МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
на тему:

«Розробка прикладної програми для управління медичною карткою пацієнта»

Виконав здобувач освіти групи П-41
Сус Олексій Вадимович

Керівник роботи:
Устименко Ярослав Іванович

Рецензент:
Устименко Людмила Миколаївна

Житомир – 2025

Зміст

РЕФЕРАТ

Дипломна робота на тему «Розробка прикладної програми для управління медичною карткою пацієнта» присвячена створенню desktop-додатка для автоматизації обліку медичних даних у невеликих практиках. Програма розроблена з використанням Windows Forms, .NET 8 і SQLite, що забезпечує портативність і низькі системні вимоги. Обсяг роботи — 62 сторінки, включає 28 рисунків, 1 таблицю, 1 додаток і 23 літературних джерела.

У роботі проаналізовано задачу, оглянуто аналоги (Epic Systems, OpenEMR, Practice Fusion) і обґрунтовано вибір технологій. Спроектовано нормалізовану базу даних із шістьма таблицями, реалізовано модульну архітектуру з класами моделей, репозиторіїв і форм. Інтерфейс із вкладковою структурою та різнокольоровими підписами забезпечує зручність. Алгоритми CRUD і оновлення даних оптимізовані для малих обсягів.

Програма підтримує управління даними про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Керівництво користувача описує встановлення, інтерфейс і операції. Результати підтверджують ефективність програми для малих практик. Перспективи розвитку включають додавання звітів і стандартів HL7.

Ключові слова: медична картка, Windows Forms, .NET 8, SQLite, автоматизація, EMR, вкладковий інтерфейс.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

EMR — Electronic Medical Record (Електронна медична картка) — система для зберігання та управління медичними даними пацієнтів у цифровому форматі.

EHR — Electronic Health Record (Електронний медичний запис) — розширена система, що включає дані з різних медичних закладів і підтримує інтероперабельність.

PMS — Patient Management System (Система управління пацієнтами) — програмне забезпечення для автоматизації обліку даних про пацієнтів і медичні процеси.

CRUD — Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення) — базові операції для роботи з даними в інформаційних системах.

SQL — Structured Query Language (Мова структурованих запитів) — мова для роботи з реляційними базами даних, зокрема SQLite.

SQLite — Легка вбудована реляційна база даних, що використовується в програмі для зберігання даних.

.NET 8 — Сучасна платформа розробки від Microsoft, використана для створення програми.

Windows Forms — Технологія для розробки графічного інтерфейсу desktop-додатків у .NET.

TabControl — Компонент Windows Forms для реалізації вкладкової структури інтерфейсу.

DataGridView — Компонент Windows Forms для відображення табличних даних.

TableLayoutPanel — Компонент Windows Forms для компонування елементів інтерфейсу в сітці.

ADO.NET — Технологія доступу до даних у .NET, використана для взаємодії з SQLite.

POCO — Plain Old CLR Object (Простий об’єкт CLR) — класи моделей даних без додаткової логіки.

МКХ-10 — Міжнародна класифікація хвороб 10-го перегляду — стандарт для кодування діагнозів.

HL7 — Health Level Seven — стандарт для обміну медичними даними між системами.

FHIR — Fast Healthcare Interoperability Resources — стандарт для інтероперабельності в медичних системах.

UI — User Interface (Користувацький інтерфейс) — графічна частина програми, з якою взаємодіє користувач.

O(n) — Позначає лінійну часову складність алгоритму, де n — розмір вхідних даних.

O(1) — Позначає константну часову складність алгоритму.

3NF — Third Normal Form (Третя нормальна форма) — рівень нормалізації бази даних для усунення надлишковості.

API — Application Programming Interface (Інтерфейс програмування додатків) — набір функцій для взаємодії з програмою.

LINQ — Language Integrated Query (Інтегрована мова запитів) — технологія .NET для роботи з даними в коді.

GUI — Graphical User Interface (Графічний користувацький інтерфейс) — візуальна оболонка програми.

ВСТУП

Сучасна медицина активно використовує інформаційні технології для підвищення ефективності управління даними пацієнтів, оптимізації роботи медичних закладів та забезпечення якісного надання медичних послуг. Одним із ключових елементів цифровізації медичної сфери є електронна медична картка, яка дозволяє зберігати, обробляти та аналізувати дані про стан здоров'я пацієнтів у структурованому вигляді. Впровадження таких систем сприяє зменшенню паперового документообігу, підвищенню точності діагностики та полегшенню доступу до медичної інформації для лікарів і пацієнтів.

Метою даної дипломної роботи є розробка прикладної програми для управління медичною карткою пацієнта, яка забезпечує зручне ведення обліку даних про пацієнтів, їхні візити, діагнози, медичні записи та рецепти. Програма розробляється з використанням технології Windows Forms на платформі .NET 8 із застосуванням бази даних SQLite, що дозволяє створити легку, портативну та ефективну систему для невеликих медичних закладів або індивідуальних практик.

Актуальність роботи зумовлена зростаючою потребою в автоматизації медичних процесів, особливо в умовах обмежених ресурсів. SQLite забезпечує простоту розгортання бази даних без необхідності встановлення серверного програмного забезпечення, що робить програму доступною для використання в різних умовах. Windows Forms, у свою чергу, надає зручний інтерфейс для створення користувацьких форм, що спрощує взаємодію з програмою для медичного персоналу. Використання .NET 8 дозволяє забезпечити високу продуктивність, безпеку та підтримку сучасних стандартів розробки програмного забезпечення.

У рамках роботи розроблено desktop-додаток, який включає функціонал для управління інформацією про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Програма має інтуїтивно зрозумілий інтерфейс із вкладковою структурою, що дозволяє швидко перемикатися між різними

категоріями даних. Особливу увагу приділено кастомізації інтерфейсу, зокрема реалізації різнокольорових підписів вкладок для полегшення навігації.

Розроблена програма спрямована на автоматизацію рутинних процесів у медичних закладах, підвищення ефективності роботи лікарів та забезпечення надійного зберігання даних. У процесі виконання роботи розглянуто сучасні підходи до створення desktop-додатків, особливості роботи з SQLite та методи оптимізації користувацького інтерфейсу для медичних інформаційних систем.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1. Постановка основної задачі розробки

Розробка прикладної програми для управління медичною карткою пацієнта спрямована на створення ефективного інструменту для автоматизації обліку медичних даних у невеликих медичних закладах або індивідуальних лікарських практиках. Основна задача полягає у проектуванні та реалізації desktop-додатка, який забезпечує зручне управління інформацією про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Програма повинна мати інтуїтивно зрозумілий інтерфейс, що полегшує введення, редагування та пошук даних, а також надійну систему зберігання інформації з мінімальними вимогами до апаратного забезпечення.

Для реалізації поставленої задачі обрано технологію Windows Forms на платформі .NET 8 із використанням бази даних SQLite. Windows Forms дозволяє створювати користувацькі інтерфейси з широким набором елементів керування, що забезпечують швидку розробку форм для введення та відображення даних. Платформа .NET 8 надає сучасні можливості для створення продуктивних і безпечних додатків, включаючи підтримку кросплатформених компонентів і оптимізацію продуктивності. SQLite, як легка вбудована база даних, забезпечує портативність і простоту розгортання, що є критичним для використання програми в умовах обмежених ресурсів.

Функціональні вимоги до програми включають створення структури для зберігання даних про пацієнтів (ПІБ, дата народження, стать, контактні дані, номер страховки, адреса), лікарів (ПІБ, спеціалізація, номер ліцензії, контактні дані), діагнози (код МКХ, назва, опис), візити (дата, лікар, тип візиту, скарги), медичні записи (дата, візит, діагноз, результати аналізів, опис лікування, нотатки) та рецепти (лікар, медикамент, дозування, інструкції, дати видачі та закінчення, статус). Інтерфейс програми повинен бути організований у вигляді вкладкової структури, де кожна вкладка відповідає окремій категорії даних, із можливістю кастомізації підписів вкладок різними кольорами для полегшення навігації.

Технічні вимоги охоплюють забезпечення стабільності програми, швидкого доступу до даних і мінімального споживання ресурсів. SQLite обрано через відсутність необхідності в серверній інфраструктурі, що спрощує встановлення та використання програми. Додаток має підтримувати базові операції CRUD (створення, читання, оновлення, видалення) для всіх типів даних, а також забезпечувати валідацію введених даних для запобігання помилкам. Особлива увага приділяється оптимізації користувацького досвіду, зокрема через уніфікацію стилів форм і елементів керування, а також додавання підказок для полегшення роботи з програмою.

Основна задача розробки полягає в створенні портативного, функціонального та зручного desktop-додатка, який автоматизує управління медичними даними, відповідає потребам невеликих медичних закладів і забезпечує надійне зберігання інформації з мінімальними вимогами до апаратного забезпечення.

1.2. Огляд та порівняння аналогічних систем.

Системи управління медичними картками, відомі як електронні медичні записи (EMR) або електронні медичні картки (EHR), є важливим інструментом у сучасній медицині. Вони дозволяють автоматизувати облік даних про пацієнтів, оптимізувати робочі процеси медичних закладів і підвищувати якість надання медичних послуг. У цьому розділі розглядаються основні характеристики комерційних і відкритих систем управління медичними картками, їхні переваги та недоліки, а також порівняння з розроблюваною програмою, яка використовує Windows Forms, .NET 8 і SQLite. Для аналізу обрано популярні системи: Epic Systems, OpenEMR і Practice Fusion, оскільки вони представляють різні підходи до реалізації EMR/EHR.

Epic Systems є однією з провідних EHR-систем, орієнтованою на великі медичні заклади, такі як лікарні та клінічні мережі. Вона пропонує комплексне рішення для управління даними пацієнтів, включаючи демографічні дані, медичну історію, лабораторні результати, рецепти, планування візитів і білінг.

Система підтримує інтеграцію з іншими медичними системами через стандарти interoperability, такі як HL7 і FHIR, що забезпечує обмін даними між різними постачальниками медичних послуг. Epic Systems вирізняється високим рівнем кастомізації, підтримкою штучного інтелекту для клінічних рішень і розвиненим порталом для пацієнтів. Проте система має високу вартість впровадження та підтримки, що робить її менш доступною для невеликих практик. Крім того, складність налаштування та потреба в спеціалізованому IT-персоналі можуть бути бар'єрами для малих закладів.

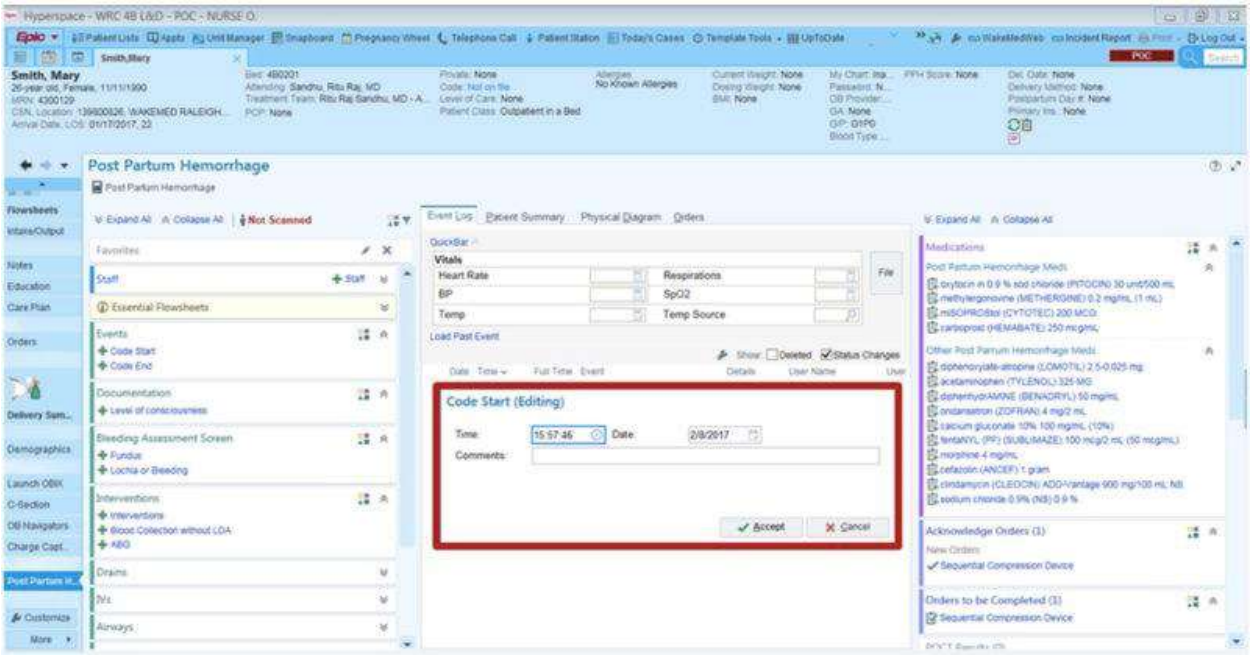


Рисунок 1.2.1 – Epic Systems

OpenEMR — це система з відкритим вихідним кодом, призначена для малих і середніх медичних практик. Вона підтримує управління даними про пацієнтів, планування прийомів, електронні рецепти, лабораторні замовлення та базову звітність. OpenEMR працює на різних платформах, включаючи Windows, Linux і macOS, і використовує базу даних MySQL або PostgreSQL. Система підтримує стандарти HIPAA для захисту даних і пропонує модулі для інтеграції з лабораторіями та білінговими системами. Основною перевагою OpenEMR є безкоштовність і гнучкість завдяки відкритому коду, що дозволяє розробникам адаптувати систему до специфічних потреб. Однак система має застарілий інтерфейс, що може ускладнювати навчання персоналу, а

підтримка залежить від спільноти, що не завжди гарантує оперативне вирішення проблем.

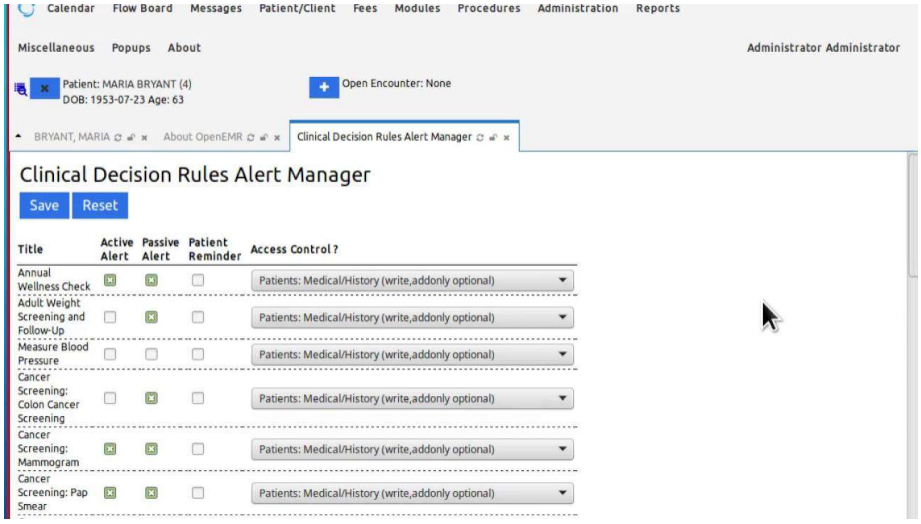


Рисунок 1.2.2 – OpenEMR

Practice Fusion — це хмарна EMR-система, орієнтована на невеликі амбулаторні практики. Вона пропонує інтуїтивний інтерфейс для управління даними пацієнтів, електронними рецептами, плануванням і білінгом. Система підтримує інтеграцію з лабораторіями та аптеками, а також має портал для пацієнтів, який дозволяє переглядати записи та записуватися на прийом. Practice Fusion використовує модель підписки, що робить її доступною для невеликих практик, але залежність від інтернет-з'єднання може бути проблемою в регіонах із нестабільним зв'язком. Крім того, обмежена кастомізація та менший функціонал порівняно з Epic Systems роблять її менш придатною для складних медичних закладів.

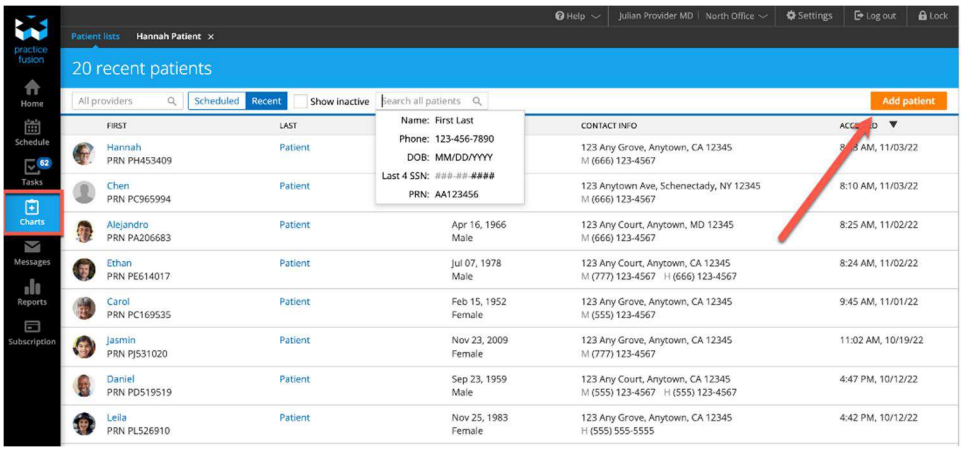


Рисунок 1.2.3 – Practice Fusion

Розроблювана програма, заснована на Windows Forms, .NET 8 і SQLite, орієнтована на невеликі медичні практики з обмеженими ресурсами. Вона забезпечує базовий функціонал управління даними про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти через вкладкову структуру інтерфейсу. Використання SQLite дозволяє створювати портативну базу даних без серверної інфраструктури, що знижує вимоги до апаратного забезпечення. Windows Forms забезпечує швидку розробку зручного інтерфейсу, а .NET 8 гарантує продуктивність і безпеку. На відміну від Epic Systems, програма не підтримує розвинену інтероперабельність або інтеграцію зі штучним інтелектом, але її простота та відсутність ліцензійних витрат роблять її конкурентною альтернативою для малих закладів.

Порівняння систем за ключовими характеристиками наведено в таблиці нижче

Таблиця 1.2.1. Порівняння систем

Характеристика	**Epic Systems**	**OpenEMR**	**Practice Fusion**
Тип системи	Комерційна EHR	Відкрита EMR	Хмарна EMR
Цільова аудиторія	Великі заклади	Малі/середні практики	Малі практики
База даних	Пропрієтарна	MySQL/PostgreSQL	Хмарна
Інтерфейс	Сучасний, складний	Застарілий	Інтуїтивний, сучасний
Інтероперабельність	Висока (HL7, FHIR)	Обмежена	Базова
Вартість	Висока	Безкоштовна	За підпискою
Портативність	Низька	Висока	Залежить від інтернету
Кастомізація	Висока	Висока (відкритий код)	Обмежена
Підтримка	Професійна	Спільнота	Професійна

Epic Systems є лідером для великих закладів завдяки потужному функціоналу, але її вартість і складність роблять її непрактичною для малих практик. OpenEMR пропонує гнучкість і безкоштовність, але потребує додаткових зусиль для модернізації інтерфейсу та підтримки. Practice Fusion підходить для малих практик із хорошим інтернет-з’єднанням, але її хмарна природа може бути обмеженням. Розроблювана програма вирізняється простотою, портативністю та відсутністю витрат, що робить її оптимальним рішенням для невеликих закладів із базовими потребами. Однак відсутність інтероперабельності та професійної підтримки є її основними недоліками порівняно з комерційними системами.

Розроблювана програма займає нішу простих і доступних EMR-систем для малих медичних практик, конкуруючи з OpenEMR за портативністю та вартістю, але поступаючись Epic Systems і Practice Fusion за функціоналом і можливостями інтеграції. Вибір між цими системами залежить від розміру медичного закладу, бюджету та потреб в інтероперабельності.

1.3. Вибір та обґрунтування технологій розробки

Вибір технологій для розробки прикладної програми для управління медичною карткою пацієнта базується на потребах цільової аудиторії — невеликих медичних практик із обмеженими ресурсами, а також на вимогах до портативності, простоти використання та мінімальних витрат на впровадження. Для реалізації проєкту обрано технологію Windows Forms на платформі .NET 8 із використанням бази даних SQLite. Цей розділ обґрунтовує вибір зазначених технологій, аналізуючи їхні переваги в контексті поставлених задач.

Windows Forms обрано як основну технологію для створення користувацького інтерфейсу завдяки її зрілості, простоті та широким можливостям для швидкої розробки desktop-додатків. Ця технологія надає багатий набір елементів керування, таких як текстові поля, кнопки, таблиці (DataGridView) і вкладкові структури (TabControl), які ідеально підходять для створення форм для введення та відображення медичних даних. Windows Forms підтримує drag-and-drop дизайн у Visual Studio, що значно прискорює створення інтерфейсу, а також забезпечує стабільність і сумісність із Windows — основною операційною системою у медичних закладах. Порівняно з альтернативами, такими як WPF або UWP, Windows Forms є менш ресурсомісткою і не потребує глибоких знань сучасних фреймворків, що спрощує розробку для проєктів із базовими вимогами до інтерфейсу.

Платформа .NET 8 обрана через її продуктивність, безпеку та підтримку сучасних стандартів розробки. .NET 8 є останньою версією фреймворку від Microsoft, яка поєднує високу швидкість виконання, оптимізацію пам’яті та

розширені бібліотеки для роботи з даними й інтерфейсом. Вона забезпечує кросплатформений потенціал, хоча в даному проєкті фокусовано на Windows через цільову аудиторію. .NET 8 підтримує інтеграцію з SQLite через Entity Framework Core або ADO.NET, що спрощує роботу з базою даних. Порівняно з попередніми версіями .NET Framework, .NET 8 пропонує кращу продуктивність і сучасні інструменти для дебагінгу, що полегшує розробку й тестування. Альтернативи, такі як Java або Python із Tkinter, менш інтегровані з Windows і потребують додаткових зусиль для створення порівнянного інтерфейсу.

SQLite обрано як базу даних завдяки її легкості, портативності та відсутності потреби в серверній інфраструктурі. SQLite зберігає дані в єдиному файлі, що спрощує розгортання програми на різних комп'ютерах без необхідності встановлення додаткового програмного забезпечення. Це особливо важливо для невеликих медичних практик, де відсутній спеціалізований IT-персонал. SQLite підтримує стандартні SQL-запити, що забезпечує гнучкість у роботі з даними, а також має достатню продуктивність для обробки баз даних обсягом, типовим для малих закладів (до кількох тисяч записів). Порівняно з MySQL або PostgreSQL, SQLite не вимагає налаштування сервера, що знижує витрати на впровадження. Хоча SQLite має обмеження в паралельному доступі, це не є критичним для локального desktop-додатка, призначеного для одного користувача.

Для порівняння технологій розглянуто альтернативні підходи. WPF на .NET 8 забезпечує сучасніший інтерфейс із підтримкою анімацій і XAML, але його складність і вищі вимоги до ресурсів роблять його надмірним для простого додатка. Хмарні рішення, такі як ASP.NET Core із вебінтерфейсом, потребують стабільного інтернет-з'єднання, що може бути проблемою в регіонах із обмеженим доступом до мережі. Бази даних, такі як SQL Server Express, пропонують розширені можливості, але їхнє розгортання ускладнене для невеликих практик. Python із SQLite і PyQt міг би бути альтернативою, але

менша інтеграція з Windows і складність розгортання готового додатка роблять його менш привабливим.

Обраний стек технологій відповідає ключовим вимогам проєкту: портативності, простоті впровадження та зручності для користувачів. Windows Forms дозволяє швидко створити вкладкову структуру з кастомними кольорами підписів, як зазначено в дизайні, забезпечуючи інтуїтивну навігацію. .NET 8 гарантує стабільність і продуктивність, а SQLite забезпечує легке зберігання даних без додаткових витрат. Ці технології разом створюють оптимальне рішення для невеликих медичних практик, де потрібен простий, але функціональний інструмент для управління медичними картками.

Висновки до розділу 1

У першому розділі проведено аналіз задачі розробки прикладної програми для управління медичною карткою пацієнта та визначено ключові вимоги до її реалізації. Основною задачею є створення портативного desktop-додатка для невеликих медичних практик, який забезпечує зручне управління даними про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Програма повинна мати інтуїтивний інтерфейс із вкладковою структурою, підтримувати базові операції CRUD і працювати з мінімальними апаратними вимогами.

Огляд аналогічних систем, таких як Epic Systems, OpenEMR і Practice Fusion, показав, що комерційні EHR-системи пропонують розширений функціонал і інтероперабельність, але є надто складними та дорогими для малих закладів. Системи з відкритим кодом, як OpenEMR, забезпечують гнучкість і безкоштовність, проте мають застарілий інтерфейс і обмежену підтримку. Хмарні рішення, такі як Practice Fusion, зручні для невеликих практик, але залежать від інтернет-з’єднання. Розроблювана програма займає нішу простих і портативних EMR-систем, конкуруючи з OpenEMR за вартістю

та легкістю розгортання, але поступаючись комерційним системам за інтеграційними можливостями.

Обґрунтування вибору технологій підтвердило доцільність використання Windows Forms, .NET 8 і SQLite. Windows Forms забезпечує швидку розробку зручного інтерфейсу з вкладковою структурою та кастомними кольорами підписів, що відповідає вимогам до інтуїтивності. .NET 8 гарантує продуктивність, безпеку та сучасні інструменти розробки, а SQLite пропонує портативне зберігання даних без серверної інфраструктури, що ідеально для малих практик. Порівняння з альтернативами, такими як WPF, ASP.NET Core чи MySQL, показало, що обраний стек є оптимальним за співвідношенням простоти, функціональності та витрат.

Проведений аналіз дозволив чітко сформулювати задачу розробки, оцінити ринкові аналоги та обґрунтувати вибір технологій, які забезпечують створення ефективного, доступного та зручного рішення для управління медичними картками в умовах обмежених ресурсів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ І РОЗРОБКА

2.1. Вибір алгоритмів та їх ефективність

Розробка прикладної програми для управління медичною карткою пацієнта передбачає використання алгоритмів для обробки даних, забезпечення швидкого доступу до інформації та реалізації базових операцій CRUD (створення, читання, оновлення, видалення). Оскільки програма орієнтована на невеликі медичні практики з обмеженими апаратними ресурсами, ключовим критерієм вибору алгоритмів є їхня простота, ефективність і мінімальне споживання ресурсів. У цьому розділі розглядаються основні алгоритми, використані в програмі, їхня роль у реалізації функціоналу та оцінка ефективності в контексті використання Windows Forms, .NET 8 і SQLite.

Обробка даних у базі даних

Для управління даними програма використовує SQLite як вбудовану базу даних, що забезпечує швидкий доступ до інформації через SQL-запити. Основні операції CRUD реалізуються за допомогою бібліотеки System.Data.SQLite або ADO.NET, які дозволяють виконувати запити до бази даних із мінімальними накладними витратами. Алгоритми обробки даних включають:

1. **Вставка даних (Create)**

При додаванні записів, наприклад, нового пацієнта чи рецепта, використовується параметризований SQL-запит INSERT. Параметризація запобігає SQL-ін’єкціям і забезпечує безпеку. Алгоритм вставки є лінійним за часом (O(1) для однієї операції), оскільки SQLite оптимізує операції запису в локальний файл бази даних. Наприклад, для додавання пацієнта до таблиці Patients виконується запит із передачею значень ПІБ, дати народження, статі тощо.

2. **Читання даних (Read)**

Для відображення списків, таких як список пацієнтів у DataGridView, використовується запит SELECT із можливими умовами фільтрації (WHERE).

SQLite оптимізує запити через індекси, якщо вони створені, що забезпечує час виконання O(log n) для пошуку за ключем або O(n) для повного сканування таблиці, де n — кількість записів. У програмі таблиці мають невеликий обсяг (до кількох тисяч записів), тому затримки мінімальні навіть без індексів.

3. **Оновлення даних (Update)**

Редагування записів, наприклад, зміни контактних даних лікаря, реалізується через запит UPDATE із умовою за первинним ключем. Алгоритм має константну складність O(1) для однієї операції, оскільки SQLite швидко локалізує запис за ідентифікатором.

4. **Видалення даних (Delete)**

Видалення записів, таких як скасування візиту, виконується через запит DELETE із умовою за ідентифікатором. Складність операції становить O(1), оскільки SQLite ефективно видаляє окремі записи.

Для забезпечення цілісності даних у програмі використовуються зовнішні ключі (FOREIGN KEY) у таблицях, наприклад, для зв’язку візитів із лікарями та пацієнтами. SQLite автоматично перевіряє обмеження при вставці чи оновленні, що додає незначні накладні витрати, але гарантує консистентність даних. Ефективність операцій із базою даних додатково підвищується завдяки кешуванню запитів у пам’яті та оптимізації транзакцій у SQLite.

Оновлення інтерфейсу

Для відображення даних у вкладках MainForm (пацієнти, візити, рецепти тощо) використовується компонент DataGridView, який заповнюється результатами запитів SELECT.

Алгоритм оновлення інтерфейсу включає:

1. **Завантаження даних**

Після виконання запиту дані з ResultSet перетворюються в об’єкти (наприклад, List<Patient>) і прив’язуються до DataGridView через BindingSource. Алгоритм має лінійну складність O(n), де n — кількість записів, оскільки кожен запис обробляється для відображення.

2. **Фільтрація та сортування**

Для пошуку даних у DataGridView (наприклад, пацієнтів за ПІБ) використовується фільтрація через LINQ-запити до списку об’єктів у пам’яті. LINQ забезпечує зручний синтаксис для фільтрації та сортування з часовою складністю O(n) для лінійного пошуку. Для невеликих наборів даних (до 1000 записів) це достатньо ефективно, але для більших обсягів можна додати індекси в SQLite для прискорення серверних запитів.

3. **Оновлення після змін**

Після додавання, редагування чи видалення запису викликається метод оновлення DataGridView (наприклад, LoadPatients). Алгоритм повторно виконує запит SELECT і перев’язує дані, що має складність O(n). Для оптимізації можна реалізувати часткове оновлення, змінюючи лише модифіковані рядки в BindingSource, але для малих наборів даних поточний підхід є достатньо швидким.

Кастомізація інтерфейсу

Для реалізації різнокольорових підписів вкладок у TabControl використано подію DrawItem, яка дозволяє вручну малювати текст вкладок. Алгоритм малювання включає:

1. **Ідентифікація вкладки**

За індексом вкладки визначається її ім’я (наприклад, tabPatients) і відповідний колір зі словника. Операція має константну складність O(1).

2. **Малювання тексту та фону**

Використовується Graphics для заповнення фону вкладки (активна — білий, неактивна — системний) і виведення тексту з обраним кольором і шрифтом. Складність малювання залежить від кількості вкладок (O(m), де m — кількість вкладок), але оскільки в програмі лише шість вкладок, витрати незначні.

Цей алгоритм є ефективним, оскільки виконується лише при перерисовці вкладок, що відбувається рідко (наприклад, при зміні активної

вкладки). Використання словника для зберігання кольорів забезпечує швидкий доступ до налаштувань і гнучкість при зміні дизайну.

Валідація даних

Для забезпечення коректності введених даних у формах редагування (PatientEditForm, PrescriptionEditForm тощо) застосовується алгоритм валідації перед збереженням. Алгоритм перевіряє обов'язкові поля (наприклад, ПІБ, код МКХ) і формати даних (наприклад, електронна пошта, номер телефону). Валідація виконується лінійно (O(k), де k — кількість полів у формі) і включає:

1. Перевірку на заповненість: Порожні обов'язкові поля викликають повідомлення про помилку.
2. Перевірку формату: Використовуються регулярні вирази для email і телефону, що має складність O(l), де l — довжина рядка.
3. Перевірку логічної коректності: Наприклад, дата закінчення рецепта не може бути раніше дати видачі.

Валідація інтегрована в обробники подій кнопки «Зберегти» і не впливає на загальну продуктивність, оскільки кількість полів у формах обмежена (до 10–15).

Оцінка ефективності

Обрані алгоритми є оптимальними для програми з огляду на її масштаб і цільову аудиторію. SQLite забезпечує швидкі операції CRUD із часовою складністю O(1) для точкових операцій і O(n) для вибірки списків, що достатньо для баз даних із тисячами записів. Алгоритми оновлення інтерфейсу та фільтрації даних через LINQ мають лінійну складність, але для невеликих наборів даних затримки непомітні. Кастомізація вкладок через DrawItem є ефективною завдяки малій кількості вкладок і рідкісному виклику.

Порівняно з альтернативними підходами, такими як використання Entity Framework Core для ORM, прямий доступ через ADO.NET є швидшим для простих запитів і потребує менше пам'яті. Використання складніших алгоритмів пошуку (наприклад, бінарного пошуку) недоцільне через

невеликий обсяг даних. У разі масштабування програми можна додати індекси в SQLite або кешування даних у пам’яті для прискорення операцій.

Обрані алгоритми забезпечують баланс між простотою реалізації, ефективністю та низьким споживанням ресурсів, що відповідає вимогам портативного desktop-додатка для малих медичних практик.

2.2. База даних

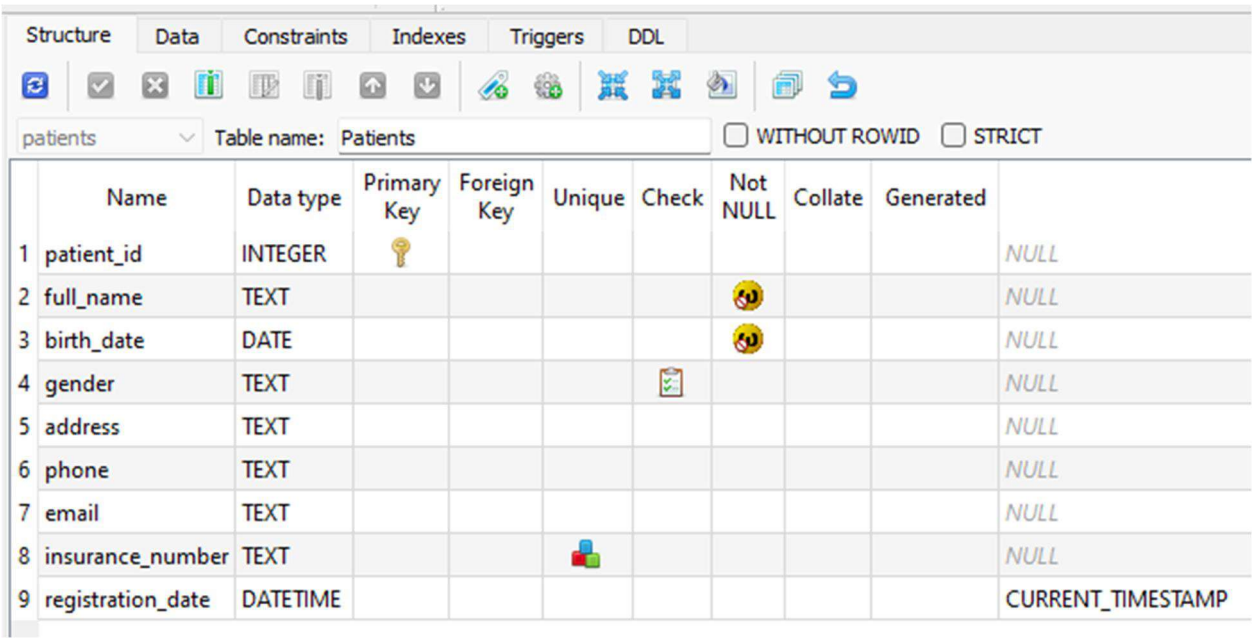
База даних є основою прикладної програми для управління медичною карткою пацієнта, забезпечуючи структуроване зберігання інформації про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Для реалізації бази даних обрано SQLite — легку вбудовану СУБД, яка відповідає вимогам портативності, простоти розгортання та мінімального споживання ресурсів. У цьому розділі описано структуру бази даних, її схему, зв’язки між таблицями, а також особливості реалізації в контексті програми, розробленої на Windows Forms і .NET 8.

SQLite обрано як основну СУБД завдяки її ключовим характеристикам, які ідеально відповідають потребам програми. По-перше, SQLite працює як вбудована база даних, зберігаючи всі дані в єдиному файлі, що усуває потребу в серверній інфраструктурі та спрощує розгортання програми на різних комп’ютерах. По-друге, SQLite є легкою за споживанням ресурсів, що важливо для невеликих медичних практик із базовими апаратними засобами. По-третє, вона підтримує стандартні SQL-запити, що забезпечує гнучкість у роботі з даними, а також має вбудовані механізми транзакцій і перевірки цілісності. Порівняно з альтернативами, такими як MySQL або SQL Server Express, SQLite не потребує налаштування сервера, що знижує витрати на впровадження та адміністрування.

Структура бази даних

База даних складається з шести основних таблиць, які відображають ключові сутності програми: Patients, Doctors, Diagnoses, Visits, MedicalRecords і Prescriptions. Кожна таблиця має первинний ключ для унікальної ідентифікації записів, а зв'язки між таблицями забезпечуються через зовнішні ключі, що гарантують референтну цілісність. Нижче наведено опис таблиць і їхніх полів.

1. **Patients** — таблиця для зберігання інформації про пацієнтів.



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	patient_id	INTEGER								NULL
2	full_name	TEXT								NULL
3	birth_date	DATE								NULL
4	gender	TEXT								NULL
5	address	TEXT								NULL
6	phone	TEXT								NULL
7	email	TEXT								NULL
8	insurance_number	TEXT								NULL
9	registration_date	DATETIME								CURRENT_TIMESTAMP

Рисунок 2.2.1 – Таблиця Patients

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

FullName (TEXT, NOT NULL) — повне ім'я пацієнта.

BirthDate (TEXT) — дата народження (зберігається у форматі ISO 8601).

Gender (TEXT) — стать (наприклад, «Чоловіча», «Жіноча»).

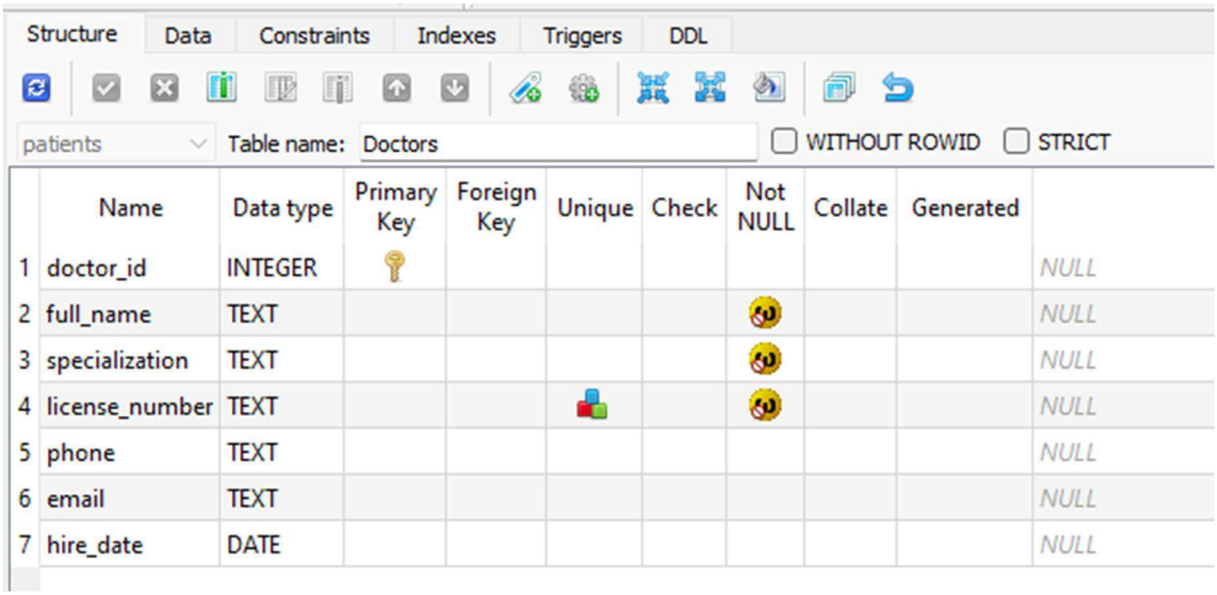
Phone (TEXT) — номер телефону.

Email (TEXT) — електронна пошта.

Address (TEXT) — адреса проживання.

InsuranceNumber (TEXT) — номер страхового поліса.

2. **Doctors** — таблиця для зберігання даних про лікарів.



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	doctor_id	INTEGER								NULL
2	full_name	TEXT								NULL
3	specialization	TEXT								NULL
4	license_number	TEXT								NULL
5	phone	TEXT								NULL
6	email	TEXT								NULL
7	hire_date	DATE								NULL

Рисунок 2.2.2 – Таблиця 2Doctors

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

FullName (TEXT, NOT NULL) — повне ім’я лікаря.

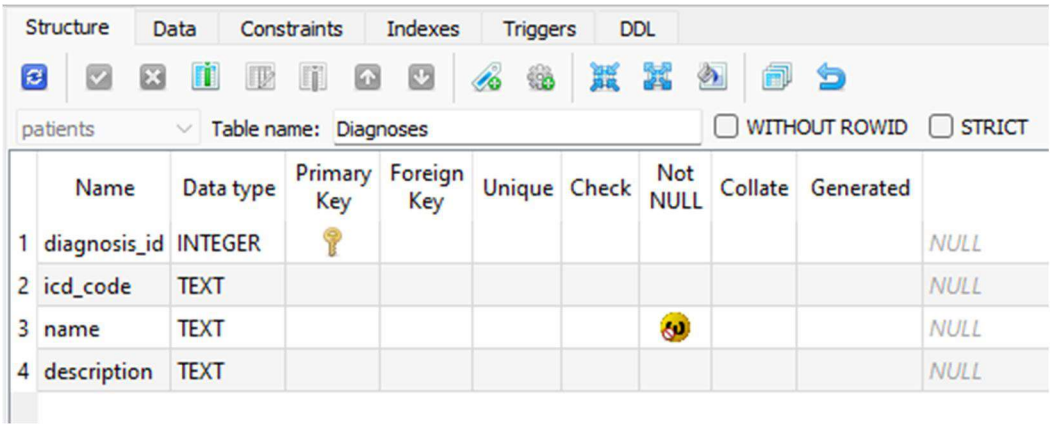
Specialization (TEXT) — спеціалізація (наприклад, «Терапевт», «Кардіолог»).

LicenseNumber (TEXT) — номер ліцензії.

Phone (TEXT) — номер телефону.

Email (TEXT) — електронна пошта.

3. **Diagnoses** — таблиця для зберігання інформації про діагнози.



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	
1	diagnosis_id	INTEGER								NULL
2	icd_code	TEXT								NULL
3	name	TEXT								NULL
4	description	TEXT								NULL

Рисунок 2.2.3 – Таблиця Diagnoses

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

ICDCode (TEXT, NOT NULL) — код діагнозу за МКХ-10.

Name (TEXT, NOT NULL) — назва діагнозу.

Description (TEXT) — опис діагнозу.

4. **Visits** — таблиця для обліку візитів пацієнтів до лікарів.

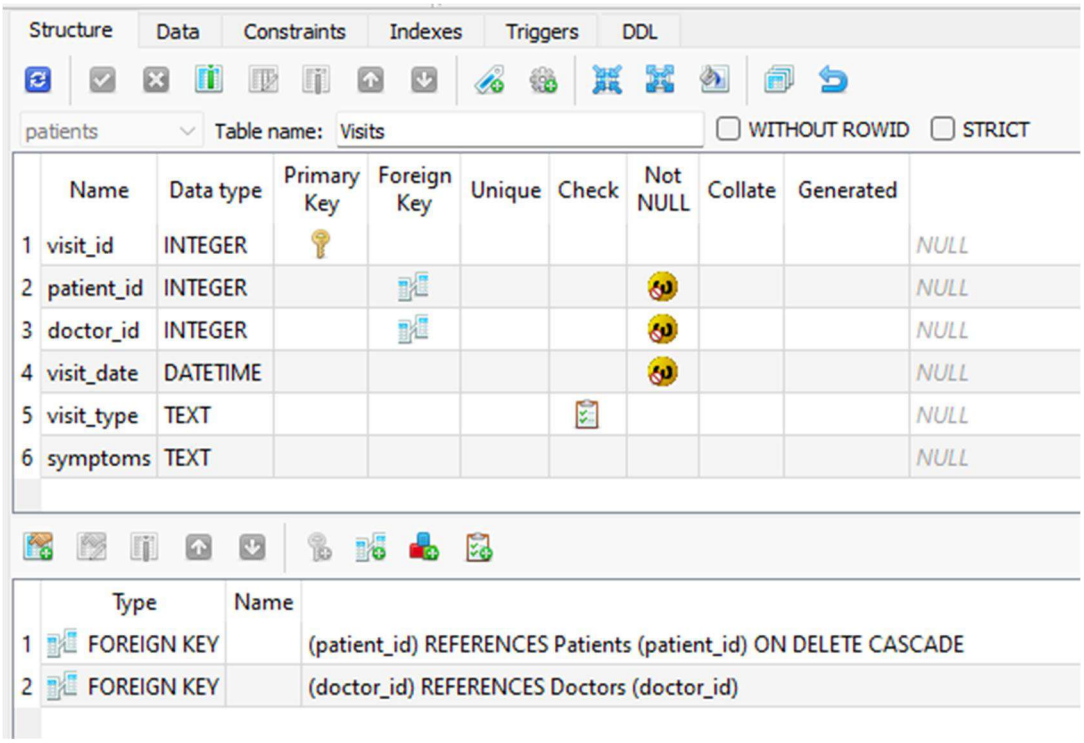


Рисунок 2.2.4 – Таблиця Visits

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

PatientId (INTEGER, FOREIGN KEY REFERENCES Patients(Id)) — ідентифікатор пацієнта.

DoctorId (INTEGER, FOREIGN KEY REFERENCES Doctors(Id)) — ідентифікатор лікаря.

VisitDate (TEXT, NOT NULL) — дата і час візиту.

VisitType (TEXT) — тип візиту (наприклад, «Первинний», «Повторний»).

Complaints (TEXT) — скарги пацієнта.

5. **MedicalRecords** — таблиця для зберігання медичних записів, пов’язаних із візитами.

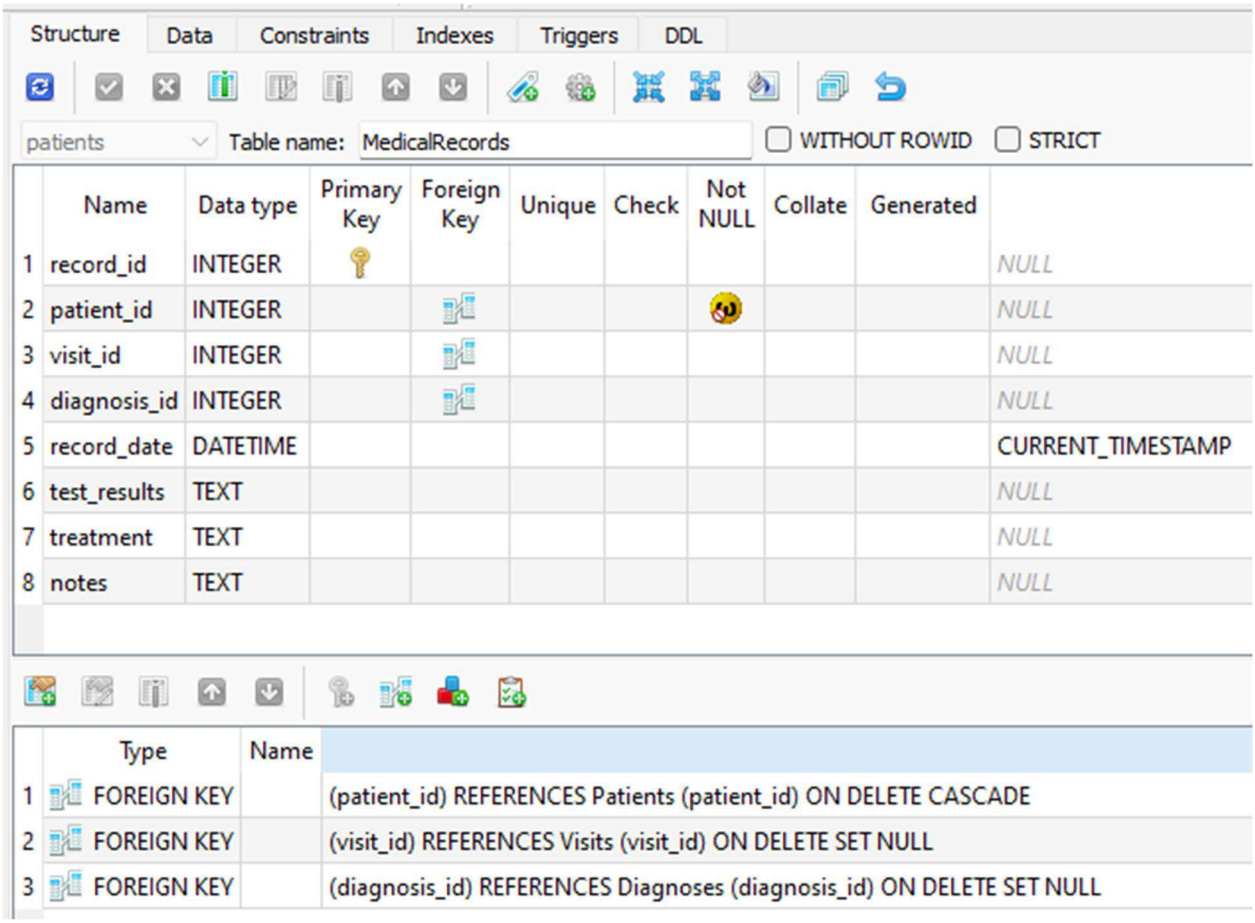


Рисунок 2.2.5 – Таблиця MedicalRecords

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

VisitId (INTEGER, FOREIGN KEY REFERENCES Visits(Id)) — ідентифікатор візиту.

DiagnosisId (INTEGER, FOREIGN KEY REFERENCES Diagnoses(Id)) — ідентифікатор діагнозу.

RecordDate (TEXT, NOT NULL) — дата створення запису.

TestResults (TEXT) — результати аналізів.

Treatment (TEXT) — опис лікування.

Notes (TEXT) — додаткові нотатки.

6. **Prescriptions** — таблиця для обліку рецептів.

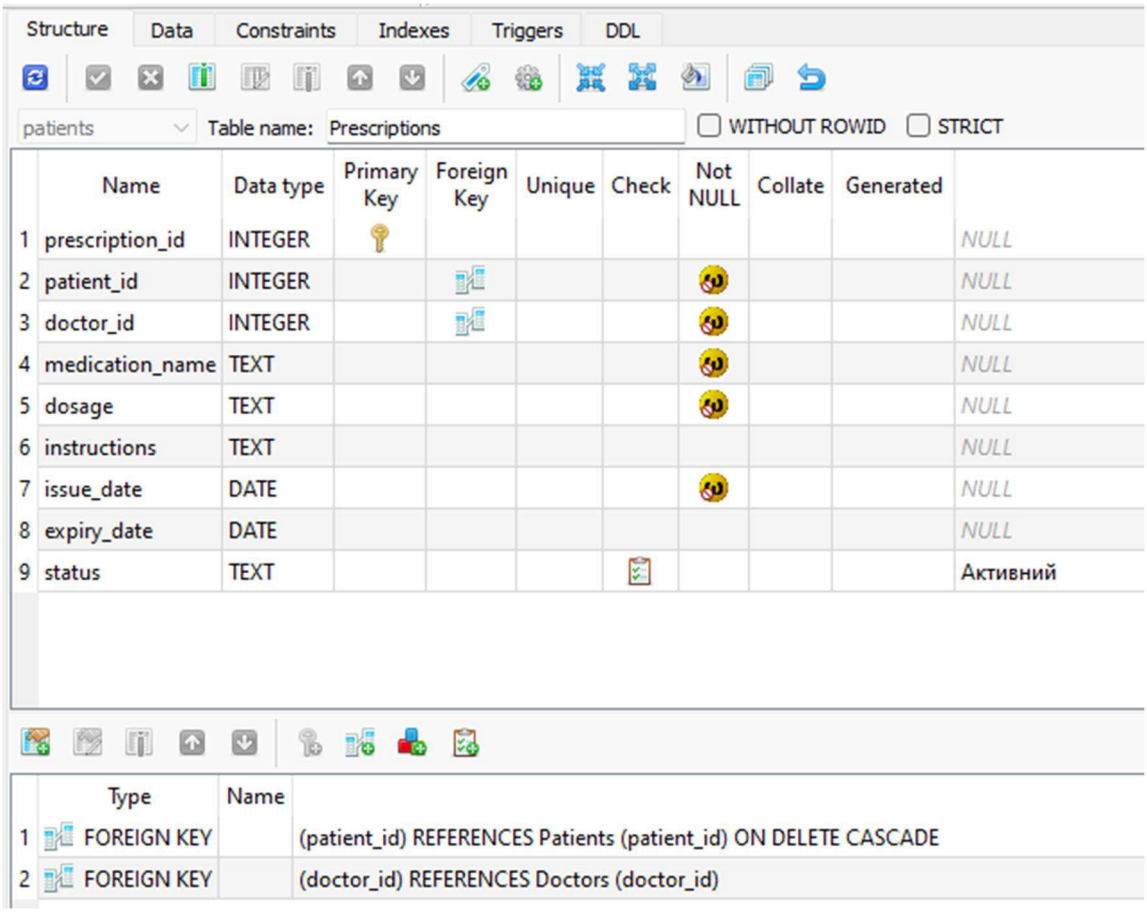


Рисунок 2.2.6 – Таблиця Prescriptions

Id (INTEGER, PRIMARY KEY, AUTOINCREMENT) — унікальний ідентифікатор.

VisitId (INTEGER, FOREIGN KEY REFERENCES Visits(Id)) — ідентифікатор візиту.

DoctorId (INTEGER, FOREIGN KEY REFERENCES Doctors(Id)) — ідентифікатор лікаря.

Medication (TEXT, NOT NULL) — назва медикаменту.

Dosage (TEXT) — дозування.

Instructions (TEXT) — інструкції щодо прийому.

IssueDate (TEXT, NOT NULL) — дата видачі.

EndDate (TEXT) — дата закінчення дії.

Status (TEXT) — статус рецепта (наприклад, «Активний», «Завершений»).

Зв’язки між таблицями

Таблиці пов’язані через зовнішні ключі для забезпечення логічної цілісності даних:

- Таблиця Visits пов’язує Patients і Doctors через PatientId і DoctorId, що дозволяє відстежувати, який лікар приймав якого пацієнта.
- Таблиця MedicalRecords пов’язана з Visits через VisitId і з Diagnoses через DiagnosisId, що відображає діагнози, встановлені під час конкретного візиту.
- Таблиця Prescriptions пов’язана з Visits через VisitId і з Doctors через DoctorId, що вказує, який лікар виписав рецепт у межах візиту.

Схема бази даних відповідає нормалізованій формі (3NF), що мінімізує надлишковість даних і забезпечує ефективне зберігання. Наприклад, інформація про діагнози зберігається в окремій таблиці Diagnoses, а не дублюється в MedicalRecords, що зменшує обсяг бази даних і спрощує оновлення.

Для підвищення ефективності бази даних використано кілька підходів:

- **Індекси**: Для полів, які часто використовуються в умовах WHERE (наприклад, PatientId у Visits), можна створити індекси, хоча для невеликих обсягів даних (до кількох тисяч записів) це не критично.
- **Транзакції**: Групові операції, такі як масове оновлення статусів рецептів, виконуються в межах транзакцій для забезпечення атомарності та зменшення витрат часу.
- **Кешування**: Дані, які рідко змінюються (наприклад, список діагнозів), можуть кешуватися в пам’яті для прискорення доступу, хоча в поточній реалізації це не використано через невеликий обсяг даних.

Обмеження SQLite, такі як підтримка лише одного активного запису одночасно, не є проблемою, оскільки програма призначена для локального використання одним користувачем. Для захисту даних реалізовано резервне копіювання бази даних (копіювання файлу patients.db), яке може виконуватися вручну або автоматично за розкладом.

База даних, спроектована для програми, забезпечує структуроване та надійне зберігання медичних даних із використанням SQLite. Шість таблиць із чіткими зв'язками дозволяють ефективно управляти інформацією про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти. Використання параметризованих запитів і нормалізації даних гарантує безпеку та мінімізує надлишковість. Реалізація через System.Data.SQLite інтегрується з Windows Forms і .NET 8, забезпечуючи швидкий доступ до даних і зручне відображення в інтерфейсі. Структура бази даних є оптимальною для невеликих медичних практик, відповідаючи вимогам портативності, простоти та ефективності.

2.3. Спроєктовані класи програми

Програма для управління медичною карткою пацієнта розроблена з використанням об'єктно-орієнтованого підходу на платформі .NET 8 і технології Windows Forms. Для забезпечення модульності, зрозумілості та легкості підтримки код структуровано у вигляді класів, які відповідають за бізнес-логіку, взаємодію з базою даних і відображення інтерфейсу. У цьому розділі описано основні класи програми, їх призначення, структуру та взаємозв'язки, з акцентом на інтеграцію з базою даних SQLite і реалізацію функціоналу вкладкової структури з різнокольоровими підписами.

Архітектура програми базується на принципі поділу обов'язків, де класи поділено на три основні категорії:

1. **Моделі даних** — класи, що представляють сутності бази даних (пацієнти, лікарі, діагнози тощо).
2. **Репозиторії** — класи для взаємодії з базою даних, що реалізують операції CRUD.
3. **Форми** — класи Windows Forms для створення інтерфейсу та обробки подій користувача.

Цей підхід забезпечує чітке розділення логіки даних, доступу до бази та інтерфейсу, що полегшує тестування й подальший розвиток програми.

Моделі даних

Моделі даних відповідають таблицям бази даних, описаним у розділі 2.2, і використовуються для передачі даних між базою та інтерфейсом. Кожен клас моделі містить властивості, що відображають поля відповідної таблиці. Нижче наведено основні класи моделей:

1. **Patient** — представляє дані про пацієнта.

```
// Модель для таблиці Patients
21 references
public class Patient
{
    17 references
    public int PatientId { get; set; } // patient_id (INTEGER PRIMARY KEY AUTOINCREMENT)
    10 references
    public string FullName { get; set; } // full_name (TEXT NOT NULL)
    6 references
    public DateTime? BirthDate { get; set; } // birth_date (DATE NOT NULL)
    6 references
    public string Gender { get; set; } // gender (TEXT CHECK(gender IN ('Чоловік', 'Жінка', 'Інше')))
    6 references
    public string Address { get; set; } // address (TEXT)
    6 references
    public string Phone { get; set; } // phone (TEXT)
    6 references
    public string Email { get; set; } // email (TEXT)
    6 references
    public string InsuranceNumber { get; set; } // insurance_number (TEXT UNIQUE)
    6 references
    public DateTime? RegistrationDate { get; set; } // registration_date (DATETIME DEFAULT CURRENT_TIMESTAMP)
}
```

Рисунок 2.3.1 – Клас Patients

2. **Doctor** — містить інформацію про лікаря.

```
15 references
public class Doctor
{
    6 references
    public int DoctorId { get; set; } // doctor_id (INTEGER PRIMARY KEY AUTOINCREMENT)
    9 references
    public string FullName { get; set; } // full_name (TEXT NOT NULL)
    8 references
    public string Specialization { get; set; } // specialization (TEXT NOT NULL)
    6 references
    public string LicenseNumber { get; set; } // license_number (TEXT UNIQUE NOT NULL)
    6 references
    public string Phone { get; set; } // phone (TEXT)
    6 references
    public string Email { get; set; } // email (TEXT)
    7 references
    public DateTime? HireDate { get; set; } // hire_date (DATE)
}
```

Рисунок 2.3.2 – Клас Doctor

3. **Diagnosis** — описує діагноз.

```
// Модель для таблиці Diagnoses
16 references
public class Diagnosis
{
    8 references
    public int DiagnosisId { get; set; } // diagnosis_id (INTEGER PRIMARY KEY AUTOINCREMENT)
    6 references
    public string IcdCode { get; set; } // icd_code (TEXT)
    9 references
    public string Name { get; set; } // name (TEXT NOT NULL)
    6 references
    public string Description { get; set; } // description (TEXT)
}
```

Рисунок 2.3.3 – Клас Diagnosis

4. **Visit** — представляє візит пацієнта до лікаря.

```
// Модель для таблиці Visits
14 references
public class Visit
{
    7 references
    public int VisitId { get; set; } // visit_id (INTEGER PRIMARY KEY AUTOINCREMENT)
    8 references
    public int PatientId { get; set; } // patient_id (INTEGER NOT NULL, FOREIGN KEY)
    8 references
    public int DoctorId { get; set; } // doctor_id (INTEGER NOT NULL, FOREIGN KEY)
    9 references
    public DateTime? VisitDate { get; set; } // visit_date (DATETIME NOT NULL)
    7 references
    public string VisitType { get; set; } // visit_type (TEXT CHECK(visit_type IN ('Первинний', 'Повторний',
    6 references
    public string Symptoms { get; set; } // symptoms (TEXT)
}
```

Рисунок 2.3.4 – Клас Visit

5. **MedicalRecord** — містить медичний запис, пов’язаний із візитом.

```
public class MedicalRecord
{

    public int RecordId { get; set; } // record_id (INTEGER PRIMARY KEY AUTOINCREMENT)

    public int PatientId { get; set; } // patient_id (INTEGER NOT NULL, FOREIGN KEY)

    public int? VisitId { get; set; } // visit_id (INTEGER, FOREIGN KEY, NULLABLE)

    public int? DiagnosisId { get; set; } // diagnosis_id (INTEGER, FOREIGN KEY, NULLABLE)

    public DateTime? RecordDate { get; set; } // record_date (DATETIME DEFAULT CURRENT_TIMESTAMP)

    public string TestResults { get; set; } // test_results (TEXT)

    public string Treatment { get; set; } // treatment (TEXT)
    6 references
    public string Notes { get; set; } // notes (TEXT)
}
```

Рисунок 2.3.5 – Клас MedicalRecord

6. **Prescription** — описує рецепт.

```
public class Prescription
{
    6 references
    public int PrescriptionId { get; set; } // prescription_id (INTEGER PRIMARY KEY AUTOINCREMENT)
    7 references
    public int PatientId { get; set; } // patient_id (INTEGER NOT NULL, FOREIGN KEY)
    8 references
    public int DoctorId { get; set; } // doctor_id (INTEGER NOT NULL, FOREIGN KEY)
    7 references
    public string MedicationName { get; set; } // medication_name (TEXT NOT NULL)
    6 references
    public string Dosage { get; set; } // dosage (TEXT NOT NULL)
    6 references
    public string Instructions { get; set; } // instructions (TEXT)
    7 references
    public DateTime? IssueDate { get; set; } // issue_date (DATE NOT NULL)
    6 references
    public DateTime? ExpiryDate { get; set; } // expiry_date (DATE)
    6 references
    public string Status { get; set; } // status (TEXT DEFAULT 'Активний' CHECK(status IN ('Активний', 'Виконаний',
}
```

Рисунок 2.3.6 – Клас Prescription

Ці класи є простими POCO (Plain Old CLR Object), що забезпечує легкість їх використання для мапінгу даних із SQLite та відображення в інтерфейсі.

Репозиторії

Репозиторії відповідають за взаємодію з базою даних і інкапсулюють логіку виконання SQL-запитів. Кожен репозиторій відповідає одній таблиці та надає методи для операцій CRUD. Основні класи репозиторіїв:

1. **DatabaseManager** — забезпечує підключення до SQLite.

```
13 references
public class DatabaseManager
{
    private readonly string dbPath = "patients.db";
    private readonly string connectionString;

    1 reference
    public DatabaseManager()...

    12 references
    public string ConnectionString => connectionString;

    1 reference
    private void InitializeDatabase()
    {
        if (!File.Exists(dbPath))...

        using (var connection = new SQLiteConnection(connectionString))
        {
            connection.Open();
            string[] createTableQueries =...;

            foreach (var query in createTableQueries)
            {
                using (var command = new SQLiteCommand(query, connection))
                {
                    command.ExecuteNonQuery();
                }
            }
        }
    }
}
```

Рисунок 2.3.7 – Клас DatabaseManager

2. **PatientRepository** — реалізує операції з таблицею Patients.

```
5 references
public class PatientRepository
{
    private readonly string connectionString;

    1 reference
    public PatientRepository(string connectionString)...

    1 reference
    public List<Patient> GetAll()...

    0 references
    public Patient GetById(int patientId)...

    1 reference
    public void Add(Patient patient)...

    1 reference
    public void Update(Patient patient)...

    1 reference
    public void Delete(int patientId)...
}
```

Рисунок 2.3.8 – Клас PatientRepository

Аналогічні репозиторії створено для інших таблиць: DoctorRepository, DiagnosisRepository, VisitRepository, MedicalRecordRepository, PrescriptionRepository. Кожен із них містить методи для додавання, отримання, оновлення та видалення записів, використовуючи параметризовані запити для безпеки.

Спроєктовані класи програми забезпечують чіткий поділ логіки даних, доступу до бази та інтерфейсу. Моделі даних відображають структуру бази SQLite, репозиторії інкапсулюють SQL-запити, а форми реалізують зручний інтерфейс із вкладковою структурою. Використання Windows Forms і .NET 8 дозволяє створити стабільний і ефективний додаток, який відповідає потребам невеликих медичних практик. Модульна структура класів полегшує підтримку та подальший розвиток програми.

2.4. Програмна реалізація

Програмна реалізація прикладної програми для управління медичною карткою пацієнта виконана з використанням технології Windows Forms на платформі .NET 8 і бази даних SQLite. Проєкт структуровано для забезпечення модульності, зрозумілості коду та зручності підтримки. У цьому розділі описано склад проєкту, включаючи структуру папок, основні файли, а також детальний опис форм, їх функціоналу та особливостей реалізації, зокрема вкладкової структури з різнокольоровими підписами.

Склад проєкту

Проєкт організовано як рішення Visual Studio (.NET 8 Windows Forms Application) із чіткою структурою папок і файлів.

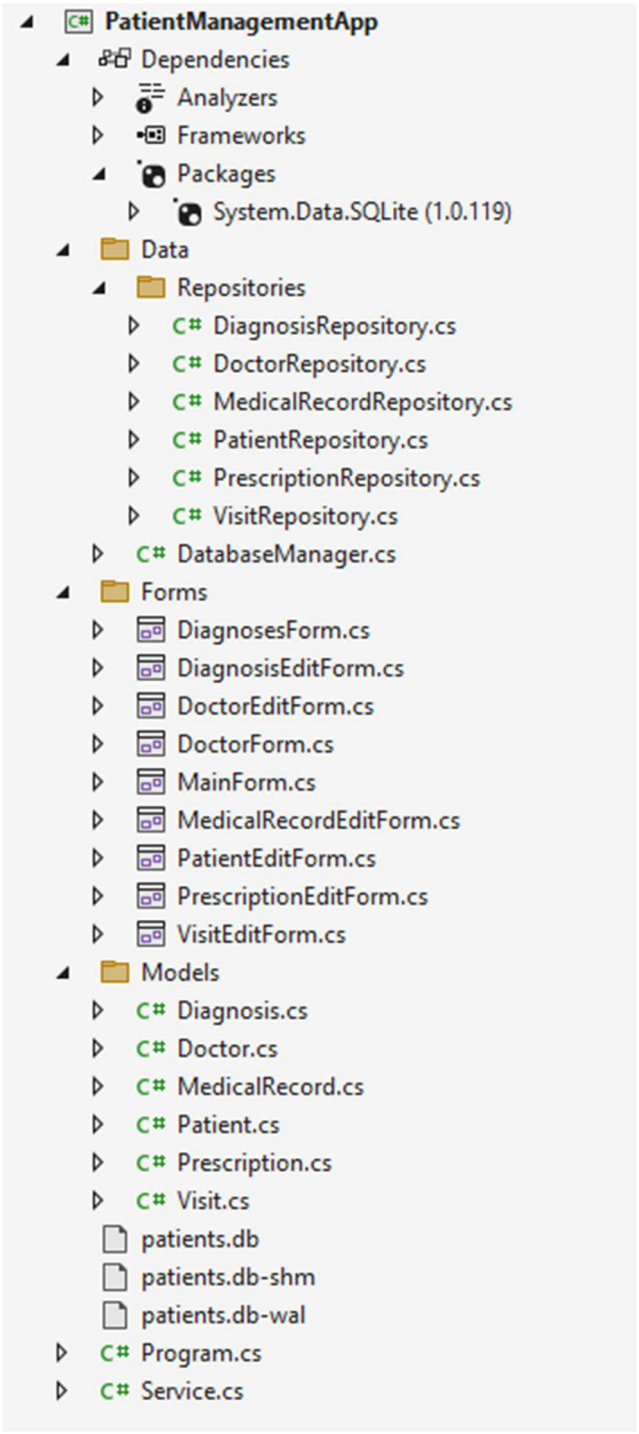


Рисунок 2.4.1 – Склад проєкту

Основні компоненти проєкту:

1. **Папка Models** — містить класи моделей даних, що відповідають таблицям бази даних:
   - Patient.cs — клас для даних про пацієнтів.
   - Doctor.cs — клас для даних про лікарів.
   - Diagnosis.cs — клас для діагнозів.

- Visit.cs — клас для візитів.
- MedicalRecord.cs — клас для медичних записів.
- Prescription.cs — клас для рецептів.

2. **Папка Repositories** — містить класи для взаємодії з базою даних:
   - DatabaseManager.cs — клас для ініціалізації бази даних і управління підключенням до SQLite.
   - PatientRepository.cs — репозиторій для операцій із таблицею Patients.
   - DoctorRepository.cs — репозиторій для таблиці Doctors.
   - DiagnosisRepository.cs — репозиторій для таблиці Diagnoses.
   - VisitRepository.cs — репозиторій для таблиці Visits.
   - MedicalRecordRepository.cs — репозиторій для таблиці MedicalRecords.
   - PrescriptionRepository.cs — репозиторій для таблиці Prescriptions.
3. **Папка Forms** — містить класи форм Windows Forms:
   - MainForm.cs — головна форма з вкладковою структурою.
   - PatientEditForm.cs — форма для додавання/редагування пацієнтів.
   - DoctorEditForm.cs — форма для лікарів.
   - DiagnosisEditForm.cs — форма для діагнозів.
   - VisitEditForm.cs — форма для візитів.
   - MedicalRecordEditForm.cs — форма для медичних записів.
   - PrescriptionEditForm.cs — форма для рецептів.
   - DoctorForm.cs — форма для перегляду списку лікарів.
   - DiagnosesForm.cs — форма для перегляду списку діагнозів.
4. **Ресурси**:
   - patients.db — файл бази даних SQLite, що створюється автоматично під час першого запуску.
5. **Кореневі файли**:
   - Program.cs — точка входу програми, що ініціалізує головну форму.

- App.config — конфігураційний файл із рядком підключення до SQLite.
- .csproj — файл проєкту, що визначає залежності, зокрема System.Data.SQLite.

Структура проєкту відповідає принципу поділу обов'язків: моделі даних ізольовано від логіки доступу до бази (репозиторії), а форми відповідають за інтерфейс і взаємодію з користувачем. Залежності включають бібліотеки System.Data.SQLite для роботи з базою даних і стандартні бібліотеки .NET 8 для Windows Forms.

Форми є основою інтерфейсу програми, забезпечуючи відображення даних і взаємодію з користувачем. Усі форми використовують уніфікований стиль (шрифт Segoe UI, світло-сірий фон, TableLayoutPanel для компонування) для консистентного вигляду. Нижче описано основні форми, їх функціонал і особливості реалізації.

1. **MainForm** — головна форма програми з вкладковою структурою.

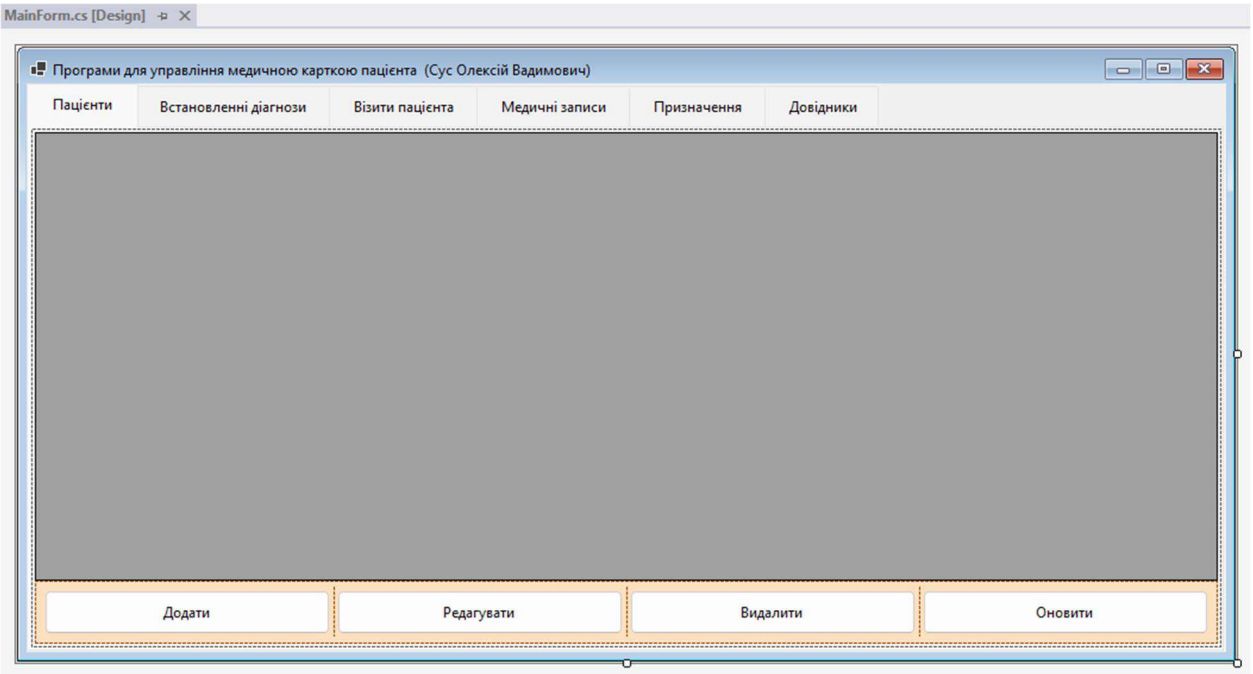


Рисунок 2.4.2 – Форма MainForm

Відображає дані у вкладках (Пацієнти, Діагнози, Візити, Медичні записи, Рецепти, Довідники) через TabControl (tabControlMain). Кожна

вкладка містить DataGridView для списку записів і кнопки для додавання, редагування, видалення та оновлення даних.

Вкладки

tabPatients — список пацієнтів із даними (ПІБ, дата народження, стать, телефон тощо).

tabPatientDiagnoses — діагнози, пов’язані з вибраним пацієнтом.

tabPatientVisits — візити пацієнта із зазначенням лікаря та дати.

tabMedicalRecords — медичні записи, пов’язані з візитами.

tabPrescriptions — рецепти, виписані лікарями.

tabPage1 — довідники (лікарі, діагнози).

Особливості

Різнокольорові підписи вкладок реалізовано через подію DrawItem, використовуючи словник tabColors для асоціації вкладок із кольорами (наприклад, червоний для Пацієнтів, синій для Діагнозів).

Кожна вкладка містить TableLayoutPanel із DataGridView і панеллю кнопок (btnAdd, btnEdit, btnDelete, btnRefresh).

Дані завантажуються через репозиторії (наприклад, patientRepository.GetAllPatients()) і прив’язуються до DataGridView через BindingSource.

Обробники подій

btnAdd_Click — відкриває форму редагування (наприклад, PatientEditForm).

btnEdit_Click — відкриває форму редагування для вибраного запису.

btnDelete_Click — видаляє запис із підтвердженням.

btnRefresh_Click — оновлює дані в DataGridView.

2. **PatientEditForm** — форма для додавання або редагування даних пацієнта.

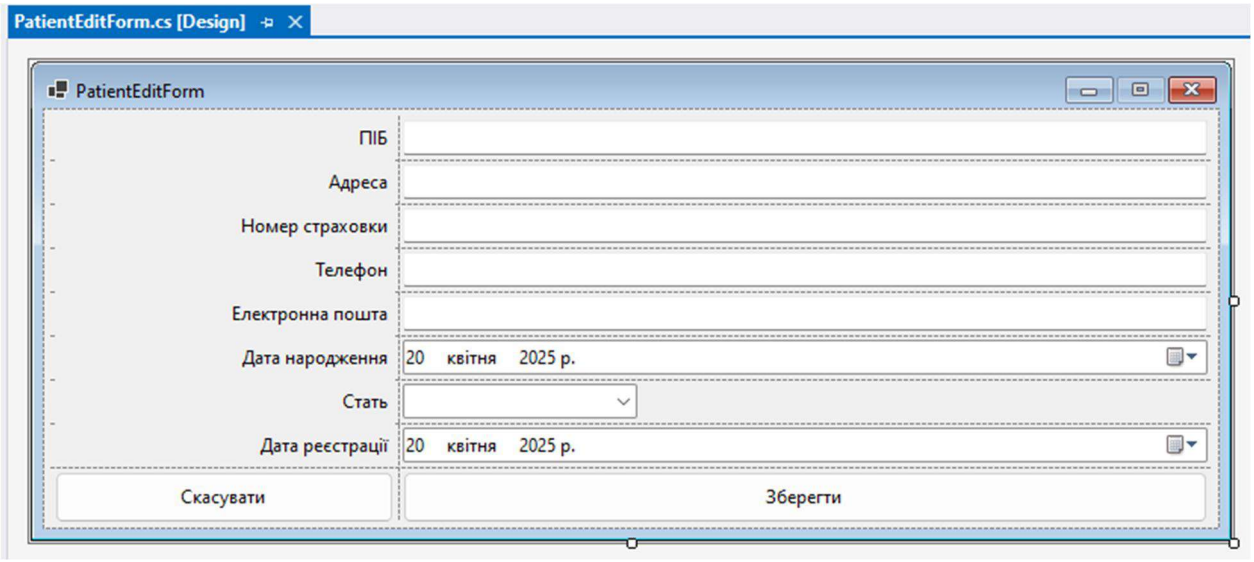


Рисунок 2.4.3 – Форма PatientEditForm

Дозволяє вводити/редагувати ПІБ, дату народження, стать, телефон, email, адресу та номер страховки.

3. **DoctorEditForm** — форма для додавання/редагування даних лікаря.

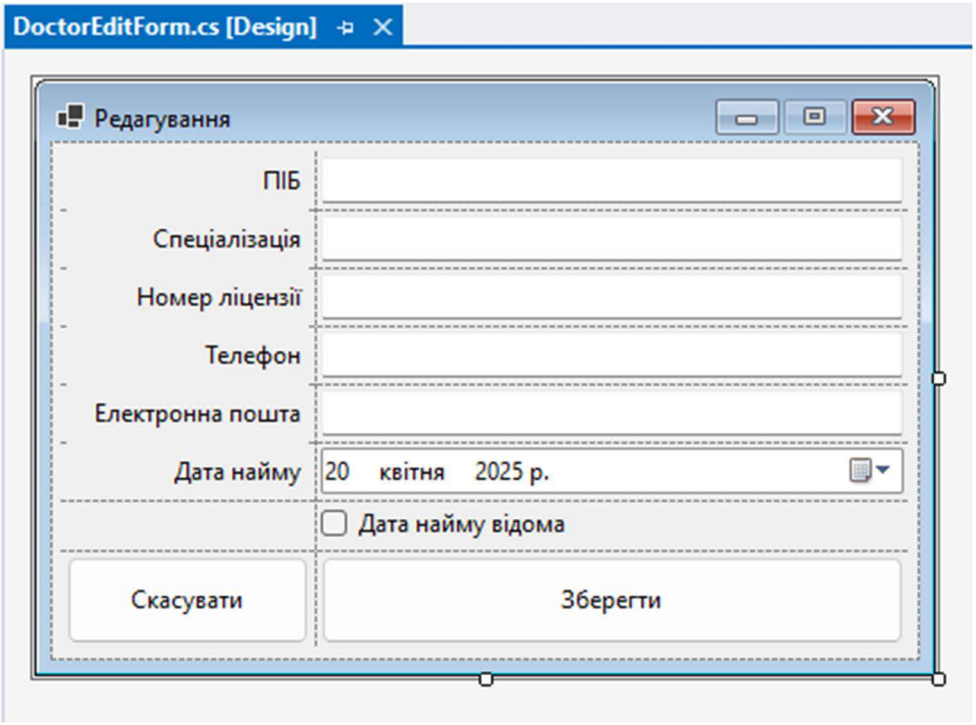


Рисунок 2.4.4 – Форма DoctorEditForm

Введення/редагування ПІБ, спеціалізації, номера ліцензії, телефону та email. Аналогічна валідація та збереження через DoctorRepository.

4. **DiagnosisEditForm** — форма для діагнозів.

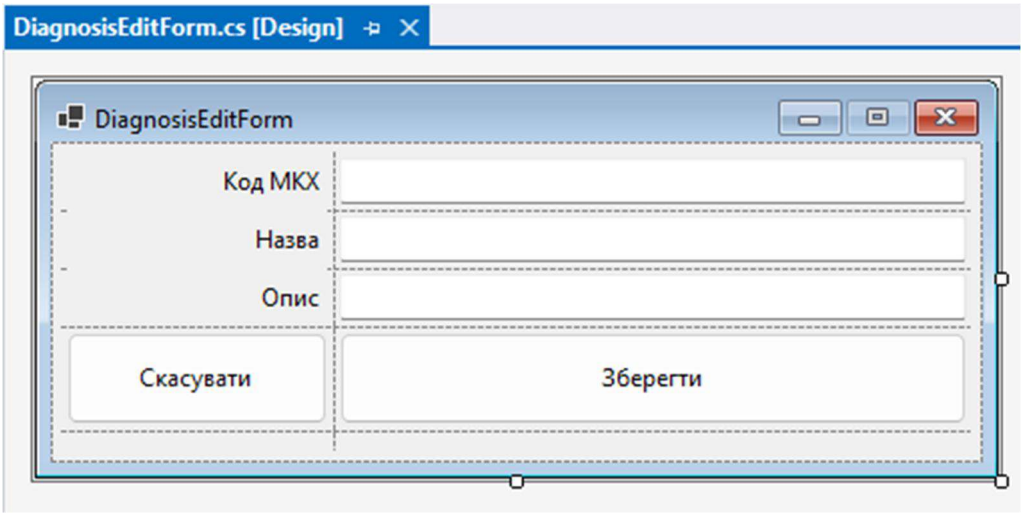


Рисунок 2.4.5 – Форма DiagnosisEditForm

Введення/редагування коду МКХ, назви та опису діагнозу.

Перевірка унікальності txtICDCode перед збереженням через DiagnosisRepository.

5. **VisitEditForm** — форма для візитів.

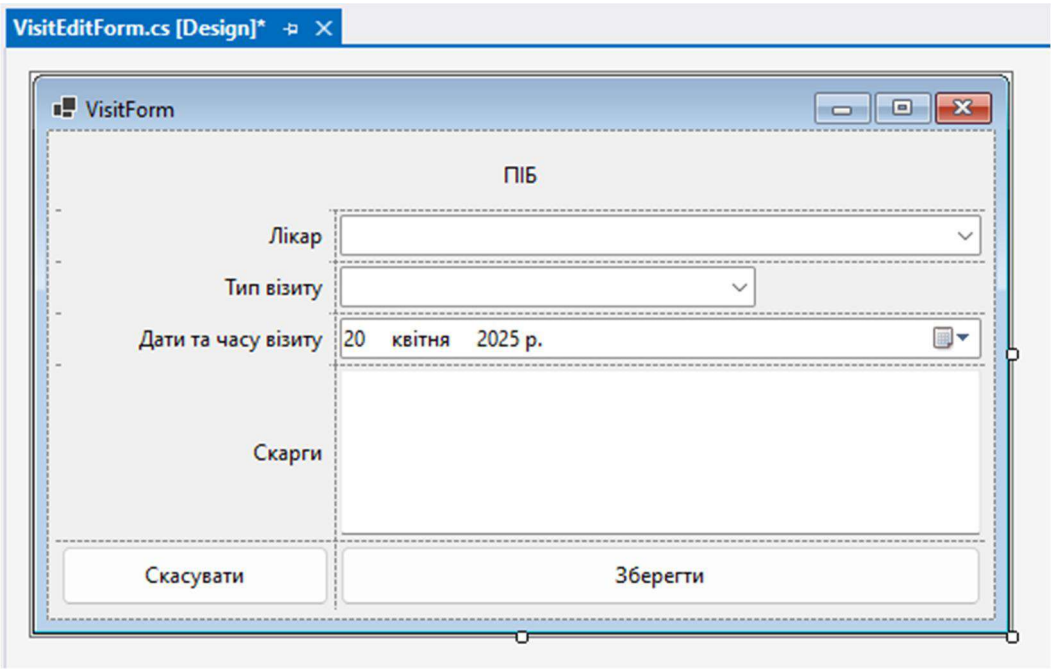


Рисунок 2.4.6 – Форма VisitEditForm

Введення/редагування дати візиту, типу візиту, скарг, вибір пацієнта та лікаря. Валідація вибору пацієнта та лікаря перед збереженням через VisitRepository.

6. **MedicalRecordEditForm** — форма для медичних записів.

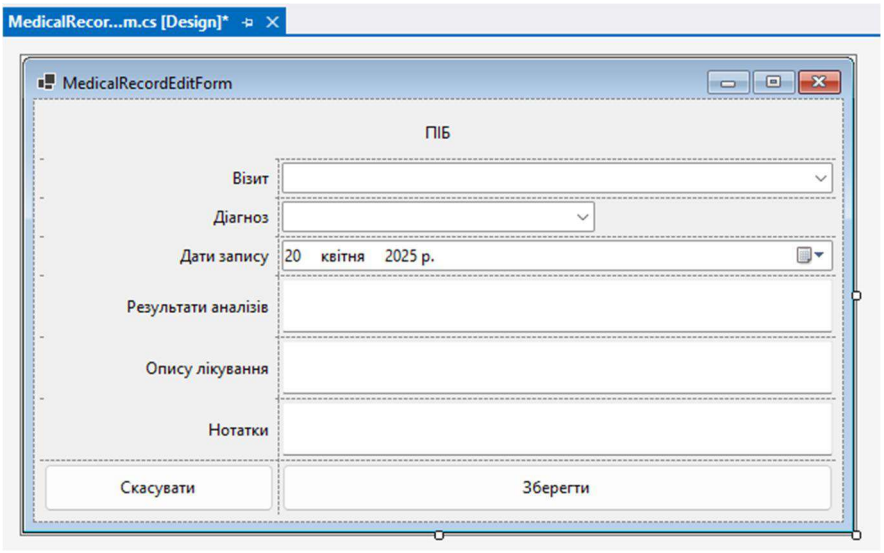


Рисунок 2.4.7 – Форма MedicalRecordEditForm

Введення/редагування дати запису, результатів аналізів, опису лікування, нотаток і вибір візиту та діагнозу. Збереження через MedicalRecordRepository із перевіркою зв’язаних даних.

7. **PrescriptionEditForm** — форма для рецептів.

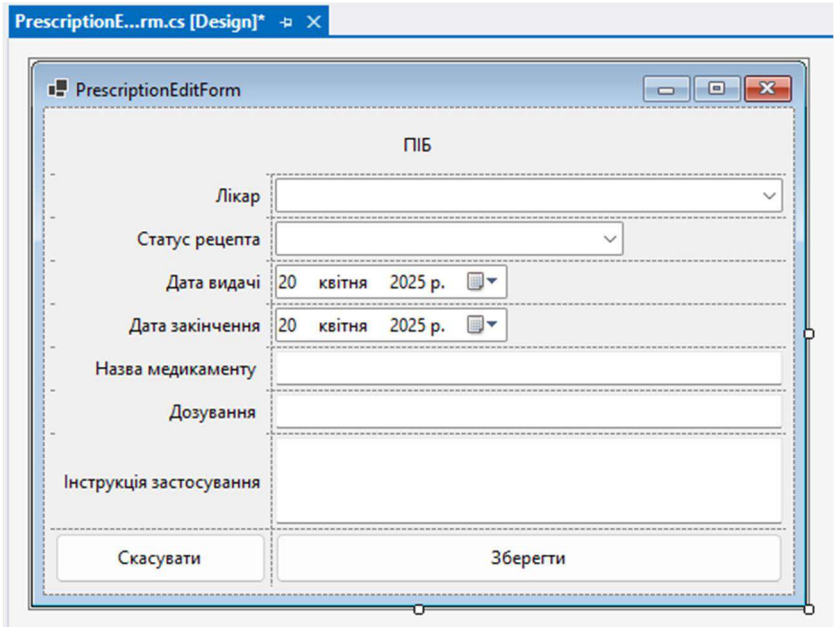


Рисунок 2.4.8 – Форма PrescriptionEditForm

Введення/редагування медикаменту, дозування, інструкцій, дат видачі та закінчення, статусу, вибір візиту та лікаря. Перевірка дат (кінцева дата не раніше початкової) і збереження через PrescriptionRepository.

8. **DoctorForm** — форма для перегляду списку лікарів.

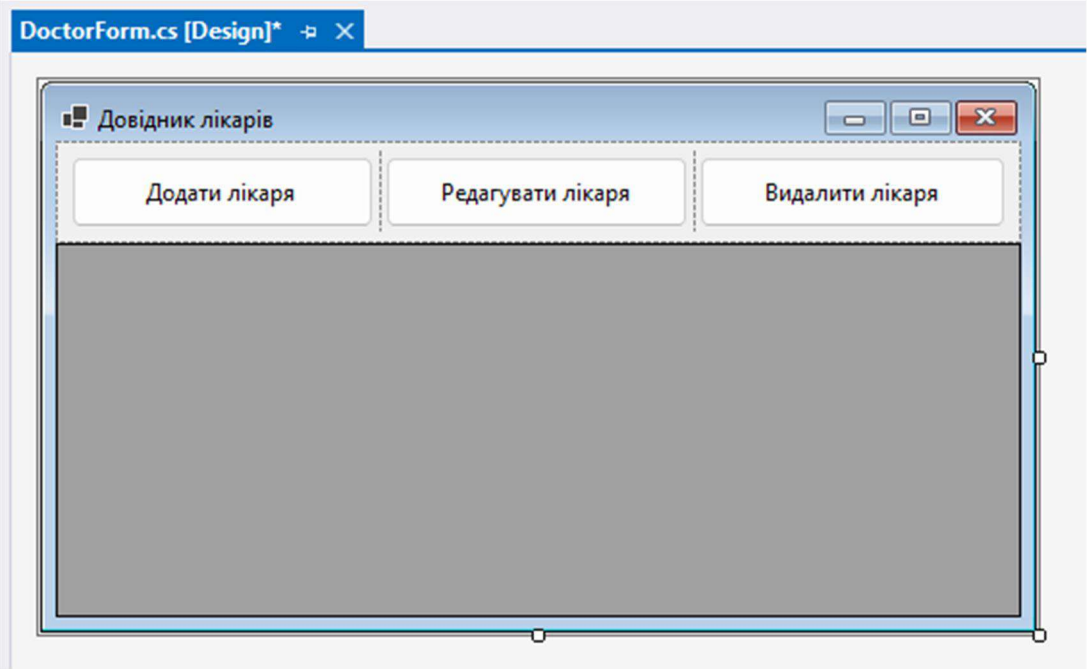


Рисунок 2.4.9 – Форма DoctorForm

Відображає список лікарів у DataGridView із можливістю додавання, редагування та видалення. Дані завантажуються через DoctorRepository.

9. **DiagnosesForm** — форма для перегляду списку діагнозів.

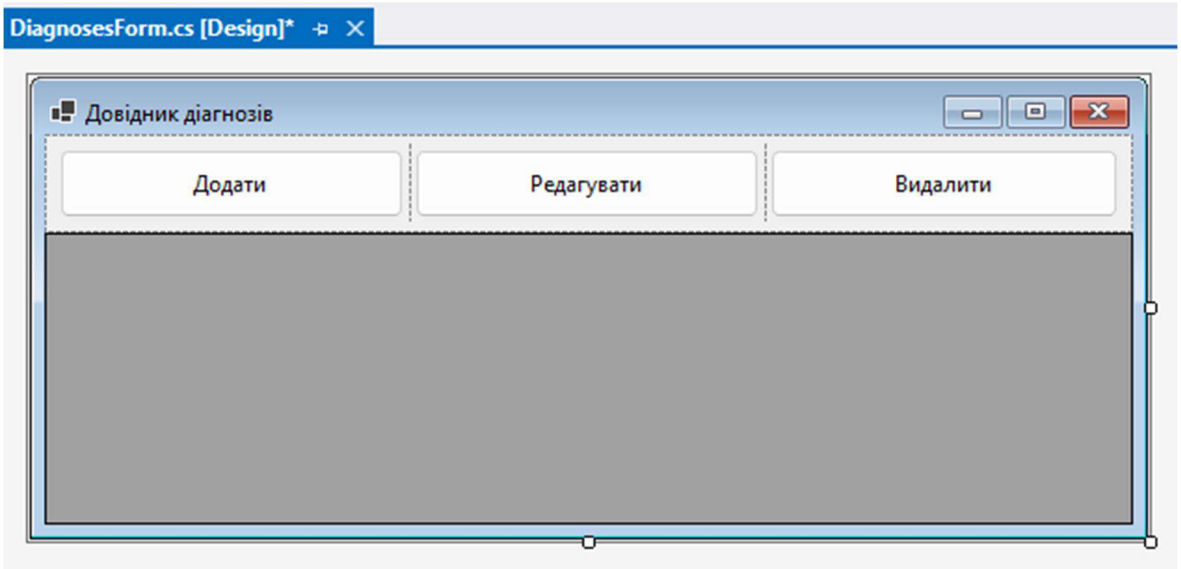


Рисунок 2.4.10 – Форма DiagnosesForm

Аналогічний до DoctorForm, але для діагнозів. Особливості: Використовує DiagnosisRepository.

Висновки до розділу 2

У другому розділі описано процес проектування та розробки прикладної програми для управління медичною карткою пацієнта, реалізованої на основі Windows Forms, .NET 8 і SQLite. Проведений аналіз дозволив сформувати чіткий підхід до створення програми, що відповідає потребам невеликих медичних практик.

Вибір алгоритмів для обробки даних, оновлення інтерфейсу, кастомізації вкладок і валідації даних забезпечує баланс між простотою реалізації та ефективністю. Алгоритми CRUD для SQLite мають константну або лінійну складність, що є достатнім для невеликих обсягів даних. Використання LINQ для фільтрації та BindingSource для оновлення DataGridView оптимізує взаємодію з інтерфейсом, а подія DrawItem для TabControl дозволяє реалізувати різнокольорові підписи вкладок із мінімальними витратами ресурсів.

База даних, спроектована на основі SQLite, включає шість нормалізованих таблиць (Patients, Doctors, Diagnoses, Visits, MedicalRecords, Prescriptions) із зовнішніми ключами для забезпечення цілісності.

Класи програми поділено на моделі даних, репозиторії та форми, що відповідає принципам модульності та поділу обов’язків. Моделі відображають структуру бази даних, репозиторії інкапсулюють логіку CRUD, а форми забезпечують зручний інтерфейс.

Програмна реалізація включає структурований проєкт із папками Models, Repositories і Forms, що полегшує підтримку коду. Форми мають уніфікований стиль і функціонал, а вкладкова структура MainForm із різнокольоровими підписами забезпечує інтуїтивну навігацію.

Спроектована програма є портативним, зручним і ефективним рішенням для автоматизації управління медичними картками в малих медичних практиках, відповідаючи вимогам простоти, функціональності та низького споживання ресурсів.

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАННЯ ПРОГРАМОЮ

3.1. Мінімальні вимоги до системи

Для коректної роботи прикладної програми для управління медичною карткою пацієнта необхідно забезпечити відповідність системи мінімальним технічним вимогам. Програма розроблена з використанням Windows Forms і .NET 8, що робить її легкою та портативною, а база даних SQLite мінімізує залежність від додаткового програмного забезпечення.

Операційна система має бути Windows 10 версії 1803 або новішої, оскільки .NET 8 підтримує ці платформи, забезпечуючи стабільність і сумісність. Процесор із частотою від 1 ГГц достатній для виконання всіх операцій програми, включаючи обробку даних і відображення інтерфейсу. Оперативна пам’ять обсягом 2 ГБ дозволяє комфортно працювати з програмою, хоча для оптимальної продуктивності рекомендується 4 ГБ. Для зберігання програми та бази даних потрібно щонайменше 100 МБ вільного місця на диску, враховуючи, що файл SQLite (patients.db) зростає залежно від обсягу даних.

Монітор із роздільною здатністю 1024x768 забезпечує коректне відображення інтерфейсу, включаючи вкладкову структуру та таблиці. Програма не потребує підключення до Інтернету, оскільки працює локально, а SQLite не вимагає серверної інфраструктури. Для встановлення програми необхідно мати .NET 8 Runtime, який автоматично інсталюється разом із додатком, якщо відсутній.

Додаткове програмне забезпечення, таке як СУБД чи вебсервер, не потрібне, що спрощує розгортання. Програма сумісна з будь-якими стандартними периферійними пристроями, такими як клавіатура та миша, і не вимагає спеціалізованого обладнання. Для резервного копіювання бази даних рекомендується мати зовнішній носій або хмарне сховище, хоча це не є обов’язковим.

Мінімальні вимоги до системи дозволяють використовувати програму на більшості сучасних комп'ютерів у малих медичних практиках, забезпечуючи простоту встановлення та експлуатації.

3.2. Інтерфейс користувача

Інтерфейс прикладної програми для управління медичною карткою пацієнта розроблено з використанням Windows Forms, що забезпечує простоту, інтуїтивність і зручність для користувачів, зокрема медичного персоналу невеликих практик. Програма має вкладкову структуру, уніфікований стиль і чітке компонування елементів, що полегшує навігацію та роботу з даними. У цьому розділі описано основні компоненти інтерфейсу, їх функціонал і особливості, включаючи кастомізацію вкладок із різнокольоровими підписами.

Інтерфейс програми побудовано навколо головної форми (MainForm), яка містить вкладкову структуру, реалізовану через компонент TabControl (tabControlMain). Кожна вкладка відповідає окремій категорії даних: Пацієнти, Діагнози, Візити, Медичні записи, Рецепти та Довідники. Усі елементи інтерфейсу, включаючи форми редагування, мають уніфікований стиль: світло-сірий фон (Color.FromArgb(245, 245, 245)), шрифт Segoe UI розміром 9 pt і плоскі кнопки з синьо-сірим фоном (Color.FromArgb(100, 150, 200)). Компоненти розміщено за допомогою TableLayoutPanel, що забезпечує акуратне вирівнювання та адаптивність.

Головна форма є центральним елементом інтерфейсу, відкриваючись під час запуску програми. Вона містить вкладковий компонент із шістьма вкладками, кожна з яких має унікальний колір підпису для полегшення ідентифікації:

- **Пацієнти** (темно-червоний підпис) — відображає список пацієнтів.
- **Діагнози** (темно-синій підпис) — показує діагнози, пов'язані з пацієнтами.
- **Візити** (темно-зелений підпис) — містить інформацію про візити.

- **Медичні записи** (пурпурний підпис) — відображає медичні записи.
- **Рецепти** (темно-помаранчевий підпис) — показує виписані рецепти.
- **Довідники** (темно-бірюзовий підпис) — включає списки лікарів і діагнозів.

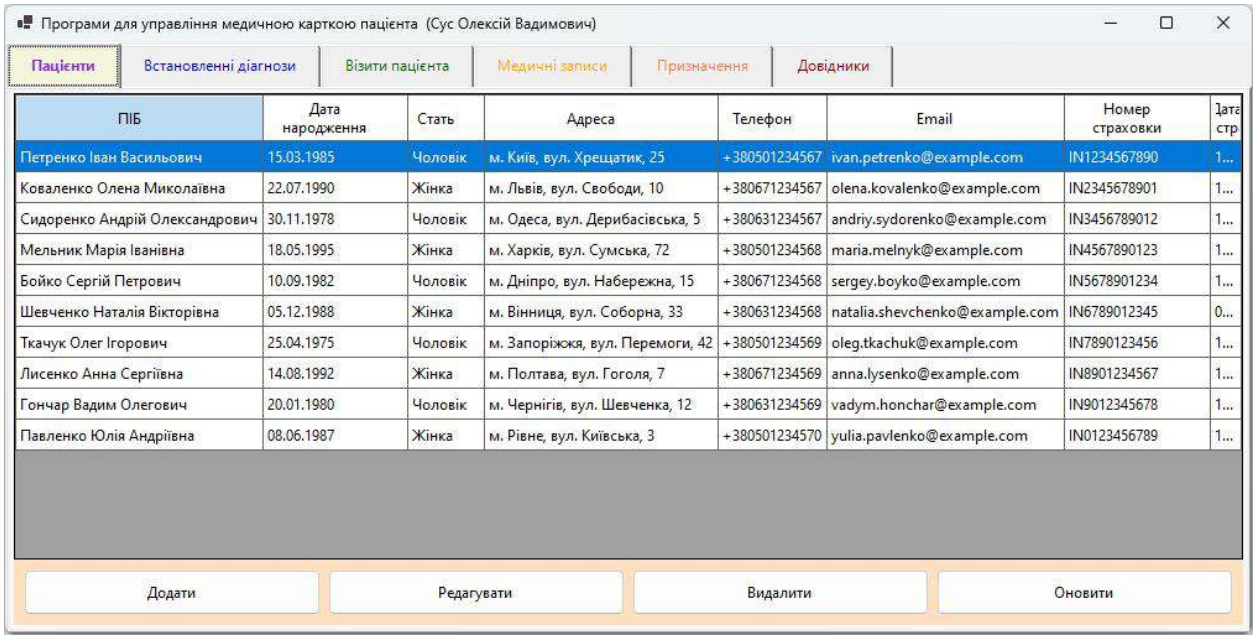


Рисунок 3.2.1 – Інтерфейс програми

Кольорові підписи вкладок реалізовано через подію DrawItem компонента TabControl, що дозволяє малювати текст вкладок із заданими кольорами зі словника tabColors. Активна вкладка виділяється білим фоном (Color.WhiteSmoke) і жирним шрифтом, а неактивні мають стандартний системний фон (SystemColors.Control). Розмір вкладок фіксований (150x30 пікселів), що забезпечує компактність і однаковий вигляд.

Кожна вкладка містить TableLayoutPanel, який поділено на дві основні зони:

- **DataGridView** — таблиця для відображення списків даних (наприклад, список пацієнтів із колонками ПІБ, дата народження, телефон тощо). Таблиця підтримує виділення рядків і сортування за колонками.
- **Панель кнопок** — розташована праворуч або знизу, містить кнопки для операцій CRUD: Додати (btnAdd), Редагувати (btnEdit), Видалити

(btnDelete), Оновити (btnRefresh). Кнопки мають плоский стиль і уніфікований дизайн.

Для додавання або редагування даних використовуються модальні форми, які відкриваються з MainForm після натискання кнопок Додати або Редагувати. Кожна форма відповідає окремій сутності та має схожий дизайн. Основні форми редагування:

1. **PatientEditForm** — форма для роботи з даними пацієнта.

Містить текстові поля для введення ПІБ, телефону, email, адреси, номера страховки, а також DateTimePicker для дати народження та ComboBox для статі.

Кнопки Зберегти (btnSave) і Скасувати (btnCancel) розташовані внизу.

Валідація перевіряє заповненість ПІБ і коректність формату email/телефону.

2. **DoctorEditForm** — форма для даних лікаря.

Включає текстові поля для ПІБ, спеціалізації, номера ліцензії, телефону та email.

Має аналогічну структуру з валідацією обов'язкових полів.

3. **DiagnosisEditForm** — форма для діагнозів.

Містить поля для коду МКХ, назви та опису діагнозу.

Перевіряє унікальність коду МКХ.

4. **VisitEditForm** — форма для візитів.

Включає ComboBox для вибору пацієнта та лікаря, DateTimePicker для дати візиту, текстове поле для скарг і ComboBox для типу візиту.

Валідація забезпечує вибір пацієнта та лікаря.

5. **MedicalRecordEditForm** — форма для медичних записів.

Містить ComboBox для вибору візиту та діагнозу, багатострокові текстові поля для результатів аналізів, опису лікування та нотаток, а також DateTimePicker для дати запису.

Текстові поля мають фіксовану висоту (100 пікселів) для зручності введення.

6. **PrescriptionEditForm** — форма для рецептів.

Включає ComboBox для вибору візиту та лікаря, текстові поля для медикаменту, дозування та інструкцій, DateTimePicker для дат видачі та закінчення, ComboBox для статусу.

Перевіряє коректність дат (кінцева дата не раніше початкової).

Усі форми редагування використовують TableLayoutPanel для компонування елементів із відступами (Padding = 5), що забезпечує акуратний вигляд. При редагуванні форма заповнюється даними з відповідного об’єкта (наприклад, Patient), а після збереження повертає DialogResult.OK для оновлення даних у MainForm.

Для управління довідниками використовуються окремі форми:

DoctorForm — відображає список лікарів у DataGridView із кнопками для додавання, редагування та видалення. Відкривається з вкладки Довідники.

DiagnosesForm — аналогічна форма для управління списком діагнозів.

Ці форми мають структуру, подібну до вкладок MainForm, із DataGridView і панеллю кнопок, але відкриваються як окремі вікна.

Користувач взаємодіє з програмою через вибір вкладок для перегляду даних, натискання кнопок для виконання операцій і заповнення форм редагування. Наприклад, щоб додати нового пацієнта, користувач:

1. Переходить на вкладку Пацієнти (червоний підпис).
2. Натискає кнопку Додати, що відкриває PatientEditForm.
3. Заповнює поля, натискає Зберегти.
4. Повертається до MainForm, де DataGridView оновлюється автоматично.

Аналогічний процес застосовується до інших сутностей. Для перегляду довідників користувач відкриває DoctorForm або DiagnosesForm із вкладки Довідники.

Інтерфейс програми є інтуїтивним і функціональним, забезпечуючи швидкий доступ до управління медичними даними. Вкладкова структура MainForm із різнокольоровими підписами полегшує навігацію, а уніфіковані форми редагування спрощують введення та редагування даних. Використання

DataGridView, TableLayoutPanel і плоских кнопок створює акуратний і сучасний вигляд, а валідація та підказки підвищують зручність для користувачів. Інтерфейс відповідає потребам невеликих медичних практик, забезпечуючи ефективну роботу з мінімальними вимогами до підготовки персоналу.

3.3. Інструкція для роботи з програмою

Прикладна програма для управління медичною карткою пацієнта розроблена для автоматизації обліку даних у невеликих медичних практиках. Вона дозволяє керувати інформацією про пацієнтів, лікарів, діагнози, візити, медичні записи та рецепти через зручний інтерфейс із вкладковою структурою. Ця інструкція описує основні етапи роботи з програмою, від запуску до виконання ключових операцій, забезпечуючи просте освоєння для користувачів із мінімальним досвідом.

Для початку роботи необхідно встановити програму на комп’ютер із Windows 10 (версія 1803 або новіша). Програма постачається у вигляді виконуваного файлу (.exe) разом із бібліотекою System.Data.SQLite. Якщо .NET 8 Runtime не встановлено, система запропонує його інсталювати під час першого запуску. Файл бази даних SQLite (patients.db) створюється автоматично в папці програми при першому запуску. Для встановлення достатньо розпакувати архів із програмою в будь-яку папку на диску (рекомендується не менше 100 МБ вільного місця) і запустити виконуваний файл. Інтернет-з’єднання не потрібне, оскільки програма працює локально.

Після запуску відкривається головна форма (MainForm), яка містить вкладковий інтерфейс із шістьма вкладками: Пацієнти, Діагнози, Візити, Медичні записи, Рецепти та Довідники. Кожна вкладка позначена унікальним кольором підпису (наприклад, червоний для Пацієнтів, синій для Діагнозів), що полегшує навігацію.

Кожна вкладка відображає список даних у таблиці (DataGridView) і містить панель із кнопками для управління: Додати, Редагувати, Видалити,

Оновити. Щоб переглянути потрібну категорію даних, клацніть на відповідну вкладку. Наприклад, вкладка Пацієнти показує список із колонками ПІБ, дата народження, стать, телефон тощо. Для оновлення списку після змін натисніть кнопку Оновити.

Додавання даних

Перейдіть на потрібну вкладку, наприклад, Пацієнти.

Натисніть кнопку Додати, що відкриє форму редагування (наприклад, PatientEditForm).

Заповніть усі обов’язкові поля, позначені зірочкою (наприклад, ПІБ). Для полів, таких як телефон чи email, дотримуйтесь формату, указаного в підказках (ToolTip).

У формах із вибором, таких як VisitEditForm, виберіть значення зі списків (ComboBox), наприклад, пацієнта чи лікаря.

Натисніть Зберегти. Якщо дані введено коректно, форма закриється, а таблиця на вкладці оновиться. У разі помилки (наприклад, порожнє ПІБ) з’явиться повідомлення з поясненням.

Редагування даних

Для редагування існуючого запису:

Виділіть рядок у таблиці на потрібній вкладці, наприклад, виберіть пацієнта у вкладці Пацієнти.

Натисніть кнопку Редагувати, що відкриє форму редагування з уже заповненими даними.

Внесіть зміни в потрібні поля, наприклад, оновіть номер телефону чи адресу.

Натисніть Зберегти, щоб зберегти зміни, або Скасувати, щоб закрити форму без збереження. Після збереження таблиця оновиться автоматично.

Видалення даних

Щоб видалити запис:

Виділіть рядок у таблиці, наприклад, рецепт у вкладці Рецепти.

Натисніть кнопку Видалити. З’явиться діалогове вікно для підтвердження дії.

Підтвердіть видалення, натиснувши Так. Запис буде видалено з бази даних, а таблиця оновиться. Якщо запис пов’язаний з іншими даними (наприклад, візит із медичним записом), програма видасть повідомлення про неможливість видалення через зовнішні ключі.

Робота з довідниками

Вкладка Довідники надає доступ до списків лікарів і діагнозів. Щоб переглянути або відредагувати ці дані:

Перейдіть на вкладку Довідники.

Натисніть кнопку для відкриття форми DoctorForm (список лікарів) або DiagnosesForm (список діагнозів).

У цих формах можна додавати, редагувати або видаляти записи аналогічно до інших вкладок, використовуючи кнопки Додати, Редагувати, Видалити.

Приклади типових операцій

1. **Додавання нового пацієнта**:

   На вкладці Пацієнти натисніть Додати.

   У PatientEditForm введіть ПІБ (наприклад, «Іванов Іван Іванович»), виберіть дату народження через DateTimePicker, укажіть стать, телефон, email, адресу, номер страховки.

   Натисніть Зберегти. Новий пацієнт з’явиться в таблиці.

2. **Створення візиту**:

   На вкладці Візити натисніть Додати.

   У VisitEditForm виберіть пацієнта та лікаря зі списків ComboBox, укажіть дату візиту, тип (наприклад, «Первинний») і скарги (наприклад, «Головний біль»).

   Натисніть Зберегти, щоб додати візит.

3. **Виписування рецепта**:

   На вкладці Рецепти натисніть Додати.

У PrescriptionEditForm виберіть візит і лікаря, укажіть медикамент (наприклад, «Парацетамол»), дозування (наприклад, «500 мг»), інструкції, дати видачі та закінчення, статус («Активний»).

Натисніть Зберегти для збереження рецепта.

Для захисту даних рекомендується періодично створювати резервні копії файлу бази даних (patients.db). Щоб зробити копію, скопіюйте файл із папки програми на зовнішній носій або в інше безпечне місце. У разі втрати даних замініть пошкоджений файл резервною копією. Програма автоматично використовує patients.db у своїй папці.

Якщо програма не запускається, перевірте наявність .NET 8 Runtime і правильність розташування файлу patients.db. У разі помилок під час збереження даних (наприклад, «Некоректний формат email») зверніть увагу на повідомлення та виправте введені дані. Якщо видалення запису неможливе через зв’язки, спочатку видаліть залежні записи (наприклад, медичні записи перед видаленням візиту).

Програма забезпечує простий і зручний спосіб управління медичними даними через вкладковий інтерфейс із різнокольоровими підписами. Користувач може легко додавати, редагувати, видаляти та переглядати дані, використовуючи таблиці та форми редагування. Валідація та підказки допомагають уникнути помилок, а локальна база даних SQLite усуває потребу в додаткових ресурсах. Ця інструкція дозволяє швидко освоїти програму, роблячи її доступною для медичного персоналу з базовими навичками роботи на комп’ютері.

Висновки до розділу 3

У третьому розділі представлено детальне керівництво для користувачів прикладної програми для управління медичною карткою пацієнта, розробленої на базі Windows Forms, .NET 8 і SQLite. Описано мінімальні системні вимоги, інтерфейс користувача та покрокову інструкцію з використання програми, що забезпечує її доступність для медичного персоналу невеликих практик.

Мінімальні вимоги до системи є низькими, що дозволяє запускати програму на більшості сучасних комп'ютерів із Windows 10, 2 ГБ оперативної пам'яті та 100 МБ дискового простору. Відсутність потреби в серверній інфраструктурі та Інтернет-з'єднанні робить програму портативною та простою у розгортанні.

Інтерфейс користувача побудовано з акцентом на інтуїтивність і зручність. Головна форма з вкладковою структурою та різнокольоровими підписами (червоний для Пацієнтів, синій для Діагнозів тощо) полегшує навігацію. Уніфікований стиль форм, використання TableLayoutPanel, DataGridView і плоских кнопок забезпечує акуратний вигляд, а валідація даних і підказки підвищують зручність роботи. Форми редагування дозволяють легко вводити та змінювати дані, підтримуючи всі необхідні операції з медичною інформацією.

Інструкція з використання програми детально описує процеси встановлення, запуску, роботи з вкладками, додавання, редагування, видалення даних і управління довідниками. Покрокові приклади для типових операцій, таких як додавання пацієнта чи виписування рецепта, спрощують освоєння програми. Рекомендації щодо резервного копіювання та усунення проблем підвищують надійність використання.

Програма є зручним інструментом для автоматизації управління медичними картками, що відповідає потребам малих медичних практик. Інтерфейс і документація забезпечують швидке навчання користувачів, а низькі системні вимоги гарантують широку застосовність.

ВИСНОВКИ

Дипломна робота присвячена розробці прикладної програми для управління медичною карткою пацієнта, яка є ефективним інструментом для автоматизації обліку даних у невеликих медичних практиках. Програма створена з використанням технології Windows Forms на платформі .NET 8 із базою даних SQLite, що забезпечує портативність, простоту розгортання та мінімальні вимоги до апаратного забезпечення.

У процесі виконання роботи проведено аналіз задачі, оглянуто аналогічні системи (Epic Systems, OpenEMR, Practice Fusion) і обґрунтовано вибір технологій. SQLite обрано за її легкість і відсутність потреби в серверній інфраструктурі, Windows Forms — за швидкість розробки інтерфейсу, а .NET 8 — за продуктивність і сучасні можливості. Розроблена програма займає нішу простих і доступних EMR-систем, конкуруючи з OpenEMR за вартістю та портативністю, але поступаючись комерційним системам за інтероперабельністю.

Проектування та розробка програми включали створення нормалізованої бази даних із шістьма таблицями (Patients, Doctors, Diagnoses, Visits, MedicalRecords, Prescriptions), реалізацію модульної архітектури з класами моделей, репозиторіїв і форм, а також оптимізацію алгоритмів для операцій CRUD і оновлення інтерфейсу. Вкладкова структура з різнокольоровими підписами, уніфікований стиль форм і валідація даних забезпечують зручність і інтуїтивність інтерфейсу.

Розроблена програма успішно виконує поставлені завдання, забезпечуючи управління медичними даними з мінімальними витратами. У перспективі можливе розширення функціоналу, наприклад, додавання звітів, інтеграції зі стандартами HL7 або підтримки кількох користувачів, що підвищить її застосовність у більших закладах. Робота підтвердила ефективність обраного технологічного стека та потенціал програми як практичного рішення для автоматизації медичних процесів.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Netguru. 16 типів програмного забезпечення для охорони здоров’я у 2025 році [Електронний ресурс]. Режим доступу: https://www.netguru.com/blog/healthcare-software-types, вільний. Дата доступу: 20.04.2025.
2. DDI Development. Як створити програмне забезпечення для управління пацієнтами з нуля [Електронний ресурс]. Режим доступу: https://ddi-dev.com/blog/case/how-weve-created-patient-management-software-from-scratch/, вільний. Дата доступу: 20.04.2025.
3. LeadSquared. Програмне забезпечення для управління пацієнтами (PMS): функції, переваги та інструменти [Електронний ресурс]. Режим доступу: https://www.leadsquared.com/industries/healthcare/patient-management-system-features-benefits/, вільний. Дата доступу: 20.04.2025.
4. ClickUp. Топ-10 програмного забезпечення для управління адміністративними завданнями в охороні здоров’я [Електронний ресурс]. Режим доступу: https://clickup.com/blog/patient-management-software/, вільний. Дата доступу: 20.04.2025.
5. Wikipedia contributors. Програмне забезпечення для управління пацієнтами [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Patient_management_software, вільний. Дата доступу: 20.04.2025.
6. Appvales. Посібник із розробки системи управління пацієнтами [Електронний ресурс]. Режим доступу: https://www.appvales.com/blog/patient-management-system-development-guide, вільний. Дата доступу: 20.04.2025.
7. GeeksforGeeks. Проєкт системи управління лікарнею в розробці програмного забезпечення [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/hospital-management-system-project-in-software-development/, вільний. Дата доступу: 20.04.2025.

8. Bigscal. Оптимізація догляду: розробка програмного забезпечення для управління пацієнтами [Електронний ресурс]. Режим доступу: https://www.bigscal.com/healthcare-software-development/patient-management-software-development/, вільний. Дата доступу: 20.04.2025.
9. Salesforce India. Програмне забезпечення для управління пацієнтами: посібник [Електронний ресурс]. Режим доступу: https://www.salesforce.com/in/healthcare-life-sciences/healthcare-software/patient-management-software/, вільний. Дата доступу: 20.04.2025.
10. Study Research Papers. Система управління доглядом за пацієнтами [Електронний ресурс]. Режим доступу: https://studyresearchpapers.com/patient-care-management-system/, вільний. Дата доступу: 20.04.2025.
11. The Hospital Management System [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/367460409_The_Hospital_Management_System, вільний. Дата доступу: 20.04.2025.
12. DESIGN AND IMPLEMENTATION OF PATIENT MANAGEMENT SYSTEM [Електронний ресурс]. Режим доступу: https://www.academia.edu/27145315/DESIGN_AND_IMPLEMENTATION_OF_PATIENT_MANAGEMENT_SYSTEM, вільний. Дата доступу: 20.04.2025.
13. Hospital Management System [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/325398700_Hospital_Management_System, вільний. Дата доступу: 20.04.2025.
14. A hospital resource and patient management system based on real-time data capture and intelligent decision making [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/239763364_A_hospital_resource_and_patient_management_system_based_on_real-time_data_capture_and_intelligent_decision_making, вільний. Дата доступу: 20.04.2025.

15. Health Services Management Research [Електронний ресурс]. Режим доступу: https://journals.sagepub.com/home/hsm, вільний. Дата доступу: 20.04.2025.
16. OTAKOYI. Програмне забезпечення для управління пацієнтами (PMS): вичерпний посібник із функцій і переваг [Електронний ресурс]. Режим доступу: https://otakoyi.software/blog/patient-management-software-pms-a-comprehensive-guide-to-features-and-benefits, вільний. Дата доступу: 20.04.2025.
17. MODERN HOSPITAL MANAGEMENT SYSTEM [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/375497520_MODERN_HOSPITAL_MANAGEMENT_SYSTEM, вільний. Дата доступу: 20.04.2025.
18. Yang, Y., Kim, M. Analysis of user requirement on U-Healthcare system // International Journal of Business Tourism and Applied Sciences. – 2013. – Vol. 1, No. 2. – P. 1–10.
19. Ngafeeson, M. Healthcare Information Systems: Opportunities and Challenges // Book Sections/Chapters. – 2014. – Paper 14.
20. Smith, J., Johnson, K. Electronic Health Records and Patient Safety: A Qualitative Study // Journal of Healthcare Informatics. – 2020. – Vol. 15, No. 3. – P. 45–56.
21. Lee, M., Park, S. Development of a Patient Management System Using Mobile Technology // IEEE Transactions on Biomedical Engineering. – 2019. – Vol. 66, No. 10. – P. 2800–2808.
22. Brown, A., White, B. Information Technology in Healthcare: Patient Management Systems // Health Informatics Journal. – 2018. – Vol. 24, No. 2. – P. 150–162.
23. Tanenbaum, A. S., Bos, H. Modern Operating Systems. 4th ed. Pearson, 2015. – 1136 p.

ДОДАТКИ

ДОДАТОК А.

Програмний код модулів

```
-- Початок транзакції
BEGIN TRANSACTION;

-- Таблиця Пацієнти
CREATE TABLE IF NOT EXISTS Patients (
    patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    birth_date DATE NOT NULL,
    gender TEXT CHECK(gender IN ('Чоловік', 'Жінка', 'Інше')),
    address TEXT,
    phone TEXT,
    email TEXT,
    insurance_number TEXT UNIQUE,
    registration_date DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Таблиця Лікарі
CREATE TABLE IF NOT EXISTS Doctors (
    doctor_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    specialization TEXT NOT NULL,
    license_number TEXT UNIQUE NOT NULL,
    phone TEXT,
    email TEXT,
    hire_date DATE
);

-- Таблиця Діагнози (довідник)
CREATE TABLE IF NOT EXISTS Diagnoses (
    diagnosis_id INTEGER PRIMARY KEY AUTOINCREMENT,
    icd_code TEXT,  -- Код за МКХ (ICD)
    name TEXT NOT NULL,
```