Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка додатку для обліку пацієнтів УЗД на базі Python з використанням PyQt5 та SQLite

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Круцяк В. В. (прізвище та ініціали)

Керівник (підпис) Цуприк Г. Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) Матійчук Л. П. (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Круцяку Володимиру Володимировичу
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка додатку для обліку пацієнтів УЗД на базі Python з використанням PyQt5 та SQLite

Керівник роботи Цуприк Галина Богданівна, кандидат технічних наук, доцент кафедри ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи _____
3. Вихідні дані до роботи Завдання замовника, потреби замовника, вимоги замовника, інтернет джерела, аналоги

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
Аналіз предметної області та постановка задачі
Проєктування архітектури та інтерфейсу додатку
Розробка та тестування програмного забезпечення
Безпека життєдіяльності та охорони праці
Висновок
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Головне меню системи «Доктор Елекс», 2. Інтерфейс системи «HEALTH24»,
3. Головне меню власної системи «EchoMed», 4. Форма створення нового огляду пацієнта,
5. Вікно попереднього перегляду карти пацієнта перед друком,
6. Головне меню з чітким розміщенням кнопок, 7. Схема архітектури EchoMed,

8. UML-діаграма класів EchoMed, 9. Деталізована UML-діаграма класів,
10. Узагальнена UML-діаграма класів, 11. Блок-схема алгоритму додавання пацієнта в EchoMed
11. Блок-схема алгоритму пошуку пацієнта за ім'ям, 12. Головне вікно програми у виконанні,
13. Фільтрація пацієнтів за статусом «Вибрані», 14. Вікно редагування карти пацієнта,
15. Підказки при наведенні на рядок заключення, 16. Вікно з повним описом заключення,
17. Вікно перегляду та друку карти пацієнта з заключенням, 18. Таблиця тестування

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності,	Лазарюк В. В., доц., доц. каф. МТ		
основи охорони праці			
Нормоконтроль	Стоянов Ю. М., к.т.н.		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи	28.01.2025 - 30.01.2025	*Виконано*
2	Підбір джерел та відомостей для розробки ПЗ EchoMed	31.01.2025 - 03.02.2025	*Виконано*
3	Переклад та опрацювання джерел з теми медичного обліку та інтерфейсів	03.02.2025 - 05.02.2025	*Виконано*
4	Аналіз предметної області та формулювання технічних вимог	06.02.2025 - 10.02.2025	*Виконано*
5	Оформлення першого розділу кваліфікаційної роботи	11.02.2025 - 14.02.2025	*Виконано*
6	Проєктування БД, архітектури програми, інтерфейсу	15.02.2025 - 28.02.2025	*Виконано*
7	Оформлення другого розділу кваліфікаційної роботи	01.03.2025 - 08.03.2025	*Виконано*
8	Розробка модулів додатка	09.03.2025 - 23.03.2025	*Виконано*
9	Реалізація ключового функціоналу додатка	25.03.2025 - 15.04.2025	*Виконано*
10	Тестування, формування таблиць, усунення помилок	16.04.2025 - 30.04.2025	*Виконано*
11	Виконання завдань до підрозділу «Безпека життєдіяльності»	1.05.2025 - 06.05.2025	*Виконано*
12	Виконання завдань до підрозділу «Основи охорони праці»	08.05.2025 - 11.05.2025	*Виконано*
13	Оформлення висновків, списку літератури, додатків	13.05.2025 - 18.05.2025	*Виконано*
14	Перевірка на плагіат	09.06.2025	*Виконано*
15	Нормоконтроль	14.06.2025	*Виконано*
16	Попередній захист кваліфікаційної роботи	09.06.2025	*Виконано*
17	Захист кваліфікаційної роботи	23.06.2025	

Студент _____ *Круцяк В. В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Цуприк Г. Б.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка додатку для обліку пацієнтів УЗД на базі Python з використанням PyQt5 та SQLite // Кваліфікаційна робота освітнього рівня «Бакалавр» // Круцяк Володимир Володимирович // Тернопільський національний технічний університету імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії // спеціальність 121 «Інженерія програмного забезпечення» // група СП-42 // Тернопіль, 2025 // С. 73, рис. — 18, таблиць — 1, додатки — 5, джерел — 32.

Робота присвячена розробці десктопного додатку для обліку пацієнтів кабінету ультразвукової діагностики із застосуванням сучасних технологій Python, PyQt5 та SQLite.

Кваліфікаційна робота складається з чотирьох основних розділів. У першому розділі розглянуто предметну область, проведено аналіз проблеми та сформульовано вимоги до функціоналу системи.

У другому розділі виконано проєктування архітектури програмного забезпечення, визначено структуру бази даних, створено UML-діаграми та описано алгоритми взаємодії модулів.

У третьому розділі описано процес тестування додатку, подано методику перевірки працездатності, наведено результати функціонального тестування та сформульовано висновки щодо надійності системи.

Ключові слова: інформаційна система, Python, PyQt5, SQLite, облік пацієнтів, УЗД, медичне програмне забезпечення.

Об'єкт дослідження: Десктопний додаток для обліку пацієнтів кабінету ультразвукової діагностики.

Предмет дослідження: Процес розробки десктопного додатку для обліку пацієнтів УЗД на базі Python з використанням PyQt5 та SQLite.

ABSTRACT

Development of an application for ultrasound patient accounting based on Python using PyQt5 and SQLite // Bachelor's qualification work // Krutsiak Volodymyr Volodymyrovych // Ternopil Ivan Pul'yui National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering // Specialty 121 "Software Engineering" // Group SP-42 // Ternopil, 2025 // P. 73, fig. — 18, tabl. — 1, app. — 5, sources — 32.

The work is devoted to the development of a desktop application for accounting for patients of an ultrasound diagnostics office using modern Python, PyQt5, and SQLite technologies.

The qualification work consists of four main sections. The first section examines the subject area, analyzes the problem, and formulates requirements for the system's functionality.

The second section performs the design of the software architecture, defines the database structure, creates UML diagrams, and describes the interaction algorithms of the modules.

The third section describes the application testing process, presents the methodology for checking performance, provides the results of functional testing, and formulates conclusions regarding the system's reliability.

Keywords: information system, Python, PyQt5, SQLite, patient accounting, ultrasound, medical software.

Object of research: Desktop application for accounting for patients of an ultrasound diagnostics office.

Subject of research: The process of developing a desktop application for ultrasound patient accounting based on Python using PyQt5 and SQLite.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

УЗД – ультразвукова діагностика

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

GUI – Graphical User Interface (графічний інтерфейс користувача)

СУБД – система управління базами даних

SQL – Structured Query Language (структурована мова запитів)

ID – Identifier (ідентифікатор)

PDF – Portable Document Format (переносний формат документів)

UML – Unified Modeling Language (уніфікована мова моделювання)

API – Application Programming Interface

МІС – медична інформаційна система

VDT – Visual Display Terminal

КПО – коефіцієнт природного освітлення

DB – database

EchoMed – умовна назва ПЗ, не є абревіатурою, вживається як позначення програми

ЗМІСТ

ВСТУП

Ультразвукова діагностика (УЗД) є одним із найважливіших методів сучасної клінічної медицини. Це неінвазивний, інформативний, оперативний та відносно недорогий спосіб виявлення патологій внутрішніх органів і тканин людини.

Широке застосування УЗД у повсякденній практиці лікарів вимагає чіткого, зручного та систематизованого способу збереження й обробки результатів обстеження. Попри актуальність цього напрямку, більшість кабінетів УЗД у державних або приватних медичних установах досі використовують паперові носії, що не відповідає сучасним вимогам до обліку та захисту медичної інформації.

Під час попередніх спостережень діяльності лікарів в кабінетах УЗД виявлено стандартні перешкоди: труднощі з пошуком досліджень за попередні періоди, подвійне внесення пацієнтів, брак централізованої інформації про діагнози та відсутність автоматизованої звітності. Переважна частина процесів передбачає повторне введення даних вручну, що призводить до втрати часу та збільшує ймовірність помилок. Ці проблеми підкреслюють потребу в розробці сучасного, зручного програмного забезпечення, яке дозволить автоматизувати ключові етапи роботи в УЗД-кабінетах.

Актуальність дипломної роботи полягає в тому, що вона націлена на розробку спеціалізованого десктопного додатку для обліку пацієнтів УЗД. На відміну від складних медичних інформаційних систем, запропоноване рішення не потребує серверного забезпечення, підключення до інтернету або складного навчання персоналу. Реалізація на базі мови програмування Python із застосуванням бібліотек PyQt5 для побудови графічного інтерфейсу та SQLite для збереження даних дає змогу створити компактний, швидкий і зручний у використанні додаток.

Програма отримала назву EchoMed. Вона забезпечує введення персональних даних пацієнта, інформації про тип та дату обстеження, висновків лікаря, а також дозволяє формувати карту огляду, переглядати історію обстежень і зберігати

висновки у форматі PDF. Для друку результатів передбачено зручний модуль із попереднім переглядом.

Таким чином, EchoMed охоплює весь цикл роботи з пацієнтом — від первинного запису до архівації даних. У межах даної дипломної роботи було проведено аналіз предметної області, спроєктовано структуру та інтерфейс додатку, реалізовано повнофункціональну програму та виконано її тестування.

Мета цієї дипломної роботи - розробити програмний продукт для організації локального обліку пацієнтів, котрі проходять ультразвукову діагностику. Таке ПЗ має забезпечити ефективне зберігання даних, полегшити роботу медичного персоналу та надати швидкий доступ до історії обстежень. Для досягнення визначеної мети необхідно виконати наступні завдання: аналіз існуючих програм, чітке визначення вимог, проєктування структури бази даних, розробка алгоритмів збереження та виведення інформації, створення та оформлення інтерфейсу користувача, реалізація функціоналу друку результатів та тестування працездатності системи.

Об'єктом цього дослідження є процес обліку та управління медичними картками пацієнтів, які проходять УЗД-діагностику. Предметом дослідження є розробка програмного забезпечення для автоматизації обліку, включаючи організацію бази даних, дизайн графічного інтерфейсу та алгоритми роботи з інформацією про пацієнтів. В роботі використано принципи об'єктно-орієнтованого програмування, методи структурного моделювання, системного аналізу, графічного проектування інтерфейсу та функціонального тестування для забезпечення коректної роботи програми.

Практична значущість дипломної роботи полягає в тому, що програмний продукт може бути впроваджений у реальних умовах кабінетів ультразвукової діагностики без додаткового навчання або налаштувань. Завдяки простоті архітектури, він не вимагає складного технічного обслуговування та забезпечує повну автономність. Крім того, гнучка структура бази даних дозволяє у майбутньому розширити функціональність програми, інтегрувати її у більші медичні системи або адаптувати під інші види медичних обстежень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Розробка прикладного програмного забезпечення неможлива без детального вивчення предметної області, у межах якої воно функціонуватиме. У даному випадку йдеться про сферу медичної діагностики, зокрема — про організацію обліку пацієнтів в кабінеті ультразвукової діагностики. Аналіз ключових характеристик, типових проблем та очікувань користувачів є основою для формулювання технічних рішень і вибору засобів реалізації. Саме на цьому етапі визначається, які функціональні потреби має задовольняти програмне забезпечення, яку архітектуру воно повинне мати та які методи доцільно застосовувати при його створенні.

Розділ включає загальну характеристику об’єкта автоматизації, огляд існуючих аналогів, постановку задачі та уточнення вимог до функціоналу та структури системи. Це дозволяє не тільки теоретично обґрунтувати необхідність розробки, а й підготувати цілісне технічне бачення кінцевого продукту.

1.1 Загальна характеристика об’єкта

Об'єкт автоматизації цієї роботи — процес обліку пацієнтів у кабінеті ультразвукової діагностики (УЗД). Це важлива частина щоденної медичної практики, яка потребує точності, структури й швидкості. УЗД — один із найбільш універсальних і безпечних методів діагностики, що використовується у клініках усіх рівнів.

Пацієнти приходять, проходять обстеження, отримують висновки — і так щодня, десятками. Кожен із них залишає інформацію, яку треба фіксувати. Зібрані відомості — ім’я, дата, тип процедури, медичні результати — мають зберігатися без помилок.

Зазвичай усе це оформлюється вручну або в Excel. Але Excel — це не медична система, і вона не здатна швидко показати динаміку обстежень. Більше того — при великому потоці пацієнтів виникають дублікати, пропуски, втрати записів. А це вже не просто незручність, а фактор ризику для здоров'я [1,2].

У закладах, де немає централізованої електронної системи, ці проблеми виникають майже щодня. Особливо в невеликих приватних кабінетах або діагностичних центрах. Там лікар зазвичай працює сам і відповідає не тільки за діагностику, а й за документообіг. Саме такі умови вимагають простої, надійної програми, яка не потребує багато часу на навчання.

Інша проблема — повторні звернення пацієнтів.

Лікар повинен мати змогу швидко знайти попередній висновок, подивитися, що було минулого разу, порівняти. Ідеально — якщо це все доступно в кілька кліків, а не вимагає перегляду старих карток або роздруківок. EchoMed враховує це й дає можливість переглядати архіви обстежень однієї особи без втрати часу [3].

Медичне середовище — це завжди тиск часу. Прості рішення виграють перед складними. Великі системи, як-от MEDOC чи МІС «Доктор Елекс», часто перевантажені функціоналом і потребують адміністрування. Для одного кабінету це надлишково. А от локальна автономна програма — саме те, що потрібно в щоденній рутині лікаря УЗД [4].

Ще один важливий аспект — повторюваність дій у роботі лікаря. Практично кожен пацієнт проходить схожий шлях: реєстрація → обстеження → висновок → друк. І все це повторюється щодня. Саме це дозволяє автоматизувати процеси максимально ефективно, без втрати гнучкості.

На практиці це означає, що програма має зберігати шаблони висновків, пам'ятати останні дії користувача, запам'ятовувати структуру даних. EchoMed реалізує це через функцію автоматичного підвантаження останнього обстеження при повторному зверненні. Це дає змогу працювати швидше, а головне — точніше.

Інтерфейс — не менш важлива складова. У медичному середовищі користувачами часто є люди з різним рівнем комп'ютерної підготовки. Тому візуально зрозуміле меню, великі кнопки, чітка структура вкладок — усе це не

просто зручність, а необхідність. У додатку реалізовано саме таку логіку: мінімум кліків, максимум доступу.

EchoMed не вимагає авторизації в мережі, не потребує оновлень через інтернет, не передає дані стороннім серверам. Це критично для багатьох кабінетів, де робоча станція не має доступу до мережі з міркувань безпеки або просто через технічні обмеження.

І насамкінець, важливо, що розробка такого рішення дозволяє зробити не тільки крок у бік цифровізації, а й сформувати платформу для подальшого вдосконалення. У майбутньому EchoMed може бути інтегрований у більші медичні системи або доповнений модулем статистики, аналітики чи формування звітів.

Таким чином, предметна область — це не лише набір технічних вимог. Це реальна картина щоденної роботи, яка формує запит на просте, автономне, ефективне рішення. Саме в цій точці починається актуальність розробки, її зміст і практична цінність.

1.2 Аналіз аналогічного програмного забезпечення

Ринок медичного програмного забезпечення в Україні та світі поступово зростає, хоча залишається фрагментованим і орієнтованим переважно на великі медичні заклади. Серед доступних рішень — як масштабні медичні інформаційні системи (МІС), так і окремі модулі чи спрощені програми для автоматизації вузькопрофільних процесів. Аналіз аналогів дозволяє виявити типові підходи до обліку пацієнтів, оцінити функціонал конкурентів і виявити недоліки, які можна усунути при розробці власного додатку.

Одним із найпоширеніших вітчизняних продуктів є система МІС «Доктор Елекс». Вона використовується у приватних клініках і охоплює широкий спектр завдань: електронний запис, фінансовий облік, формування медичних карток, призначення лікування, ведення складу тощо. Програма має веб-інтерфейс і

потребує підключення до мережі. Її функціонал гнучкий, але водночас перевантажений для потреб одного лікаря УЗД. Установка, ліцензія, навчання персоналу — усе це створює додаткові витрати й обмеження [5].

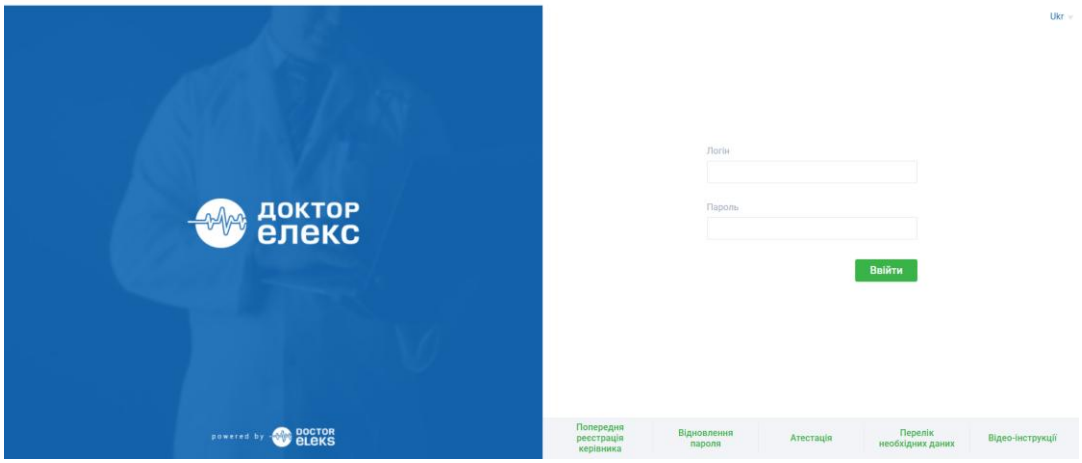

Рис 1.1 – Головне меню системи

Ще один приклад — HEALTH24, платформа для онлайн-обліку пацієнтів. Вона дозволяє лікарям вести картки, зберігати історію звернень, додавати документи й вести статистику. Однак це рішення передбачає щомісячну оплату, обов’язкову реєстрацію та підключення до інтернету. У багатьох кабінетах УЗД — особливо в державних установах або приватних практиках — відсутня стабільна мережа або не дозволено зберігати дані в хмарі. Тому HEALTH24 не є придатним для автономної роботи [6].

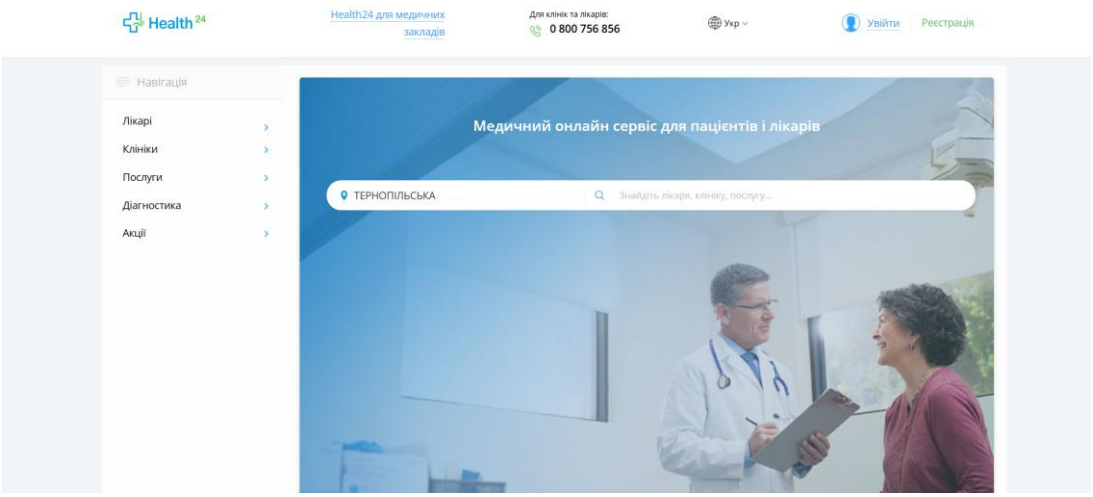

Рис 1.2 – Головне меню системи «HEALTH24»

Існують також рішення у вигляді Excel-шаблонів чи самописних локальних баз, створених самими лікарями або ентузіастами. Вони мають низький поріг входу, проте є вкрай вразливими до втрати даних, дублювання записів, помилок при пошуку або сортуванні. До того ж більшість таких рішень не підтримують збереження висновків у зручному вигляді, тим більше — генерацію PDF-файлів або фільтрацію за обстеженнями.

На міжнародному ринку варто згадати OpenEMR — систему з відкритим кодом для комплексного ведення медичних записів. Її перевага — безкоштовність і масштабованість. Втім, вона орієнтована на клініки з IT-персоналом і вимагає знань адміністрування для налаштування. Для лікаря УЗД, який працює самостійно, це надмірно складне рішення [7].

Усі зазначені системи мають спільну ознаку: вони або занадто складні й об’ємні, або не враховують специфіки щоденної роботи лікаря-діагноста. УЗД — це швидкий потік обстежень, де немає часу на заповнення десятків форм, реєстрацію в мережі чи довгі оновлення. Більшість універсальних МІС не оптимізовані для цієї задачі.

EchoMed, на відміну від перелічених рішень, створювався з нуля під конкретну модель роботи: локально, без інтернету, без реєстрацій, із мінімалістичним, але функціональним інтерфейсом. Він вирішує лише ті задачі, які справді потрібні лікарю: створення картки пацієнта, додавання висновку, фільтрація, пошук, генерація результату. Все інше — прибране.

Додаток також має перевагу в швидкості запуску, адаптації до будь-якої робочої машини й можливості миттєвого друку. Його структура проста: база SQLite, форма для введення даних і кнопка для збереження результату. Але ця простота — не обмеження, а фокус на головному.

Рис 1.3 – Головне меню системи «EchoMed»

В результаті, EchoMed не є прямим конкурентом жодній із вищезгаданих систем. Він — альтернатива. Це компактний інструмент для лікаря УЗД, який цінує свій час, не має доступу до великих IT-ресурсів і хоче просто працювати, а не адмініструвати. Його створено як відповідь на ті недоліки, які систематично спостерігаються в аналогах: перевантаженість, залежність від інтернету, складність налаштування й не прорахованість вузькопрофільної специфіки.

Саме тому наш додаток пропонує зручні рішення, простий, інтуїтивно зрозумілий та швидкісний інтерфейс, з котрим зможе розібратися людина будь якого віку. Що відповідає загальнодоступним вимогам та потребам одного із приватних кабінетів, власне під який і розроблялось дане програмне забезпечення.

1.3 Постановка задачі

Проєктування будь-якого прикладного програмного забезпечення починається з чіткого розуміння цілей, які воно повинне реалізувати. У цьому

випадку завданням стало створення автономного десктопного додатку для обліку пацієнтів кабінету ультразвукової діагностики, який буде простим у використанні, швидким у роботі та зручним для зберігання й аналізу даних.

Програма повинна вирішувати низку пов’язаних між собою задач. Перша — забезпечити можливість швидкої реєстрації пацієнта. Інформація про особу, тип обстеження, дата й висновок мають вноситися за декілька секунд. Дані зберігаються у внутрішній базі й відображаються у загальній таблиці. Це дозволяє уникнути дублювання та втрати записів, які трапляються під час ведення ручного журналу [8, 9].

Другою задачею стало формування висновків. Необхідно, щоб лікар міг одразу підготувати текст результату обстеження, відредагувати його, зберегти, а за потреби — надрукувати або експортувати у форматі PDF. Таким чином, пацієнт отримає документ, а дані залишаться в електронному архіві.

Рис 1.4 – Форма створення нового огляду пацієнта

Третій напрям — можливість пошуку та фільтрації. Програма повинна надати засіб перегляду історії обстежень для одного пацієнта, а також загальну вибірку за датами чи типами процедур. Це критично для клінічного аналізу та формування тенденцій.

Ще одне завдання — створити гнучку, але просту структуру бази даних. У межах цієї дипломної використовується SQLite, що забезпечує автономність, високу швидкість і відсутність залежності від серверів або зовнішніх бібліотек [10].

Окрему увагу слід звернути на інтерфейс користувача. Завдання — зробити його інтуїтивним, з мінімумом дій для виконання ключових операцій. Наприклад, при натисканні «Новий пацієнт» відкривається вікно з порожніми полями, які заповнюються вручну — без випадаючих довгих меню чи зайвих кнопок.

Для цього передбачена структура головного вікна, де відображається таблиця пацієнтів, а також основні кнопки дій: додати, змінити, видалити, додаткові кнопки: сформувати висновок, експорт, друк — розміщені в відповідному вікні.

З технічного боку, програма повинна:

- зберігати всі дані у локальному середовищі;
- не потребувати встановлення серверної частини;
- працювати без доступу до інтернету;
- підтримувати друк або збереження у форматі PDF;
- мати стабільну структуру бази з можливістю розширення.

Останнім компонентом задачі стало створення структури вихідного коду, яка дозволить у майбутньому легко змінювати логіку додатку, додавати поля, змінювати шаблон висновку або структуру таблиці без переписування всієї програми. Це важливо з огляду на практичне використання розробки у медичній сфері, де вимоги можуть змінюватися досить часто [11].

Таким чином, задача розробки EchoMed включає одночасно і технічну, і функціональну, і практичну складові. Усі вони спрямовані на створення інструменту, який буде корисним у реальному щоденному середовищі лікаря-діагноста.

1.4 Вимоги до системи

Щоб створити справді зручний та функціональний інструмент для лікаря УЗД, важливо чітко визначити вимоги до системи. Це стосується як функціональної, так і технічної частини проєкту. Всі параметри мають бути обґрунтованими з точки зору практичного застосування — у конкретному робочому середовищі, без ускладнень чи залежностей.

Функціональні вимоги поділяються на базові та розширені. До базових належать: можливість додавати, редагувати та видаляти записи пацієнтів; збереження висновків; вивід інформації у вигляді таблиці; пошук за ПІБ та фільтрація за датами. Усе це реалізовано через компактний інтерфейс без складних меню.

Розширені функції включають: експорт висновку у PDF, автоматичне збереження копії результату в базі, можливість друку з попереднього перегляду, підтримку архіву пацієнтів і ведення історії звернень. Програма також повинна мати базову перевірку коректності введених даних (наприклад, запобігання збереженню порожніх полів або повторів).

З технічного боку, програма повинна мати мінімальні системні вимоги, щоб працювати навіть на старих офісних комп’ютерах без оновлень. EchoMed не потребує інсталяції: достатньо просто запустити файл. Уся інформація зберігається локально у файлі бази даних *.db.

Системні вимоги виглядають так:

- ОС: Windows 7/10/11;
- Оперативна пам’ять: від 512 МБ;
- Місце на диску: від 50 МБ;
- Наявність принтера (для друку висновків);
- Бажано: встановлений PDF-принтер (наприклад, Microsoft Print to PDF);

Рис 1.5 - Вікно попереднього перегляду карти пацієнта перед друком

Окрема категорія — вимоги до інтерфейсу. Вони не менш важливі, ніж функціональні. Програма повинна запускатися за 1–2 секунди, мати великі, чітко підписані кнопки, світлий або темний фон і контрастні поля введення. Усі елементи повинні мати логічне розміщення кнопок.

Рис 1.6 – Головне меню з чітким розміщенням кнопок

Серед вимог безпеки — відсутність мережевих запитів і збереження всіх даних локально. Це забезпечує конфіденційність і дозволяє використовувати додаток навіть у кабінетах без доступу до інтернету. Дані не передаються

стороннім серверам, не синхронізуються з хмарою й не потребують зовнішніх API [12].

У проєкті не закладено підтримки користувачів або багаторівневого доступу — це свідомий вибір. Весь функціонал розраховано на одного лікаря, який і веде прийом, і фіксує обстеження. Тому логіки авторизації або розмежування прав не передбачено.

На основі наведеного можна сказати, що вимоги до EchoMed є мінімалістичними, але чіткими. Вони повністю відповідають ситуації, коли потрібне надійне рішення для щоденного застосування без зайвого коду чи адміністрування.

Це дозволяє зосередитися на головному — забезпеченні стабільної, швидкої та зручної роботи з пацієнтами, не перевантажуючи систему зайвими модулями. Такий підхід зменшує кількість потенційних точок відмови, що критично важливо для роботи в реальному часі. Відсутність авторизації не є недоліком у контексті індивідуального використання: вона спрощує доступ до програми та усуває потребу у збереженні додаткових облікових даних.

У разі розширення функціоналу, коли додаток буде використовуватися в колективній практиці або в умовах багатопрофільного центру, архітектура дозволяє впровадити багаторівневу авторизацію через додаткові класи контролю доступу та ролей. Але на початковому етапі проєкт залишився легким і ефективним, що відповідає його першочерговій меті — створення інструменту для індивідуального лікаря, який хоче автоматизувати власний прийом.

Простота інтерфейсу також є перевагою. Оскільки користувач працює з однією основною формою, навігація не викликає труднощів навіть у людей без технічної освіти. Інтуїтивно зрозумілий підхід до введення, збереження й перегляду інформації мінімізує ймовірність помилок під час заповнення

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ІНТЕРФЕЙСУ ДОДАТКУ

Щоб будь-яка програма працювала не лише правильно, а й зручно — її потрібно добре спроєктувати. Архітектура програмного забезпечення — це як каркас будинку: від неї залежить і стійкість, і простота розширення, і комфорт у використанні. Саме тому перед реалізацією EchoMed була побудована чітка логічна структура компонентів.

На цьому етапі я сформував загальну схему взаємодії модулів, визначив основні сутності, продумав внутрішню логіку бази даних та передбачив основні сценарії користування. Це дало змогу уникнути помилок у майбутньому та створити систему, яка не заплутує, а навпаки — допомагає швидко зорієнтуватися в потоці дій. Окрім функціональної архітектури, критично важливим став дизайн інтерфейсу. У програмі, якою користуватиметься лікар упродовж всього робочого дня, не має бути нічого зайвого. Кожна кнопка, кожен елемент вікна має бути на своєму місці. Саме тому проєктування зовнішнього вигляду велося паралельно з логікою — крок за кроком, враховуючи реальні сценарії використання в медичному кабінеті.

2.1 Вибір архітектури

Під час розробки EchoMed я обрав модель монолітної архітектури з чітко розділеними логічними блоками. Це дозволило створити просту, автономну систему, яка працює без сервера, інтернету чи додаткових служб. Основна ідея — уся логіка, база даних і графічний інтерфейс об’єднані в одному програмному модулі, що запускається одним кліком.

Такий підхід має кілька важливих переваг. По-перше, це зменшує складність підтримки. Програма не потребує налаштування середовища чи інсталяції

залежностей. Все вже зібрано в одному каталозі, включно з базою даних, і може бути перенесено на інший комп’ютер без втрати функціоналу. По-друге, монолітна структура дозволяє швидше реагувати на зміни, не розділяючи логіку по різних сервісах чи компонентах [13, 14].

У структурі EchoMed закладено базовий розподіл на три ключові частини: графічний інтерфейс користувача (GUI), модуль логіки та модуль роботи з базою даних. GUI побудовано за допомогою бібліотеки PyQt5, що забезпечує кросплатформеність і зручність побудови форм. Візуальні елементи організовані в окремі вікна або вкладки, кожне з яких відповідає за певну функцію: реєстрація, історія, висновки, налаштування.

Модуль логіки відповідає за обробку подій, перевірку даних, управління інтерфейсом. Тут прописані всі дії, що активуються при натисканні кнопок: додавання нового запису, редагування, очищення полів, експорт у PDF. Код розбитий на класи, які відповідають за окремі частини — наприклад, обробка висновків винесена в окрему функцію, що спрощує супровід.

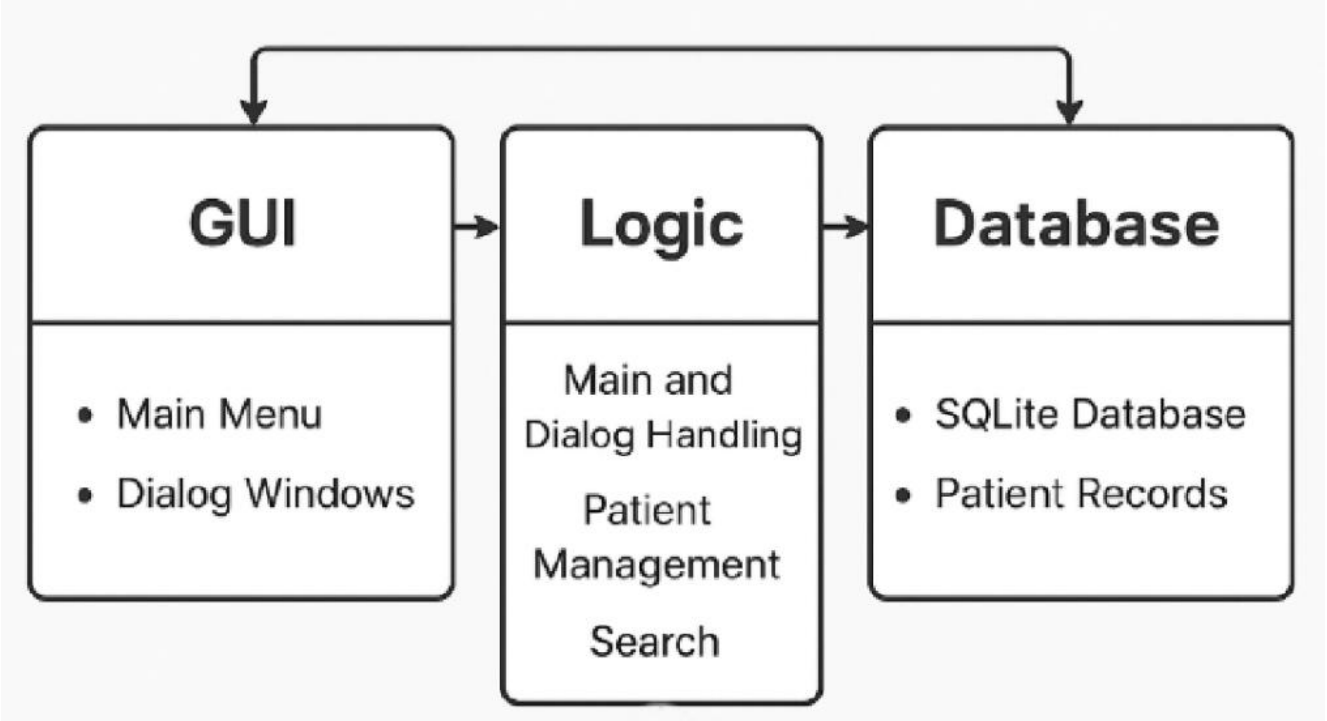

Рис 2.1 - Схема архітектури EchoMed: взаємодія GUI, логіки та бази даних

Окремо стоїть модуль бази даних. Він реалізований на SQLite — вбудованій СУБД, яка не потребує окремого сервера. Це дозволило уникнути складного налаштування й зробити збереження даних автономним. Кожна таблиця має свою чітко визначену структуру: наприклад, таблиця patients містить поля для імені, дати, висновку, унікального ID.

Для збереження й читання інформації використовуються SQL-запити, які прописані прямо в коді. Такий підхід спрощує контроль і робить логіку прозорою — можна в будь-який момент вручну переглянути файл бази, змінити дані або створити резервну копію [15].

Підхід із поділом на модулі також дозволяє в подальшому модифікувати систему. Наприклад, замінити GUI на веб-інтерфейс або додати нові поля — без переробки всієї структури. Це відкриває перспективу розвитку додатку за межі кабінету УЗД, наприклад, у кабінети функціональної діагностики чи лабораторій.

У підсумку, обрана архітектура виявилася не лише зручною, а й гнучкою. Вона дозволила створити програму, яка просто запускається, швидко працює, не потребує оновлень і дає змогу лікарю повністю зосередитися на своїй основній роботі — діагностиці.

2.2 UML-діаграми класів

Для зручності проєктування та кращого розуміння структури програми EchoMed було створено UML-діаграму класів, яка відображає основні сутності додатку, їхні зв’язки та реалізовані методик [16]. Вона стала фундаментом під час побудови логіки взаємодії між графічним інтерфейсом, базою даних і функціональними модулями.

На діаграмі чітко видно, що центральне місце займає клас MainWindow, який є спадкоємцем QMainWindow. Саме він відповідає за завантаження інтерфейсу, керування подіями та взаємодію з користувачем. У його складі — методи для

роботи з таблицею пацієнтів (`setup_table, load_patients, reloading_table`), обробки подій (`mousePressEvent, mouseMoveEvent, mouseReleaseEvent`), а також функції додавання, редагування, сортування й пошуку пацієнтів (`open_add_patient, search_patients_by_name, sort_table_by_date, toggle_favorite`).

Безпосередньо з головним вікном взаємодіють діалоги, побудовані на основі QDialog. Найпростіші з них — `ConclusionDialog`, `PatientFoundDialog` — мають лише метод __init__, що вказує на їхню допоміжну роль. Основна логіка сконцентрована у `PatientMenu`, де реалізовано такі події, як `save_patient(), setup_comboboxes()` та обробку взаємодії з мишкою. Цей клас працює з формою додавання нового пацієнта.

```
def add_patient(self):
    name = self.name_input.text()
    birthdate = self.birthdate_input.date().toString("yyyy-MM-dd")
    diagnosis = self.diagnosis_input.text()

    if name:
        self.db.insert_patient(name, birthdate, diagnosis)
        self.accept()
    else:
        QMessageBox.warning(self, "Помилка", "Поле імені не може бути порожнім.")
```

Лістинг 1 — Збереження нового пацієнта з AddPatientDialog

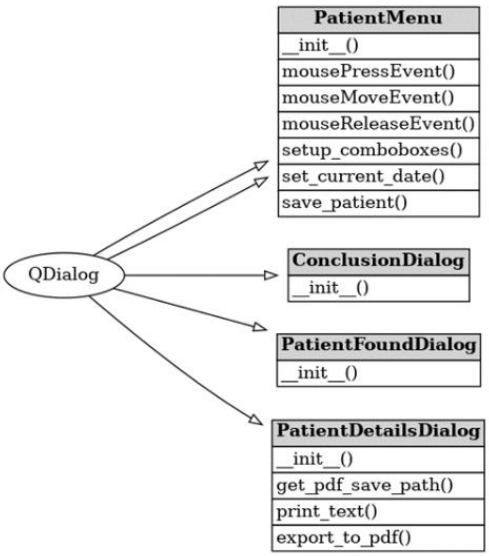

Рис 2.2 – UML-діаграма класів EchoMed

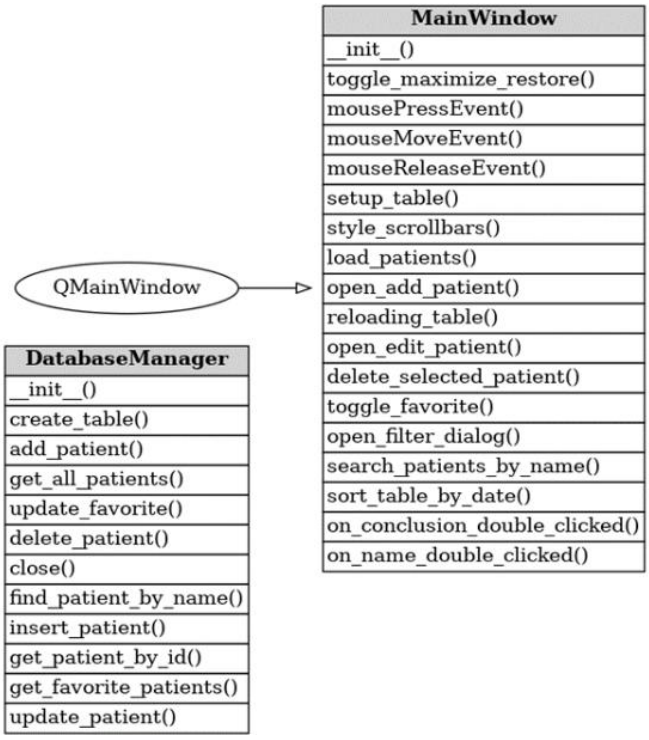

Рис 2.3 – Детальніша UML-діаграма класів EchoMed

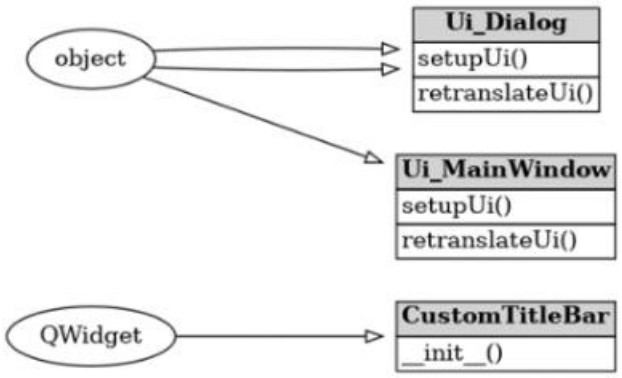

Рис 2.4 - UML-діаграма класів

Ще один важливий елемент — `PatientDetailsDialog`. Цей діалог виконує функцію виводу й друку медичного висновку. Тут містяться методи `get_pdf_save_path`, `print_text`, `export_to_pdf`. Вони відповідають за генерацію підсумкового документа, його форматування та збереження у вигляді PDF-файлу [17].

Окремо в структурі представлено клас `DatabaseManager`, який не пов’язаний із вікнами напряму. Він реалізує усю логіку взаємодії з базою даних: створення таблиць (`create_table`), додавання й редагування пацієнтів (`add_patient`, `update_patient`, `delete_patient`), пошук (`find_patient_by_name`, `get_patient_by_id`), а також роботу з обраними записами (`update_favorite`, `get_favorite_patients`).

```
class DatabaseManager:
    def __init__(self, db_name="patients.db"):
        self.connection = sqlite3.connect(db_name)
        self.cursor = self.connection.cursor()
        self.create_tables()

    def create_tables(self):
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS patients (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                birthdate TEXT,
                diagnosis TEXT,
                favorite INTEGER DEFAULT 0
            )
```

```
        ''')
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS reports (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                patient_id INTEGER,
                report_text TEXT,
                FOREIGN KEY (patient_id) REFERENCES patients(id)
            )
        ''')
        self.connection.commit()
```

Лістинг 2 — Створення таблиць у класі DatabaseManager

Такий поділ дозволяє утримувати код чистим і розділити обов'язки між інтерфейсом та даними.

Інтерфейсні класи `Ui_Dialog` та `Ui_MainWindow`, як видно з діаграми, мають лише методи `setupUi()` і `retranslateUi()` — вони генеруються автоматично за допомогою Qt Designer. Їх призначення — побудова зовнішнього вигляду вікон.

Ще однією цікавою деталлю є клас `CustomTitleBar`, який успадковує `QWidget` і відповідає за кастомізацію заголовка вікна. У ньому реалізовано лише метод __init__, адже його функціонал мінімальний: візуальна адаптація стандартного вигляду PyQt-вікон під дизайн EchoMed.

Усі ці класи не пов'язані між собою через спадкування, крім зв'язку з базовими Qt-компонентами (`QMainWindow, QDialog, QWidget`). Це дозволяє утримати архітектуру простою та легкою для розширення.

UML-діаграма дала змогу зберегти порядок у структурі програми, уникнути дублювання логіки та забезпечити стабільність. Якщо знадобиться розширити функціонал або винести частину логіки в окремий модуль, зміни не зачеплять основні компоненти. Такий підхід значно підвищує життєздатність проєкту на практиці.

2.3 Модуль бази даних (SQLite)

Для збереження даних у програмі EchoMed використано локальну реляційну базу даних SQLite. Це легка вбудована СУБД, яка не потребує окремого сервера або клієнта — достатньо одного файлу *.db, що зберігається поруч із додатком.

Такий підхід повністю відповідає цілям проєкту: автономна робота, швидкий доступ, конфіденційність [17, 18].

База даних містить одну основну таблицю — `patients`, яка включає всі необхідні поля для ведення обліку пацієнтів:

- id — первинний ключ (INTEGER, AUTOINCREMENT)
- name — ПІБ пацієнта (TEXT)
- birthdate — дата народження (TEXT)
- gender — стать (TEXT)
- address — місце проживання (TEXT)
- exam_type — тип обстеження (TEXT)
- exam_date — дата обстеження (TEXT)
- conclusion — висновок лікаря (TEXT)
- is_favorite — статус обраного запису (INTEGER: 0 або 1)

Формат збереження обрано текстовий (TEXT), оскільки він найзручніший для подальшого відображення в інтерфейсі. Тип поля `is_favorite` дозволяє реалізувати додаткову функцію — швидкий доступ до вибраних записів, наприклад, для частих пацієнтів.

Модуль взаємодії з базою реалізовано у вигляді окремого класу `DatabaseManager`. Саме він забезпечує повний набір функцій для роботи з таблицею: створення, додавання, оновлення, пошук, видалення, вибірка. Під час ініціалізації класу викликається метод `create_table()`, який перевіряє наявність базової структури та створює її, якщо файл порожній або відсутній.

Додавання нового запису виконується через `add_patient()`, що приймає набір даних із форми. Для оновлення інформації використовуються методи

`update_patient()` та `update_favorite()`, а вибірка реалізована функціями `get_all_patients()`, `find_patient_by_name()`, `get_patient_by_id()`.

Кожен SQL-запит виконується безпосередньо з Python-коду за допомогою модуля `sqlite3`.

Такий формат дозволяє гнучко керувати структурою таблиці — у разі потреби її можна змінити без втрати даних. Також легко створюються резервні копії — достатньо скопіювати файл `patients.db`.

Однією з переваг SQLite є її прозорість: базу можна переглянути в будь-якому SQLite-менеджері, наприклад, DB Browser for SQLite. Це корисно у разі технічного супроводу або аналізу помилок. Також база працює без драйверів, тому її підтримка доступна «з коробки» на більшості систем.

Ще однією перевагою є швидкодія: всі дані завантажуються локально, без затримок, пов’язаних із мережею. Це особливо важливо в умовах швидкого потоку пацієнтів, коли кожна секунда взаємодії з інтерфейсом має значення.

І хоча структура зараз обмежується однією таблицею, за потреби її можна розширити. Наприклад, додати таблицю users для авторизації або `logs` для журналу дій. Така можливість уже закладена в архітектуру, і реалізація не потребуватиме повного переписування логіки.

Модуль `DatabaseManager` також відповідає за контроль помилок — якщо під час виконання запиту виникає виняток, програма обробляє його без завершення роботи. Це робить систему більш стійкою та готовою до щоденного використання.

2.4 Алгоритми взаємодії

Для забезпечення зручності роботи з додатком EchoMed необхідно було не лише реалізувати функції, а й продумати логіку взаємодії — тобто, як саме користувач буде виконувати ті чи інші дії. Це включає послідовність натискань,

обробку подій, збереження інформації, і навіть такі дрібниці, як повернення до таблиці після введення даних.

Розглянемо приклад — додавання нового пацієнта. Цей сценарій є базовим, тому саме з нього починався проєкт. Лікар натискає кнопку «Додати пацієнта» — запускається метод `open_add_patient()`, який відкриває форму `PatientMenu`. Там відображаються порожні поля для введення ПІБ, дати народження, типу обстеження тощо.

Після введення даних лікар натискає кнопку «Зберегти», яка викликає метод `save_patient()`. Спочатку запускається валідація: якщо поля не заповнено або дати мають некоректний формат — з'являється повідомлення про помилку. Якщо все коректно — інформація передається до `DatabaseManager.add_patient()`, який створює SQL-запит і записує дані в таблицю `patients`.

Після цього вікно закривається, а головна таблиця оновлюється методом `reloading_table()` — і новий запис одразу з'являється на екрані.

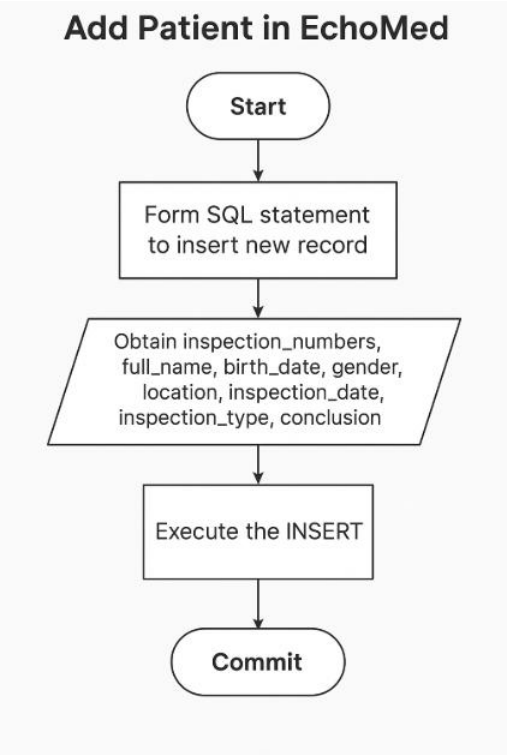

Рис 2.5 - Блок-схема алгоритму додавання пацієнта в EchoMed

Інший тип взаємодії — перегляд історії пацієнта. Якщо лікар хоче подивитись деталі — він клікає двічі по ПІБ, що викликає `on_name_double_clicked()`. Система звертається до `DatabaseManager.get_patient_by_id()` і відкриває діалог `PatientDetailsDialog`, де відображено повну інформацію, а також доступні дії — «Друк» або «Експорт у PDF».

Алгоритм обробки натискання на кнопку «Експорт» такий: метод `export_to_pdf()` бере текст з висновку, форматує його відповідно до шаблону, відкриває діалог вибору шляху (через `get_pdf_save_path()`), генерує документ і зберігає його.

Якщо натискається «Улюблене» — змінюється статус `is_favorite` запису. Це реалізовано через метод `update_favorite()`, який оновлює поле у таблиці `patients`. Зміни одразу видно в інтерфейсі, бо таблиця автоматично оновлюється.

Окремо варто описати логіку пошуку пацієнтів. Вона починається з виклику `search_patients_by_name()` — функція зчитує введений текст і виконує SQL-запит до бази. Якщо знайдено декілька результатів — відкривається вікно `PatientFoundDialog` із вибором. Якщо один — одразу відкривається його картка. У разі відсутності збігів виводиться повідомлення.

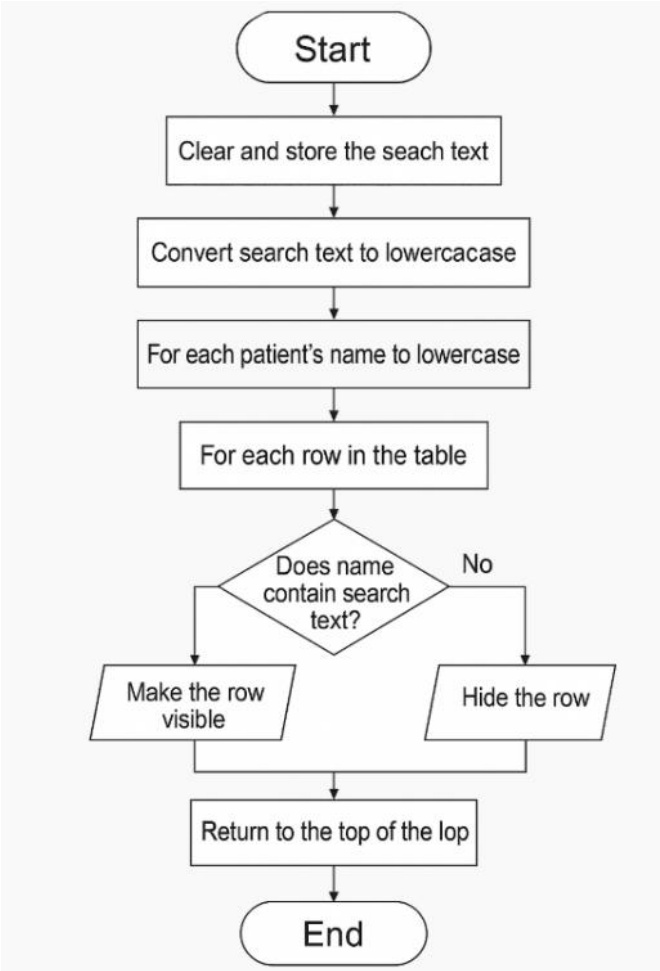

Рис 2.6 - Блок-схема алгоритму пошуку пацієнта за ім’ям

Логіка всіх дій у програмі чітка, без зайвих кроків. Кожен сценарій взаємодії — від реєстрації до друку — відбувається в межах кількох кліків. Це дозволяє лікарю зосередитися на діагностиці, не витрачаючи час на технічні операції

Перевагою такої структури є також ізольованість обробки даних. Головне вікно керує лише інтерфейсом. Уся база — в `DatabaseManager`, усі документи — у відповідному діалозі. Завдяки цьому код не змішується, і зміни в одному модулі не порушують роботу іншого [19]. Так побудовані алгоритми забезпечують просту, але ефективну модель взаємодії, яка добре працює в умовах реального кабінету УЗД — швидко, надійно й без перенавантаження інтерфейсу.

Такий поділ відповідальності суттєво полегшує підтримку та масштабування системи. Наприклад, у разі оновлення бази даних достатньо змінити лише методи

`DatabaseManager`, не торкаючись UI-компонентів. А при додаванні нового формату експорту — зміни відбуваються лише в модулі створення документів.

Це дає змогу швидко адаптувати програму до потреб конкретного медичного закладу або розширити її функціональність. Крім того, ізоляція логіки сприяє підвищенню безпеки — користувач не має прямого доступу до внутрішніх даних, і кожна дія проходить через контрольовані методи.

Завдяки такому підходу також полегшується тестування окремих компонентів — кожен модуль можна перевіряти незалежно, що знижує ймовірність помилок при впровадженні змін. Наприклад, функції збереження та пошуку пацієнтів можна протестувати без запуску графічного інтерфейсу, використовуючи лише підключення до бази. Це значно прискорює процес налагодження.

Ще однією перевагою є гнучкість розробки: за потреби можна легко замінити або оновити окремі елементи системи без порушення цілісності застосунку. Наприклад, якщо згодом буде потрібно інтегрувати хмарне сховище або реалізувати онлайн-доступ до даних, це можна зробити, модифікуючи лише відповідний програмний модуль, не змінюючи загальну архітектуру. Такий рівень адаптивності є ключовим фактором довготривалої підтримки та розвитку програмного забезпечення в умовах швидкоплинних технологічних змін.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Після того як було сформовано архітектуру, спроєктовано базу даних і створено інтерфейс, розпочався найвідповідальніший етап — власне реалізація логіки додатку EchoMed. Від початку метою було створити рішення, яке буде стабільним, простим у використанні та достатньо гнучким для подальшого розширення. Цей розділ містить опис процесу програмування основних модулів, тестування реалізованих функцій, аналіз результатів та рекомендації щодо впровадження розробленої системи у реальні медичні установи.

3.1 Розробка програмного забезпечення

Розробка додатку EchoMed почалася з підготовки середовища. Для створення настільного застосунку я обрав Python як основну мову програмування, а для побудови графічного інтерфейсу — бібліотеку PyQt5. Обробку даних забезпечує вбудована СУБД SQLite, яка не потребує встановлення серверів і дозволяє працювати з одним файлом бази, зручним для резервного копіювання. Такий стек технологій був обраний не випадково — він забезпечує гнучкість, простоту підтримки та достатню продуктивність для типових завдань лікаря.

Структура програми розділена на окремі файли й модулі, кожен із яких відповідає за свій функціонал. Так, наприклад, файл main.py відповідає за ініціалізацію програми та запуск головного вікна. У ньому створюється екземпляр класу `MainWindow`, який побудований на основі PyQt-класу `QMainWindow` і є головним контейнером для виводу таблиці пацієнтів, панелей керування та кнопок.

Рис 3.1 - Головне вікно програми EchoMed у режимі виконання

Інтерфейс створено за допомогою Qt Designer [20], а потім перетворено в Python-код через pyuic5. Графічна частина інтерфейсу міститься у файлі `ui_mainwindow.py`, де реалізовано метод `setupUi()`. Там описано компонування таблиці пацієнтів, кнопки «Додати», «Редагувати», «Пошук», фільтрацію за улюбленими, а також кастомізовану панель заголовка через клас `CustomTitleBar`.

Усі дії користувача обробляються у відповідних слотах — методах, пов’язаних із подіями. Наприклад, натискання кнопки «Додати пацієнта» викликає метод `open_add_patient()`, який відкриває діалог `PatientMenu`. У цьому діалозі користувач заповнює дані, які потім передаються в базу даних. Аналогічно працюють `edit_patient()`, `delete_patient()` і `search_patients_by_name()`.

База даних створюється і підтримується у класі `DatabaseManager`, що розміщений у файлі `database.py`. Він містить усю логіку взаємодії з SQLite: створення таблиці, додавання нових записів (add_patient), оновлення даних (`update_patient`), видалення (`delete_patient`) та вибірки (`get_all_patients`, `get_patient_by_id`). Запити виконуються за допомогою стандартного модуля sqlite3.

Особливу увагу під час реалізації я приділив правильному збереженню та обробці медичних висновків. У діалозі `PatientDetailsDialog` реалізовано

функцію `export_to_pdf()`, яка дозволяє зберегти висновок лікаря у вигляді PDF-файлу. Для цього використано бібліотеку `reportlab`, яка формує документ із заданими полями, включаючи дату, ПІБ пацієнта, тип обстеження та сам висновок. Це дозволяє створити цифровий архів висновків, який можна легко друкувати або пересилати.

Для зручності користувача реалізовано фільтрацію пацієнтів за статусом «Вибрані». Ця опція корисна, коли один і той самий пацієнт часто звертається за обстеженням. У базі це реалізується через поле `is_favorite`, а в інтерфейсі — через зміну значка у таблиці. Функціонал оновлення запису забезпечується методом `update_favorite()`.

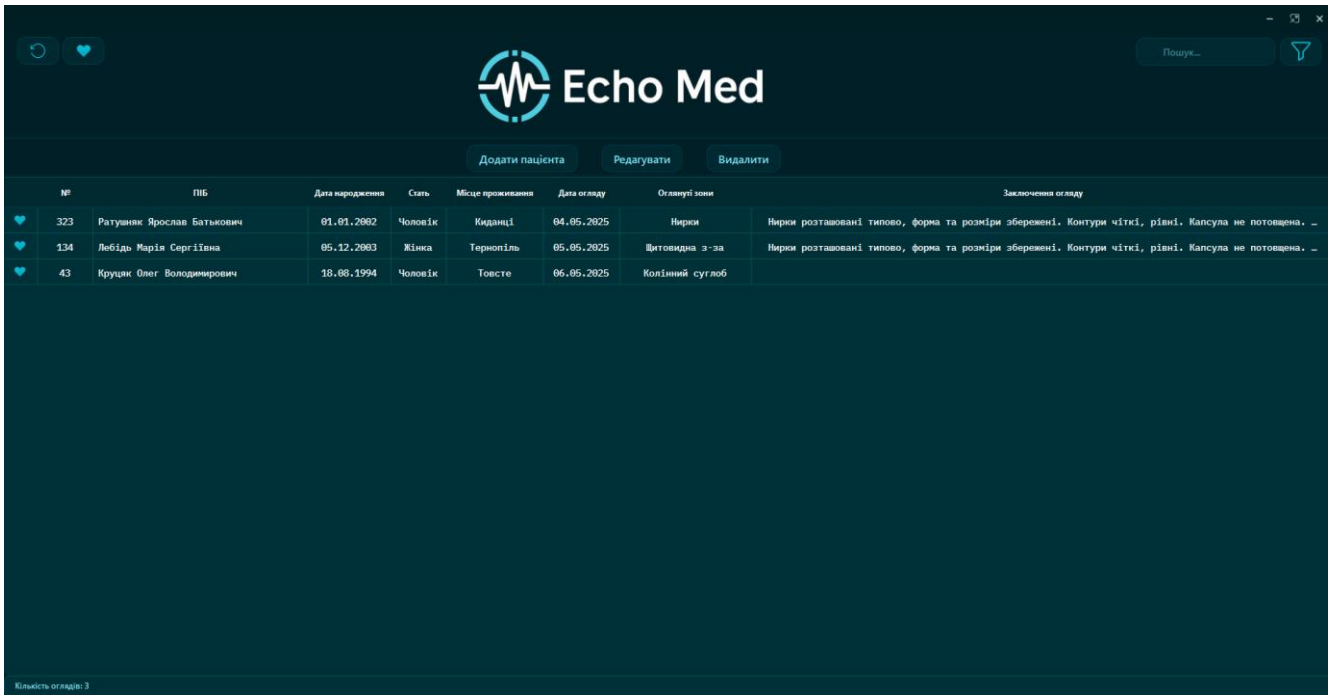

Рис 3.2 – Головне вікно ПЗ з застосуванням фільтрації з статусом «Вибрані»

Рис 3.3 – Вікно редагування карти пацієнта

Слід зауважити що функції додавання, редагування, видалення оглядів пацієнта – реалізовані через вибір певного ID та кліком на необхідну кнопку. В свою чергу повний опис заключення, та карта огляду пацієнта – викликається подвійним кліком на необхідне поле.

Також було реалізовано підказки при наведенні буквально на будь яку кнопку чи рядок, але основну роль це грає саме на рядку «заключення», адже там часто буде обширний текст який може не поміщатися в візуальне відображення таблиці.

Таке рішення дозволить заощадити час лікаря при перегляді попереднього заключення огляду пацієнта, який умовно приходив раніше.

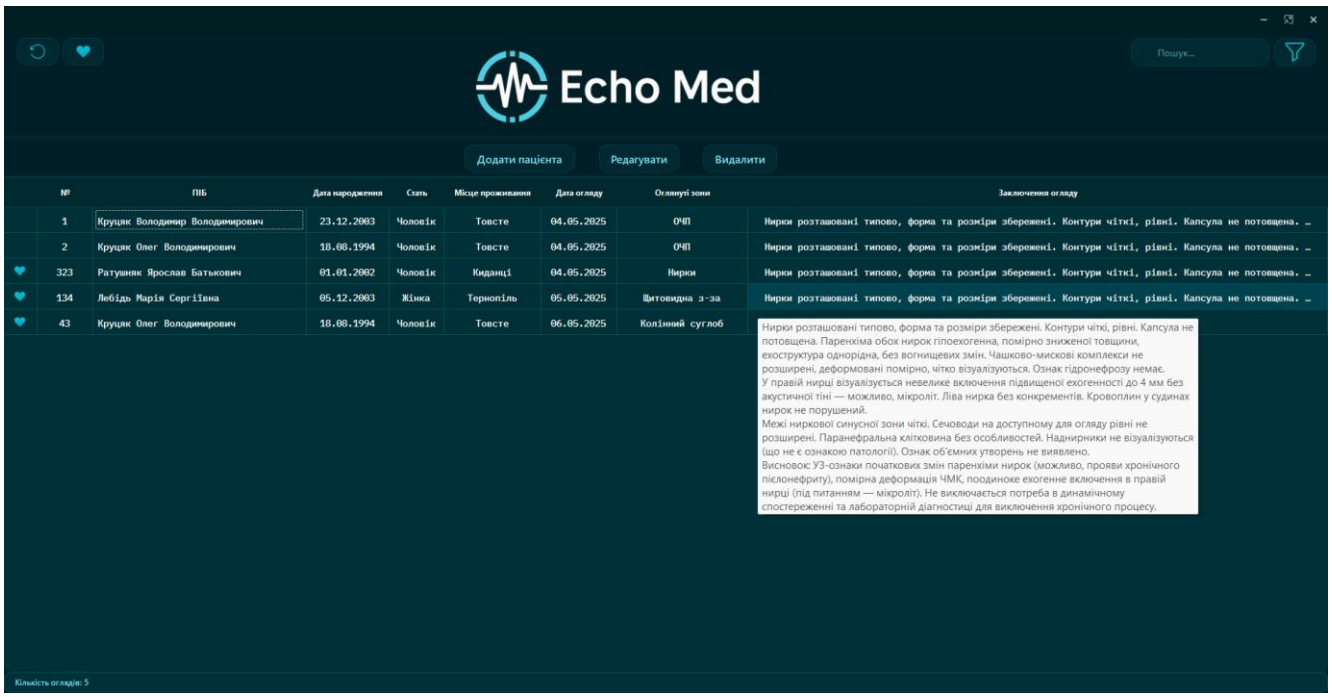

Рис 3.4 – Реалізація підказок для економії часу під час перегляду заключень

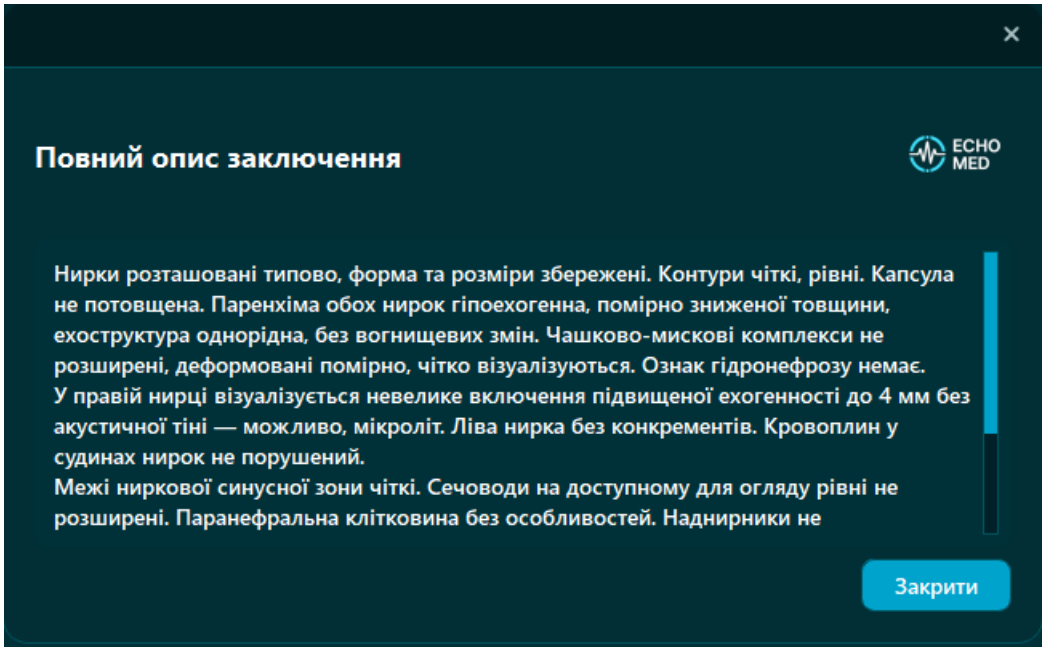

Рис 3.5 – Вікно з повним описом заключення

Рис 3.6 – Вікно перегляду та друку карти пацієнта з заключенням

Під час розробки я також передбачив можливість масштабування — у майбутньому до програми можна додати авторизацію, облік кількох користувачів, шифрування бази або хмарну синхронізацію. Структура коду дозволяє легко розширити функціонал без переробки основної логіки.

Таким чином, кожен етап розробки — від підключення базових модулів до реалізації візуальної частини — проводився з урахуванням зручності, надійності та простоти підтримки. На завершення було виконано перевірку працездатності всіх компонентів, після чого розпочалось повноцінне тестування, про яке детально йтиметься у наступному підрозділі.

3.2 Тестування програмного забезпечення

Після завершення етапу розробки кожного функціонального блоку програми було розпочато повноцінне тестування. Оскільки додаток EchoMed розроблявся як практичний інструмент для лікарів УЗД, основний акцент було зроблено на функціональній надійності та стійкості до помилкових дій. Усі тести проводилися вручну, з послідовним виконанням типових сценаріїв роботи користувача, а також з навмисним провокуванням нетипових ситуацій [21].

Однією з перших перевірених функцій було додавання пацієнта. Було протестовано як стандартний сценарій з повним набором даних, так і випадки з незаповненими обов'язковими полями. У випадку порожнього імені або некоректної дати висвітлювалося попередження — це підтвердило правильну роботу валідації у методі `save_patient()`. Також перевірено, що після натискання «Зберегти» форма очищується й новий запис відображається в таблиці.

Наступним етапом було тестування перегляду інформації про пацієнта. Для цього було створено кілька записів із різними наборами даних. Подвійне натискання на ПІБ у таблиці відкривало діалог `PatientDetailsDialog`, у якому успішно відображались усі поля, включно з висновком лікаря. Також було перевірено роботу кнопок «Друк» і «Експорт у PDF». Документи створювались без помилок, збереження відбувалося у вказану папку, форматування було коректним [22].

Важливу роль відіграло тестування бази даних. Спеціально було створено базу з великою кількістю записів — понад 100 фіктивних пацієнтів — для перевірки продуктивності. Таблиця завантажувалася без затримок, а всі запити працювали з нормальною швидкістю. Особливо тестувалась функція `search_patients_by_name()`: як за повним ПІБ, так і за частиною імені. У кожному випадку результати відображались коректно, включаючи ситуації з кількома збігами.

Окремо перевірялася робота з полем "Вибране". Було додано декілька записів, яким вручну змінювали статус через іконку. Після натискання відбувалася зміна значка, а стан фільтрації у головній таблиці також оновлювався. Зміни зберігались навіть після перезапуску програми, що підтвердило правильне збереження `is_favorite` у базі.

Тестувалися і нестандартні ситуації. Наприклад, раптове закриття програми під час редагування пацієнта, одночасне відкриття кількох вікон, спроба зберегти порожній PDF. Усі події оброблялись без аварійного завершення. Система або блокувала дію, або виводила повідомлення [23].

Також було виконано тестування оновлень — зміна даних у записі та їхнє збереження. У вікні редагування після зміни типу обстеження й дати висновок залишався недоторканим, а нові дані одразу з’являлися у таблиці.

Під час тестування було зафіксовано одну критичну помилку на ранньому етапі — при збереженні нового пацієнта без дати народження система не блокувала запис. Після внесення додаткової перевірки до `save_patient()` ця проблема була усунена.

Загалом у межах ручного тестування було опрацьовано понад 30 сценаріїв.

Усі основні функції спрацювали стабільно, жодної критичної помилки на фінальному етапі виявлено не було. Програма коректно реагувала як на правильні, так і на помилкові дії користувача, забезпечуючи стабільну роботу й зрозумілі повідомлення у разі винятків.

3.3 Результати тестування

Результати проведеного тестування дозволили дійти обґрунтованих висновків щодо працездатності, надійності та зручності використання програмного забезпечення EchoMed. Програма успішно пройшла всі основні перевірки, зокрема

перевірку введення, збереження та обробки даних, обробку винятків, а також сценарії взаємодії з базою та інтерфейсом [24].

Найперше варто відзначити стабільність роботи всіх основних функцій. Додавання, редагування, перегляд та видалення пацієнтів відбувалося без помилок. При спробах ввести некоректні або порожні дані програма адекватно реагувала — користувач отримував повідомлення з поясненням, а запис не зберігався. Це свідчить про коректно реалізовану валідацію та перевірку введення, що є критичним для медичного програмного забезпечення [23].

Особливо позитивно зарекомендував себе механізм експорту висновків до PDF. Документи створювалися швидко, із збереженням форматування, що дозволяє лікарям одразу отримати професійний вигляд результату. Усі перевірені поля відображались точно, без втрати важливої інформації, навіть при великому обсязі тексту у висновку.

Окрема увага приділялась тестуванню сценаріїв пошуку та фільтрації. Програма без затримок обробляла запити навіть при великій кількості записів. Алгоритм пошуку працював як за повним ім’ям, так і за його частиною, а статус «Вибрані» зберігався після перезапуску. Це важливо, зважаючи на реальні навантаження в кабінеті УЗД, де пацієнтів може бути кілька сотень.

Таблиця 1 - Результати функціонального тестування основних сценаріїв

№	Сценарій тестування:	Очікуваний результат:	Результат тесту:
1	Додавання пацієнта з усіма даними	Пацієнта додано, запис збережено у базі	Успішно
2	Спроба зберегти без імені	Виведено повідомлення про помилку, запис не створено	Успішно
3	Перегляд деталей про огляд пацієнта	Всі поля відображаються правильно	Успішно
4	Експорт заключення у PDF	Файл створено, форматування збережено	Успішно
5	Пошук за частиною імені	Коректний результат у таблиці	Успішно

Продовження таблиці 1

№	Сценарій тестування:	Очікуваний результат:	Результат тесту:
6	Редагування запису	Дані оновлено, відображено у таблиці	Успішно
7	Робота з позначкою «Вибрані»	Іконка змінюється, запис зберігається	Успішно
8	Раптове закриття програми під час редагування	Програма не аварійно завершується	Успішно
9	Створення запису без дати народження	Виведено повідомлення про помилку	Успішно
10	Фільтрація вибраних після перезапуску	Список оновлюється правильно	Успішно

Згідно з таблицею, усі ключові сценарії завершились успішно. Це дозволяє зробити висновок, що логіка взаємодії компонентів реалізована коректно, а виняткові ситуації — передбачені і належним чином обробляються. Виявлена на початковому етапі помилка зі збереженням запису без дати була усунена, після чого вона більше не проявлялась.

Позитивним є також те, що під час роботи не виявлено жодної критичної помилки, яка б призводила до аварійного завершення застосунку або втрати даних. Це особливо важливо для сфери охорони здоров’я, де збої в роботі можуть вплинути на якість обслуговування пацієнтів. Поведінка програми виявилась передбачуваною та стабільною.

Ще один аспект — продуктивність. Навіть при навантаженні в 100+ записів таблиця залишалась інтерактивною, а інтерфейс не гальмував. Це говорить про оптимальну реалізацію обробки бази даних та рендерингу UI-елементів, що базується на асинхронній логіці Qt та легкості SQLite [22].

Таким чином, проведене тестування підтвердило, що створене програмне забезпечення не лише відповідає заданим функціональним вимогам, а й має потенціал для реального використання в медичній практиці. Виявлені незначні

недоліки були усунені під час налагодження, що дозволило досягти високого рівня стабільності та функціональної завершеності.

Крім цього, варто відзначити загальну зручність використання системи. Інтерфейс виявився інтуїтивно зрозумілим, а усі базові дії — доступними навіть для користувачів без спеціальної технічної підготовки. Чітка структура вікон, зрозумілі підписи до кнопок, логічна послідовність дій дозволяють звести навчання персоналу до мінімуму. Це особливо актуально у сфері охорони здоров’я, де програмне забезпечення повинно бути не тільки функціональним, але й швидко адаптивним до користувача, який працює з великим потоком пацієнтів щодня.

У ході тестування також було відзначено, що програма швидко запускається, не створює навантаження на систему та не вимагає інсталяції сторонніх компонентів. Це робить її придатною для встановлення на будь-який сучасний комп’ютер в межах медичного кабінету без додаткових витрат чи ускладнень.

Завдяки використанню SQLite як вбудованого механізму зберігання даних, EchoMed не залежить від зовнішніх серверів чи мережевої інфраструктури. Це дозволяє забезпечити повну автономність роботи — навіть у випадках відсутності інтернету чи нестабільного з'єднання. Усі дані зберігаються локально, а їхній резервний експорт може бути організований вручну або через додаткові скрипти. Такий підхід особливо зручний для невеликих медичних закладів, де немає окремого ІТ-відділу.

Крім того, автономна архітектура дозволяє уникнути ризиків, пов’язаних із вразливостями мережевих з’єднань, що є критично важливим при роботі з конфіденційними медичними даними. Локальне збереження забезпечує кращий контроль над доступом до інформації, а відсутність необхідності в онлайн-сервісах мінімізує потенційні точки вторгнення.

Програма також не потребує спеціальних прав адміністратора для встановлення, що спрощує процес її розгортання в закладах з обмеженим технічним супроводом. EchoMed може запускатись з портативних носіїв або папок користувача, а оновлення застосунку можуть виконуватись вручну шляхом заміни виконуваного файлу без впливу на наявні бази даних.

Іншим важливим аспектом є енергоефективність та низьке споживання ресурсів: навіть при тривалому використанні програма не спричиняє навантаження на процесор чи оперативну пам’ять. Це дозволяє запускати EchoMed паралельно з іншими медичними чи адміністративними системами без зниження продуктивності робочого місця. Усе це робить програму привабливим рішенням у сфері цифровізації медичних послуг.

Крім того, враховано зручність перенесення програми між комп’ютерами — достатньо скопіювати каталог з виконуваним файлом і базою даних. Немає потреби в установці драйверів або окремих бібліотек, що значно скорочує час впровадження.

Програма не вимагає доступу до мережі, працює локально, а це виключає ризики втрати даних через нестабільне підключення або зовнішні втручання. Такий рівень автономності та простоти особливо актуальний для невеликих клінік або мобільних УЗД-кабінетів.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

4.1 Медичний захист і забезпечення санітарного та епідемічного благополуччя населення

Система медичного захисту та забезпечення санітарного й епідемічного благополуччя населення в Україні є складовою державної політики у сфері охорони здоров'я та безпеки життєдіяльності. Її правову основу складає Закон України «Про забезпечення санітарного та епідемічного благополуччя населення» від 24.02.1994 № 4004-XII, який визначає засади державної санітарно-епідеміологічної служби, повноваження органів виконавчої влади, обов'язки роботодавців та громадян щодо дотримання санітарного законодавства [25].

Метою медичного захисту є зменшення впливу шкідливих та небезпечних чинників виробничого середовища на організм людини, профілактика захворювань, попередження епідемій і підтримка належного рівня гігієни на робочих місцях. Санітарне благополуччя досягається завдяки комплексному контролю умов праці, водопостачання, харчування, поводження з відходами, вентиляції та дотримання гігієнічних нормативів [26].

Особливу увагу в рамках техногенної безпеки приділяють організації первинної та періодичної медичної допомоги для працівників підприємств. Згідно з Законом України «Про охорону праці» № 2694-XII, роботодавець зобов'язаний організовувати проведення попередніх (до прийняття на роботу) та періодичних (протягом трудової діяльності) медичних оглядів для працівників, зайнятих на важких роботах, роботах із шкідливими або небезпечними умовами праці, а також для осіб віком до 21 року [27].

Медичні огляди проводяться на підставі Наказу Міністерства охорони здоров'я України від 21.05.2007 № 246, затвердженого як обов'язковий для підприємств всіх форм власності. Порядок організації цих заходів дозволяє

виявляти ранні ознаки професійних захворювань і проводити медико-профілактичні втручання. Працівники, які за результатами огляду визнані непридатними до певних видів робіт, підлягають переведенню на інші посади відповідно до рекомендацій медичних закладів.

У межах забезпечення епідемічної безпеки особливої актуальності набуває дотримання вимог санітарної охорони підприємств. Зокрема, на підставі Закону України «Про основи законодавства України про охорону здоров'я» № 2801-XII, керівники організацій несуть відповідальність за створення безпечного середовища, профілактику розповсюдження інфекційних захворювань та забезпечення належного рівня гігієни у виробничих приміщеннях [25].

Особливе місце у цій сфері займає контроль за вентиляцією та мікрокліматом, оскільки на робочих місцях при порушенні повітряного режиму можуть виникати осередки поширення інфекцій. Згідно з державними будівельними нормами ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування», повітрообмін у приміщеннях офісного типу повинен забезпечувати щонайменше 20 м³/год на одну особу при постійній присутності працівників. Рекомендована температура в опалювальний сезон становить 22–24 °С, а відносна вологість — у межах 40–60 % [26].

Контроль за якістю водопостачання на підприємствах регламентується нормами ДСанПіН 2.2.4-171-10, згідно з якими питна вода, що використовується в офісах і на виробництві, повинна відповідати гігієнічним вимогам за показниками органолептики, токсикології, мікробіології та вмісту важких металів. У випадку централізованого водопостачання обов'язкове проходження води через лабораторний контроль не рідше ніж один раз на квартал, а при наявності бюветів – щомісячно [28, с. 148].

У ситуаціях виникнення надзвичайних подій біологічного характеру (спалахи інфекцій, викиди біологічно небезпечних речовин), на підприємствах має бути розроблений план ліквідації аварій та проведення евакуації відповідно до вимог наказу ДСНС України та МОЗ від 27.05.2022 № 290/841 «Про затвердження Вимог до планів реагування на надзвичайні ситуації та аварії на об'єктах» [29]. Цей

документ встановлює обов’язкову наявність алгоритму дій персоналу в разі виникнення загрози життю чи здоров’ю через зараження інфекційними агентами.

Одним із важливих аспектів санітарного захисту є робота з відходами. Відповідно до Закону України «Про управління відходами» № 2320-IX, небезпечні та біологічні відходи повинні збиратися, зберігатися, транспортуватися і знищуватися згідно з ліцензованими процедурами, з обов’язковим контролем дотримання вимог екологічної та санітарної безпеки [30].

Крім того, в межах вимог наказу Міністерства соціальної політики України № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», передбачено надання працівникам регулярних гігієнічних перерв, забезпечення контролю за зоровим навантаженням, а також організацію заходів психофізіологічного розвантаження [31]. Практика свідчить, що дотримання цих вимог суттєво знижує кількість звернень до лікарів з ознаками стомлення, головного болю, зорового дискомфорту та м’язово-скелетних болей.

Таким чином, ефективне функціонування системи медичного захисту й санітарного контролю забезпечується завдяки інтеграції правових, організаційних і технічних заходів. Дотримання гігієнічних нормативів, організація профілактичних заходів, контроль за епідемічною ситуацією та забезпечення належних умов праці є основою збереження працездатності й здоров’я працівників у виробничих умовах.

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Організація безпечних умов праці користувачів персональних комп’ютерів (ПК) регламентується комплексом нормативно-правових актів, які враховують фізіологічні, психоемоційні та санітарно-гігієнічні фактори, що впливають на працездатність людини.

Відповідно до Закону України «Про охорону праці» № 2694-XII роботодавець зобов’язаний створити умови праці, які відповідають вимогам безпеки, гігієни праці, та не завдають шкоди здоров’ю працівника [27].

Спеціалізований норматив у цій сфері — Наказ Міністерства соціальної політики України від 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [31]. Документ визначає основні вимоги до обладнання, організації робочих місць та режимів праці користувачів ВДТ (відеодисплейних терміналів). Так, безперервна робота за ПК не повинна перевищувати 2 години, після чого працівнику надається перерва тривалістю не менше 15 хвилин. Альтернативно дозволяється чергування з іншими видами діяльності, що зменшують навантаження на зір та нервову систему [31].

Робоче місце з ПК повинно відповідати ергономічним нормам. Висота поверхні столу має бути 680–800 мм, а відстань від очей користувача до екрана — 500–700 мм. Центр екрана рекомендується розташовувати нижче рівня очей на 15–20° [26]. Екран повинен мати чітке, стабільне зображення без мерехтінь, з яскравістю, яка регулюється користувачем. Клавіатура має бути розташована на столі окремо від дисплея, з можливістю регулювання кута нахилу та матовою поверхнею для уникнення відблисків [31].

Умови мікроклімату в приміщенні визначаються за показниками температури, вологості та швидкості повітря. Оптимальні значення для операторських робочих місць — температура 22–25 °С, вологість 40–60 %, швидкість повітряного потоку — до 0,1 м/с [26, 32]. Вентиляція повинна забезпечувати повітрообмін щонайменше 20 м³/год на одну особу. Порушення цих параметрів призводить до зниження концентрації уваги, швидшого настання втоми та загального зниження працездатності.

Освітлення є критично важливим чинником для зорової роботи з ПК. Згідно з нормами, рівень освітленості повинен становити 300–500 лк при роботі з ВДТ, коефіцієнт пульсації освітлення — не більше 10 %, а коефіцієнт природного освітлення (КПО) — не нижче 1,5 % [26]. Світильники необхідно розміщувати

таким чином, щоб уникати прямих або відбитих блисків на екрані, які викликають додаткове напруження зорового аналізатора.

Зорове перенавантаження, статика м’язів шиї та рук, відсутність змін у робочій позі зумовлюють розвиток синдрому втоми та астенопії — станів, які супроводжуються болем в очах, зниженням гостроти зору, головним болем, болем у верхній частині спини та зап’ястках [26, 32]. Особливо це проявляється у працівників, які працюють у діалоговому режимі або виконують редагування тексту.

Доцільно також враховувати психофізіологічні фактори. Високий рівень нервово-емоційного навантаження, дефіцит часу на виконання завдань і відсутність належних перерв сприяють розвитку дистресу, який негативно впливає на продуктивність і загальний стан працівника [28]. У цьому контексті важливим є регламентування тривалості змін, періодичності відпочинку та чергування типів завдань.

Профілактичні заходи включають також надання працівникам спеціальних гігієнічних перерв, організацію психофізіологічного розвантаження, професійний відбір операторів з урахуванням індивідуально-психологічних характеристик, а також ергономічне проектування робочих місць [28]. Контроль за дотриманням вимог безпеки здійснюють фахівці з охорони праці на підприємстві відповідно до нормативних актів та інструкцій, затверджених на місцевому рівні.

Таким чином, створення безпечних умов для користувачів ПК базується на суворому дотриманні чинного законодавства, гігієнічних регламентів, а також впровадженні організаційно-технічних заходів, які забезпечують збереження здоров’я та стабільну працездатність працівників у процесі їх трудової діяльності.

ВИСНОВКИ

Підсумовуючи результати виконаної кваліфікаційної роботи, можна з упевненістю сказати, що поставлена задача була виконана у повному обсязі. Було створено повнофункціональний настільний додаток для кабінету ультразвукової діагностики, який поєднує простоту використання з ефективністю зберігання і обробки медичних даних. На відміну від громіздких систем, які потребують серверів, авторизації, або складного налаштування, розроблений програмний продукт орієнтований на автономну роботу та швидкий доступ до основних функцій — саме цього часто не вистачає в умовах реального медичного прийому.

Особлива увага під час розробки приділялася побудові логічної, чітко розділеної архітектури. Було сформовано окремі класи для взаємодії з базою даних, обробки інтерфейсних подій та генерації PDF-звітів. Це дозволило досягти високої гнучкості структури проєкту, де кожен модуль працює незалежно, а всі компоненти взаємодіють через чітко визначені точки доступу. Такий підхід значно спрощує процес тестування, налагодження і потенційного розширення програми в майбутньому — наприклад, за потреби можна інтегрувати модуль роботи з іншими форматами файлів чи додати вивантаження в хмару.

Під час проєктування особливий акцент було зроблено на мінімалізм інтерфейсу. Усі дії користувача — додавання нового пацієнта, редагування запису, перегляд висновку — винесено в окремі діалогові вікна з обмеженою кількістю полів і кнопок. Це дає змогу медичному працівнику, який не має IT-освіти, інтуїтивно зрозуміти, як працює система, не читаючи інструкції. Усі поля мають підписи, є вбудовані обмеження на порожні значення, а повідомлення про помилки допомагають уникнути неочікуваних ситуацій.

Тестування проводилося як у середовищі розробки, так і вручну, безпосередньо через взаємодію з програмою. Було виявлено низку дрібних неточностей, які оперативно усувалися: від некоректного форматування дати до невиведення повідомлення при порожньому полі діагнозу. Зрештою, додаток

продемонстрував стабільну роботу при тривалому запуску, багатократному збереженні та видаленні даних, а також експорту документів. Жодних критичних збоїв або втрати даних виявлено не було.

Окремо варто відзначити ефективність реалізованої структури бази даних. Замість складних реляцій, у проєкті було використано спрощену, але надійну модель з однією таблицею пацієнтів, що містить усі необхідні поля. Кожен запис має унікальний ідентифікатор, а також дату створення, що дозволяє сортувати та відстежувати зміни. Збереження відбувається через клас DatabaseManager, що інкапсулює всі SQL-запити. Це дозволяє змінювати структуру таблиці у майбутньому без втручання в основний код інтерфейсу, що підтверджує масштабованість архітектури.

У роботі також був реалізований механізм створення звітів у форматі PDF. Цей модуль дозволяє швидко сформувати текстовий висновок і зберегти його як окремий файл. Таке рішення підвищує зручність для лікаря, оскільки не потребує зовнішніх текстових редакторів або складних форматувань. Усі дії — збереження, друк, перегляд — виконуються буквально за два кліки. Крім того, назва файлу генерується динамічно, з урахуванням імені пацієнта, що зменшує ймовірність помилки при збереженні кількох звітів поспіль.

З точки зору безпеки, у роботі враховано основні принципи обробки персональних даних: локальне зберігання, обмежений доступ до файлів, уникнення мережевих з’єднань без шифрування. Крім того, в інтерфейсі відсутні поля, які вимагали би введення надлишкової особистої інформації, що також сприяє зменшенню ризиків. Хоча автор не реалізовував окремі механізми шифрування, структура додатка дозволяє в майбутньому додати такі функції без серйозного переписування коду.

На етапі порівняння з аналогічними системами, стало очевидно, що більшість медичних ПЗ є або занадто складними у встановленні, або мають непотрібну надмірну функціональність, яка перевантажує користувача. Розроблений додаток, натомість, фокусується лише на найнеобхідніших функціях і робить це просто та

надійно. Саме цей підхід дозволяє інтегрувати систему в практичну роботу без додаткового навчання персоналу чи залучення IT-фахівців.

Таким чином, проведена робота довела актуальність та доцільність розробки вузькоспеціалізованого програмного забезпечення для кабінету УЗД. Результат має не лише навчальну цінність, але й практичну — додаток може бути розгорнутий у реальному закладі охорони здоров’я для щоденного використання. Водночас проєкт залишається відкритим для подальшої розробки: за потреби можна додати підтримку мультимовного інтерфейсу, створити резервне копіювання, реалізувати пошук за кількома параметрами, або навіть синхронізувати дані між декількома ПК.

Усе це свідчить про те, що студент не лише виконав поставлені перед ним задачі, а й продемонстрував глибоке розуміння принципів побудови прикладного програмного забезпечення, а також готовність до практичного застосування набутих знань. Це дозволяє розглядати розробку як успішний приклад проєкту, орієнтованого на реальні потреби користувача, і підтверджує високий рівень професійної підготовки здобувача освіти.

Крім технічної реалізації, ця робота продемонструвала вміння планувати, поетапно впроваджувати та тестувати складну систему з урахуванням практичного застосування. Зокрема, було важливо не лише написати функціональний код, а й забезпечити його читабельність, простоту супроводу та можливість масштабування. Такий підхід наближає студентську розробку до рівня професійного інженерного продукту.

Ретельна увага до деталей — від оформлення форм до формату PDF-звіту — свідчить про відповідальне ставлення до користувача і розуміння, що навіть дрібниці можуть мати значення в реальному середовищі. Саме ця орієнтація на практику, з реальними обмеженнями і сценаріями, відрізняє цю роботу серед типових студентських проєктів. На основі отриманого досвіду можна не тільки продовжити вдосконалення поточної системи, а й сміливо розпочинати розробку нових, більш масштабних медичних рішень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Касевич О. А. Інформаційні технології в медицині : підручник. 2-ге вид., перероб. і допов. Київ : ВСВ «Медицина», 2021. 368 с.

2. Яворська, ЄБ, Кінаш, РВ, Цуприк, ГБ, Николайчук ВІ Проблеми виявлення біосигналів у медичних інформаційних системах/ЄБ Яворська, Кінаш РВ, ГБ Цуприк, ВІ Николайчук//IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині»(ІСМ–2021)[Текст]: зб. наук. пр.– Харків: Нац. аерокосм. ун-т ім. МЄ Жуковського «Харків. авіац. ін-т», 2021.–25-26 с.

3. Леонтьєва, Т. В. Документознавство в діяльності медичних установ. Київ : Видавничий Дім "МЕДЕК", 2012. 352 с.

4. Волощук В. Я. Інформаційні технології в системі охорони здоров’я України. Медична інформатика та інженерія. 2013. № 1. С. 62–67.

5. МІС «Доктор Елекс» : офіційний сайт. URL: https://ehealth.eleks.com/ (дата звернення: 04.06.2025).

6. HEALTH24 : сервіс обліку пацієнтів : офіційний сайт. URL: https://health24.ua/ (дата звернення: 04.06.2025).

7. OpenEMR : офіційний сайт. URL: https://www.open-emr.org/ (дата звернення: 04.06.2025).

8. Глущенко, Ю. В. Особливості обліку в закладах охорони здоров'я. Облік і фінанси АПК. 2017. № 4. С. 97–104.

9. Yavorskyy B., Yavorska E., Tsupryk H., Kinash R. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli // Scientific Journal of TNTU. – Ternopil: TNTU, 2023. – Vol. 112, No. 4. – P. 82–90.

10. SQLite : Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 04.06.2025).

11. Кузьмінов, А. В. Сучасні тенденції розвитку та застосування медичних інформаційних систем. Вісник СумДУ. Серія «Технічні науки». 2019. № 2. С. 49–55.

12. ДСТУ ISO/IEC 27001:2023 Інформаційна безпека, кібербезпека та захист конфіденційності. Системи керування інформаційною безпекою. Вимоги (ISO/IEC 27001:2022, IDT). [Чинний від 2023–08–22]. Київ : УкрНДНЦ, 2023.

13. Дудзяний І. М. Вступ до програмної інженерії : конспект лекцій. Тернопіль : ТНТУ, 2017. 132 с.

14. Petryk M., Chyzh V., Tsupryk H., Petryk O. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion // ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems. Ternopil, Ukraine ; Opole, Poland, 2024. – С. 486–493.

15. SQLite Developer Guide. URL: https://sqlite.org/docs.html (дата звернення: 04.06.2025)

16. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. 2-ге вид., перероб. і допов. Київ : ВНВ «Промінь», 2017. 488 с

17. Boyko I., Tsupryk H., Mudryk I., Stoianov Y. A Theoretical Model of Thermal Conductivity for Multilayer Nitride-Based Nanosystems // EasyChair Preprint. 2022. – № 8909. URL: https://easychair.org/publications/preprint/35d796 (дата звернення: 04.06.2025).

18. Boyko I., Mudryk I., Stoianov Y., Yavorska E. Nonlinear Model of the Three-Components Competitive Adsorption Using // Modeling and Simulation of Systems (MODS 2020). 2020. – Vol. 2707. – С. 37–43. URL: http://ceur-ws.org/Vol-2707/paper05.pdf (дата звернення: 04.06.2025).

19. Гончаренко, В. О. *Основи дизайну користувацьких інтерфейсів : навчальний посібник*. Київ : НАУ, 2018. 164 с.

20. Qt Designer : documentation. URL: https://doc.qt.io/qtforpython-6/overviews/qtdesigner-manual.html (дата звернення: 04.06.2025).

21. Глинський, Я. М. Тестування програмного забезпечення : навчальний посібник. Львів : ТНТУ, 2021. 176 с.

22. Beizer, B. Software Testing Techniques. 2nd ed. New York : Van Nostrand Reinhold, 1990. 696 p

23. Рашковський, О. М. Тестування програмного забезпечення : навчальний посібник. Харків : ХНУРЕ, 2021. 176 с.

24. Kaner C., Falk J., Nguyen H. Q. Testing Computer Software. 3rd ed. New York : Wiley, 1999. 640 p.

25. Основи законодавства України про охорону здоров'я : Закон України від 19.11.1992 № 2801-XII. Відомості Верховної Ради України. 1993. № 4. Ст. 19. URL: https://zakon.rada.gov.ua/laws/show/2801-12 (дата звернення: 04.06.2025).

26. Сакун М. М., Москалюк І. В., Нагорнюк В. Ф. Безпека життєдіяльності та основи охорони праці : навчально-методичний комплекс. Одеса, 2017. 168 с.

27. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. Відомості Верховної Ради України. 1992. № 49. Ст. 668. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 04.06.2025).

28. Зацарний В. В., Зацарна О. В., Землянська О. В., Праховнік Н. А. Безпека життєдіяльності : навчальний посібник. Київ : НТУУ «КПІ», 2016. 242 с.

29. Наказ ДСНС України та МОЗ України від 27.05.2022 № 290/841 «Про затвердження Вимог до планів реагування на надзвичайні ситуації та аварії на об'єктах». Офіційний вісник України. 2022. № 35. Ст. 1899. URL: https://zakon.rada.gov.ua/laws/show/z0407-22 (дата звернення: 04.06.2025).

30. Про управління відходами : Закон України від 20.06.2022 № 2320-IX. Відомості Верховної Ради України. 2022. № 35. Ст. 250. URL: https://zakon.rada.gov.ua/laws/show/2320-IX (дата звернення: 04.06.2025).

31. Наказ Міністерства соціальної політики України від 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». URL: https://zakon.rada.gov.ua/laws/show/z0508-18

32. Охорона праці в галузі : конспект лекцій / уклад. В. Я. Яскілка, М. З. Олійник. Тернопіль : ТНТУ імені Івана Пулюя, 2016. 56 с.

ДОДАТКИ

ДОДАТОК А

Тези для Міжнародної студентської науково-технічної конференції

УДК 004.41
В.В. Круцяк, Г.Б. Цуприк, к.т.н., доц..
Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА ПРИКЛАДНОГО СПЕЦІАЛІЗОВАНОГО ЗАСТОСУНКУ ДЛЯ ОПТИМІЗАЦІЇ ОПРАЦЮВАННЯ МАСИВІВ ДАННИХ З ЗАСТОСУВАННЯМ ПАРАДИГМ ТА ТЕХНОЛОГІЙ PYTHON, PYQT5 ТА SQLITE

V.V. Krutsiak, H.B. Tsupryk, Ph.D., Assoc. Prof.

DEVELOPMENT OF A SPECIALIZED APPLICATION FOR OPTIMIZING DATA PROCESSING USING PYTHON, PYQT5, AND SQLITE TECHNOLOGIES AND PARADIGMS

В умовах сучасних медичних закладів, особливо у відділеннях ультразвукової діагностики (УЗД), ефективне управління інформацією про пацієнтів є критично важливим завданням. Традиційні підходи до обліку, що базуються на паперових носіях або застарілих програмних рішеннях, часто виявляються неефективними, призводячи до втрати даних, затримок у пошуку необхідної інформації та ускладнень при аналізі статистичних даних. Це, в свою чергу, негативно впливає на швидкість та якість обслуговування пацієнтів, а також на ефективність роботи медичного персоналу. Тому, розробка сучасних, зручних та надійних систем обліку пацієнтів є актуальною і важливою задачею, спрямованою на оптимізацію робочих процесів та підвищення рівня медичних послуг.

У даній роботі представлено розробку додатку для обліку пацієнтів УЗД, який забезпечує комплексне вирішення проблеми управління інформацією. Метою розробки є створення інструменту, що дозволяє ефективно зберігати, обробляти та аналізувати дані про пацієнтів, а також надає зручний та інтуїтивно зрозумілий інтерфейс користувача для взаємодії з системою. Для реалізації поставленої мети було обрано мову програмування Python, фреймворк PyQt5 для створення графічного інтерфейсу та систему управління базами даних (СУБД) SQLite для організації зберігання даних.

Вибір Python обумовлений його потужністю, простотою та читабельністю коду, що значно полегшує розробку та підтримку програмного забезпечення. Крім того, Python має велику кількість доступних бібліотек, що розширюють його функціональні можливості. PyQt5 є крос-платформним фреймворком, який дозволяє створювати сучасні графічні інтерфейси користувача з використанням широкого спектру віджетів Qt. Це забезпечує зручність та ефективність взаємодії користувача з додатком. SQLite – це легка вбудована СУБД, яка не потребує окремого сервера та ідеально підходить для проектів, де не потрібна висока масштабованість. Її використання спрощує розгортання та адміністрування додатку.

Розроблений додаток забезпечує широкий спектр функціональних можливостей, спрямованих на оптимізацію процесу обліку пацієнтів УЗД. Серед основних функцій слід виділити:

- Зберігання та управління інформацією про пацієнтів, включаючи особисті дані (ПІБ, дата народження, стать, контактна інформація, місце проживання) та дані про обстеження (дата обстеження, тип обстеження, направлення, заключення, лікар).
- Здійснення швидкого та ефективного пошуку пацієнтів за різними критеріями, такими як ПІБ, номер медичної картки, дата обстеження або тип обстеження.
- Фільтрація даних за заданими параметрами для зручного перегляду та аналізу інформації, наприклад, фільтрація за періодом часу, типом обстеження або лікарем.
- Відображення інформації про пацієнтів у зручному табличному вигляді з можливістю сортування за стовпцями, що полегшує навігацію та аналіз даних.

- Генерація звітів на основі збережених даних, що дозволяє отримувати статистичну інформацію про кількість та типи обстежень, завантаженість відділення УЗД та інші важливі показники.
- Експорт даних у різні формати, такі як CSV або Excel, для подальшого аналізу або інтеграції з іншими інформаційними системами медичного закладу.

Архітектура розробленого додатку базується на трьох основних компонентах:

- Модуль бази даних, який відповідає за всі операції взаємодії з базою даних SQLite, включаючи створення таблиць, виконання запитів на додавання, оновлення, видалення та вибірку даних.
- Модуль графічного інтерфейсу, реалізований з використанням фреймворку PyQt5, який забезпечує зручний та інтуїтивно зрозумілий інтерфейс користувача для взаємодії з додатком.
- Модуль обробки даних, який містить логіку обробки та перетворення даних, отриманих з бази даних, перед їх відображенням у графічному інтерфейсі, а також обробку дій користувача.

При розробці додатку особлива увага приділялась забезпеченню його гнучкості, масштабованості та зручності підтримки. Для досягнення цієї мети були застосовані принципи модульності, інверсії управління (IoC) та шаблон проектування MVVM. Модульна архітектура дозволяє легко додавати нові функції та модифікувати існуючі модулі без впливу на інші частини системи. Принцип IoC зменшує залежність між компонентами, що спрощує їх тестування та повторне використання. Шаблон MVVM розділяє логіку інтерфейсу користувача та бізнес-логіку додатку, що полегшує розробку, тестування та підтримку інтерфейсу користувача.

Результати розробки підтвердили ефективність використання Python, PyQt5 та SQLite для створення додатків обліку пацієнтів УЗД. Розроблений додаток забезпечує повний набір необхідних функціональних можливостей, має зручний та інтуїтивно зрозумілий інтерфейс користувача, а також є масштабованим та гнучким рішенням, готовим до подальшого розвитку та інтеграції в існуючу інфраструктуру медичного закладу.

Подальші дослідження можуть бути спрямовані на розширення функціональності додатку, включаючи:

- Інтеграцію з іншими медичними інформаційними системами (МІС) для обміну даними про пацієнтів та результати обстежень.
- Реалізацію підтримки мережевої роботи для забезпечення доступу до даних з різних робочих місць у відділенні УЗД.
- Розробку мобільної версії додатку для зручного доступу до інформації з мобільних пристроїв.
- Впровадження системи підтримки прийняття рішень (СППР) на основі аналізу даних про пацієнтів для допомоги лікарям у діагностиці.

Література

1. PyQt5 Reference Guide. Official documentation. Riverbank Computing Limited. URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата звернення: 05.05.2025).

2. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 05.05.2025)..

3. Mark Summerfield. Rapid GUI Programming with Python and Qt. Pearson Education, 2015. – 648 p.

4. David Beazley, Brian K. Jones. Python Cookbook: Recipes for Mastering Python 3. O'Reilly Media, 2013. – 706 p.

5. Allen Downey. Think Python: How to Think Like a Computer Scientist. O'Reilly Media,

6. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

7. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

8. Яворська, Є.Б., Кінаш, Р.В., Цуприк, Г.Б., Николайчук В.І. Проблеми виявлення біосигналів у медичних інформаційних системах / Є.Б. Яворська, Кінаш Р.В., Г.Б. Цуприк, В.І. Николайчук // IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині» (ІСМ–2021) [Текст] : зб. наук. пр. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2021. – 25-26 с.

ДОДАТОК Б

КОД РЕАЛІЗАЦІЇ ВІКНА ДОДАВАННЯ / РЕДАГУВАННЯ КАРТИ ПАЦІЄНТА

```
from PyQt5.QtWidgets import QDialog, QLabel, QPushButton, QVBoxLayout,
QHBoxLayout, QWidget
from PyQt5.QtGui import QPixmap, QIcon
from PyQt5.QtCore import Qt, QSize

class CustomTitleBar(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.setFixedHeight(32)
        self.setStyleSheet("background-color: #012229;")
        self.btn_close = QPushButton()
        self.btn_close.setFixedSize(32, 32)
        self.btn_close.setIconSize(QSize(32, 32))
        self.btn_close.setFlat(True)
        self.btn_close.setStyleSheet("""
            QPushButton {
                background-color: transparent;
                border-radius: 6px;
            }
            QPushButton:hover {
                background-color: #00545f;
            }
        """)
        self.btn_close.setIcon(QIcon(":/img/close.png"))
        self.btn_close.clicked.connect(parent.close)

        layout = QHBoxLayout(self)
        layout.setContentsMargins(4, 0, 4, 0)
        layout.setSpacing(4)

        layout.addStretch()
        layout.addWidget(self.btn_close)

class PatientFoundDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowFlags(Qt.FramelessWindowHint | Qt.Dialog)
        self.setFixedSize(540, 180)

        self.setStyleSheet("""
            QDialog {
                background-color: #022f36;
                border-radius: 10px;
            }
        """)

        self.title_bar = CustomTitleBar(self)

        # === Лого у вікні===
        logo = QLabel()
        logo.setPixmap(QPixmap(":/img/EchoMed2.png").scaled(120, 120,
Qt.KeepAspectRatio, Qt.SmoothTransformation))
        logo.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        logo.setStyleSheet("background: transparent;")

        # === Текст підсказка ===
```

```
        message = QLabel("Пацієнт з таким ім'ям вже є в базі.\nБажаєте автоматично
заповнити його дані?")
        message.setStyleSheet("""
            color: #ffffff;
            font-family: 'Segoe UI SemiBold';
            font-size: 14px;
        """)
        message.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        message.setWordWrap(True)

        message_row = QHBoxLayout()
        message_row.addWidget(logo)
        message_row.addSpacing(15)
        message_row.addWidget(message)
        message_row.addStretch()

        # === Кнопки ===
        yes_button = QPushButton("Так")
        no_button = QPushButton("Ні")
        for btn in (yes_button, no_button):
            btn.setCursor(Qt.PointingHandCursor)
            btn.setFixedWidth(90)
            btn.setStyleSheet("""
                QPushButton {
                    background-color: #00A3CC;
                    color: white;
                    padding: 6px;
                    border-radius: 8px;
                    font-family: 'Segoe UI SemiBold';
                    font-size: 13px;
                }
                QPushButton:hover {
                    background-color: #0090B8;
                }
            """)
        yes_button.clicked.connect(lambda: self.done(QDialog.Accepted))
        no_button.clicked.connect(lambda: self.done(QDialog.Rejected))

        buttons = QHBoxLayout()
        buttons.addStretch()
        buttons.addWidget(yes_button)
        buttons.addWidget(no_button)

        # === Основний layout ===
        layout = QVBoxLayout(self)
        layout.setContentsMargins(10, 5, 10, 10)
        layout.setSpacing(10)
        layout.addWidget(self.title_bar)
        layout.addLayout(message_row)
        layout.addStretch()
        layout.addLayout(buttons)
```

ДОДАТОК В

КОД РЕАЛІЗАЦІЇ ВІКНА ДРУКУ АБО СТВОРЕННЯ ПДФ ЗАКЛЮЧЕННЯ

```
from PyQt5.QtWidgets import QDialog, QLabel, QVBoxLayout, QHBoxLayout, QTextEdit,
QGridLayout, QPushButton, QFileDialog, QWidget
from PyQt5.QtGui import QFont, QPixmap, QPainter, QIcon
from PyQt5.QtCore import Qt, QRect, QSize
from PyQt5.QtPrintSupport import QPrinter, QPrintDialog
import os

class PatientDetailsDialog(QDialog):
    def __init__(self, patient_data, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Карта огляду пацієнта")
        self.setFixedSize(600, 750)
        self.setAttribute(Qt.WA_TranslucentBackground)
        self.setWindowFlags(Qt.FramelessWindowHint | Qt.Dialog)
        self.patient_data = patient_data

        self.setStyleSheet("QDialog { background: transparent; } QLabel { color:
white; font-family: 'Segoe UI'; background: transparent; border: none; }")

        self.verticalLayout_root = QVBoxLayout(self)
        self.verticalLayout_root.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_root.setSpacing(0)

        # Бар
        self.widget_bar = QWidget(self)
        self.widget_bar.setFixedHeight(40)
        self.widget_bar.setStyleSheet("background-color: #011f22; border-top-left-
radius: 16px; border-top-right-radius: 16px;")

        self.button_close = QPushButton(self.widget_bar)
        self.button_close.setFixedSize(40, 40)
        self.button_close.setIcon(QIcon(":/img/close.png"))
        self.button_close.setIconSize(QSize(24, 24))
        self.button_close.setStyleSheet("QPushButton { border: none; background-
color: transparent; } QPushButton:hover { background-color: #ff4d4d; }")
        self.button_close.clicked.connect(self.close)

        bar_layout = QHBoxLayout(self.widget_bar)
        bar_layout.setContentsMargins(0, 0, 0, 0)
        bar_layout.addStretch()
        bar_layout.addWidget(self.button_close)

        # Основной віджет (контейнер)
        self.widget = QWidget(self)
        self.widget.setStyleSheet("background-color: #022f36; border-bottom-left-
radius: 16px; border-bottom-right-radius: 16px; border: 1px solid #00545f;")

        content_layout = QVBoxLayout(self.widget)
        content_layout.setContentsMargins(20, 20, 20, 20)

        title = QLabel("Карта огляду пацієнта")
        title.setFont(QFont("Segoe UI", 20, QFont.Bold))
        title.setAlignment(Qt.AlignLeft | Qt.AlignVCenter)
        title.setStyleSheet("background: transparent; border: none;")

        logo = QLabel()
```

```
        logo.setPixmap(QPixmap(":/img/EchoMed2.png").scaled(100, 100,
Qt.KeepAspectRatio, Qt.SmoothTransformation))
        logo.setAlignment(Qt.AlignRight | Qt.AlignVCenter)
        logo.setStyleSheet("background: transparent; border: none;")

        top_layout = QHBoxLayout()
        top_layout.setSpacing(20)
        top_layout.addWidget(title)
        top_layout.addStretch()
        top_layout.addWidget(logo)

        grid = QGridLayout()
        grid.setVerticalSpacing(8)
        grid.setContentsMargins(0, 10, 0, 0)

        self.labels = [
            ("Номер обстеження:", str(patient_data[1])),
            ("ПІБ:", patient_data[2]),
            ("Дата народження:", patient_data[3]),
            ("Стать:", patient_data[4]),
            ("Місце проживання:", patient_data[5]),
            ("Дата обстеження:", patient_data[6]),
            ("Оглянуті зони:", patient_data[7]),
        ]

        for row, (label_text, value) in enumerate(self.labels):
            label = QLabel(label_text)
            label.setFont(QFont("Segoe UI", 10, QFont.DemiBold))
            label.setStyleSheet("background: transparent; border: none;")

            value_label = QLabel(str(value))
            value_label.setFont(QFont("Segoe UI", 10, QFont.Bold))
            value_label.setStyleSheet("color: white; background: transparent;
border: none;")

            grid.addWidget(label, row, 0, alignment=Qt.AlignLeft)
            grid.addWidget(value_label, row, 1, alignment=Qt.AlignLeft)

        conclusion_title = QLabel("Заключення огляду пацієнта:")
        conclusion_title.setFont(QFont("Segoe UI", 13, QFont.Bold))
        conclusion_title.setAlignment(Qt.AlignCenter)
        conclusion_title.setStyleSheet("background: transparent; border: none;")

        self.conclusion_box = QTextEdit()
        self.conclusion_box.setReadOnly(True)
        self.conclusion_box.setText(patient_data[8])
        self.conclusion_box.setMinimumHeight(180)
        self.conclusion_box.setStyleSheet("""
            QTextEdit {
                background-color: #003138;
                color: #ffffff;
                font-family: 'Segoe UI SemiBold';
                font-size: 14px;
                border-radius: 8px;
                padding: 8px;
                border: none;
            }
            QScrollBar:vertical {
                background: #002128;
                width: 10px;
                margin: 0;
            }
            QScrollBar::handle:vertical {
                background: #00A3CC;
```

```
            min-height: 20px;
            border-radius: 5px;
        }
        QScrollBar::add-line:vertical,
        QScrollBar::sub-line:vertical {
            background: none;
            height: 0px;
        }
        QScrollBar::add-page:vertical,
        QScrollBar::sub-page:vertical {
            background: none;
        }
    """)

    print_button = QPushButton("Друкувати")
    print_button.setCursor(Qt.PointingHandCursor)
    print_button.setStyleSheet("""
        QPushButton {
            background-color: #00A3CC;
            color: white;
            padding: 8px 16px;
            border-radius: 8px;
            font-family: 'Inter SemiBold';
            font-size: 13px;
        }
        QPushButton:hover {
            background-color: #0090B8;
        }
    """)
    print_button.clicked.connect(lambda: self.print_text(self.conclusion_box))

    export_button = QPushButton("Зберегти PDF (перегляд)")
    export_button.setCursor(Qt.PointingHandCursor)
    export_button.setStyleSheet(print_button.styleSheet())
    export_button.clicked.connect(self.export_to_pdf)

    button_layout = QHBoxLayout()
    button_layout.addStretch()
    button_layout.addWidget(print_button)
    button_layout.addSpacing(10)
    button_layout.addWidget(export_button)
    button_layout.addStretch()

    content_layout.addLayout(top_layout)
    content_layout.addSpacing(10)
    content_layout.addLayout(grid)
    content_layout.addSpacing(20)
    content_layout.addWidget(conclusion_title)
    content_layout.addWidget(self.conclusion_box)
    content_layout.addSpacing(10)
    content_layout.addLayout(button_layout)

    self.verticalLayout_root.addWidget(self.widget_bar)
    self.verticalLayout_root.addWidget(self.widget)
```

```
    def get_pdf_save_path(self):
        default_name = f"{self.patient_data[2]}.pdf"  # ПІБ пацієнта
        path, _ = QFileDialog.getSaveFileName(
            self,
            "Зберегти PDF",
            default_name,
            "PDF Files (*.pdf)"
        )
        return path

    def print_text(self, text_edit):
        printer = QPrinter(QPrinter.HighResolution)
        printer.setPageSize(QPrinter.A4)
        printer.setOrientation(QPrinter.Portrait)
        printer.setFullPage(False)

        dialog = QPrintDialog(printer, self)
        if dialog.exec_() != QPrintDialog.Accepted:
            return

        def mm(value):
            return int(value * printer.logicalDpiX() / 25.4)

        painter = QPainter()
        if not painter.begin(printer):
            return

        # Налаштування
        margin_left = mm(15)
        margin_right = mm(15)
        margin_top = mm(10)
        line_spacing = mm(6)
        y = margin_top
        page_width = printer.pageRect().width()

        # Шрифти
        title_font = QFont("Arial", 14, QFont.Bold)
        label_font = QFont("Arial", 10, QFont.Bold)
        value_font = QFont("Arial", 10)

        # Заголовок
        painter.setFont(title_font)
        title = "Карта огляду пацієнта"
        painter.drawText(QRect(margin_left, y, page_width - margin_left -
margin_right, mm(10)),
                         Qt.AlignCenter, title)
        y += mm(20)
```

```
# Дані пацієнта
        for label, value in self.labels:
            painter.setFont(label_font)
            painter.drawText(margin_left, y, label)

            painter.setFont(value_font)
            painter.drawText(margin_left + mm(55), y, str(value))
            y += line_spacing


 # Заключення
        y += mm(10)
        painter.setFont(label_font)
        painter.drawText(margin_left, y, "Заключення огляду пацієнта:")
        y += mm(7)

        painter.setFont(value_font)
        conclusion_rect = QRect(
            margin_left,
            y,
            page_width - margin_left - margin_right,
            printer.pageRect().height() - y - mm(20)
        )
        painter.drawText(conclusion_rect, Qt.TextWordWrap | Qt.AlignJustify,
self.patient_data[8])

        painter.end()



    def export_to_pdf(self):

        filename, _ = QFileDialog.getSaveFileName(
            self,
            "Зберегти PDF",
            f"{self.patient_data[2]}.pdf",
            "PDF Files (*.pdf)"
        )
        if not filename:
            return

        printer = QPrinter(QPrinter.HighResolution)
        printer.setOutputFormat(QPrinter.PdfFormat)
        printer.setOutputFileName(filename)
        printer.setPageSize(QPrinter.A4)
        printer.setOrientation(QPrinter.Portrait)
        printer.setFullPage(False)

        def mm(value):
            return int(value * printer.logicalDpiX() / 25.4)

        painter = QPainter()
        if not painter.begin(printer):
            return
```

```
# Налаштування
        margin_left = mm(15)
        margin_right = mm(15)
        margin_top = mm(10)
        line_spacing = mm(6)
        y = margin_top
        page_width = printer.pageRect().width()
# Шрифти
        title_font = QFont("Arial", 14, QFont.Bold)
        label_font = QFont("Arial", 10, QFont.Bold)
        value_font = QFont("Arial", 10)

# Заголовок по центру
        painter.setFont(title_font)
        title = "Карта огляду пацієнта"
        painter.drawText(QRect(margin_left, y, page_width - margin_left -
margin_right, mm(10)),
                         Qt.AlignCenter, title)
        y += mm(20)


        for label, value in self.labels:
            painter.setFont(label_font)
            painter.drawText(margin_left, y, label)

            painter.setFont(value_font)
            painter.drawText(margin_left + mm(55), y, str(value))
            y += line_spacing

        # Заголовок заключення
        y += mm(10)
        painter.setFont(label_font)
        painter.drawText(margin_left, y, "Заключення огляду пацієнта:")
        y += mm(7)

        # Текст заключення по ширині
        painter.setFont(value_font)
        conclusion_rect = QRect(
            margin_left,
            y,
            page_width - margin_left - margin_right,
            printer.pageRect().height() - y - mm(20)
        )
        painter.drawText(conclusion_rect, Qt.TextWordWrap | Qt.AlignJustify,
self.patient_data[8])

        painter.end()
        os.startfile(filename)
```

ДОДАТОК Д

РЕАЛІЗАЦІЯ КОДУ БАЗИ ДАНИХ

```
import sqlite3
import os
import shutil
from pathlib import Path

class DatabaseManager:
    def __init__(self):

        self.user_data_dir = Path(os.getenv('LOCALAPPDATA')) / "EchoMed"
        self.user_data_dir.mkdir(parents=True, exist_ok=True)

        self.db_path = self.user_data_dir / "patients.db"


        if not self.db_path.exists():
            try:

                base_dir = os.path.dirname(os.path.abspath(__file__))
                source_db = os.path.join(base_dir, "patients.db")
                if os.path.exists(source_db):
                    shutil.copy2(source_db, self.db_path)
                else:
                    print("⚠️ Шаблон бази даних не знайдено.")
            except Exception as e:
                print(f"Помилка при копіюванні бази даних: {e}")

        self.connection = sqlite3.connect(str(self.db_path))
        self.cursor = self.connection.cursor()
        self.create_table()

    def create_table(self):
        query = """
        CREATE TABLE IF NOT EXISTS patients (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            inspection_numbers TEXT,
            full_name TEXT,
            birth_date TEXT,
            gender TEXT,
            location TEXT,
            inspection_date TEXT,
            inspection_type TEXT,
            conclusion TEXT,
            is_favorite INTEGER DEFAULT 0
        );
        """
        self.cursor.execute(query)
        self.connection.commit()

    def add_patient(self, inspection_numbers ,full_name, birth_date, gender,
location, inspection_date, inspection_type, conclusion):
        query = """
        INSERT INTO patients (inspection_numbers, full_name, birth_date, gender,
location, inspection_date, inspection_type, conclusion)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?);
        """
        self.cursor.execute(query, (inspection_numbers, full_name, birth_date,
gender, location, inspection_date, inspection_type, conclusion))
```

```
        self.connection.commit()

    def get_all_patients(self):
        query = "SELECT * FROM patients;"
        self.cursor.execute(query)
        return self.cursor.fetchall()

    def update_favorite(self, patient_id, is_favorite):
        query = "UPDATE patients SET is_favorite = ? WHERE id = ?;"
        self.cursor.execute(query, (is_favorite, patient_id))
        self.connection.commit()

    def delete_patient(self, patient_id):
        query = "DELETE FROM patients WHERE id = ?;"
        self.cursor.execute(query, (patient_id,))
        self.connection.commit()

    def close(self):
        self.connection.close()

    def find_patient_by_name(self, full_name):
        query = "SELECT * FROM patients WHERE LOWER(full_name) = LOWER(?) LIMIT 1"
        self.cursor.execute(query, (full_name,))
        return self.cursor.fetchone()

    def insert_patient(self, inspection_numbers, full_name, birth_date, gender,
location, inspection_date, inspection_type, conclusion):
        query = """
        INSERT INTO patients (inspection_numbers, full_name, birth_date, gender,
location, inspection_date, inspection_type, conclusion)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?);
        """
        self.cursor.execute(query, (inspection_numbers, full_name, birth_date,
gender, location, inspection_date, inspection_type, conclusion))
        self.connection.commit()

    def get_patient_by_id(self, patient_id):
        query = "SELECT * FROM patients WHERE id = ?"
        self.cursor.execute(query, (patient_id,))
        return self.cursor.fetchone()

    def get_favorite_patients(self):
        query = "SELECT * FROM patients WHERE is_favorite = 1;"
        self.cursor.execute(query)
        return self.cursor.fetchall()

    def update_patient(self, patient_id, inspection_numbers, full_name,
birth_date, gender, location, inspection_date, inspection_type, conclusion):
        query = """
        UPDATE patients
        SET inspection_numbers=?, full_name = ?, birth_date = ?, gender = ?,
location = ?, inspection_date = ?, inspection_type = ?, conclusion = ?
        WHERE id = ?
        """
        self.cursor.execute(query, (inspection_numbers, full_name, birth_date,
gender, location, inspection_date, inspection_type, conclusion, patient_id))
        self.connection.commit()
```

ДОДАТОК Ж

ДИСК