблена архітектура має високий потенціал для подальшого розвитку, зокрема шляхом інтеграції з центральними компонентами eHealth та розширення функціоналу системами підтримки прийняття клінічних рішень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 World Health Organization. Digital health. 2023. URL: https://www.who.int/health-topics/digital-health (дата звернення: 19.04.2026).

2 Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed.: Pearson, 2017. 1200 p.

3 Mell P., Grance T. The NIST Definition of Cloud Computing, 2022. 7 p.

4 Python Software Foundation. Python 3 Documentation. 2023. URL: https://docs.python.org/3/ (дата звернення: 21.04.2026).

5 Sommerville I. Software Engineering. 10th ed.: Pearson, 2023. 816 p.

6 Tanenbaum A., Steen M. Distributed Systems: Principles and Paradigms. 3rd ed.: Pearson, 2017. 686 p.

7 PostgreSQL Global Development Group. PostgreSQL Documentation. 2023. URL: https://www.postgresql.org/docs/ (дата звернення: 28.04.2026).

8 Django Software Foundation. Django Documentation. 2023. URL: https://docs.djangoproject.com/ (дата звернення: 19.05.2026).

9 FastAPI. *FastAPI* Documentation. 2023. URL: https://fastapi.tiangolo.com/ (дата звернення: 11.05.2026).

10 Docker Inc. Docker Documentation. 2023. URL: https://docs.docker.com/ (дата звернення: 05.05.2026).

12 Date C. J. *SQL* and Relational Theory. 2nd ed. : O'Reilly Media, 2015. 483 p.

13 Pressman R., Maxim B. Software Engineering: A Practitioner's Approach. 9th ed.: McGraw-Hill, 2020. 784 p.

14 Fielding R. Architectural Styles and the Design of Network-based Software Architectures., 2000. 180 p.

15 ECMA International. The JSON Data Interchange Syntax (ECMA-404). 2023. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ (дата звернення: 19.04.2026).

16 Stallings W. Network Security Essentials. 7th ed.: Pearson, 2017. 448 p.

17 Anderson R. Security Engineering. 3rd ed.: Wiley, 2020. 1248 p.

18 OWASP Foundation. OWASP Top Ten. 2023. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 22.04.2026).

19 Pytest developers. Pytest Documentation. 2023. URL: https://docs.pytest.org/ (дата звернення: 29.04.2026).

20 Pahl C. Containerization and the PaaS cloud. IEEE Cloud Computing. 2015. Vol. 2, No. 3. P. 24–31.

21 Nginx Inc. NGINX Documentation. 2023. URL: https://nginx.org/en/docs/ (дата звернення: 19.05.2026).

22 Google. Site Reliability Engineering. O’Reilly Media, 2016. 552 p.

ДОДАТОК А
СЛАЙДИ ПРЕЗЕНТАЦІЇ

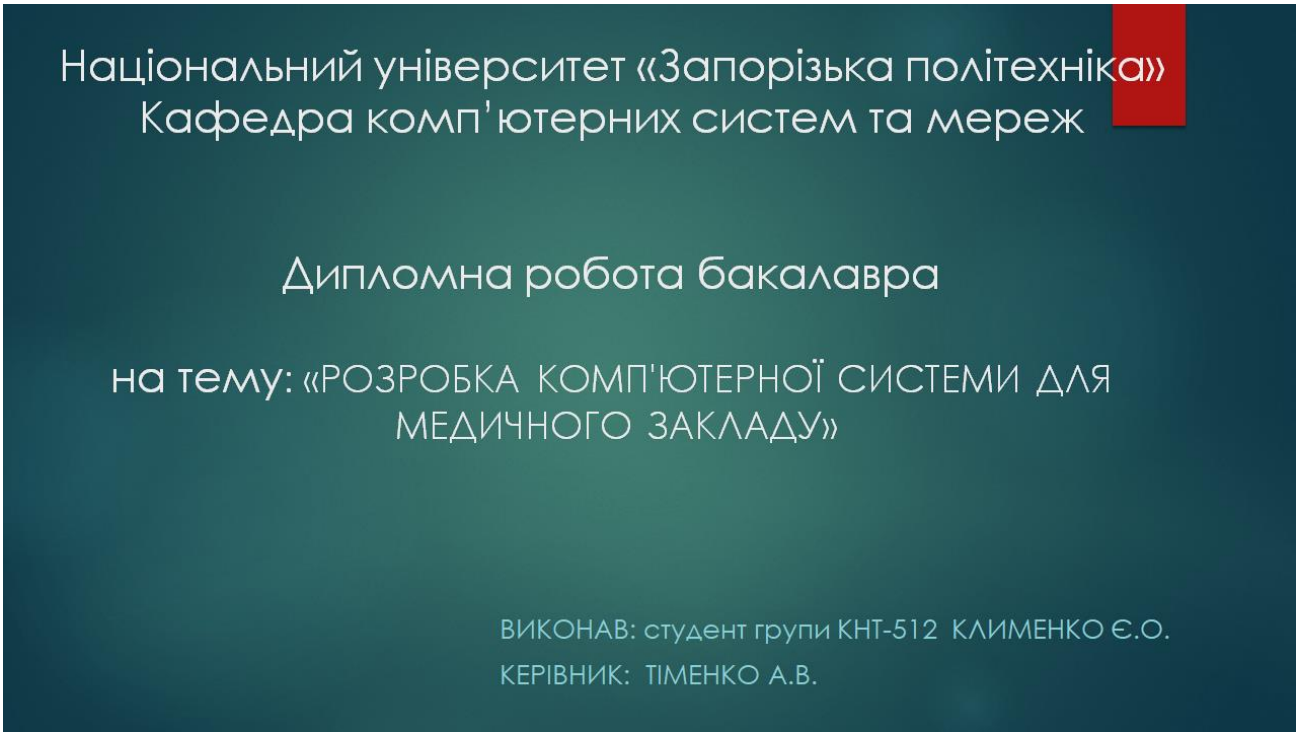

Рисунок А.1 – Слайд 1

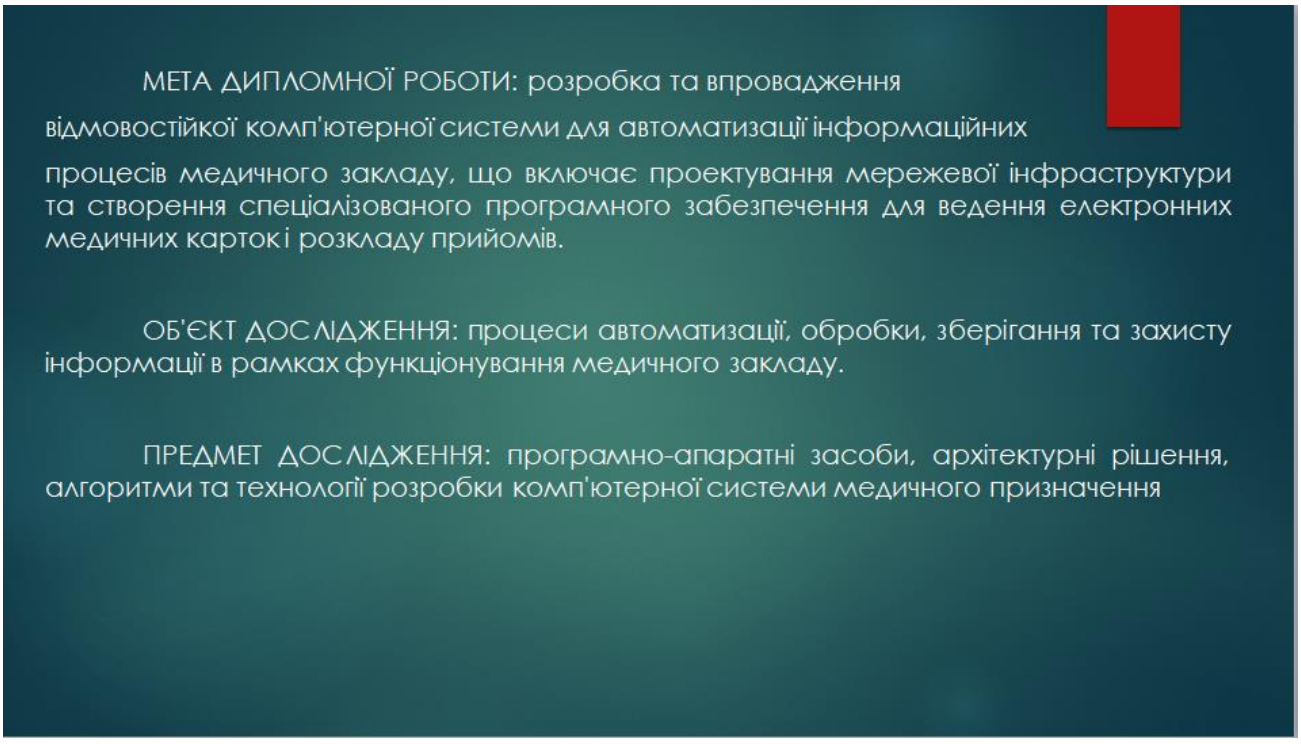

Рисунок А.2 – Слайд 2

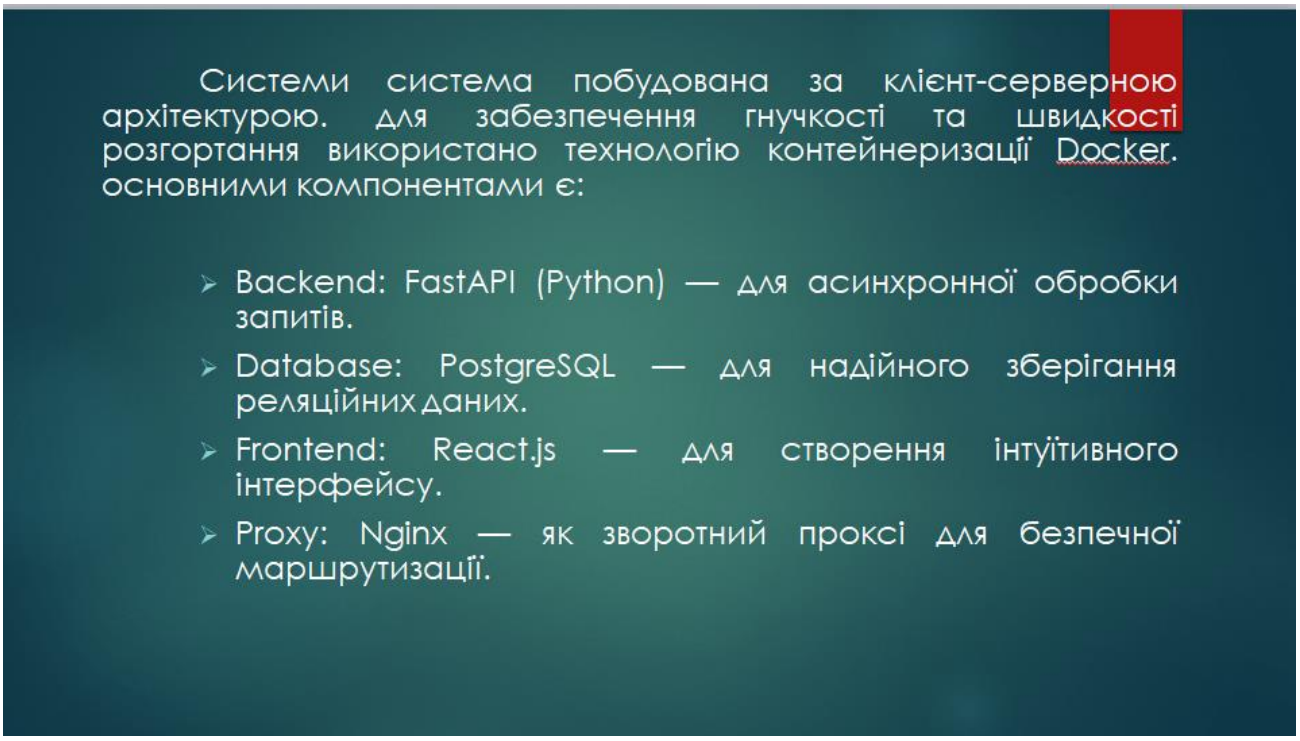

Рисунок А.3 – Слайд 3

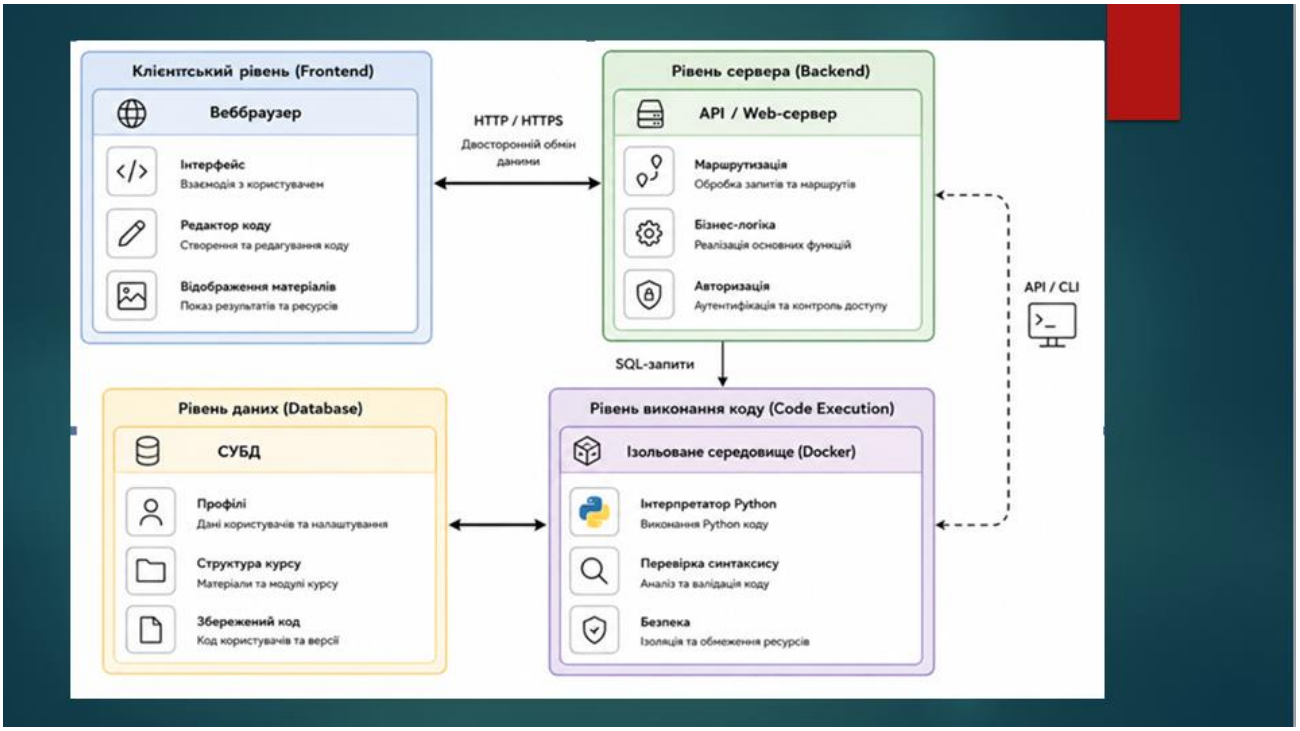

Рисунок А.4 – Слайд 4

Порівняльний аналіз популярних медичних інформаційних систем

Назва МІС	Тип архітектури	Можливості інтеграції	Переваги	Недоліки
Helsi	Хмарна (SaaS)	Високі (API, eHealth)	Зручний портал для пацієнтів, швидке розгортання	Залежність від інтернет-з'єднання, закритий код
Doctor Eleks	Клієнт-серверна / Хмарна	Високі (HL7, DICOM)	Потужний інструментарій для стаціонарів, гнучкість	Складність налаштування, високі вимоги до апаратури
ЕМСІмед	Клієнт-серверна	Середні (eHealth)	Надійність роботи в локальній мережі, захист	Застарілий інтерфейс у деяких модулях
Medstar	Хмарна	Високі	Модульна структура, хороша підтримка мобільних пристроїв	Обмежені можливості кастомізації звітів

Рисунок А.5 – Слайд 5

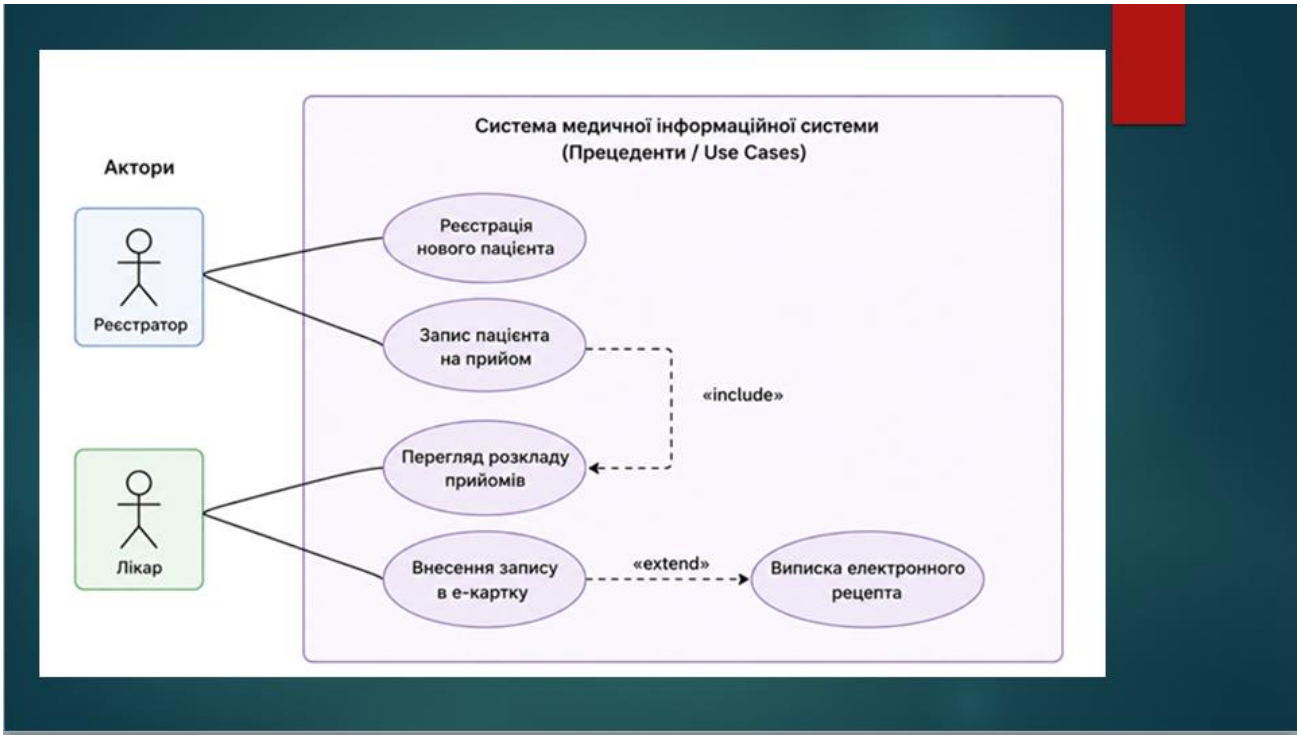

Рисунок А.6 – Слайд 6

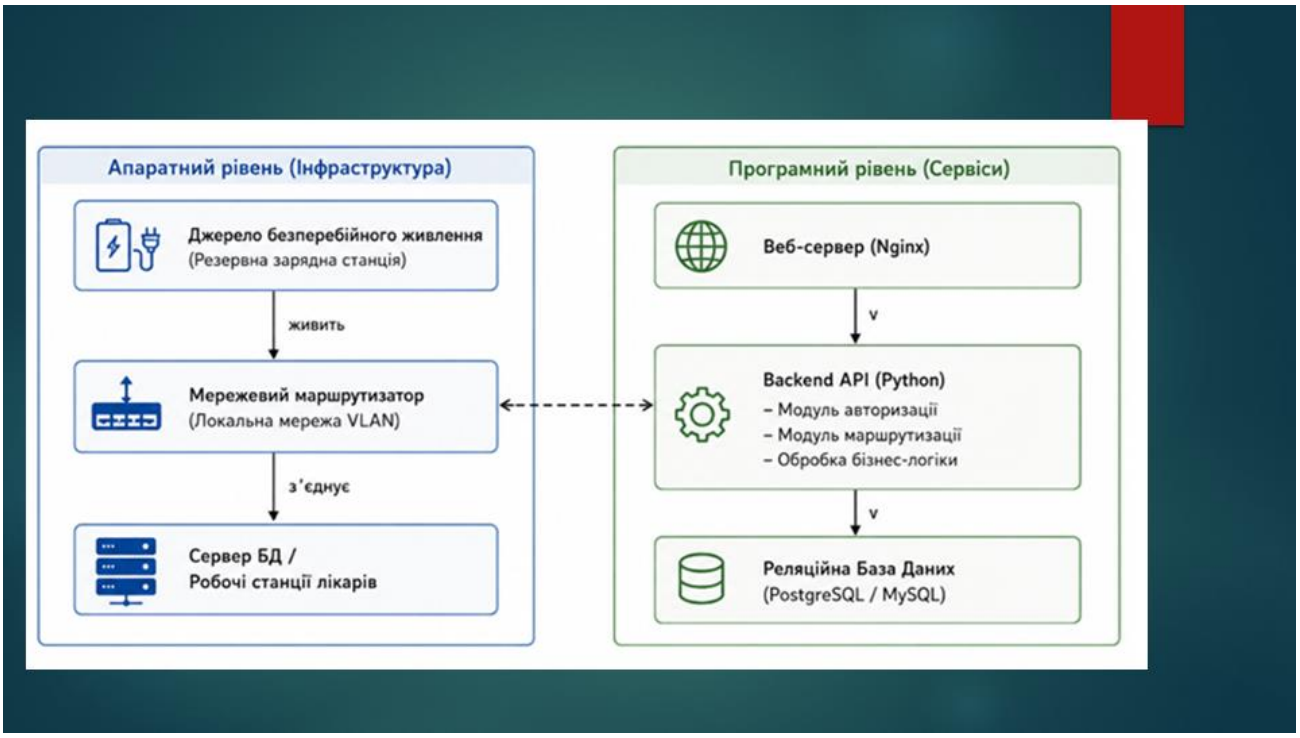

Рисунок А.7 – Слайд 7

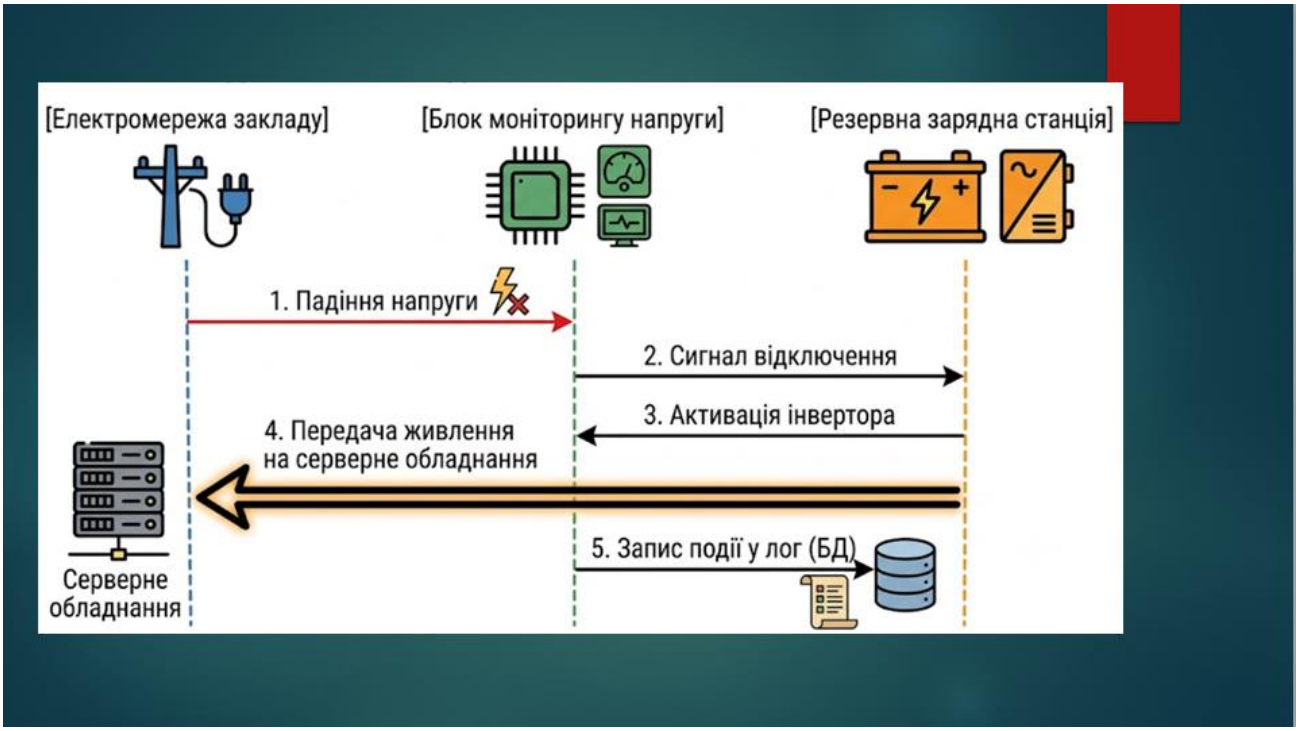

Рисунок А.8 – Слайд 8

Рисунок А.9 – Слайд 9

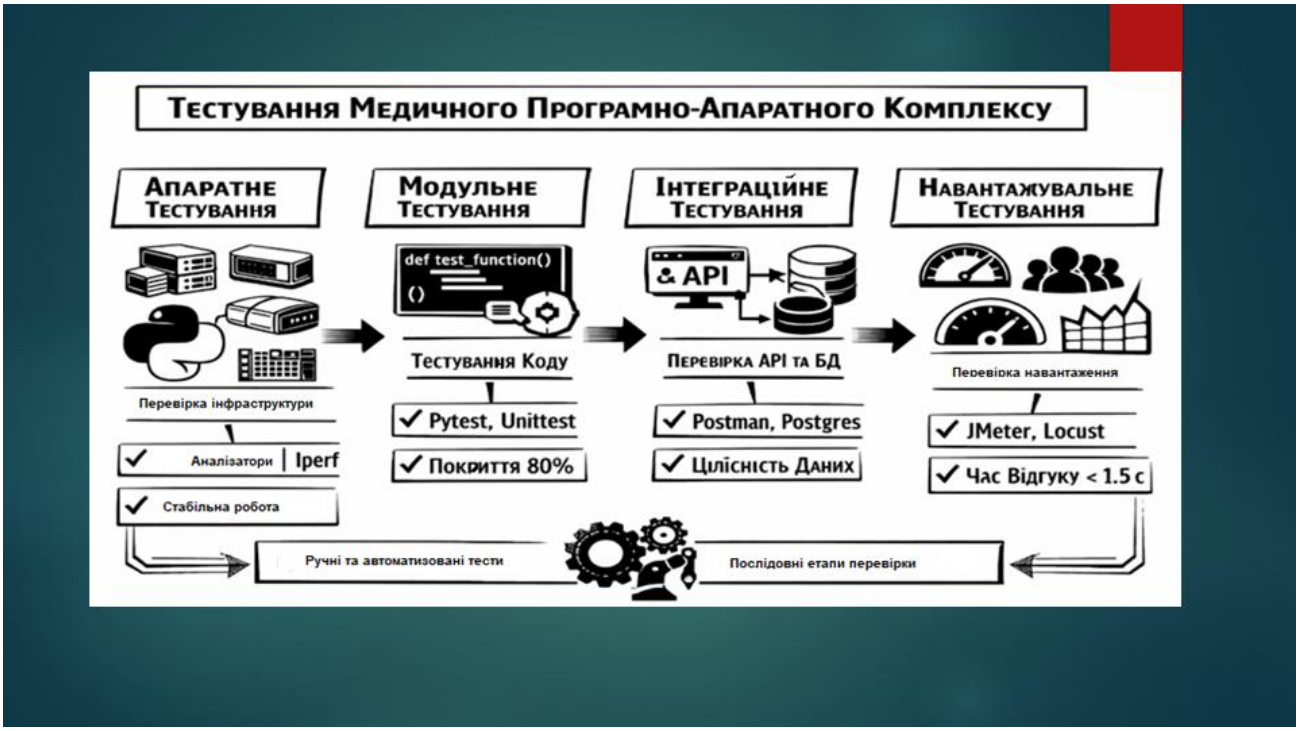

Рисунок А.10 – Слайд 10

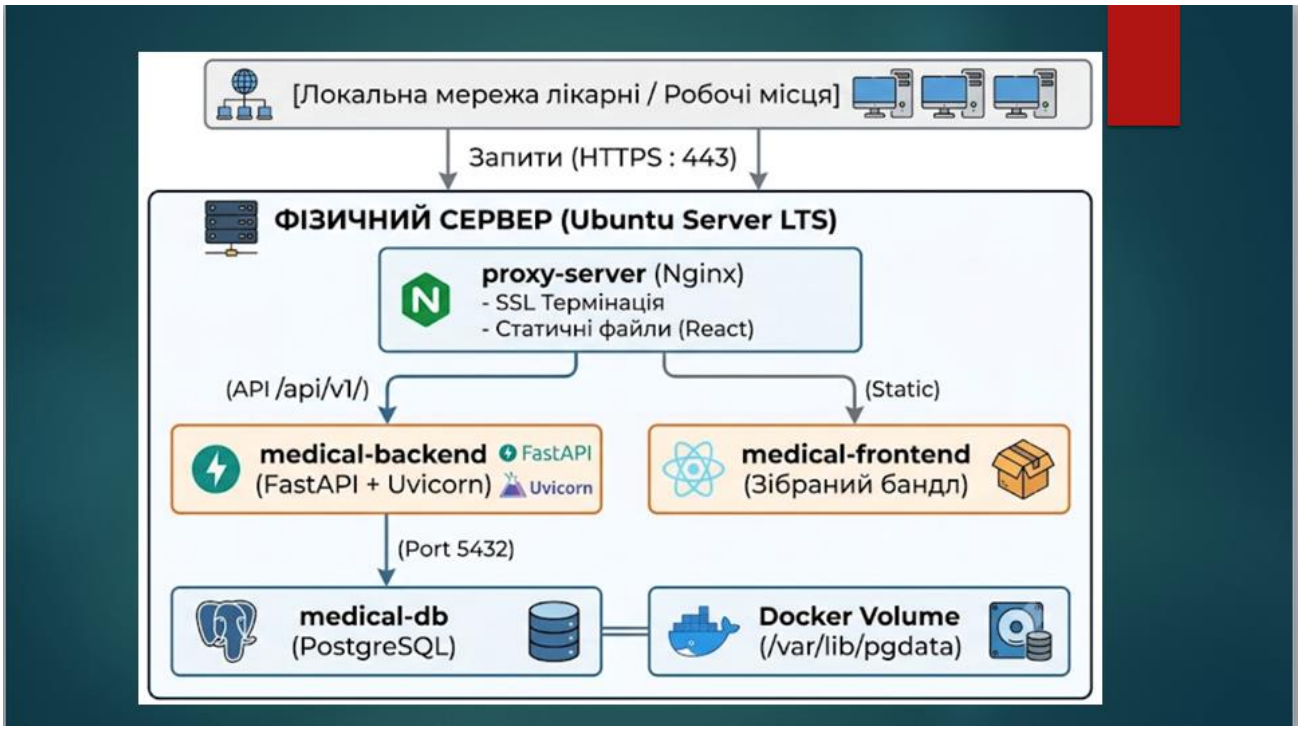

Рисунок А.11 – Слайд 11

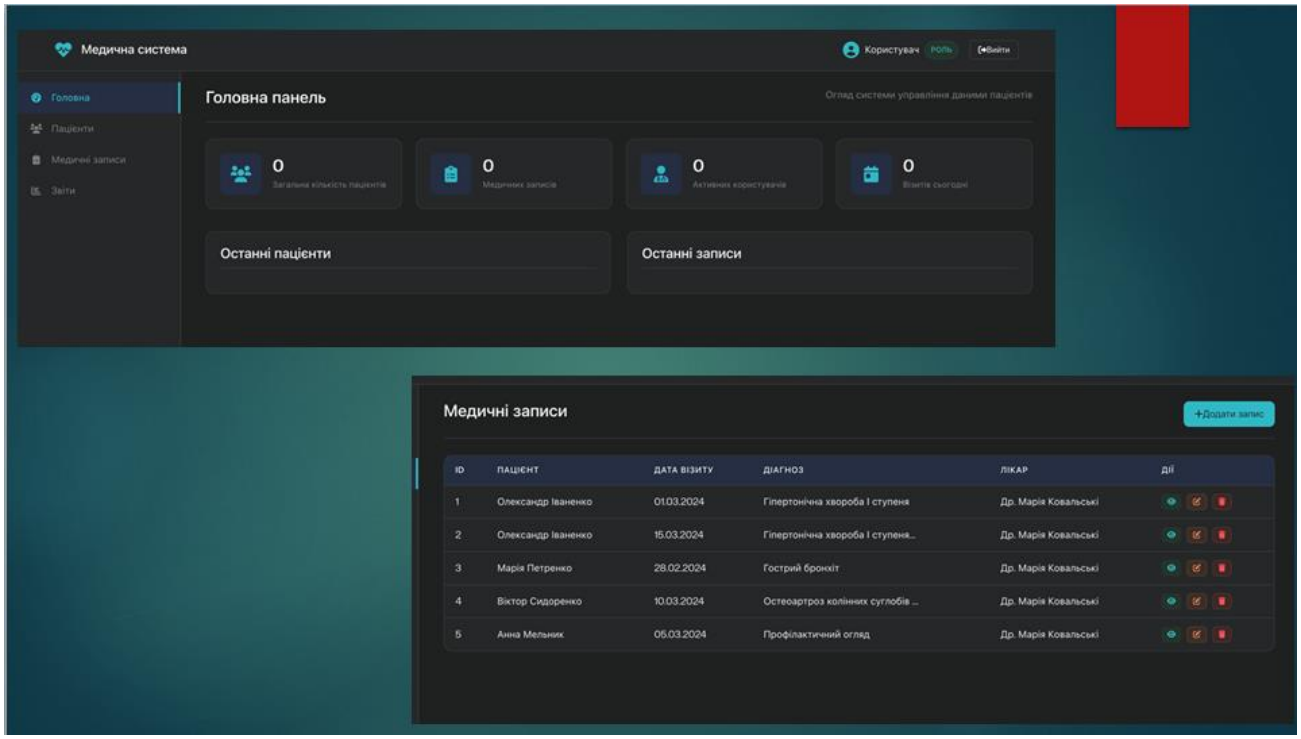

Рисунок А.12 – Слайд 12

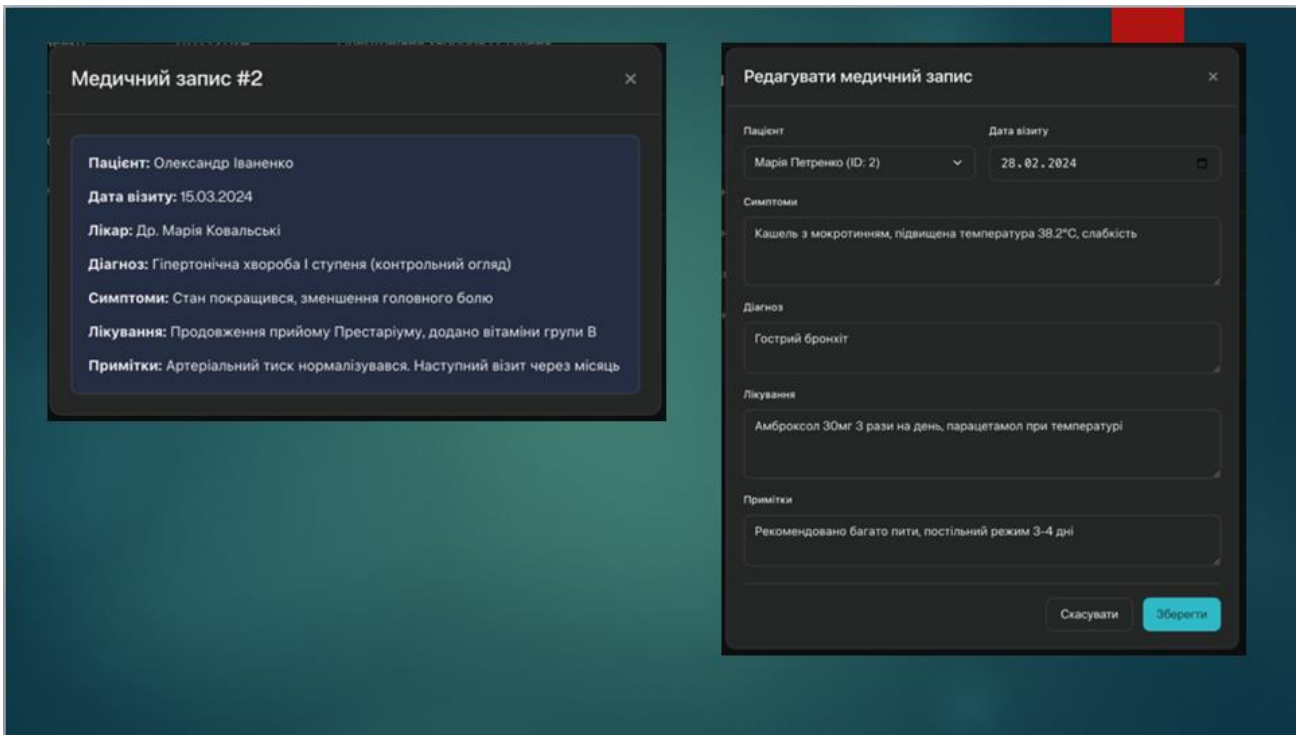

Рисунок А.13 – Слайд 13

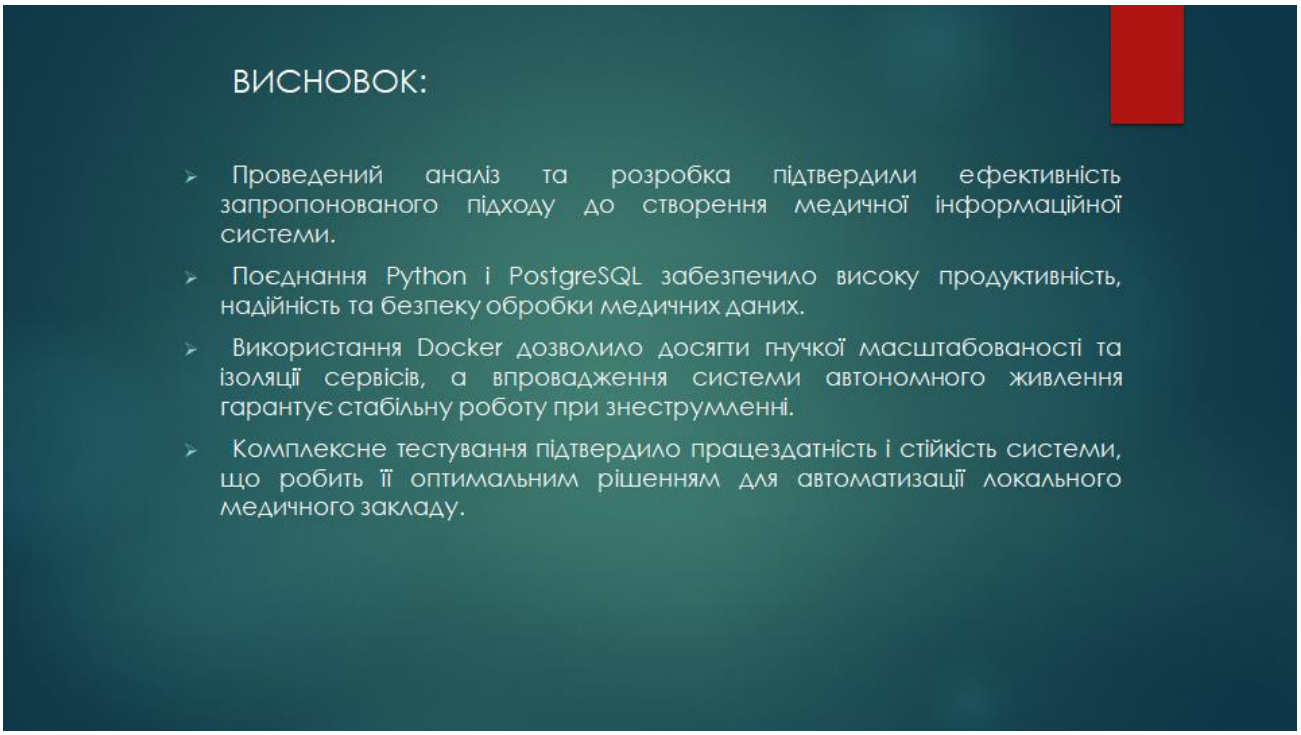

Рисунок А.14 – Слайд 14