НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри

_____Комп'ютері науки _____
(назва кафедри)

_____ ____Голуб Б.Л_____
(підпис) (ПІБ)

"___" _____ 25 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Розробка інформаційної системи для обліку індивідуальних медичних показників військових»

Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
_____ Д.е.н., професор_____ _____ ____Руденський Р.А_____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____к.е., доцент_____ _____ ___Назаренко В. А_____
_____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ ___Ведмеденко Дарина Дмитрівна_____
(підпис) (ПІБ студента)

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
“____” _____20_____р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Ведмеденко Дарина Дмитрівна
(прізвище, ім’я, по батькові)

Спеціальність 122 – «Комп’ютерні науки»

Тема бакалаврської кваліфікаційної роботи Розробка інформаційної системи для обліку індивідуальних медичних показників військових

затверджена наказом ректора НУБіП України від “16” 12 2024 р. № 2246«С»
Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
__Створення інформаційної системи для обліку індивідуальних медичних показників військових

Перелік питань, які потрібно розробити:
___Системний аналіз предметної області. Інформаційне забезпечення. Прикладне програмне забезпечення. Рекомендації щодо впровадження та експлуатації системи.

Дата видачі завдання “_____”_____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ _____
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ __Ведмеденко Д. Д._____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП

Актуальність завдання розробки інформаційної системи для обліку медичних показників військових зумовлена необхідністю забезпечення якісного медичного супроводу військовослужбовців, що є критично важливим для підтримання їх боєздатності та здоров’я. Сучасні реалії, зокрема зростання вимог до оперативного доступу до медичних даних, аналізу динаміки показників та створення звітів, підкреслюють потребу в автоматизованих системах, які здатні централізовано обробляти, зберігати та аналізувати інформацію. Відсутність таких систем або недостатня їх функціональність може ускладнити своєчасне прийняття рішень лікарями, що може мати негативний вплив на якість медичного обслуговування. Розробка такої системи не лише дозволяє оптимізувати процеси медичного обліку, але й гарантує захист даних, зручність використання, а також порівняльний аналіз стану здоров'я військовослужбовців.

Метою розробки програмного додатку є створення інформаційної системи, яка забезпечує ефективне ведення медичних карток військовослужбовців, внесення та аналіз результатів обстежень, генерацію звітів у форматі PDF, а також управління користувачами та медичними установами. Додаток покликаний підвищити якість медичного обслуговування шляхом автоматизації рутинних процесів, забезпечення швидкого доступу до історії обстежень та візуалізації динаміки показників, що є особливо актуальним для військової медицини, де швидкість і точність обробки даних відіграють ключову роль.

Проект реалізовано з використанням сучасних методів та технологій програмування. Основою застосунку є платформа WPF на базі .NET Framework, яка дозволяє розробляти адаптивний та функціональний інтерфейс користувача. Для роботи з даними використовувалися Entity Framework Core та метод Code First, що забезпечує гнучке проектування баз даних та взаємодію з MS SQL Server. Візуалізація даних реалізована за допомогою бібліотеки LiveCharts, а

генерація звітів — через iTextSharp з підтримкою кирилиці. Інтерфейс розроблено з використанням XAML, що забезпечує чітке розділення логіки та представлення даних. Модульність та легкість збереження системи забезпечуються використанням архітектури MVVM. Безпека даних реалізована за допомогою автентифікації користувачів, розподілу ролей та автоматичного виходу з системи у разі неактивності протягом 10 хвилин.

Апробація програмного додатку передусім була проведена шляхом публікації тез на конференції «Технології розробки програмного забезпечення та інформаційних управляючих систем». Водночас, система була протестована в лабораторних умовах, де підтвердила свою працездатність.

Пояснювальна записка складається з 86 сторінок, містить 38 використаних джерел та 2 додатки. Структура записки включає розділи:

– Системний аналіз предметної області, що охоплює постановку завдання, огляд аналогів і моделювання предметної області.

– Інформаційне забезпечення, присвячене логічній моделі даних, вибору СУБД і створенню бази даних.

– Прикладне програмне забезпечення, де розглянуто організаційну структуру, інструментарій і алгоритмізацію модулів.

– Рекомендації щодо впровадження, які включають тестування, вимоги до обладнання та склад інсталяційного пакету.

Така структура дозволяє послідовно розкрити всі напрямки розробки, від аналізу вимог до практичної реалізації та рекомендацій щодо використання системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Розробка інформаційної системи для обліку медичних показників військовослужбовців спрямована на автоматизацію процесів управління медичною інформацією у військових медичних установах. Основною метою системи є забезпечення централізованого зберігання, обробки та аналізу даних про стан здоров’я військовослужбовців, а також надання зручного інструментарію для медичного персоналу та адміністраторів для виконання їх професійних обов’язків. Система має сприяти підвищенню ефективності медичного обслуговування, забезпечувати швидкий доступ до актуальних даних та підтримувати прийняття обґрунтованих клінічних рішень.

Система передбачає роботу з кількома типами даних, які охоплюють ключові показники медичного обліку. До них належать:

– Дані про військовослужбовців: ПІБ, дата народження, номер військового документа, номер медичної книжки, прив’язка до статі.

– Стать: назва статі.

– Дані про медичні установи: назва, адреса.

– Дані про користувачів системи: ПІБ, логін, пароль, роль, прив’язка до медичної установи, ролі користувача та статусу користувача.

– Дані про обстеження: дата обстеження, прив’язка до інформація про користувача, військовослужбовця і медичної установи.

– Параметри обстежень: назва, категорія (наприклад, серцево-судинна система, лабораторні показники), тип даних (числовий або текстовий), одиниці вимірювання.

– Результати обстеження: вимірювання, прив’язка до обстеження та параметру обстеження.

– Параметри обстеження: назва параметра, категорія, тип даних, одиниця вимірювання.

– Запис на прийом: дата, час, опис прийому та прив’язка до військовослужбовця, користувача, медичної установи, статусу на прийом.

– Системний журнал: дія, дата та час запису, прив’язка до користувача.

– Статус користувача: назва статусу користувача.

– Роль користувача: назва ролі користувача.

Система автоматизує широкий спектр операцій, що відповідають потребам двох категорій користувачів — лікарів та адміністраторів:

– Для лікарів: авторизація, перегляд списків військовослужбовців, пошук за номером документа чи медичної книжки, внесення даних нових обстежень, перегляд історії обстежень, записи на прийом та безпосередньо їх скасування, створення графіків динаміки числових показників, генерація звітів у форматі PDF (звіт одного обстеження або порівняння двох обстежень).

– Для адміністраторів: управління користувачами (додавання, блокування, видалення), скасування запису на прийом, якщо лікар не може провести обстеження, перегляд історії дій користувачів, управління параметрами обстежень та медичними установами, створення резервних копій бази даних та її відновлення.

– Загальні операції: авторизація з автоматичним виходом після 10 хвилин бездіяльності, додавання нових військовослужбовців, формування звітів, обробка помилок введення даних.

Результатом роботи системи є звіти у форматі PDF, які включають:

– Звіт одного обстеження: інформація про військовослужбовця, лікаря, медичну установу, дата обстеження, повний перелік параметрів та їх значень.

– Порівняльний звіт двох обстежень: аналогічна інформація з відображенням зміни показників відносно часового проміжку у форматі порівняння для аналізу динаміки стану здоров’я.

За характером використання інформації система відноситься до інформаційно-аналітичних систем, оскільки вона забезпечує як зберігання даних, так і їх аналіз (графіки, порівняння). За масштабом це локальна система, призначена для використання в межах однієї або кількох медичних установ з можливістю розширення. За підтримуваними технологіями система базується на стандартах .NET Framework та SQL Server, що забезпечує надійність і сумісність. За ступенем автоматизації система є частково автоматизованою: більшість операцій (внесення даних, генерація звітів, побудова графіків) виконуються автоматично, але потребують введення первинних даних користувачем.

Система відповідає таким ключовим вимогам:

– Функціональні: підтримка авторизації, управління даними, генерація звітів, візуалізація динаміки показників.

– Нефункціональні: висока безпека (захист даних, автовихід), зручність інтерфейсу (адаптивний дизайн, підтримка української мови), надійність (обробка помилок, резервне копіювання), продуктивність (швидкий доступ до даних).

– Інтерфейс: інтуїтивно зрозумілий, з чіткою навігацією, підтримкою вкладок для одночасної роботи з кількома записами, повідомленнями про помилки та діалоговими вікнами для підтвердження дій.

– Безпека: забезпечення конфіденційності даних, журналювання дій користувачів, захист від несанкціонованого доступу.

Отже, розроблювана система є цілісним рішенням для автоматизації медичного обліку, яке враховує особливості діяльності військових медичних закладів, забезпечуючи зручність, безпеку та аналітичні можливості для користувачів.

1.2 Огляд інформаційних джерел та існуючих рішень

Сучасні інформаційні системи для обліку медичних показників, зокрема у військовій медицині, є важливою складовою забезпечення ефективного управління охороною здоров'я. Аналіз літератури та наукових джерел дозволяє глибше зрозуміти технологічні підходи, стандарти та виклики, пов'язані з розробкою таких систем.

Одним із ключових етапів є впровадження електронних медичних карток (ЕМК), які забезпечують централізоване зберігання даних про стан здоров'я військовослужбовців. У дослідженні Torab-Miandoab et al. [1] проведено систематичний огляд використання ЕМК у військових медичних системах, підкреслюючи їх роль у підвищенні якості медичного обслуговування та оперативності прийняття рішень. Автори зазначають, що ЕМК сприяють інтеграції даних між різними медичними установами, що є критично важливим у військових умовах, де потрібна швидка координація [1].

Важливим елементом є стандартизація даних. HL7 (Health Level Seven International) є провідним стандартом для обміну медичною інформацією, зокрема через FHIR (Fast Healthcare Interoperability Resources), який забезпечує гнучкість і масштабованість сучасних систем [5]. Використання таких стандартів дозволяє створювати сумісні системи, здатні інтегруватися з іншими медичними платформами, що є актуальним для військових систем, де дані можуть надходити з різних джерел [5].

Візуалізація медичних даних також відіграє неабияку роль для підвищення ефективності аналізу. У статті Achanta та Bo [2] підкреслюється, що інструменти візуалізації, такі як графіки та дашборди, допомагають лікарям швидше виявляти тенденції та аномалії в медичних показниках [2]. Це особливо важливо для моніторингу динаміки стану здоров'я військовослужбовців, де своєчасне виявлення змін може мати критичне значення.

Серед прикладів реальних систем варто відзначити MHS GENESIS, яка є електронною медичною системою, розробленою для Збройних сил США. Вона інтегрує дані про обстеження, лабораторні результати та призначення, забезпечуючи доступ до інформації в реальному часі [3, 4]. Проте, як зазначає звіт GAO [8], впровадження таких систем супроводжується викликами, зокрема щодо навчання персоналу та забезпечення стабільності роботи [8].

В Україні електронні медичні картки також набувають поширення. За даними Українського медичного журналу, ЕМК дозволяють зберігати повну історію обстежень, що полегшує доступ до даних у різних медичних закладах [9]. Однак у військовій сфері України подібні системи ще перебувають на етапі розвитку, що підкреслює актуальність створення спеціалізованих рішень, адаптованих до потреб військових медичних служб [9].

Інший приклад — система AHLTA (Armed Forces Health Longitudinal Technology Application), яка використовувалася армією США до впровадження MHS GENESIS. Вона забезпечувала облік медичних даних, але мала обмеження щодо інтеграції та зручності інтерфейсу [7]. Сучасніші системи, такі як EHRM (Electronic Health Record Modernization) для ветеранів США, демонструють перехід до хмарних технологій і кращої взаємодії між медичними установами [6]. Таким чином, інформаційні джерела вказують на важливість інтеграції, стандартизації та візуалізації даних у медичних інформаційних системах. Водночас вони підкреслюють можливі труднощі, пов’язані з масштабуванням, навчанням користувачів і адаптацією до специфічних умов, таких як військова медицина.

На ринку існує низка інформаційних систем для обліку медичних даних, які можуть частково відповідати вимогам до системи для військових медичних показників. Тому буде доцільно розглянути кілька популярних рішень, їх переваги та недоліки.

MHS GENESIS [10] — це електронна медична система, розроблена для Збройних сил США (рис. 1.1). Вона забезпечує повний цикл управління медичними даними: від реєстрації обстежень до генерації звітів.

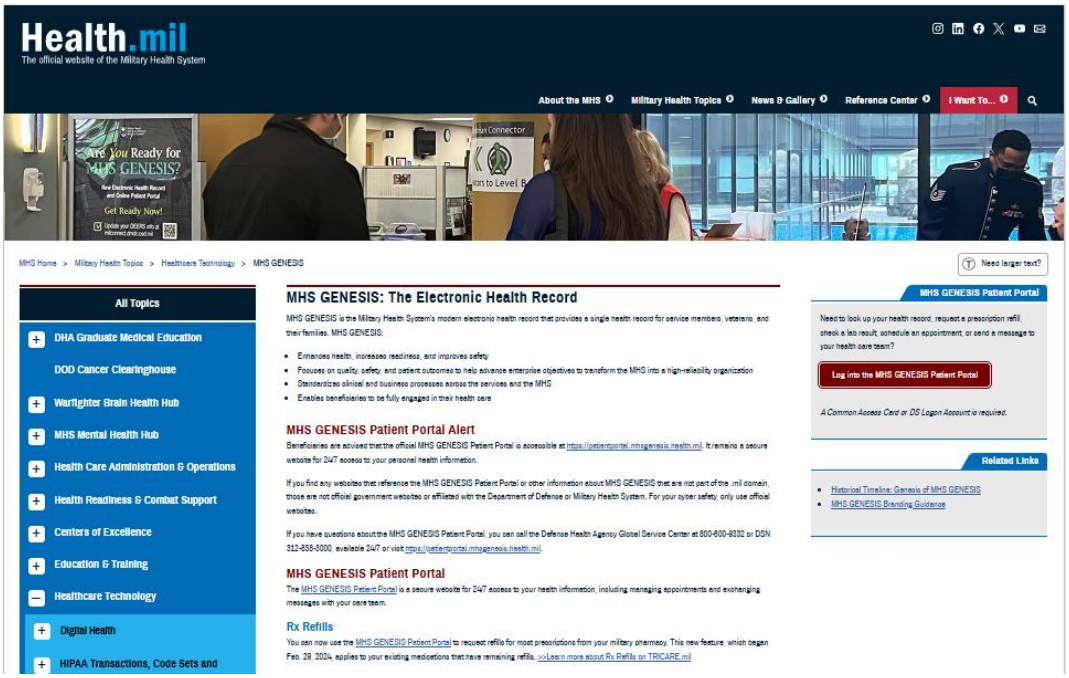

Рис.1.1 MHS GENESIS [10]

Переваги:

– Високий рівень інтеграції з різними медичними установами.
– Підтримка стандартів HL7 і FHIR, що забезпечує сумісність.
– Можливість доступу до даних у реальному часі, що критично для військових операцій.

Недоліки:

– Висока вартість, яка може спричиняти проблеми з доступністю для країн з обмеженим бюджетом.
– Складність навчання персоналу, як зазначено в звіті GAO [8].
– Орієнтація на американську систему охорони здоров’я, що потребує адаптації для інших країн.

OpenEMR [11] — це відкрита система управління медичними записами, яка використовується в різних медичних закладах, зокрема у невеликих клініках (рис. 1.2).

Рис. 1.2 OpenEMR [11]

Переваги:

– Безкоштовність і відкритий код, що дозволяє налаштування під конкретні потреби.

– Підтримка базового функціоналу: облік пацієнтів, обстежень, генерація звітів.

– Можливість локального розгортання, що підвищує безпеку даних.

Недоліки:

– Обмежені можливості інтеграції з військовими системами, які потребують високого рівня безпеки та спеціалізованих даних.

– Відсутність вбудованої підтримки для специфічних військових параметрів, таких як фізична підготовка чи психологічний стан.

– Інтерфейс може бути недостатньо інтуїтивним для користувачів без технічної підготовки.

MedTrak Systems [12] — комерційна система для управління медичними даними, орієнтована на автоматизацію робочих процесів у медичних закладах (рис .1.3).

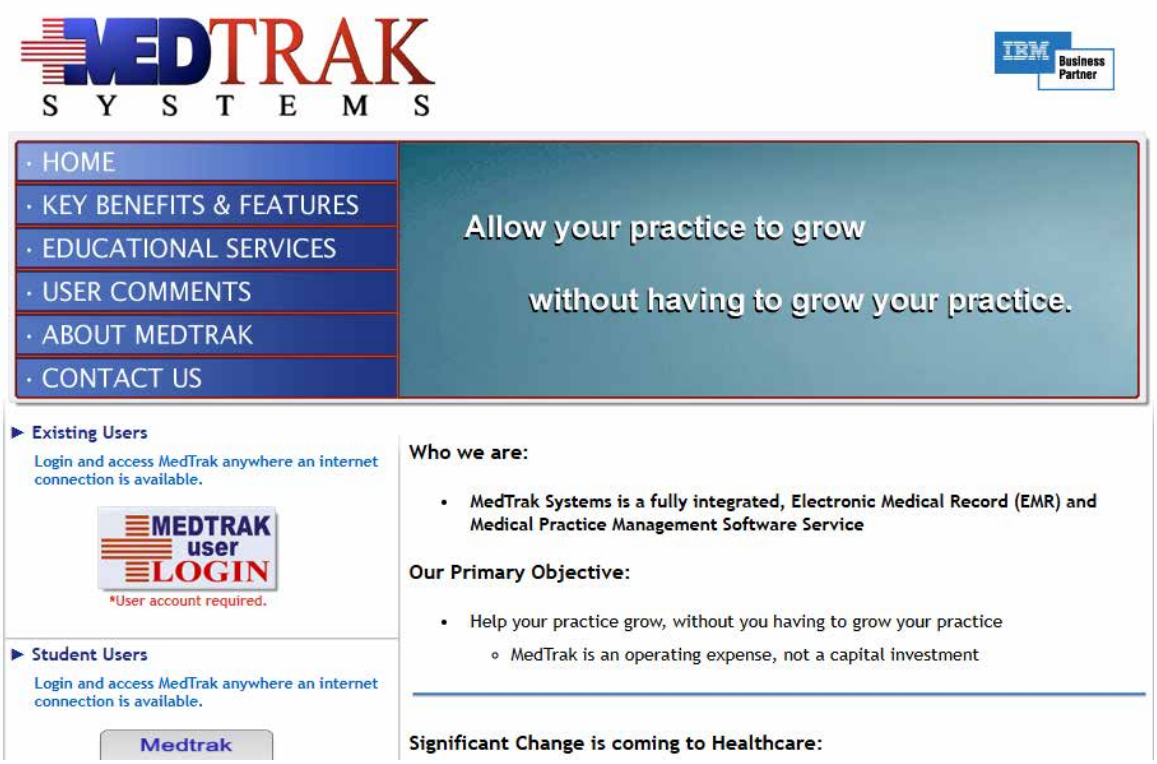

Рис. 1.3 MedTrak Systems [12]

Переваги:

– Гнучкість налаштувань для різних типів медичних установ.

– Підтримка аналітичних звітів і дашбордів для аналізу даних.

– Інтуїтивний інтерфейс, що полегшує роботу лікарів.

Недоліки:

– Висока вартість ліцензії, що може бути проблематичною для масштабного впровадження.

– Відсутність спеціалізації на військову медицину, що потребує додаткових модифікацій.

– Обмежена підтримка кирилиці, що є критичним для україномовного середовища.

TrakCare [13] — система від InterSystems, яка використовується в багатьох країнах для управління медичними даними (рис. 1.4).

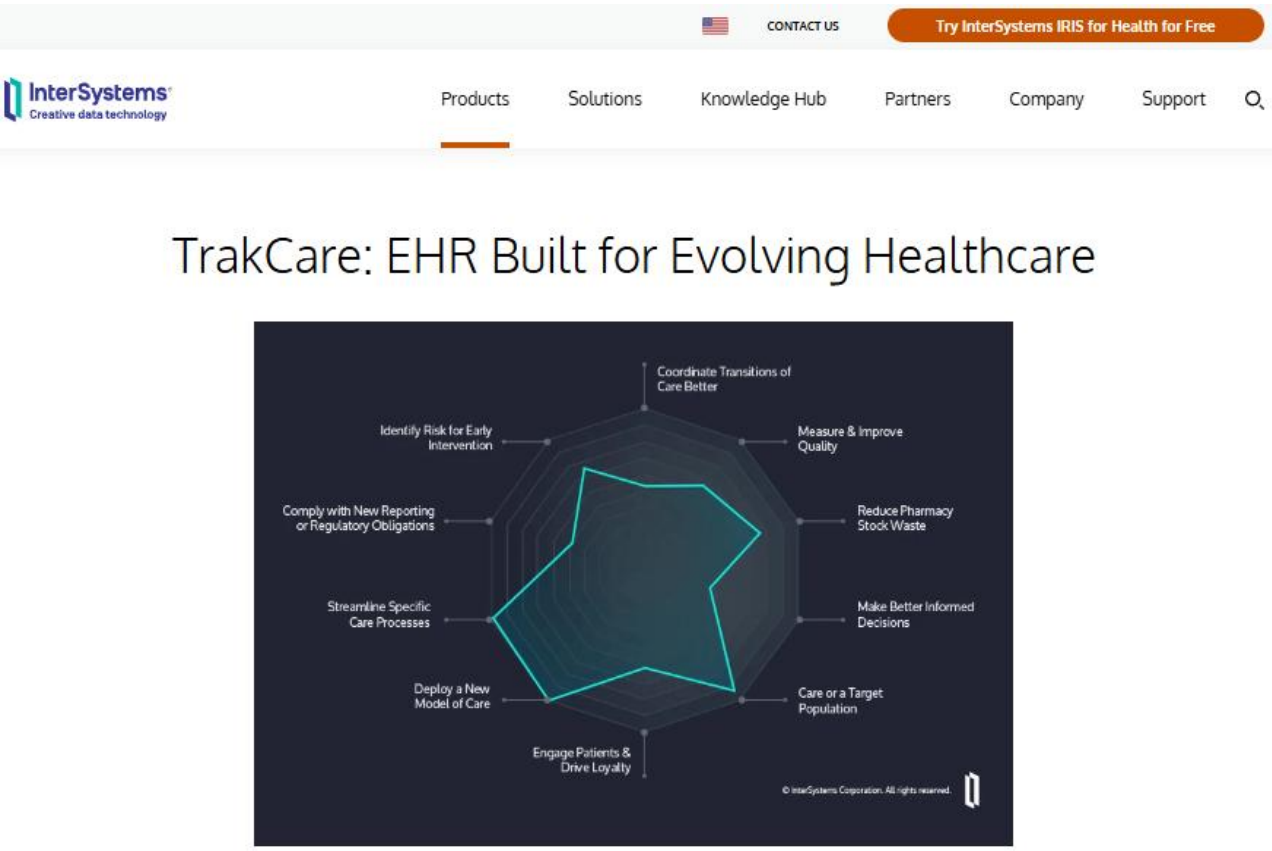

Рис. 1.4 TrakCare [13]

Переваги:

– Висока масштабованість і підтримка великих обсягів даних.

– Можливість інтеграції з іншими системами через стандарти HL7 і FHIR.

– Наявність модулів для аналізу даних і генерації звітів.

Недоліки:

– Складність впровадження, що потребує значних ресурсів і часу.

– Висока ціна, орієнтована на великі медичні установи.

– Необхідність адаптації для специфічних вимог військової медицини, таких як облік фізичної підготовки чи психологічного стану.

Проведений аналіз свідчить, що більшість існуючих рішень мають сильні сторони в загальному управлінні медичними даними, але потребують значної адаптації для використання у військовій медицині, особливо в умовах України. Розроблена система, представлена в реалізованому коді, враховує ці особливості,

пропонуючи локалізоване рішення з підтримкою української мови, спеціалізованими параметрами для військових і доступною вартістю впровадження.

1.3 Моделювання предметної області

Моделювання предметної області є ключовим етапом у розробці інформаційної системи для обліку медичних показників військовослужбовців, оскільки воно дозволяє формалізувати процеси, об’єкти та взаємозв’язки в межах системи, а також визначити вимоги до її функціональності. Для створення комплексного опису предметної області використано набір моделей, зокрема діаграми UML, які відображають як статичну структуру, так і динамічну поведінку системи [14].

Для опису динамічних показників використано діаграму прецедентів (Use Case Diagram) (рис. 1.5), яка відображає взаємодію користувачів із системою [16]. Основні актори описані у табл. 1.1.

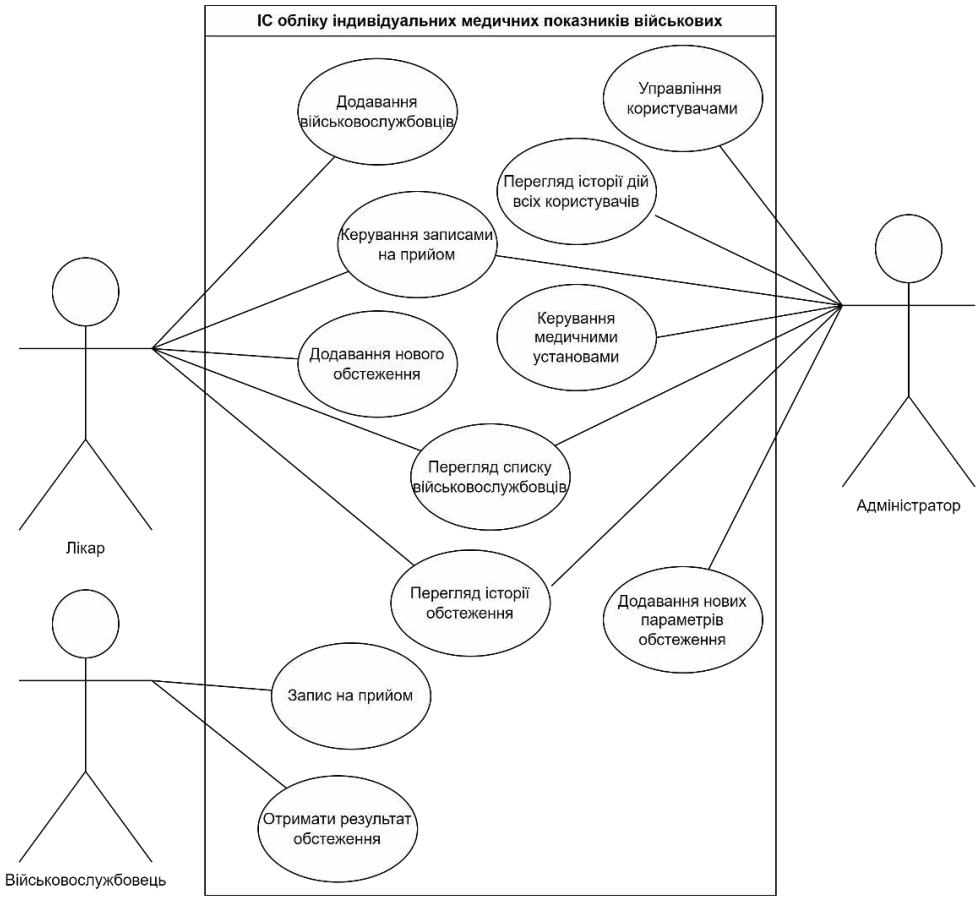

Рис. 1.5 Діаграма прецедентів

Таблиця 1.1

Основні актори системи

№	Назва	Опис
1	Лікар	Переглядає список військовослужбовців, переглядає історію обстежень, додає нові обстеження, керує записами на прийом (призначення, скасування)
2	Адміністратор	Переглядає список військовослужбовців, керує записами на прийом (скасування), керує користувачами (додавання, блокування, видалення), переглядає історію дій користувачів та обстежень, додає і видаляє медичні установи, також додає нові параметри обстежень.
3	Військовослужбовець	Записується на прийом та отримує результат обстеження через лікаря

Основні прецеденти включають:

– Додавання та перегляд військовослужбовців

– Внесення та перегляд даних обстежень.
– Керування записами на прийом.
– Додавання нових параметрів обстежень.
– Управління користувачами.
– Управління медичними установами.
– Перегляд журналу дій користувачів.

Для деталізації процесів використано діаграму активності UML [17], яка наведена у Додатку А. На діаграмі представлено стандартні кроки проведення медичних оглядів військовослужбовців. Діаграма демонструє зв'язок між трьома основними сутностями, такими як військовослужбовець, лікар і адміністратор.

Військовослужбовець ініціює процес, подаючи запит на прийом. Лікар перевіряє інформацію, за потреби, створює нову медичну карту, та вибирає відповідний час для прийому.

Процес продовжується двома різними шляхами залежно від того, чи з'являється військовослужбовець на прийом. Якщо з’являється, лікар проводить огляд, вносить медичні показники та надає результати огляду. Прийом скасовується, якщо військовослужбовець не з'являється.

Діаграма містить технічні елементи, які включають управління обліковим записом лікаря та перевірку прав доступу, а також функції створення та відновлення резервних копій. Діаграма демонструє стандартний медичний процес, який поєднує взаємодію людини з системними операціями.

Для представлення інформаційних потоків використано контекстну діаграму (рис. 1.6), яка відображає взаємодію системи з зовнішніми сутностями [18].

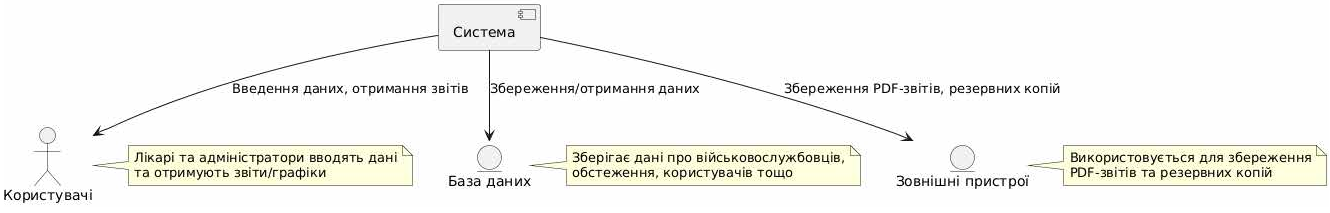

Рис. 1.6. Контекстна діаграма

Зовнішніми сутностями є:

– Користувачі (лікарі та адміністратори), які вводять дані та отримують звіти.

– База даних, яка зберігає інформацію про військовослужбовців, обстеження, користувачів тощо.

– Зовнішні пристрої (для збереження PDF-звітів або резервних копій).

Контекстна діаграма показує, як дані (наприклад, результати обстежень) надходять від користувачів до системи, обробляються, зберігаються в базі даних і повертаються у вигляді звітів чи графіків.

Інформаційна складова системи реалізована через базу даних Microsoft SQL Server, використовуючи її локальну версію LocalDB із застосуванням Entity Framework Core для об’єктно-реляційного відображення [19]. Структура бази даних відповідає діаграмі класів, забезпечуючи зберігання всіх сутностей і зв’язків. Прикладне програмне забезпечення розроблено на WPF з використанням бібліотеки LiveCharts для графіків і iTextSharp для створення PDF-звітів [20]. Архітектура MVVM забезпечує розділення логіки інтерфейсу та даних, що полегшує підтримку та масштабування системи [21].

Розроблені моделі предметної області (діаграми класів, прецедентів, активності, контекстна діаграма), враховують усі вимоги до функціоналу для лікарів і адміністраторів, обробки даних, безпеки та звітності. Імітаційне моделювання підтверджує коректність реалізації, дозволяючи протестувати систему на реалістичних даних. Подальша розробка може включати вдосконалення моделей для аналізу великих обсягів даних або інтеграцію з іншими медичними системами.

Висновок до розділу 1

В даному розділі було проведено системний аналіз предметної області інформаційної системи для обліку індивідуальних медичних показників військовослужбовців. Цей аналіз дозволив сформувати системне уявлення про її структуру, функції та вимоги. Постановка задачі окреслила основну мету системи, яка полягала в автоматизації управління медичною інформацією у військово-медичних закладах. Головною метою цієї системи було підвищення ефективності медичного лікування та підтримки прийняття клінічних рішень. Огляд наявної літератури та доступних рішень показав переваги сучасних систем, таких як MHS GENESIS або OpenEMR. Однак також було продемонстровано, що ці системи потребують адаптації для врахування специфічних особливостей української військової медицини, зокрема, щодо локалізації та захисту даних.

Для моделювання предметної області було використано певні методи, включаючи UML-діаграми і контексту діаграму, що дозволило отримати чітке уявлення про структуру, поведінку і потоки даних системи. Використання технологій архітектури .NET Framework, SQL Server та MVVM забезпечує надійність, масштабованість та простоту використання інтерфейсу. Очевидно, що окреслена концепція системи є комплексним рішенням з точки зору функціональних і нефункціональних вимог. Вона запроваджує автоматизацію обліку, аналізу даних та захисту інформації, що є надзвичайно важливим для військово-медичних закладів.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Інформаційне забезпечення інформаційної системи для обліку медичних показників військовослужбовців є ключовим елементом, що охоплює єдину систему класифікації, уніфіковану систему документації та інформаційну базу. Інформаційна база являє собою структуровану сукупність даних, організованих у пам’яті обчислювальної системи, що сприяє ефективній зберіганню, обробці та доступу до інформації. Проектування інформаційного забезпечення передбачає створення логічної моделі даних, яка відповідає вимогам реляційності та забезпечує оптимальну організацію даних для виконання функціональних завдань системи.

Логічна модель даних розроблена у вигляді ER-діаграми (рис. 2.1), яка відображає основні сутності системи та зв’язки між ними [22]. Основними сутностями є: Користувач, Медична установа, Запис на прийом, Стасус запису на прийом, Військовослужбовець, Стать, Обстеження, Результат обстеження, Параметр обстеження, Системний журнал, Статус користувача, Роль користувача.

Кожна сутність має набір атрибутів, що описують її характеристики, та зв’язки, які визначають взаємодію між сутностями [23]. Наприклад, сутність Обстеження пов’язана з Військовослужбовцем, Користувачем (лікар, який проводить обстеження) та Медичною установою (місце проведення), та має дату обстеження. А також сутність Результат обстеження, що посилається на Обстеження та Параметр обстеження і містить вимірювання. Аналогічно, Запис на прийом має дату, час, опис і пов’язує Військовослужбовця, Користувача, Медичну установу та Статус прийому.

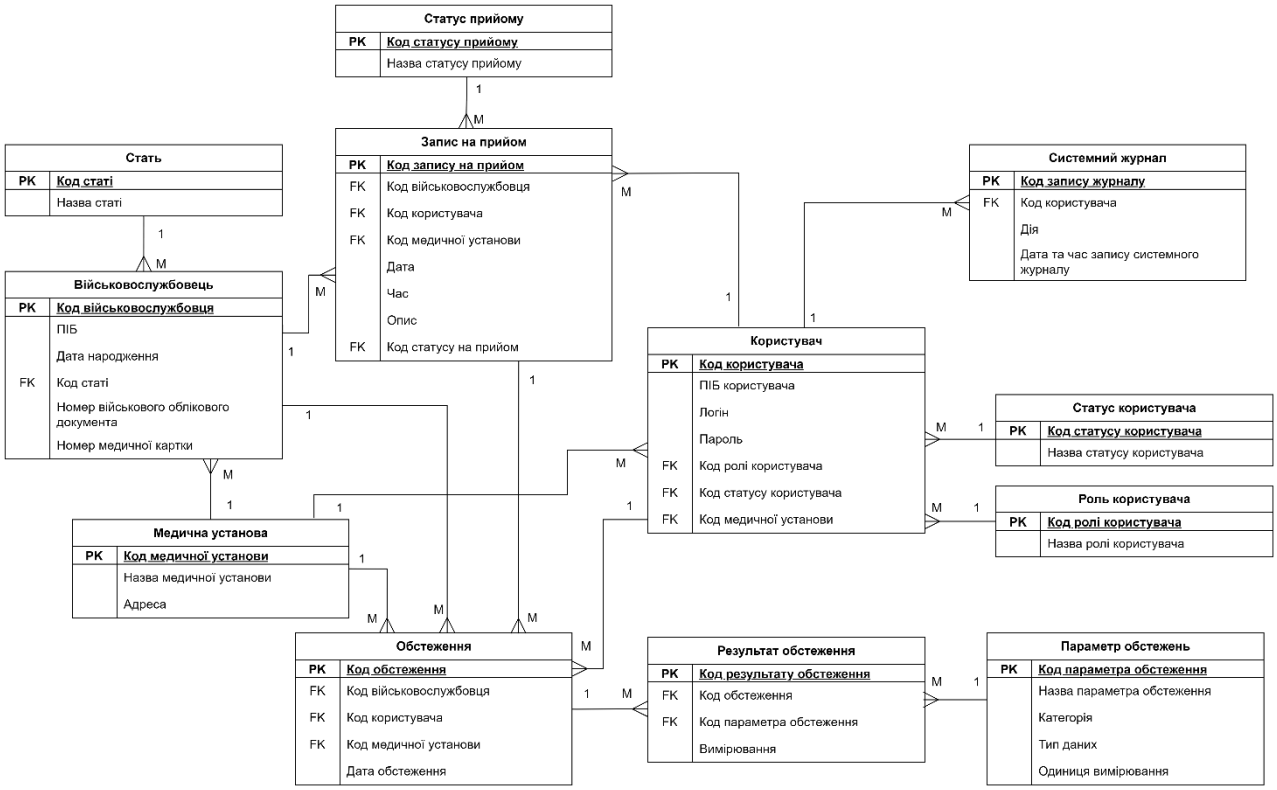

Рис. 2.1 ER-діаграма

Логічна модель даних відповідає вимогам реляційності та перебуває у третій нормальній формі (3NF). Це забезпечується такими принципами [24]:

– Перша нормальна форма (1NF): Усі атрибути сутностей є атомарними, тобто не містять множинних значень або вкладених структур. Наприклад, поле ПІБ у таблиці Військовослужбовець- це один текстовий рядок, а не набір з окремих імен. Так само і з полями типу Дата, Опис, Категорія також містять одне значення.

– Друга нормальна форма (2NF): Усі некритичні атрибути залежать від повного первинного ключа. Наприклад, у таблиці Результат обстеження- поле Вимірювання залежить від повного зв’язку між кодом Обстеження та Параметра обстеження, що виключає часткову залежність.

– Третя нормальна форма (3NF): Відсутні транзитивні залежності між атрибутами. Наприклад, у таблиці Користувача значення ролі або статусу не зберігається безпосередньо, а визначається через зовнішні ключі (Код ролі

користувача та Код статусу користувача), які ведуть до довідкових таблиць Роль користувача та Статус користувача.

– Таким чином, модель виключає надлишковість даних, мінімізує аномалії при вставці, видаленні чи оновленні даних і забезпечує ефективну роботу системи. Логічна модель є реляційною, оскільки базується на таблицях, пов’язаних через зовнішні ключі, і не включає нереляційні структури, такі як ієрархічні або мережеві моделі.

2.2 Вибір системи управління інформаційною базою

Для реалізації інформаційної бази обрано систему управління базами даних (СУБД) Microsoft SQL Server в одно користувацькій версії LocalDB [25]. Вибір обґрунтовується кількома факторами, що описані у табл. 2.1.

Таблиця 2.1

Фактори вибору Microsoft SQL Server LocalDB

№	Назва	Опис
1	Легкість інтеграції	SQL Server LocalDB безшовно інтегрується з .NET Framework та Entity Framework Core, що використовуються в проєкті, забезпечуючи ефективну роботу з даними через підхід Code First
2	Локальне розгортання	LocalDB не потребує окремого серверного обладнання, що ідеально підходить для настільного додатка, призначеного для локального використання в медичних установах
3	Підтримка реляційних функцій	СУБД забезпечує повну підтримку транзакцій, індексів, зовнішніх ключів та складних запитів, що відповідає потребам системи
4	Доступність	LocalDB є безкоштовним компонентом SQL Server, що знижує витрати на розгортання
5	Перспективи розширення	Структура бази даних створеної на LocalDB може легко мігрувати в повноцінний SQL Server.

Порівняно з альтернативами, такими як MySQL, PostgreSQL чи SQLite, SQL Server LocalDB має переваги в контексті платформи Windows завдяки тісній

інтеграції з Visual Studio та .NET. Наприклад, SQLite, хоча й легший, має обмеження в обробці складних зв’язків і паралельних транзакцій, тоді як MySQL і PostgreSQL вимагають додаткових зусиль для налаштування в локальному середовищі. Oracle і DB2, своєю чергою, є надмірно складними для настільного додатка з локальною базою даних.

Інформаційна база організована як централізована (рис. 2.2), оскільки весь обсяг даних зберігається в єдиній базі даних на локальному комп’ютері користувача [26]. Це відповідає потребам системи, яка призначена для використання в межах однієї медичної установи без необхідності розподіленої обробки. Централізована структура спрощує адміністрування, резервне копіювання та відновлення даних, а також сприяє швидкому доступу до інформації. Структурна схема системи включає один екземпляр бази даних, до якого звертаються всі компоненти додатка (авторизація, обробка обстежень, генерація звітів тощо).

Рис. 2.2 Схема централізованої системи інформаційної бази

Для гарантування безпеки та надійності даних передбачено механізми представлені у табл. 2.2.

Таблиця 2.2

Механізми забезпечення безпеки та надійності даних

№	Назва	Опис
1	Правила доступу	Доступ до даних регулюється ролями користувачів ("Doctor" або "Admin"). Лікарі можуть переглядати та додавати обстеження, військових і запис на прийом, тоді як адміністратори мають доступ до управління користувачами, медичними установами, параметрами, запису на прийом (скасування). Авторизація реалізується через логін і пароль, що зберігаються в базі.
2	Резервне копіювання	Система підтримує створення резервних копій бази даних у форматі .bak через спеціальну функцію в адміністративній панелі. Копії створюються вручну адміністратором, що дозволяє відновити дані в разі збою
3	Відновлення даних	Функція відновлення дозволяє замінити поточну базу даних із резервної копії, забезпечуючи захист від втрати даних
4	Автовихід	Для підвищення безпеки система автоматично завершує сеанс користувача після 10 хвилин бездіяльності
5	Журналювання	Усі дії користувачів (вхід, вихід, додавання даних, створення звітів) фіксуються в таблиці SystemLog, що дозволяє адміністратору відстежувати історію операцій і виявляти нестандартну активність

Оскільки база даних є централізованою, механізми реплікації даних не застосовуються [27]. Проте в майбутніх розширеннях системи можливе впровадження розподіленої архітектури з синхронізацією між кількома медичними установами, що вимагатиме реалізації реплікації через транзакційні або злиттєві моделі.

Таким чином, інформаційне забезпечення системи забезпечує надійне зберігання та обробку даних завдяки реляційній моделі, яка відповідає третій нормальній формі, використанню SQL Server LocalDB як оптимальної СУБД та

централізованій організації бази даних. Механізми безпеки, резервного копіювання та журналювання гарантують захист інформації.

2.3 Створення інформаційної бази

Створення інформаційної бази для інформаційної системи обліку медичних показників військовослужбовців є завершальним етапом проектування інформаційного забезпечення, який спрямований на реалізацію логічної моделі даних у фізичній структурі бази даних. Цей процес охоплює визначення структури таблиць, налаштування зв’язків, забезпечення цілісності даних, створення механізмів ініціалізації та наповнення бази, а також впровадження засобів для управління даними. Розроблена інформаційна база забезпечує надійне зберігання, швидкий доступ до даних та підтримку всіх функціональних можливостей системи.

Фізична структура інформаційної бази базується на логічній моделі, і реалізується через набір таблиць у реляційній базі даних Microsoft SQL Server LocalDB, що надає повну сумісність з СУБД SQL Server, в разі впровадження системи у медичних установах. Кожна сутність логічної моделі (User, MedicalFacility, Serviceman, Gender, Examination, Parameter, Result, SystemLog, Appointment, AppointmentStatus, UserStatus, UserRole) відповідає окремій таблиці, а зв’язки між сутностями забезпечуються зовнішніми ключами. Наприклад:

– Таблиця Users містить поля UserID (первинний ключ), FullName, Login, Password та зовнішні ключі UserRoleID (до таблиці UserRole) , UserStatusID (UserStatus) та MedicalFacilityID (MedicalFacility).

– Таблиця Examinations включає поля ExaminationsID, ServicemanID, UserID, MedicalFacilityID та ExaminationDate, де ServicemanID, UserID і MedicalFacilityID є зовнішніми ключами, що забезпечують зв’язок із таблицями Serviceman, User і MedicalFacility відповідно.

– Таблиця Results пов’язує обстеження з параметрами через поля ExaminationID і ParameterID, а також містить поле Value для зберігання результату вимірювання.

Для забезпечення цілісності даних у базі даних налаштовано обмеження:

– Первинні ключі (ID) у кожній таблиці гарантують унікальність записів.

– Зовнішні ключі забезпечують референційну цілісність, наприклад, неможливість видалення запису з таблиці Serviceman, якщо на нього посилаються записи в Examination.

– Обмеження на значення здійснюються за допомогою довідкових таблиць. Наприклад, у таблиці User поле UserRoleID посилається на таблицю UserRole, де можливими ролями є: лікар або адміністратор. Так само поле AppointmentStatusId у таблиці Appointment посилаються на таблицю AppointmentStatus, яка обмежена статусами: заплановано та скасовано.

Фізична модель відповідає третій нормальній формі, що виключає надлишковість даних і мінімізує аномалії при операціях вставки, оновлення чи видалення.

Діаграма моделі бази даних (рис. 2.3) створена в Microsoft SQL Server Management Studio за допомогою інструмента Database Diagram, який дозволяє бачити структури таблиць, їхні поля та зв'язки.

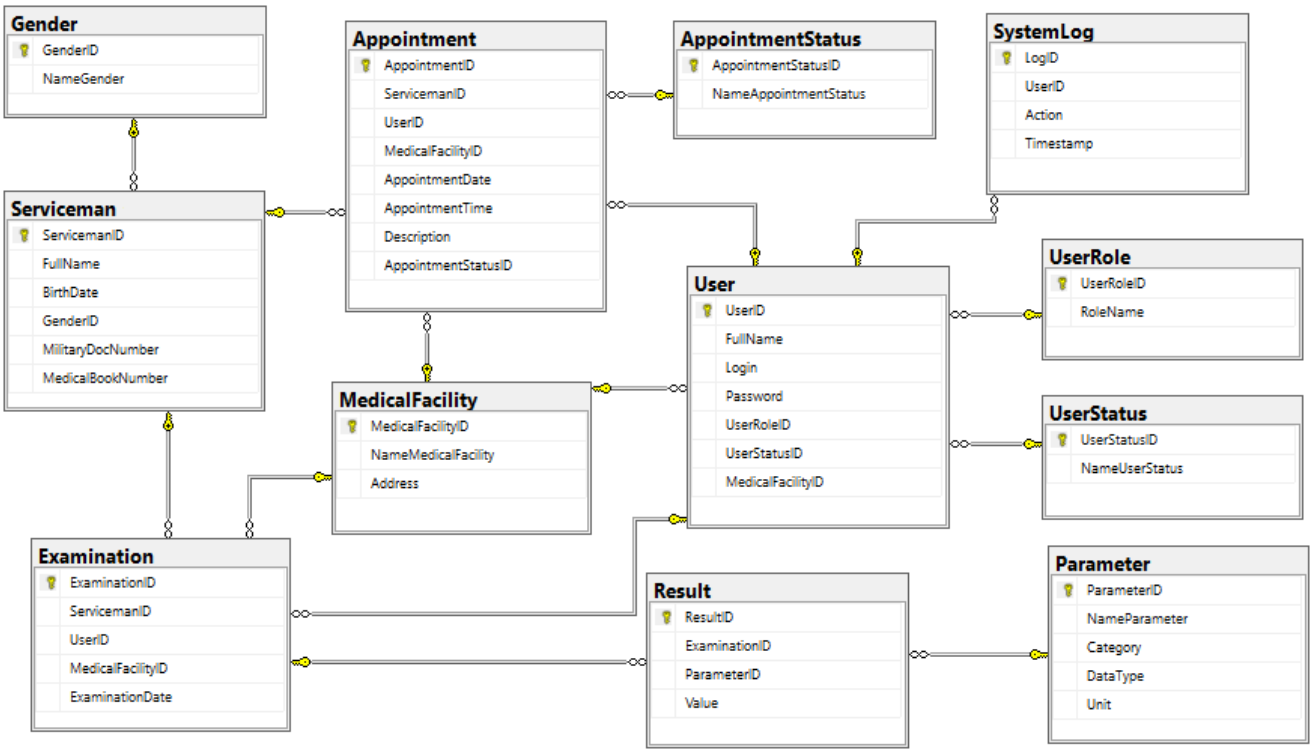

Рис. 2.3 Діаграма моделі бази даних реалізованої на фізичному рівні

Інформаційна база створюється автоматично за допомогою Entity Framework Core (EF Core) з підходом Code First, що дозволяє визначати структуру бази даних через класи C# і їх конфігурацію [28]. У коді класу MedicalContext (файл MainWindow.xaml.cs) визначено набори даних (DbSet) для кожної сутності, а метод OnConfiguring встановлює рядок підключення до SQL Server LocalDB, через використання методу LoadConnectionStringFromFile, який призначений для зчитування рядка підключення, що розміщений в окремому файлі. Рядок підключення має наступний вигляд:

```
"Server=(localdb)\\mssqllocaldb;Database=MilHealthDB;Trusted_Connection=True;TrustServerCertificate=True;"
```

Метод OnModelCreating налаштовує зв’язки між таблицями, наприклад, зв’язок один-до-багатьох між Serviceman і Examinations через ServicemanId.

При першому запуску програми викликається метод context.Database.EnsureCreated(), який створює базу даних, якщо вона ще не існує. Цей підхід забезпечує гнучкість у розгортанні системи, оскільки база

даних генерується автоматично на основі коду без необхідності ручного створення таблиць у SQL Server Management Studio.

Для того щоб система могла нормально працювати одразу після запуску, було зроблене автоматичне наповнення бази даних тестовими даними через клас DatabaseSeed (файл DatabaseSeed.cs). Процес наповнення включає етапи представлені у табл. 2.3.

Таблиця 2.3

Етапи процесу наповнення

№	Назва	Опис
1	Створення медичних установ	Додаються записи про п’ять установ із назвами та адресами, наприклад, «Військовий госпіталь №1, м. Київ, вул. Госпітальна, 18»
2	Створення користувачів	Генеруються один адміністратор (логін: admin, пароль: admin123) і п’ять лікарів із різними логінами та паролями, кожен із яких прив’язаний до певної медичної установи.
3	Створення параметрів обстежень	Додаються параметри для різних категорій (серцево-судинна система, загальні показники, лабораторні показники, психологія тощо), наприклад, «Частота серцевих скорочень» (Number, уд/хв) або «Психологічний стан» (Text)
4	Створення військовослужбовців	Генерується 12 записів із унікальними ПІБ, датами народження, статтю, номерами військових документів і медичних книжок
5	Створення обстежень і результатів	Для кожного військовослужбовця генерується від 15 до 25 обстежень за останні два роки з випадковими значеннями результатів, що моделюють реальні тренди (наприклад, сезонні коливання ваги чи рівня стресу)
6	Створення системних логів	Додаються записи про дії користувачів, такі як входи в систему, додавання обстежень чи створення звітів, із відповідними часовими мітками

Приклад створення системних логів для наповнення бази даних (рис. 2.4), де кожен запис містить користувача, дію та точний час.

```
var logEntries = new List<SystemLog>
{
    new SystemLog { UserId = admin.Id, Action = "Вхід в систему", Timestamp = DateTime.Now.AddDays(-5) },
    new SystemLog { UserId = admin.Id, Action = "Додано нового користувача", Timestamp = DateTime.Now.AddDays(-5).AddMinutes(15) },
    new SystemLog { UserId = admin.Id, Action = "Вихід із системи", Timestamp = DateTime.Now.AddDays(-5).AddMinutes(45) },
    new SystemLog { UserId = doctor1.Id, Action = "Вхід в систему", Timestamp = DateTime.Now.AddDays(-3) },
    new SystemLog { UserId = doctor1.Id, Action = "Додано нове обстеження", Timestamp = DateTime.Now.AddDays(-3).AddMinutes(20) },
    new SystemLog { UserId = doctor1.Id, Action = "Вихід із системи", Timestamp = DateTime.Now.AddDays(-3).AddMinutes(60) },
    new SystemLog { UserId = doctor2.Id, Action = "Вхід в систему", Timestamp = DateTime.Now.AddDays(-1) },
    new SystemLog { UserId = doctor2.Id, Action = "Призначено прийом", Timestamp = DateTime.Now.AddDays(-1).AddMinutes(15) },
    new SystemLog { UserId = doctor2.Id, Action = "Вихід із системи", Timestamp = DateTime.Now.AddDays(-1).AddMinutes(30) }
};

context.SystemLogs.AddRange(logEntries);
context.SaveChanges();
```

Рис. 2.4 Приклад створення системних логів

Цей механізм дозволяє протестувати всі функції системи, включаючи пошук, аналіз графіків, генерацію звітів і управління користувачами, без необхідності ручного введення даних.

Інформаційна база включає вбудовані механізми для гарантування безпеки та управління даними у табл. 2.4. Насамперед, безпека даних та ефективна обробка даних є важливим компонентом для будь-якої інформаційної системи. Вбудовані механізми гарантують захист даних від втрати, контролюють діяльність користувачів та запобігають помилкам і неправомірному доступу. Ці запобіжні заходи допомагають системі працювати без збоїв та зменшують навантаження на адміністраторів.

Таблиця 2.4

Вбудовані механізми для забезпечення безпеки та управління даними

№	Назва	Опис
1	Автовихід із системи	Реалізовано через таймер (inactivityTimer), який завершує сеанс після 10 хвилин бездіяльності, скидаючи інтерфейс до сторінки авторизації
2	Журналювання дій	Таблиця SystemLogs фіксує всі дії користувачів (вхід, вихід, додавання записів, створення звітів), що дозволяє адміністратору відстежувати активність і виявляти нестандартні операції
3	Резервне копіювання та відновлення	Функції BackupDatabaseButton_Click і RestoreDatabaseButton_Click забезпечують створення резервних копій бази даних у форматі .bak і відновлення з них з подальшим перезапуском програми

Таблиця 2.4 (закінчення)

4	Обробка помилок	При операціях із базою даних (додавання записів, оновлення, видалення) передбачено обробку винятків із виведенням повідомлень користувачу, наприклад, "Помилка додавання військовослужбовця: {ex.Message}"

Для забезпечення швидкого доступу до даних використано кілька підходів, що представлені у табл. 2.5.

Таблиця 2.5

Підходи для забезпечення швидкого доступу до даних

№	Назва	Опис
1	Індексація	Первинні та зовнішні ключі автоматично індексуються SQL Server LocalDB, що прискорює виконання запитів, наприклад, для пошуку військовослужбовців за номером медичної книжки чи військового документа
2	Ліниве завантаження	EF Core використовує ліниве завантаження (lazy loading) для зв’язаних даних, що зменшує обсяг даних, які завантажуються одночасно. Наприклад, список Examinations для Serviceman завантажується лише при переході до вкладки деталей
3	Агрегація запитів	Методи, такі як LoadServicemen чи LoadExaminations, використовують LINQ-запити з Include для ефективного завантаження пов’язаних даних, наприклад, context.Servicemen.Where(...).ToList()

Ці заходи забезпечують швидкодію системи навіть за наявності великої кількості записів.

Створення інформаційної бази для системи обліку медичних показників військовослужбовців виконано з урахуванням реляційної моделі даних, реалізованої через Entity Framework Core і SQL Server LocalDB. База даних включає таблиці для всіх ключових сутностей, забезпечує цілісність даних через зовнішні ключі та обмеження, а також підтримує автоматичну ініціалізацію тестовими даними. Механізми безпеки, резервного копіювання, журналювання та оптимізації доступу до даних забезпечують надійність, швидкодію та зручність використання системи. Такий підхід дозволяє ефективно виконувати

всі передбачені функції, від авторизації та пошуку до генерації звітів і аналізу графіків.

Висновок до розділу 2

В даному розділі було здійснене проєктування інформаційного забезпечення системи обліку медичних показників військовослужбовців. Створено логічну модель даних у вигляді ER-діаграми, яка охоплює всі ключові сутності, їх атрибути та зв'язки і відповідає принципам реляційної моделі. Модель відповідає третій нормальній формі (3НФ) - підходу до управління даними, який мінімізує надмірність і зменшує ймовірність помилок при додаванні, оновленні або видаленні записів.

Для реалізації інформаційної бази використано SQL Server LocalDB - полегшену версію Microsoft SQL Server, яка забезпечує повну сумісність з платформою .NET і не вимагає окремого серверного середовища. Такий підхід ідеально підходить для локального використання в межах одного медичного закладу. Архітектура бази даних полегшує перехід на повноцінний SQL Server, усуваючи необхідність внесення змін до її логіки або структури.

Фізична реалізація бази даних була виконана за допомогою Entity Framework Core із застосуванням підходу Code First. Це рішення дозволило створити автоматизовану генерацію структур таблиць на основі класів у коді, що полегшило встановлення зв'язків між сутностями за допомогою зовнішніх ключів. Крім того, система була налаштована на автоматичне введення даних, що дозволило швидко перевіряти функціональність системи. Це стосується таких функцій, як авторизація, обробка результатів обстежень та генерація звітів.

Для підвищення надійності впроваджено механізми безпеки, включаючи автоматичний вихід при неактивності, обробку помилок, ведення журналу дій користувача, резервне копіювання та відновлення бази даних. Оптимізація

доступу до даних досягається за рахунок індексування, лінивого завантаження та ефективних LINQ-запитів.

Таким чином, створена інформаційна база є надійною, функціональною та підготовленою до розширення. Вона забезпечує швидкий доступ до даних, зберігає їх цілісність та повністю відповідає вимогам системи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Організаційна структура програмного забезпечення

Організаційна структура прикладного програмного забезпечення (ППЗ) інформаційної системи для обліку медичних показників військових є ключовим елементом, що визначає логіку взаємодії компонентів системи та забезпечує її ефективне функціонування. Структура ППЗ побудована з урахуванням принципів модульності, що дозволяє чітко розмежувати функціональні блоки, спростити розробку, тестування та подальше масштабування системи. Відповідно до архітектурного підходу, система поділена на кілька основних пакетів: інтерфейс користувача, реалізація зв’язку з базою даних, обробка даних і формування звітів. Кожен із цих пакетів виконує спеціалізовані функції, забезпечуючи цілісність і надійність роботи системи. Важливо детально розглянути організаційну структуру ППЗ з описом кожного компонента.

Пакет інтерфейсу користувача є основним каналом взаємодії між користувачем і системою. Він реалізований з використанням технології Windows Presentation Foundation (WPF) і мови розмітки XAML, що забезпечує створення адаптивного та зручного графічного інтерфейсу [20, 29]. Цей пакет включає набір екранних форм, діалогових вікон і елементів управління, які дозволяють користувачам (лікарям і адміністраторам) виконувати авторизацію, переглядати списки військовослужбовців, вносити дані обстежень, призначати прийоми, генерувати звіти та переглядати графіки динаміки показників. Наприклад, у файлі MainWindow.xaml визначено структуру основного вікна з вкладками для різних функціональних зон, таких як пошук військовослужбовців, деталі обстежень і адміністративна панель. Для забезпечення локалізації інтерфейс підтримує українську мову, що реалізовано через налаштування культури в коді (Thread.CurrentThread.CurrentCulture = new CultureInfo("uk-UA")). Додатково,

бібліотека LiveCharts інтегрована для відображення графіків, що дозволяє візуалізувати динаміку числових медичних показників. Стилі та ресурси інтерфейсу централізовано визначені у файлі ResourceDictionary.xaml, що забезпечує одноманітність дизайну та спрощує підтримку.

Пакет зв’язку з базою даних відповідає за взаємодію системи з реляційною базою даних, яка базується на Microsoft SQL Server LocalDB. Для цього використано Entity Framework Core з підходом Code First, що дозволяє описувати моделі даних у коді та автоматично генерувати відповідну структуру бази даних. У файлі MainWindow.xaml.cs визначено клас MedicalContext, який успадковується від DbContext і містить набори даних (DbSet) для сутностей, таких як Users, Servicemen, Examinations, Appointments тощо. Метод OnConfiguring налаштовує підключення до бази даних через рядок з’єднання, а метод OnModelCreating визначає зв’язки між таблицями, наприклад, один-до-багатьох між Serviceman і Examinations. Для ініціалізації бази даних із тестовими даними використовується клас DatabaseSeed, який заповнює таблиці інформацією про медичні установи, лікарів, військовослужбовців і обстеження.

Цей пакет також забезпечує резервне копіювання та відновлення бази даних, що реалізовано через методи BackupDatabaseButton_Click та RestoreDatabaseButton_Click, які працюють із файлами формату .bak. Такий підхід гарантує цілісність даних і можливість їх відновлення у разі збоїв.

Пакет обробки даних виконує функції валідації, аналізу та трансформації інформації, отриманої від користувача або бази даних. Цей компонент включає бізнес-логіку системи, таку як пошук військовослужбовців за номером медичної книжки чи військового документа, збереження результатів обстежень, порівняння показників між двома обстеженнями та оновлення графіків. Наприклад, метод SearchButton_Click реалізує фільтрацію списку військовослужбовців за введеним текстом, використовуючи LINQ-запити до бази даних [30]. Метод SaveExaminationButton_Click перевіряє коректність введених даних (дата обстеження, медична установа тощо) перед збереженням,

а також збирає результати вимірювань із різних категорій (загальні показники, лабораторні дані тощо). Для забезпечення безпеки введених даних застосовуються перевірки на унікальність (наприклад, номерів документів у методі SaveNewServicemanButton_Click) та формат (наприклад, числові значення для певних параметрів). Пакет також підтримує автоматичний вихід із системи після 10 хвилин бездіяльності, що реалізовано через таймер у методі InitializeInactivityTimer.

Пакет формування звітів відповідає за створення PDF-документів, які містять детальну інформацію про обстеження або порівняння двох обстежень. Для цього використано бібліотеку iTextSharp, яка забезпечує генерацію документів із підтримкою кирилиці. Метод ExaminationReportButton_Click створює звіт для одного обстеження, включаючи інформацію про медичну установу, лікаря, військовослужбовця та результати вимірювань, згруповані за категоріями. Метод CompareExaminationsButton_Click генерує порівняльний звіт, відображаючи зміни показників у форматі «старий показник, новий показник». Обидва методи дозволяють користувачу вибрати місце збереження файлу через діалог SaveFileDialog, що підвищує зручність використання. Звіти формуються з урахуванням структури даних, отриманої з бази, і включають усі необхідні параметри, такі як артеріальний тиск, психологічний стан, лабораторні показники тощо, що відповідає побажанням до проекту.

Організаційна структура ППЗ, описана вище, може бути представлена у вигляді діаграми пакетів (рис. 3.1), де кожен пакет пов’язаний із іншими через чітко визначені інтерфейси [31]. Наприклад, інтерфейс користувача взаємодіє з пакетом обробки даних для відображення результатів пошуку чи графіків, а пакет зв’язку з базою даних забезпечує збереження та отримання інформації для всіх інших компонентів. Такий модульний підхід дозволяє легко розширювати систему, наприклад, додаванням нових типів обстежень чи інтеграцією з іншими базами даних, а також спрощує підтримку та усунення помилок.

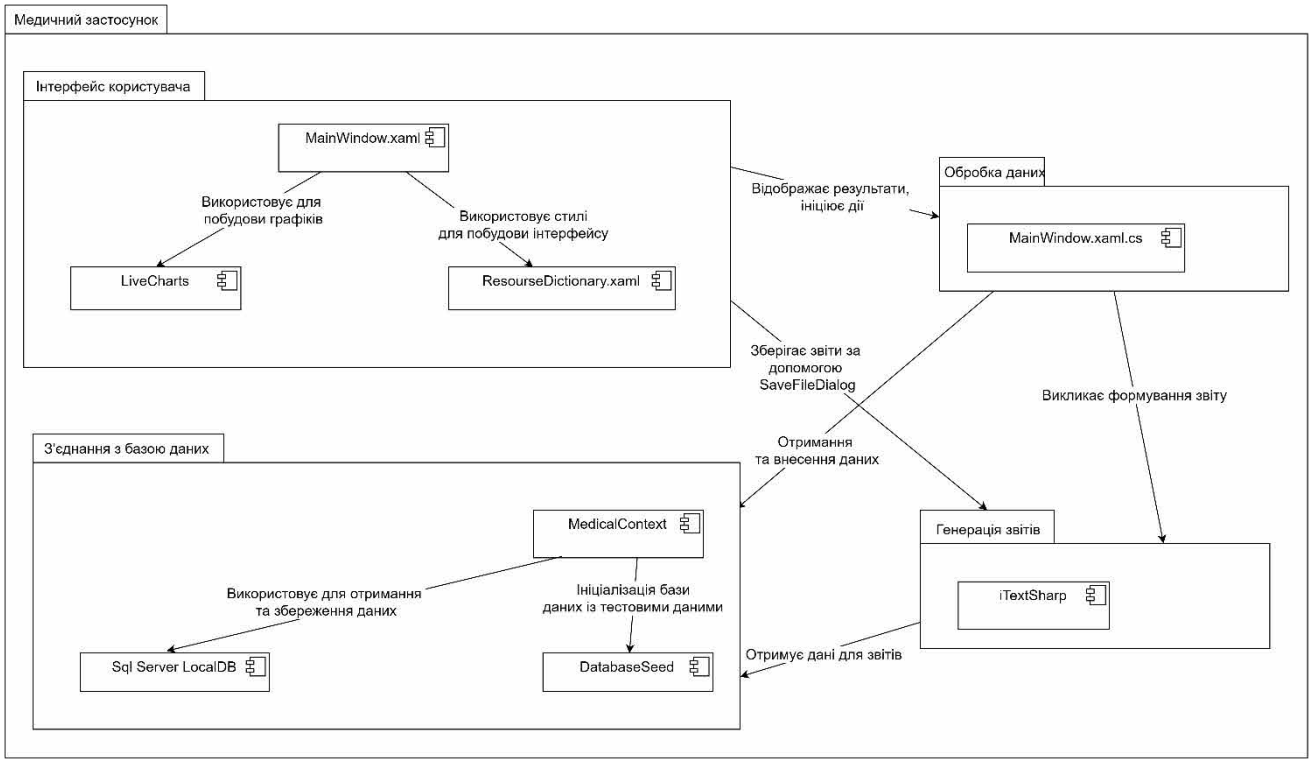

Рис. 3.1 Діаграма пакетів

У підсумку, організаційна структура ППЗ інформаційної системи для обліку медичних показників військових є добре продуманою. Використання сучасних технологій, таких як WPF, Entity Framework Core та iTextSharp, у поєднанні з модульною архітектурою забезпечує високу продуктивність, гнучкість і зручність використання системи як для лікарів, так і для адміністраторів.

Для опису структури предметної області використано діаграму класів UML (рис. 3.2), яка відображає основні сутності системи, їх атрибути та зв’язки між ними [15]. Основними класи описані у табл. 3.1.

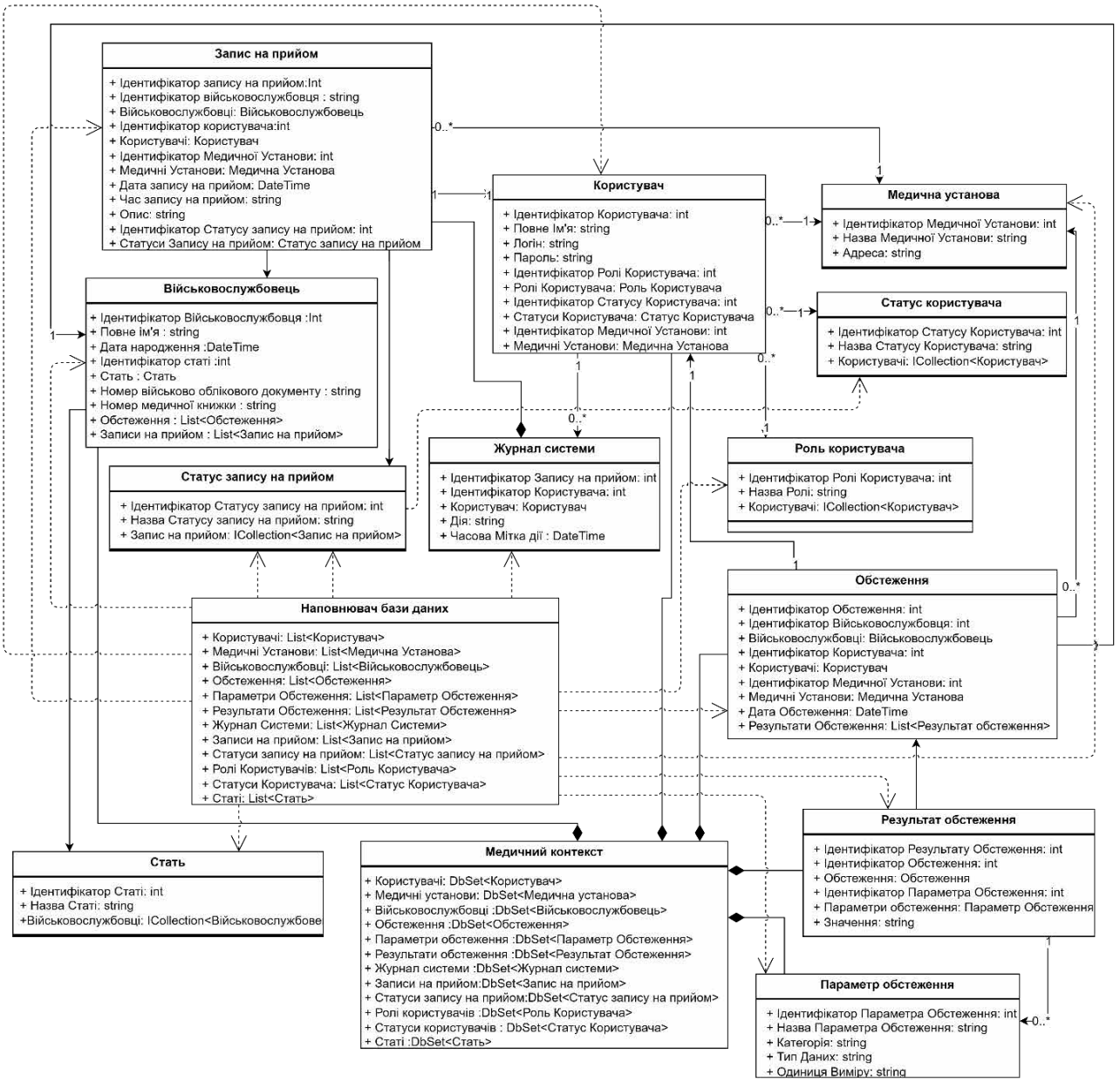

Рис. 3.2 Діаграма класів

Таблиця 3.1

Основні класи системи

№	Назва	Опис
1	Користувач	Представляє зареєстрованих користувачів системи (лікарів та адміністраторів). Атрибути: ідентифікатор користувача, Повне ім’я, логін, пароль, роль, статус, ідентифікатор медичної установи. Зв'язок із класом медична установа через зовнішній ключ ідентифікатор медичної установи

Таблиця 3.1 (продовження)

№	Назва	Опис
2	Медична установа	Описує заклади, де проводяться обстеження. Атрибути: ідентифікатор медичної установи, назва медичної установи, адреса.
3	Військовослужбовець	Містить інформацію про особу, чиї медичні показники обстежуються. Атрибути: ідентифікатор військовослужбовця, повне ім’я, дата народження, стать, номер військового документа, номер медичної книжки. Має зв'язки один-до-багатьох із записами обстеження та записами на прийом.
4	Обстеження	Фіксує дані про медичне обстеження. Атрибути: ідентифікатор обстеження, Ідентифікатор військовослужбовця, ідентифікатор лікаря, ідентифікатор медичної установи, дата обстеження. Зв'язки: один-до-багатьох із результатом обстеження, один-до-одного з військовослужбовцем, користувачем (Лікарем), медичною установою
5	Параметр обстеження	Довідник медичних показників. Атрибути: ідентифікатор параметра, назва параметра, категорія, тип даних, одиниця виміру
6	Результат обстеження	Зберігає значення конкретного параметра для обстеження. Атрибути: ідентифікатор результату, ідентифікатор обстеження, ідентифікатор параметра, значення
7	Запис на прийом	Фіксує заплановані прийоми. Атрибути: ідентифікатор запису, ідентифікатор військовослужбовця, ідентифікатор лікаря, ідентифікатор медичної установи, дата прийому, час прийому, опис, статус
8	Журнал системи	Записує дії користувачів. Атрибути: ідентифікатор запису, ідентифікатор користувача, дія, часова мітка запису.
9	Наповнювач бази даних	Клас для ініціалізації початкових даних системи. Містить колекції всіх основних сутностей: користувачі, медичні установи, військовослужбовці, обстеження, параметри, результати, записи на прийом, журнал системи

Таблиця 3.1 (закінчення)

№	Назва	Опис
10	Медичний контекст	Entity Framework DbContext для роботи з базою даних. Містить DbSet для всіх сутностей: користувачі, медичні установи, військовослужбовці, параметри, обстеження, результати, записи на прийом, журнал системи
11	Роль користувача	Довідник ролей користувачів системи. Атрибути: ідентифікатор ролі, назва ролі
12	Статус користувача	Довідник статусів користувачів. Атрибути: ідентифікатор статусу, назва статусу
13	Статус запису на прийомм	Довідник статусів на прийом. Атрибути: ідентифікатор статусу, назва статусу
14	Стать	Довідник статей. Атрибути: ідентифікатор статі, назва статі

3.2 Вибір інструментарію для створення ППЗ

Вибір інструментарію для розробки прикладного програмного забезпечення (ППЗ) є одним із ключових етапів створення інформаційної системи, оскільки від нього залежить ефективність розробки, якість кінцевого продукту та його відповідність поставленим вимогам. У контексті створення інформаційної системи обліку медичних показників військовослужбовців, описаної у реалізованому коді, вибір інструментальних засобів ґрунтується на потребах функціональності, зручності реалізації, масштабованості та відповідності сучасним стандартам розробки. Цей розділ присвячений обґрунтуванню вибору мови програмування, середовища розробки та супутніх бібліотек, а також підкреслює індивідуальність цього вибору та можливість використання різних інструментів для окремих компонентів системи.

Вибір інструментів для розробки ППЗ залежить від багатьох факторів, таких як тип задачі, вимоги до інтерфейсу користувача, необхідність інтеграції з базами даних, підтримка локалізації, а також особистий досвід розробника та рекомендації наукового керівника. У даному випадку інформаційна система

передбачає створення настільного додатка з графічним інтерфейсом, який забезпечує облік медичних даних, їх аналіз, генерацію звітів у форматі PDF та візуалізацію у вигляді графіків. Ці вимоги визначили основні критерії вибору інструментарію:

– Функціональність та продуктивність: інструменти повинні забезпечувати швидку обробку даних, надійну роботу з базами даних та можливість створення інтерактивного інтерфейсу.

– Локалізація: підтримка української мови як для інтерфейсу, так і для генерації документів.

– Масштабованість: можливість розширення функціональності в майбутньому.

– Доступність документації та спільноти: наявність ресурсів для підтримки розробки.

– Індивідуальний підхід: врахування досвіду розробника та рекомендацій керівника.

На основі цих критеріїв було обрано мову програмування C#, фреймворк .NET, технологію WPF для інтерфейсу, Entity Framework Core для роботи з даними, бібліотеки LiveCharts для візуалізації та iTextSharp для створення PDF-звітів. Середовищем розробки обрано Visual Studio, яке забезпечує інтеграцію всіх зазначених технологій.

Мова програмування C# була обрана як основна для реалізації ППЗ завдяки своїм універсальним можливостям та широкій інтеграції з екосистемою .NET. Основні переваги C# для даного проєкту описані в табл. 3.2 [32].

Таблиця 3.2

Переваги C#

№	Назва	Опис
1	Об’єктно-орієнтована парадигма	C# підтримує принципи ООП, що дозволяє створювати модульну та структуровану архітектуру системи, як видно у коді (наприклад, класи User, Serviceman, Examination)

Таблиця 3.2 (закінчення)

№	Назва	Опис
2	Інтеграція з .NET	C# є основною мовою для фреймворку .NET, який забезпечує доступ до потужних бібліотек для роботи з базами даних (Entity Framework Core), інтерфейсом користувача (WPF) та іншими компонентами
3	Підтримка локалізації	C# у поєднанні з .NET дозволяє легко налаштувати українську мову для інтерфейсу, як це реалізовано у методі MainWindow через встановлення CultureInfo(«uk-UA")
4	Безпека типів	Статична типізація C# мінімізує кількість помилок під час розробки, що дуже важливо для медичних систем, де точність даних є критичною
5	Широка спільнота та документація	C# має розвинену екосистему, що полегшує пошук рішень для специфічних задач, таких як генерація PDF-звітів чи побудова графіків

Хоча C# є оптимальним вибором, розглядалися й інші мови, такі як Java, Python та C++. Java має подібні можливості для створення настільних додатків через JavaFX, але поступається C# у зручності інтеграції з Windows-системами. Python, хоч і простий у освоєнні, менш ефективний для створення складних настільних додатків із графічним інтерфейсом. C++ гарантує високу продуктивність, але вимагає значно більше зусиль для реалізації інтерфейсу та роботи з базами даних. Таким чином, C# виявився збалансованим рішенням, яке поєднує продуктивність, зручність розробки та підтримку всіх необхідних функцій.

Visual Studio (версія 2019 або новіша) була обрана як основне середовище розробки через її комплексну підтримку проєктів на C# та .NET, а також інтеграцію з усіма необхідними інструментами [33]. Основні переваги Visual Studio надані в табл. 3.3.

Таблиця 3.3

Переваги Visual Studio

№	Назва	Опис
1	Інтегровані інструменти	Visual Studio підтримує створення WPF-додатків, роботу з Entity Framework Core, налагодження коду та управління пакетами NuGet, що використано для підключення бібліотек, таких як LiveCharts.Wpf та iTextSharp
2	Зручний дизайнер інтерфейсу	Вбудований редактор XAML дозволяє створювати та редагувати графічний інтерфейс, як видно у файлі MainWindow.xaml, де визначено вкладки, діалогові вікна та інші елементи
3	Інструменти для роботи з базами даних	Visual Studio забезпечує підтримку SQL Server, що використовується у проєкті для зберігання даних (MedicalContext)
4	Налагодження та тестування	Потужні інструменти дебагінгу дозволяють ефективно виявляти помилки, що особливо важливо для складних операцій, таких як збереження обстежень чи генерація звітів
5	Підтримка української мови	Visual Studio пропонує зручне налаштування інтерфейсу та документації для роботи з українською мовою

Розглядалися також альтернативні середовища, такі як JetBrains Rider та Visual Studio Code. Rider є потужним інструментом для розробки на C#, але його платна модель може стати проблемою за умов обмеженого бюджету. Visual Studio Code, хоч і легший і безкоштовний, потребує додаткових розширень для повноцінної роботи з WPF та Entity Framework, що ускладнює процес порівняно з Visual Studio. Таким чином, Visual Studio була обрана за її універсальність і готовність до роботи "з коробки".

Windows Presentation Foundation (WPF) обрано для створення графічного інтерфейсу завдяки її можливостям для побудови сучасних настільних додатків. Основні переваги WPF представлені в табл. 3.4 [34].

Таблиця 3.4

Переваги WPF

№	Назва	Опис
1	Гнучкість дизайну	Використання XAML дозволяє створювати адаптивні інтерфейси, як видно у структурі вкладок (MainTabControl) та діалогових вікон (AddServicemanDialog, AddExaminationDialog)
2	Прив’язка даних	WPF підтримує патерн MVVM, що спрощує зв’язок між моделями даних (наприклад, Serviceman, Examination) та інтерфейсом, як реалізовано через DataContext у MainWindow
3	Підтримка локалізації	Легке налаштування української мови для елементів інтерфейсу
4	Інтеграція з LiveCharts	WPF сумісна з бібліотекою LiveCharts для створення графіків, що використано для візуалізації медичних показників

Альтернативою WPF могла б бути Windows Forms, але вона менш гнучка для створення сучасних інтерфейсів і має обмеження в підтримці анімацій та стилізації.

Entity Framework Core (EF Core) обрано як ORM (Object-Relational Mapping) для взаємодії з базою даних SQL Server. Переваги EF Core описані в табл. 3.5 [35].

Таблиця 3.5

Переваги EF Core

№	Назва	Опис
1	Підхід Code First	Дозволяє визначати моделі даних у коді (класи User, Serviceman, Examination) і автоматично створювати базу даних, як реалізовано у MedicalContext
2	Продуктивність	EF Core оптимізує запити до бази даних, що важливо для швидкого пошуку військовослужбовців чи завантаження обстежень
3	Гнучкість	Підтримка складних зв’язків між таблицями, як видно у конфігурації OnModelCreating для зв’язків один-до-багатьох

Серед альтернатив можна назвати ADO.NET, але вона вимагає написання більших обсягів коду для SQL-запитів. Dapper, як легша ORM, також

розглядалася, але поступається EF Core у зручності роботи зі складними моделями.

Бібліотека LiveCharts обрана для створення графіків, що відображають динаміку медичних показників [36]. Її переваги показані в табл. 3.6.

Таблиця 3.6

Переваги LiveCharts

№	Назва	Опис
1	Простота інтеграції	Сумісність із WPF, як видно у використанні SeriesCollection і ChartLabels у MainWindow
2	Гнучкість	Підтримка різних типів графіків для відображення числових даних (наприклад, частота серцевих скорочень, артеріальний тиск)
3	Відкритість	Безкоштовна бібліотека з активною спільнотою

Альтернативою могла б бути OxyPlot, але LiveCharts виявилася простішою у налаштуванні для потреб проєкту.

Бібліотека iTextSharp використовується для створення PDF-звітів про обстеження та порівняння. Переваги надані в табл. 3.7 [37].

Таблиця 3.7

Переваги iTextSharp

№	Назва	Опис
1	Підтримка кирилиці	Забезпечує коректне відображення української мови у звітах
2	Гнучкість форматування	Дозволяє створювати структуровані документи, як видно у методах GenerateExaminationReport і GenerateComparisonReport
3	Безкоштовність	Відкрита версія бібліотеки відповідає потребам проєкту

Іншим варіантом є PdfSharp, але iTextSharp має ширшу функціональність для роботи з таблицями та шрифтами.

Вибір інструментів для розробки програмного забезпечення, передусім, зводиться до того, що найкраще підходить для проекту. Завжди слід враховувати

вимоги до програмного забезпечення, а потім вибирати технології, які зроблять більш плавний процес розробки. Дуже важливими чинниками будуть прості у використанні середовища, з підтримкою відповідних бібліотек, хорошою продуктивністю та якісною документацією. Врахування цих факторів дозволяє розробнику більше зосередитися над вивченням проблематики проекту, а не над функціоналом самих інструментів. У більшості випадків вибір також залежить від конкретної платформи або типу програми, наприклад, для настільних програм можуть знадобитися інструменти, відмінні від веб- або мобільних програм.

Використання різних інструментів для окремих частин ППЗ, система допускає використання різних мов і технологій для окремих компонентів. Наприклад:

– Інтерфейс користувача (WPF, XAML): XAML використовується для декларативного опису інтерфейсу, що відокремлює дизайн від логіки, написаної на C#.

– Робота з даними (Entity Framework Core): EF Core із C# забезпечує доступ до бази даних, що забезпечує використання та контроль над базою даних напряму з програмного коду, без використання стороннього програмного забезпечення. Проте для більш складних звітів можливе додавання SQL-скриптів для оптимізації запитів.

– Генерація звітів (iTextSharp): Бібліотека iTextSharp використовується разом із C#, дозволяє створювати документи програмно враховуючи такі елементи як текст, таблиці, зображення та форматування. Це дозволяє автоматично створювати добре структуровані та професійно виглядаючі звіти без ручного редагування. Python з бібліотекою ReportLab також може бути використаний для створення PDF-звітів як альтернатива iTextSharp.

– Візуалізація (LiveCharts): Використовувалась для створення інтерактивних графіків, що дозволило наочно відображати аналітичну інформацію про пацієнта. У разі розширення системи для веб-візуалізації можна

додати JavaScript із бібліотекою Chart.js. Даний підхід дозволяє гнучко адаптувати систему до нових вимог, зберігаючи основну частину коду уніфікованою.

Вибір інструментарію для створення інформаційної системи обліку медичних показників військовослужбовців ґрунтується на потребах проєкту, досвіді розробника та сучасних стандартах розробки. Мова C# у поєднанні з фреймворком .NET забезпечує надійну основу для реалізації бізнес-логіки, тоді як Visual Studio як середовище розробки пропонує комплексні інструменти для створення, налагодження та тестування додатка. Використання WPF для інтерфейсу, Entity Framework Core для роботи з даними, LiveCharts для візуалізації та iTextSharp для звітів дозволяє створити функціональний і зручний додаток, який відповідає поставленим вимогам. Водночас індивідуальність вибору інструментів та можливість використання різних технологій для окремих компонентів забезпечують гнучкість і перспективу для подальшого розвитку системи.

Цей вибір є результатом ретельного аналізу потреб проєкту та компромісу між зручністю, продуктивністю та доступністю інструментів, що підтверджує доцільність такого підходу для створення сучасного прикладного програмного забезпечення.

3.3. Алгоритмізація та програмування програмних модулів

Розробка прикладного програмного забезпечення (ППЗ) для інформаційної системи обліку медичних показників військовослужбовців вимагає чіткої алгоритмізації та програмування окремих модулів, що забезпечують функціональність системи. У цьому підрозділі висвітлено питання розробки алгоритмів для ключових програмних модулів, представлено їх у вигляді блок-схем, а також продемонстровано особливості реалізації через аналіз фрагментів коду. Основна увага приділена модулям автентифікації користувачів, додавання

нового військовослужбовця та генерації звітів про обстеження, оскільки вони є центральними для взаємодії з системою та обробки даних.

Модуль автентифікації користувачів є першим етапом взаємодії користувача з системою, забезпечуючи безпечний доступ до даних залежно від ролі ("Лікар" або "Адміністратор"). Алгоритм автентифікації включає перевірку логіну та пароля, визначення ролі користувача та налаштування інтерфейсу відповідно до рівня доступу.

Алгоритм автентифікації:

Крок 1: Отримати введені дані (логін, пароль) від користувача.

Крок 2: Перевірити, чи не є поля порожніми. Якщо порожні, вивести повідомлення про помилку.

Крок 3: Виконати запит до бази даних для пошуку користувача за логіном і паролем.

Крок 4: Якщо користувач не знайдений, виведення помилки.

Крок 5: Користувач знайдений- перевірка чи не заблокований користувач.

Крок 5: Користувач не заблокований- зберегти його дані в змінну currentUser і записати дію в системний журнал. Якщо заблокований- виведення помилки.

Крок 5: Налаштувати інтерфейс: приховати панель входу, відобразити головне меню, активувати вкладку адміністратора лише для ролі "Admin".

Крок 6: Якщо користувач не знайдений, вивести повідомлення про невірний логін або пароль.

Блок-схема представлена на рис. 3.3.

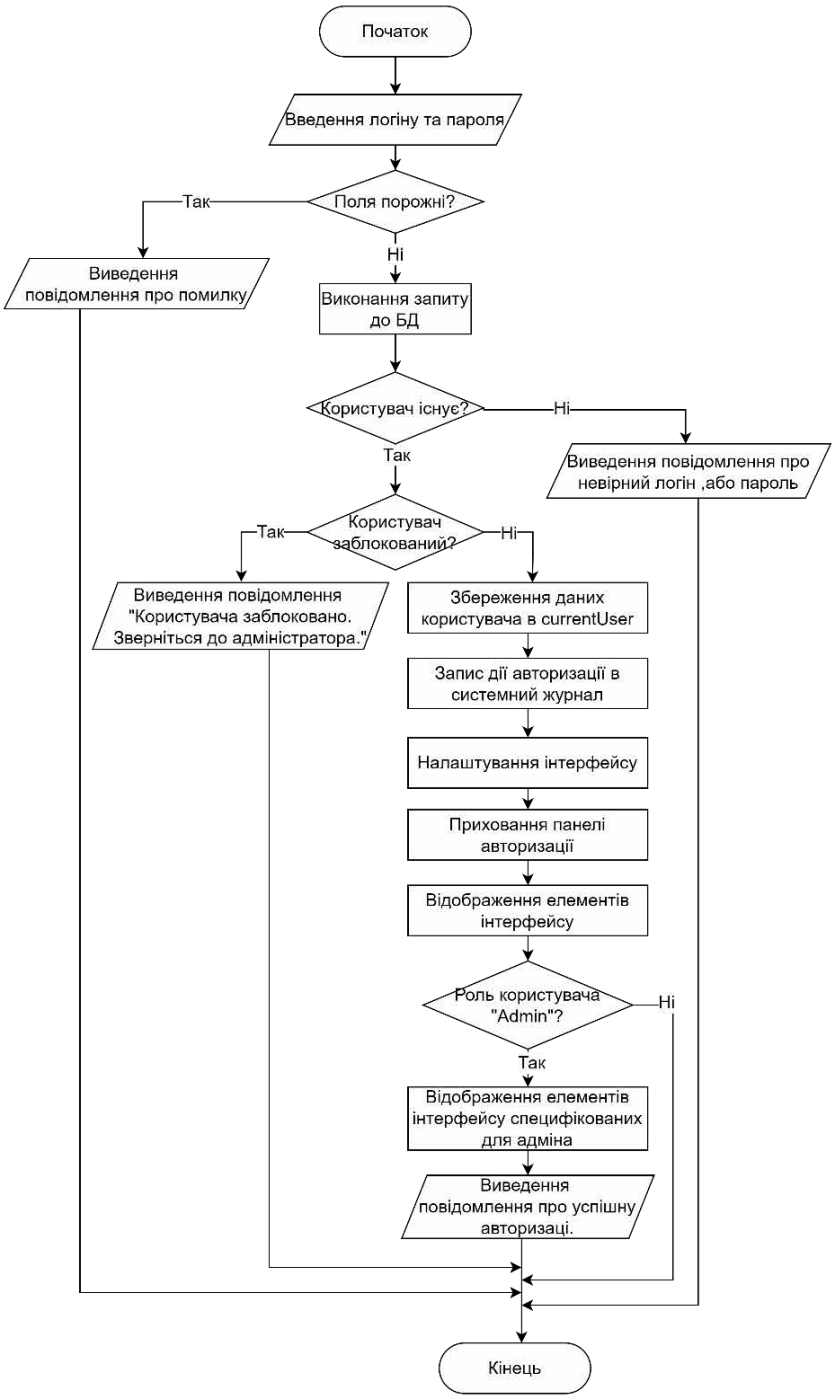

Рис. 3.3 Блок-схема алгоритму автентифікації

Фрагмент коду модуля автентифікації користувачів показано у Додатку Б.

Особливості реалізації:

– Використання Entity Framework Core для безпечного доступу до бази даних.

– Логування дій користувача через метод LogAction, що забезпечує відстеження активності.

– Динамічне налаштування інтерфейсу залежно від ролі користувача, що підвищує гнучкість системи.

Модуль додавання нового військовослужбовця дозволяє лікарям створювати записи про нових військовослужбовців, забезпечуючи унікальність даних (номер військового документа та медичної книжки).

Алгоритм додавання військовослужбовця:

Крок 1: Відобразити діалогове вікно для введення даних (ПІБ, дата народження, стать, номери документів).

Крок 2: Перевірити заповненість усіх обов’язкових полів.

Крок 3: Перевірити унікальність номерів документів у базі даних.

Крок 4: Якщо дані унікальні, створити новий запис Serviceman і зберегти його в базі.

Крок 5: Зафіксувати дію в системному журналі.

Крок 6: Оновити список військовослужбовців у інтерфейсі та приховати діалогове вікно.

Крок 7: У разі помилок (незаповнені поля, дублювання) вивести відповідне повідомлення.

Блок-схема зображена на рис. 3.4.

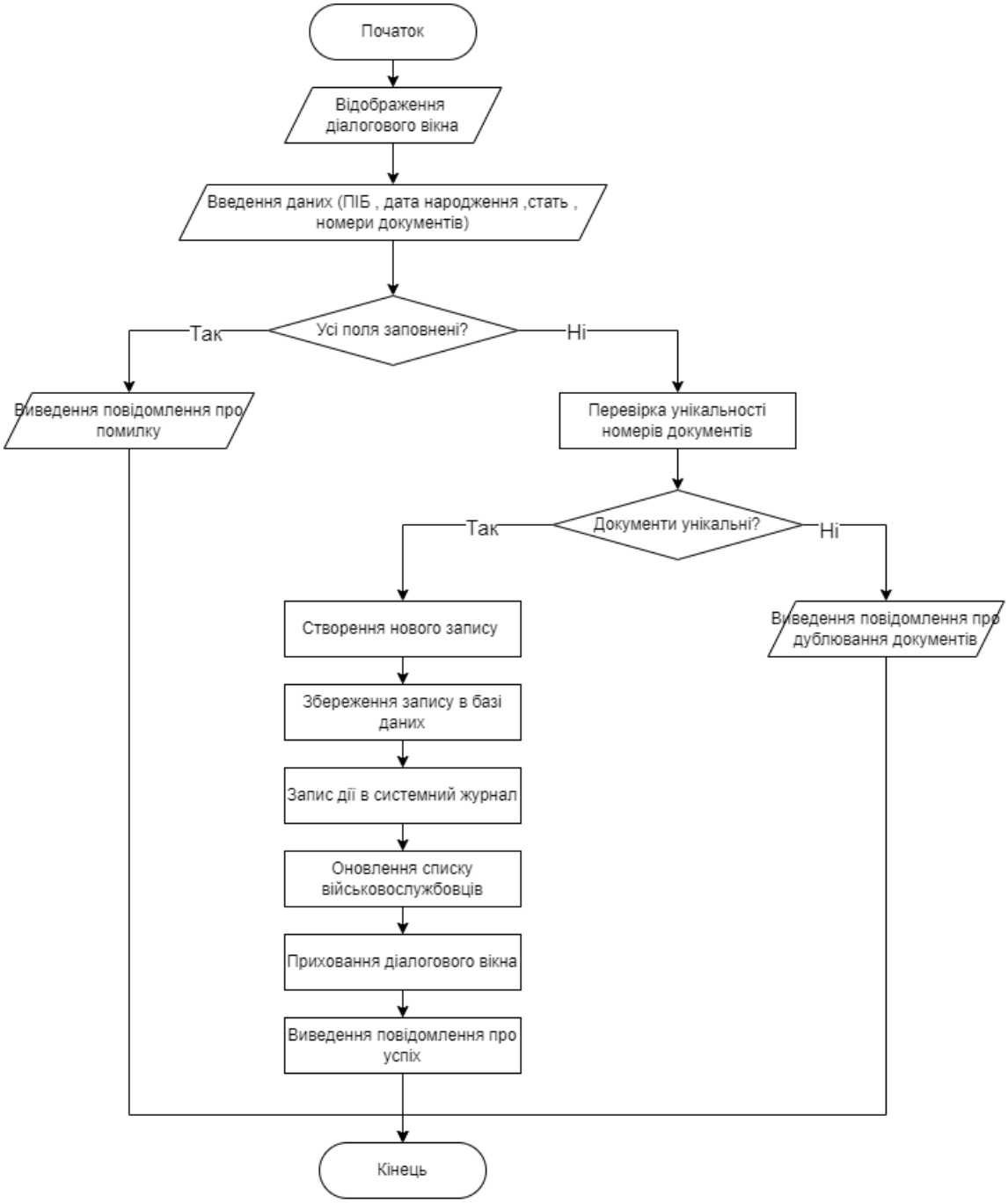

Рис. 3.4 Блок-схема алгоритму додавання військовослужбовця

Код модуля додавання військовослужбовця показано у Додатку Б.

Особливості реалізації:

– Ретельна валідація введених даних для запобігання помилкам.

– Перевірка унікальності документів через LINQ-запити, що оптимізує роботу з базою.

– Інтеграція з системним журналом для фіксації дій, що підвищує прозорість роботи системи.

Модуль генерації звітів дозволяє створювати PDF-документи з детальною інформацією про обстеження, що є важливим для документування медичних даних.

Алгоритм генерації звіту:

Крок 1: Перевірити, чи вибране обстеження в інтерфейсі.

Крок 2: Завантажити повні дані обстеження, включно з інформацією про військовослужбовця, лікаря, медичну установу та результати.

Крок 3: Відкрити діалог збереження файлу з форматом PDF.

Крок 4: Сформувати PDF-документ за допомогою бібліотеки iTextSharp, додавши структуровані дані.

Крок 5: Зберегти документ за вибраним шляхом.

Крок 6: У разі помилок вивести повідомлення.

Блок-схема представлена на рис. 3.5.

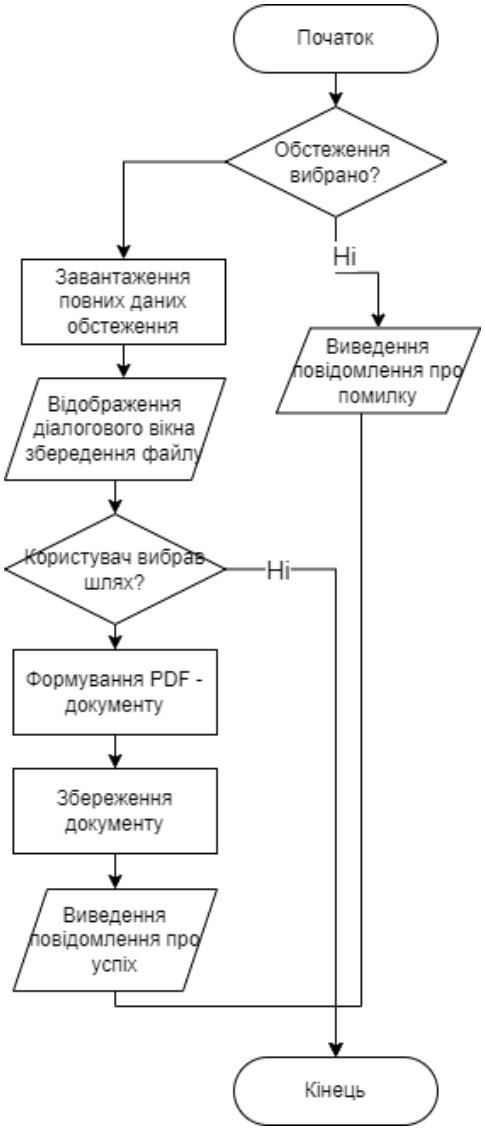

Рис. 3.5 Блок-схема алгоритму генерації звіту

Код модуля генерації звітів показано у Додатку Б.

Особливості реалізації:

– Використання iTextSharp для створення структурованих PDF-документів з підтримкою кирилиці.

– Групування результатів за категоріями для зручного представлення даних.

– Гнучке формування імені файлу з урахуванням даних військовослужбовця та дати.

Розроблені алгоритми та їх програмна реалізація демонструють комплексний підхід до створення інформаційної системи. Використання Entity Framework Core, WPF та iTextSharp забезпечує ефективну роботу з даними, зручний інтерфейс і професійне документування. Блок-схеми чітко відображають логіку роботи модулів, а фрагменти коду підкреслюють увагу до деталей, таких як валідація, безпека та логування. Ці рішення дозволяють системі бути надійною, масштабованою та зручною для користувачів.

Висновок до розділу 3

У третьому розділі було розглянуто основні аспекти, щодо розробки програмного забезпечення. Найважливішим завданням було створення архітектури системи, обґрунтування вибору інструментів та розробка основних програмних модулів.

Спочатку було визначено структуру програмного забезпечення ППЗ відповідно до принципу модульності, яку поділено на чотири базові модулі: інтерфейс користувача, реалізація зв’язку з базою даних обробка даних та формування звітності. Цей підхід дозволив розмежувати функціональні блоки, що спростило процес розробки, тестування та подальше масштабування. Крім, того було створено діаграму класів, яка передусім відображає основні сутності системи, що допомогло формалізувати струстуру системи.

Один з основних етапів полягав у виборі інструментів розробки. Було обґрунтовано вибір мови програмування C# та .NET Framework як найкращий варіант для створення настільного застосунку з графічним інтерфейсом. Для створення інтерфейсу була обрана технологія WPF через їй гнучкість та сучасний дизайн. Використано Entity Framework Core для операцій з базою даних з метою спрощення доступу до даних через підход ORM. LiveCharts використовувався для графічного представлення медичних показників у вигляді графіків, а для генерації звітності у форматі PDF- iTextSharp.

Особливий акцент було зроблено на алгоритмах та кодуванні ключових системних модулів - автентифікація користувачів, створення нових військовослужбовців та формування звітів. Кожен модуль було розроблено з урахуванням валідації даних, обробки помилок та журналювання дій для покращення надійності системи. Структури блок-схем та фрагменти програмного коду чудово демонструють логіку роботи застосунку. Таким чином створене програмне забезпечення підтверджує доцільність обраного підходу та можливість практичного використання у медичних установах.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування інформаційної системи є ключовим етапом її розробки та впровадження, спрямованим на забезпечення якості, надійності та відповідності програмного продукту поставленим вимогам. У контексті інформаційної системи обліку медичних показників військовослужбовців, розробленої на базі платформи WPF з використанням Entity Framework Core та Microsoft SQL Server LocalDB, тестування відіграє важливу роль у гарантуванні стабільності роботи, захисту даних та зручності використання для кінцевих користувачів – лікарів і адміністраторів. Необхідно детально розглянути поняття тестування, його основні показники, а також зобразити процес проведення обраного типу тестування для реалізованої системи.

Тестування програмного забезпечення – це процес перевірки коректності роботи системи шляхом виконання різних сценаріїв, з урахуванням оцінки продуктивності, функціональності, безпеки та інших характеристик з метою виявлення дефектів, забезпечення відповідності вимогам і гарантування належної роботи в реальних умовах експлуатації. Його основа полягає в порівнянні отриманих результатів з очікуваними. Тестування проводиться на різних етапах циклу розробки, що дозволяє своєчасно виявляти та усувати помилки, підвищуючи якість кінцевого продукту.

Для інформаційної системи обліку медичних показників військовослужбовців тестування має охоплювати кілька ключових напрямків, описаних у табл. 4.1.

Таблиця 4.1

Ключові напрямки тестування

№	Назва	Опис
1	Функціональне тестування	Перевірка коректності реалізації всіх функцій системи, таких як автентифікація користувачів, додавання та редагування даних про військовослужбовців, збереження результатів обстежень, генерація звітів у форматі PDF, а також управління призначеннями прийомів
2	Тестування продуктивності	Оцінка швидкості роботи системи під час обробки великих обсягів даних, наприклад, при пошуку військовослужбовців або побудові графіків на основі історичних даних
3	Тестування безпеки	Перевірка захисту даних користувачів, забезпечення коректної роботи механізмів авторизації та розмежування доступу між ролями "Лікар" і "Адміністратор"
4	Тестування інтерфейсу користувача	Аналіз зручності та інтуїтивності інтерфейсу, коректності відображення елементів на різних роздільних здатностях екрана, а також відповідності локалізації українській мові
5	Тестування сумісності	Перевірка роботи системи на різних версіях операційної системи Windows та з різними конфігураціями SQL Server
6	Тестування відновлення даних	Оцінка коректності механізмів створення резервних копій бази даних і їх відновлення

Процес тестування зазвичай складається з кількох етапів: планування, розробка тестових сценаріїв, виконання тестів, аналіз результатів і виправлення виявлених дефектів. Для автоматизації тестування можуть використовуватися спеціалізовані інструменти, такі як NUnit або MSTest, однак для даної системи значна частина тестування виконується вручну через специфіку інтерфейсу користувача та необхідність перевірки візуальних елементів.

Тестування системи зазвичай проводиться в умовах, максимально наближених до тих, що виникають в процесі експлуатації, із використанням тестової бази даних, заповненої демонстраційними даними за допомогою класу DatabaseSeed. Це дозволяє оцінити поведінку системи при обробці різноманітних сценаріїв, таких як додавання нових записів, виконання пошуку чи порівняння результатів обстежень.

Для ілюстрації процесу тестування розглянуто функціональне тестування- один із найважливіших типів тестування для даної системи, оскільки він спрямований на перевірку коректності виконання основних функцій. Функціональне тестування було обрано через його здатність охопити ключові сценарії використання системи, такі як автентифікація, управління даними військовослужбовців, збереження обстежень і генерація звітів. Необхідно розглянути детальний опис процесу його проведення.

- Крок 1. Планування тестування

На етапі планування було складено перелік функціональних вимог, які необхідно перевірити. Наприклад:

– Успішна автентифікація користувачів із ролями "Лікар" і "Адміністратор".

– Додавання нового військовослужбовця з унікальними номерами документів.

– Створення нового обстеження з різними типами параметрів (числові, текстові).

– Генерація звіту у форматі PDF із результатами обстеження.

– Порівняння двох обстежень із коректним відображенням різниці в показниках.

Для кожного сценарію визначено вхідні дані, очікувані результати та критерії успішності. Наприклад, для автентифікації: введення логіну "admin" і пароля "admin123" має відкрити інтерфейс із вкладкою адміністратора, тоді як некоректний пароль має відобразити повідомлення про помилку.

- Крок 2. Розробка тестових сценаріїв

Було створено набір тестових сценаріїв, які охоплюють типові та граничні випадки. Наприклад, для функції додавання військовослужбовця:

– Сценарій 1: Успішне додавання з коректними даними (ПІБ: "Тестовий Іван Іванович", дата народження: 01.01.1990, стать: "Чоловіча",

унікальні номери документів). Очікуваний результат: запис з’являється в базі даних, відображається в таблиці.

– Сценарій 2: Спроба додавання з неунікальним номером документа (рис. 4.1). Очікуваний результат: відображення повідомлення про помилку, запис не додається.

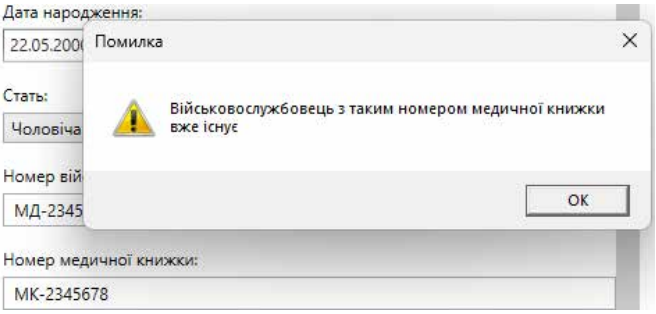

Рис 4.1 Спроба додавання з неунікальним номером документа

– Сценарій 3: Введення порожнього ПІБ (рис. 4.2). Очікуваний результат: повідомлення про необхідність заповнення поля.

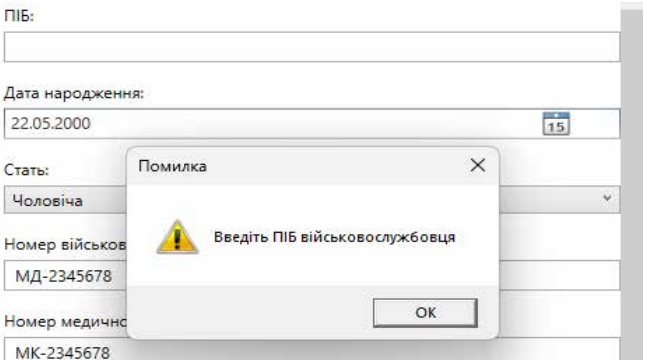

Рис. 4.2 Спроба залишити поле ПІБ порожнім

Аналогічні сценарії розроблено для інших функцій, таких як збереження обстежень і генерація звітів.

- Крок 3. Виконання тестів

Тестування проводилося вручну на підготовленому середовищі з Visual Studio 2022, SQL Server, SQL Server LocalDB. Спочатку база даних була ініціалізована демонстраційними даними за допомогою методу DatabaseSeed.SeedDatabase(). Тестувальник послідовно виконував кожен сценарій, вводячи дані через інтерфейс і фіксуючи результати. Наприклад:

– Для перевірки автентифікації тестувальник увійшов із логіном "doctor1" і паролем "doctor123", переконався, що відкрилися вкладки лікаря, а вкладка адміністратора залишалася прихованою.

– Для тестування додавання обстеження створювався запис із числовими (наприклад, частота серцевих скорочень: 75 уд/хв) і текстовими (загальний аналіз крові: "В межах норми") параметрами, після чого перевірялася їх коректна присутність у базі даних.

– Для генерації звіту тестувальник обирав обстеження, створював PDF-файл і перевіряв його вміст, зокрема правильність відображення кириличних символів.

- Крок 4. Аналіз результатів і виправлення дефектів

Під час тестування було виявлено кілька проблем, такі як:

– Некоректне відображення повідомлення про помилку при введенні невалідної дати обстеження (виправлено валідацією в методі SaveExaminationButton_Click).

– Проблеми з оновленням графіків при виборі нового параметра (усунено шляхом оптимізації методу UpdateChart).

Усі виявлені дефекти документувалися, після чого було внесено відповідні зміни до коду. Коли зміни були внесені, проводилося повторне тестування для підтвердження усунення помилок.

- Крок 5. Документація результатів

Результати тестування фіксувалися у вигляді таблиці, яка мала опис сценарію, вхідних даних, фактичних і очікувані результати, а також статус (успішно/невдало). Приклад результатів тестування зображено у табл. 4.2.

Таблиця 4.2

Приклад результатів тестування

Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
Додавання військовослужбовця	ПІБ: "Тестовий", унікальні номери	Запис додано, відображено в таблиці	Запис додано	Успішно
Некоректний логін	Логін: "invalid", пароль: "123"	Повідомлення про помилку	Повідомлення відображено	Успішно

Функціональне тестування дозволило пересвідчитися у коректності реалізації основних можливостей системи, а також виявити та усунути дрібні недоліки, що сприяло підвищенню її надійність і готовності до експлуатації.

Тестування інформаційної системи є дуже важливою частиною її впровадження, що забезпечує якість і стабільність роботи. Для підвищення ефективності тестування в майбутньому рекомендується автоматизувати повторювані сценарії, використовуючи фреймворки на кшталт MSTest, а також проводити тестування навантаження для оцінки роботи системи з великими обсягами даних. Комплексний підхід до тестування сприятиме успішному впровадженню системи в реальних умовах медичних установ.

4.2 Покрокове виконання програми

– Крок 1. Запуск програми

Процес роботи системи розпочинається з ініціалізації програми, коли користувач запускає виконуваний файл, створений на основі проєкту, розробленого у середовищі .NET із використанням технології WPF. На даному етапі програма виконує кілька важливих підготовчих дій. Спочатку встановлюється українська локалізація для забезпечення коректного відображення текстів та форматів даних, таких як дати чи числа. Далі ініціалізується контекст бази даних, який базується на Entity Framework Core та

підключається до локальної бази даних Microsoft SQL Server LocalDB. Якщо база даних ще не створена, система автоматично формує її структуру та заповнює локаційними даними, що включають інформацію про медичні установи, користувачів, військовослужбовців та параметри обстежень. Одночасно створюються діалогові вікна для подальшої взаємодії, а також налаштовується таймер неактивності, який забезпечує автоматичний вихід із системи після 10 хвилин бездіяльності для захисту даних. На екрані з’являється вікно авторизації, де користувач має ввести свої облікові дані, щоб отримати доступ до функціоналу програми.

– Крок 2. Авторизація користувача

Після запуску програми користувач переходить до етапу авторизації, який є ключовим для забезпечення безпеки та персоналізації доступу (рис. 4.3). У відповідних полях необхідно ввести логін та пароль, які перевіряються системою шляхом порівняння із записами у базі даних. У разі введення коректних даних, наприклад, логіну "admin" із паролем "admin123" для адміністратора або "doctor1" із паролем "doctor123" для лікаря, система ідентифікує користувача та визначає його роль. Ця роль впливає на доступний функціонал: адміністратори мають ширші привілеї, включаючи управління користувачами, медичними установами та параметрами, тоді як лікарі зосереджені на медичних даних. Після успішної авторизації панель входу приховується, а головне вікно програми відкриває вкладки з основними функціями. У цей момент дія користувача фіксується у системному журналі, що забезпечує відстеження активності. Якщо ж дані введено неправильно, система виводить повідомлення про помилку, пропонуючи повторити спробу.

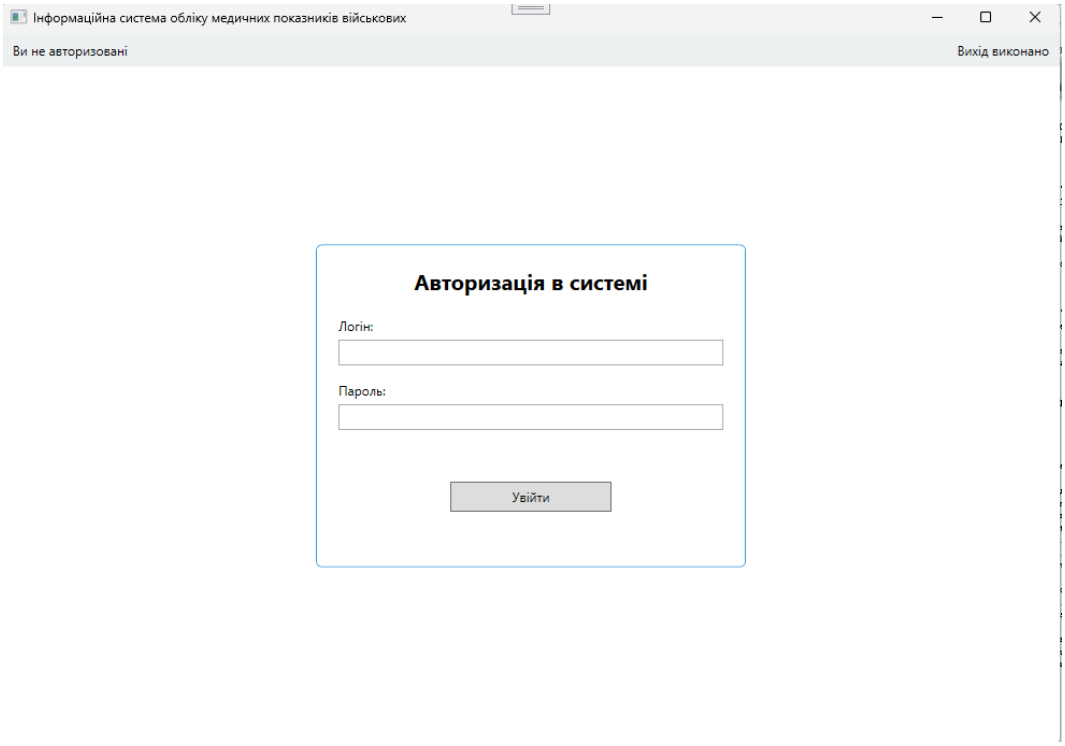

Рис. 4.3 Авторизація у системі

– Крок 3. Взаємодія з основним інтерфейсом

Після входу в систему користувач потрапляє до основного інтерфейсу, який побудовано за принципом вкладок для зручної навігації (рис. 4.4). Для лікаря доступні функції пошуку військовослужбовців, перегляду їх медичних даних, внесення результатів обстежень та призначення прийомів, додавання нового військовослужбовця (рис. 4.5). Пошук здійснюється за прізвищем, номером військового документа чи медичної книжки, що дозволяє швидко знайти потрібну особу. Вибравши військовослужбовця, користувач може переглянути його профіль, історію обстежень або призначень (рис. 4.6), а також створити нове обстеження, заповнивши дані про медичні показники, такі як артеріальний тиск чи рівень глюкози. Адміністратори, окрім цього, мають доступ до додаткових вкладок, де можуть додавати нових користувачів (рис. 4.7), медичні установи чи параметри обстежень, а також керувати резервними копіями бази даних. Усі дії супроводжуються логуванням для забезпечення прозорості та можливості аудиту. Інтерфейс дозволяє також

генерувати звіти у форматі PDF та будувати графіки для аналізу динаміки показників, що полегшує прийняття медичних рішень.

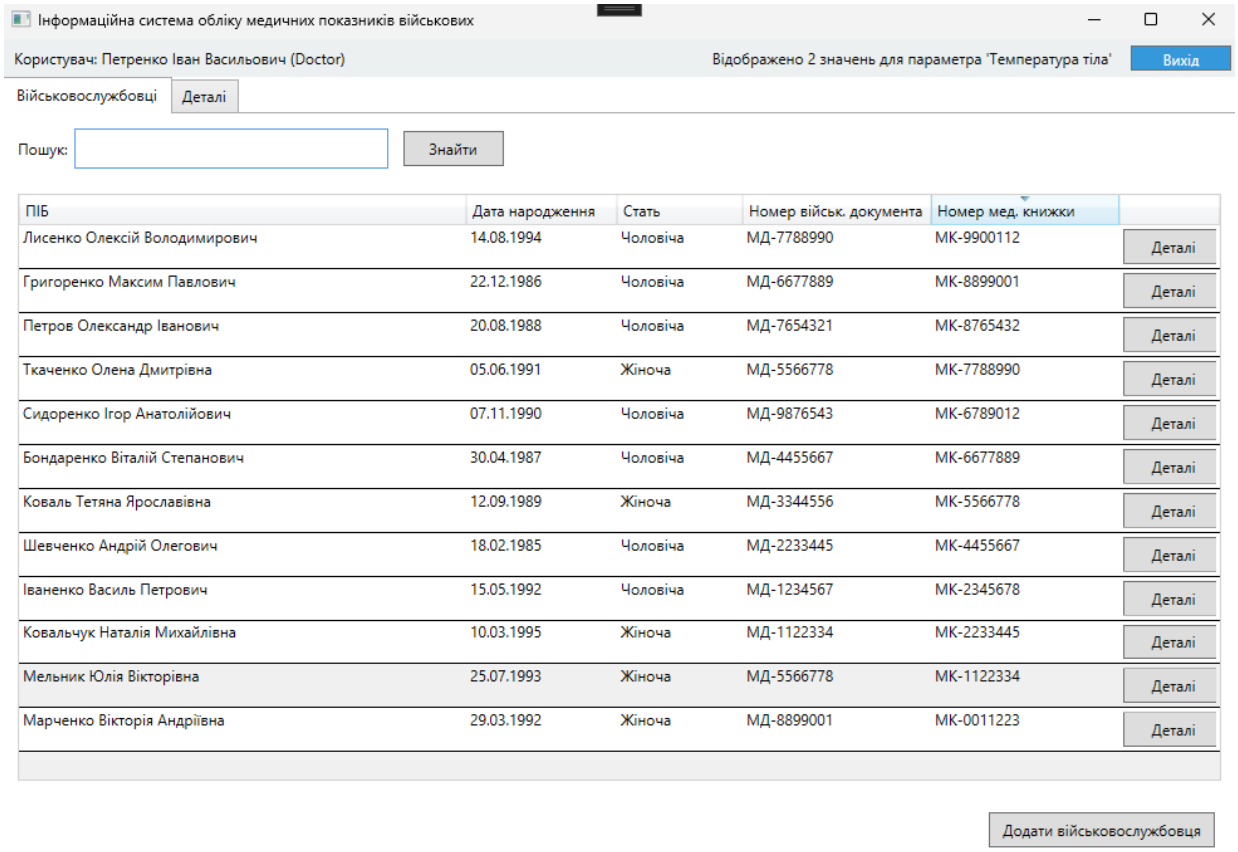

Рис. 4.4 Розділ «Військовослужбовці» для лікаря

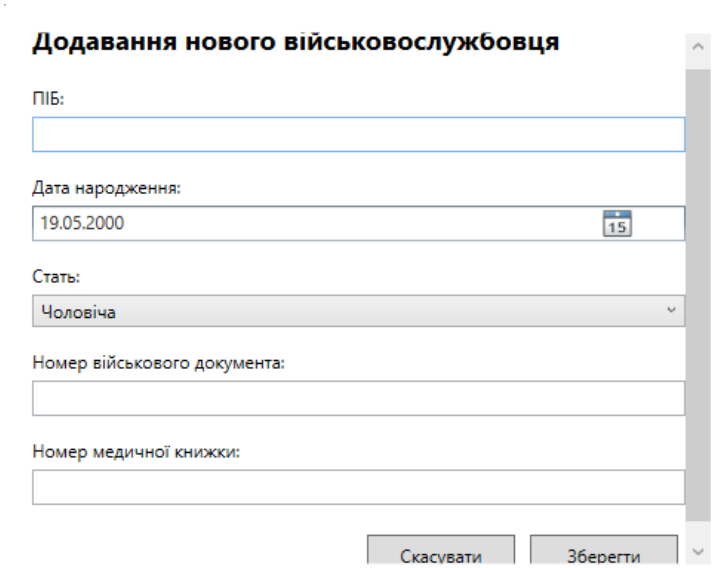

Рис. 4.5 Додавання нового військовослужбовця

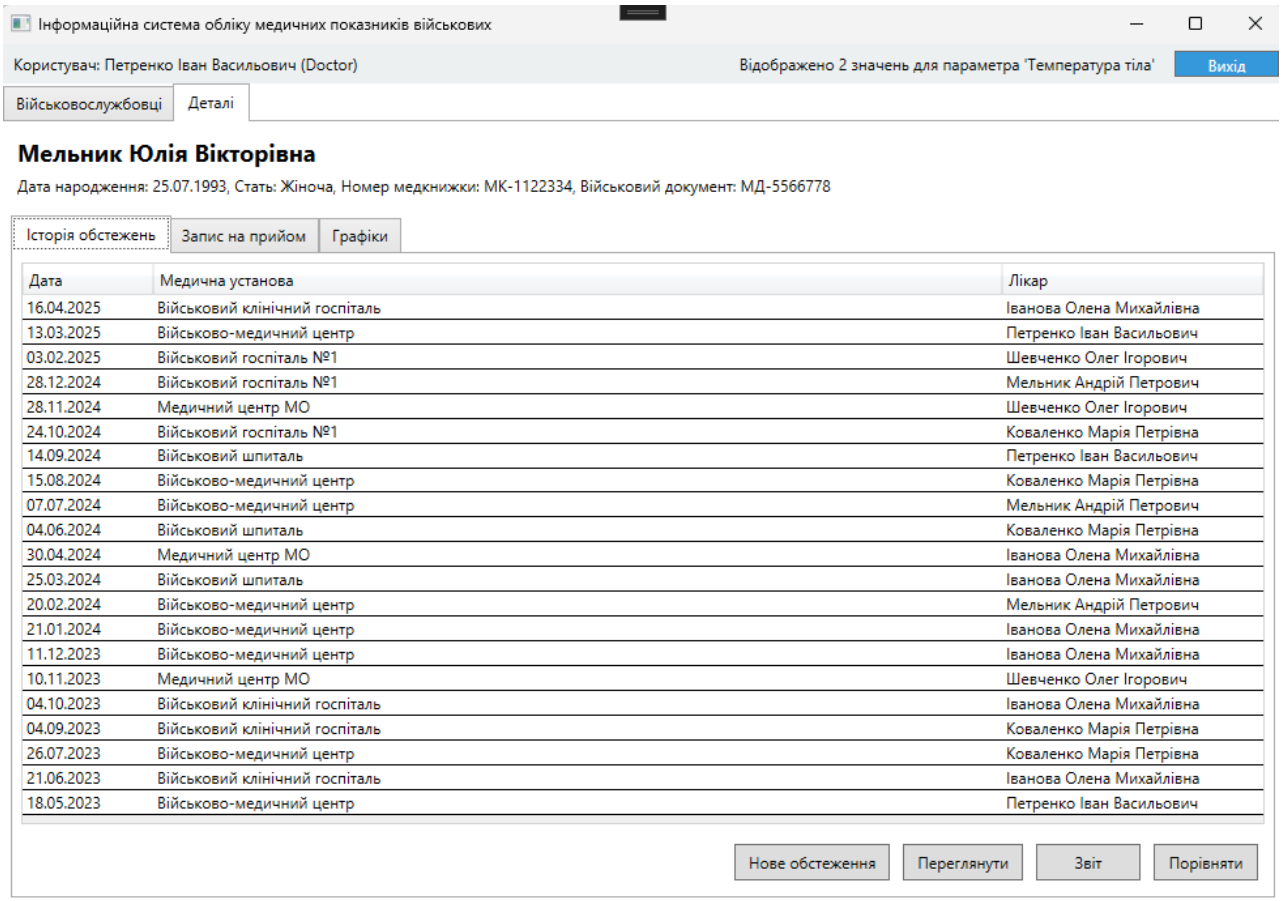

Рис. 4.6 Історія обстежень обраного військовослужбовця

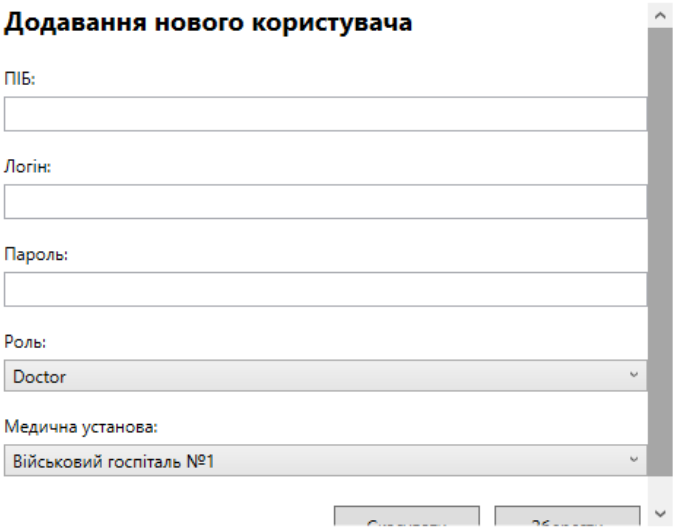

Рис. 4.7 Додавання нового користувача адміністратором

– Крок 4. Виконання спеціалізованих завдань

На цьому етапі користувач виконує цільові задачі залежно від своїх потреб та ролі. Лікар, наприклад, може додати нове обстеження (рис. 4.8), обравши відповідну медичну установу та заповнивши результати вимірювань у зручному форматі з вкладками за категоріями, такими як серцево-судинна система чи лабораторні показники, або переглянути вже наявні обстеження (рис. 4.9). Система перевіряє коректність даних, наприклад, унікальність номерів документів чи заповненість обов'язкових полів, і зберігає інформацію у базі. Для порівняння стану здоров'я програма дозволяє обрати два обстеження одного військовослужбовця та згенерувати звіт із детальним аналізом змін. Адміністратор, у свою чергу, може створювати резервні копії бази даних для захисту інформації або відновлювати її з попереднього збереження, що є важливим для безперебійної роботи, також може додавати новий параметр для обстежень (рис. 4.10) Усі ці дії супроводжуються повідомленнями про успішність або помилки, що допомагає користувачу орієнтуватися у процесі.

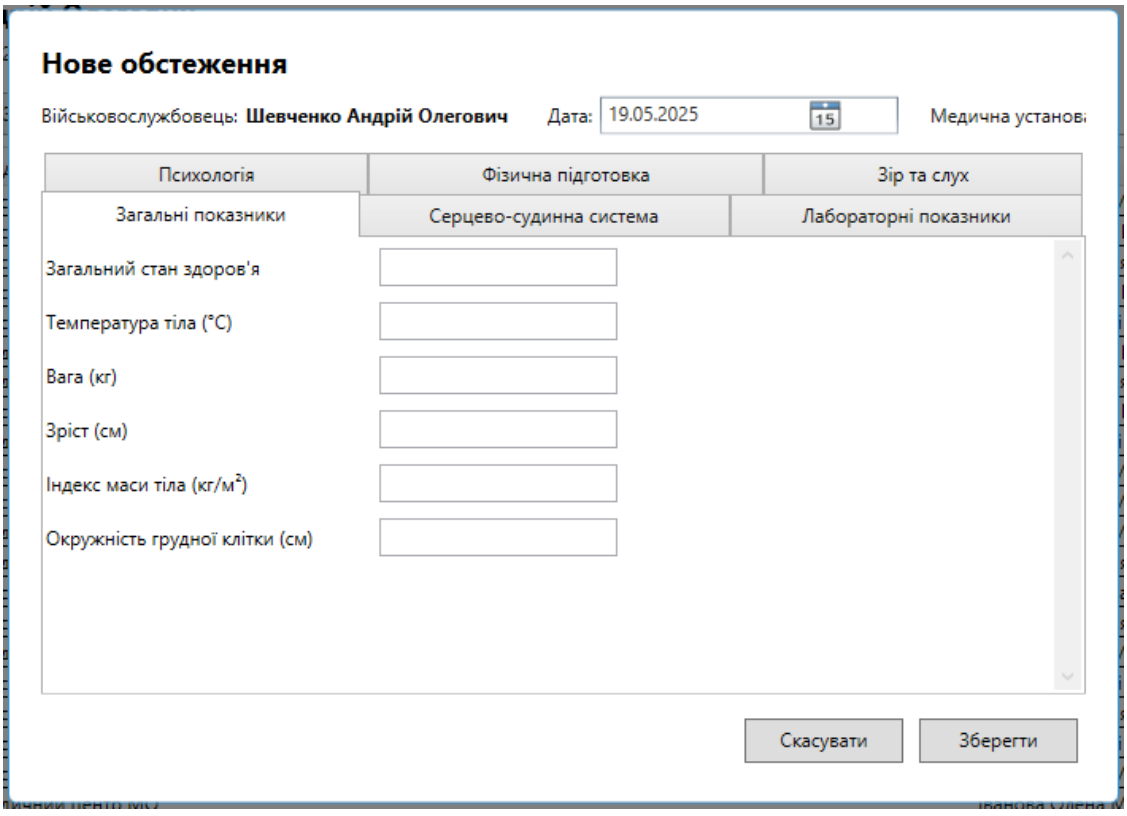

Рис. 4.8 Додавання нового обстеження

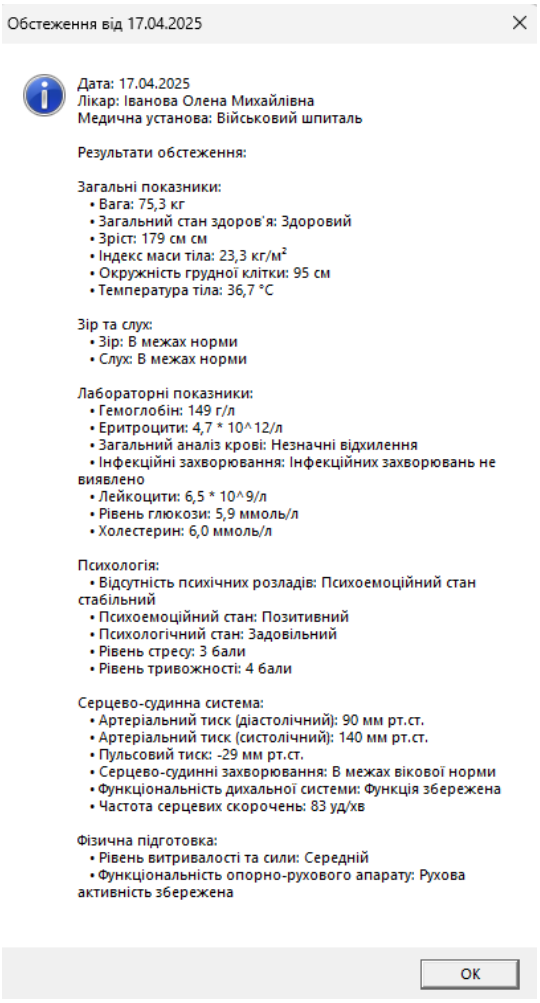

Рис. 4.9 Перегляд обстеження військовослужбовця

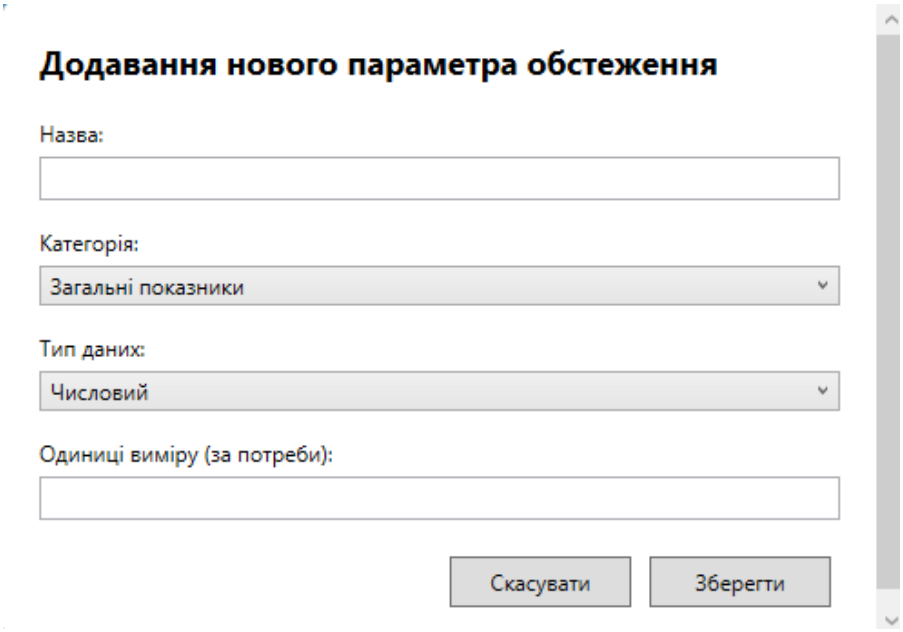

Рис. 4.10 Додавання нового параметра обстеження

– Крок 5. Візуалізація та аналіз даних

Однією з ключових можливостей системи є аналіз медичних даних, що реалізується через інтеграцію бібліотеки LiveCharts. Користувач може обрати параметр, наприклад, вагу, та період часу для побудови графіка, який відображає динаміку показника (рис. 4.11). Це дозволяє лікарям виявляти тенденції, такі як погіршення чи покращення стану здоров'я, і приймати обґрунтовані рішення. Графіки оновлюються автоматично при зміні параметрів, а їх вигляд адаптовано для зручного сприйняття.

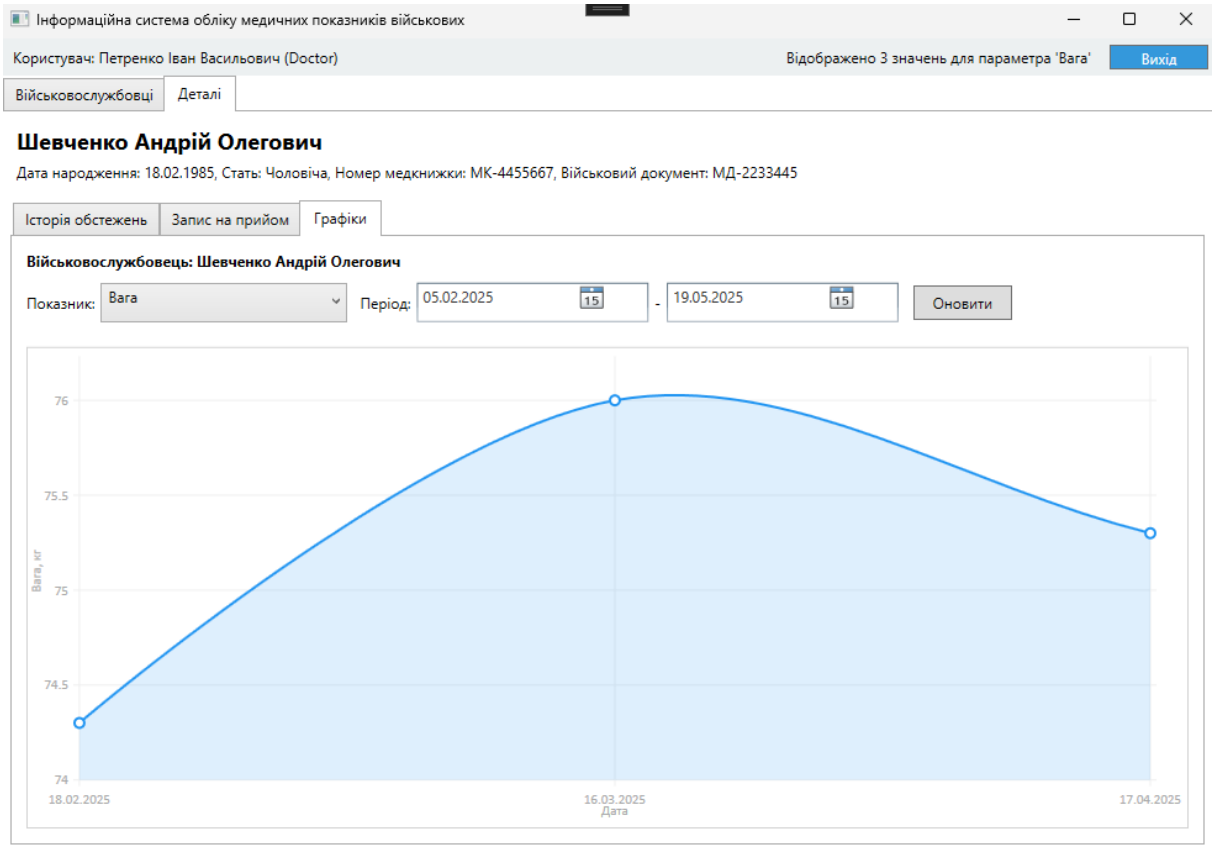

Рис. 4.11 Деталі обстежень військовослужбовця у вигляді графіків

Також система пропонує генерацію звітів у форматі PDF, які включають детальну інформацію про обстеження чи порівняння кількох обстежень (рис. 4.12, рис. 4.13, рис. 4.14, рис. 4.15, рис. 4.16, рис. 4.17). Ці звіти створюються з

урахуванням кириличних символів, що забезпечує їх читабельність та відповідність стандартам документації.

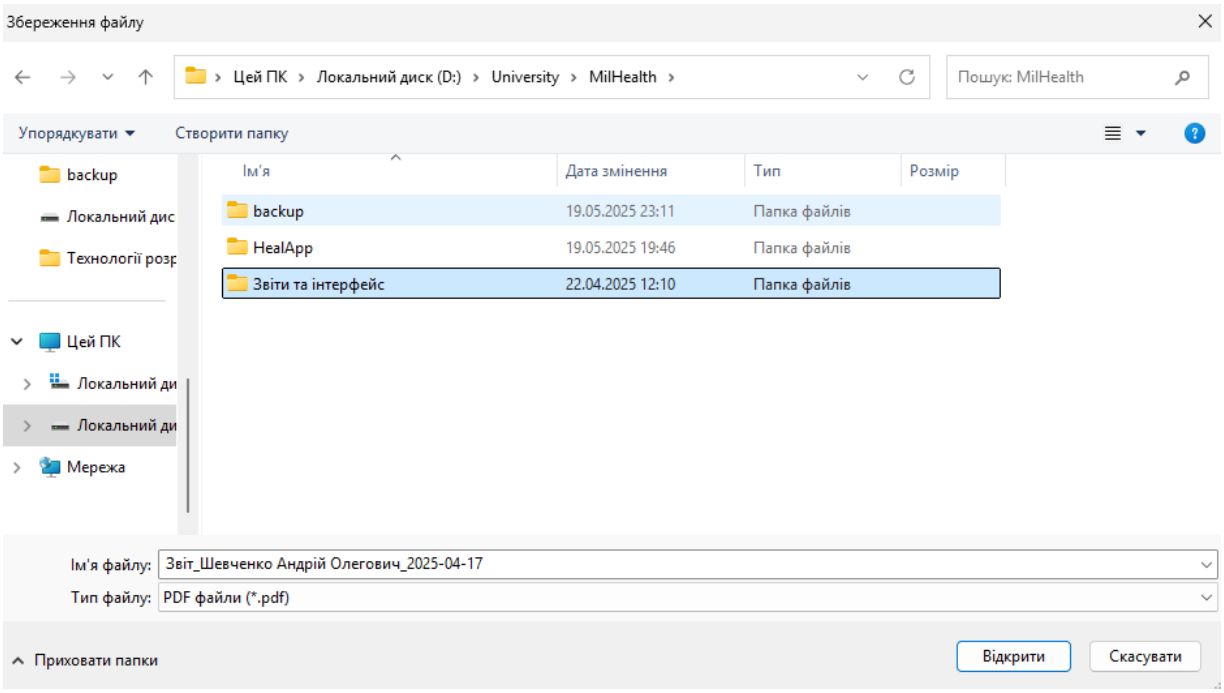

Рис. 4.12 Збереження створеного звіту

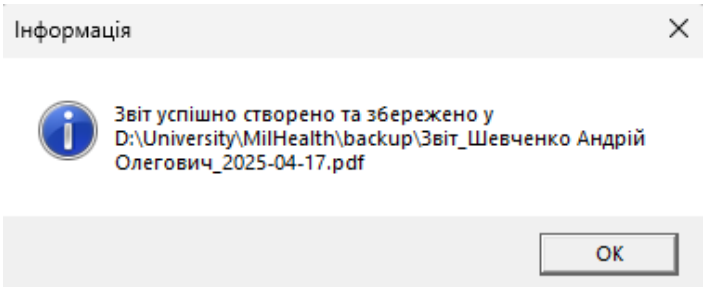

Рис. 4.13 Підтвердження створення звіту

МЕДИЧНИЙ ВИСНОВОК

Медична установа

Назва:	Військовий шпиталь
Адреса:	м. Одеса, вул. Приморська, 10

Лікар

ПІБ:	Іванова Олена Михайлівна

Військовослужбовець

ПІБ:	Шевченко Андрій Олегович
Дата народження:	18.02.1985
Стать:	Чоловіча
Військ.-обліковий №:	МД-2233445
№ мед. книжки:	МК-4455667

Результати обстеження

Загальні показники	
Вага	75,3 кг
Загальний стан здоров'я	Здоровий
Зріст	179 см см
Індекс маси тіла	23,3 кг/м²
Окружність грудної клітки	95 см
Температура тіла	36,7 °C

Рис. 4.14 Перегляд створеного звіту

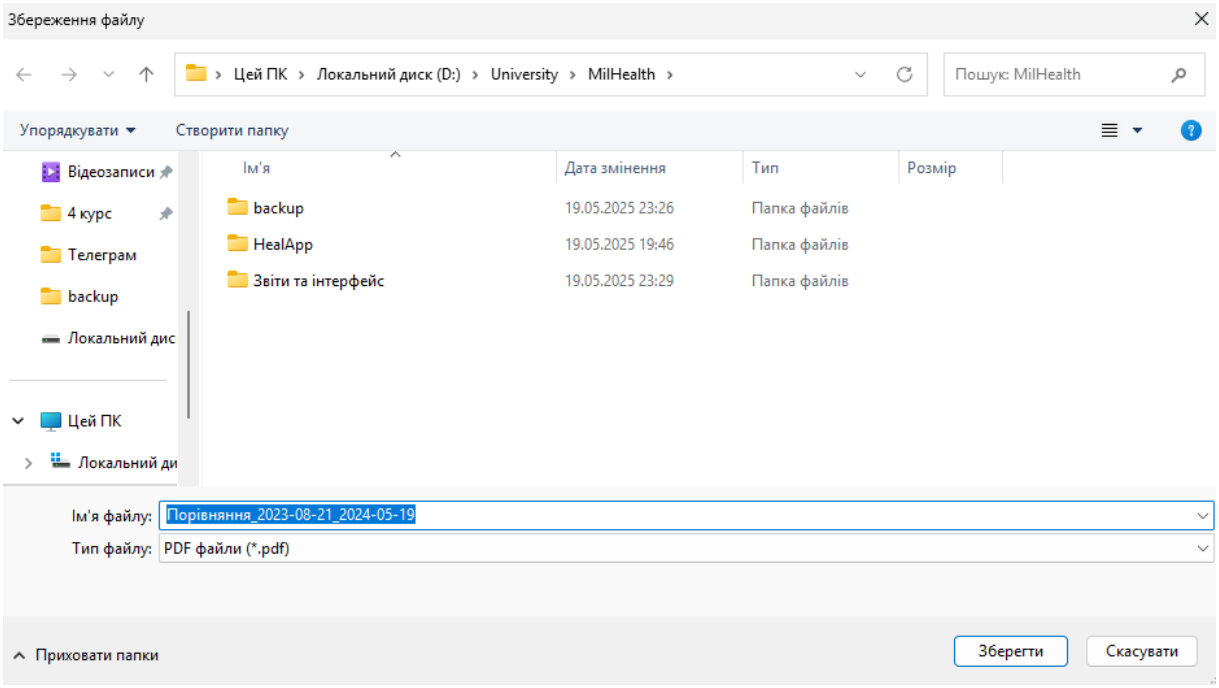

Рис. 4.15 Збереження порівняльного звіту

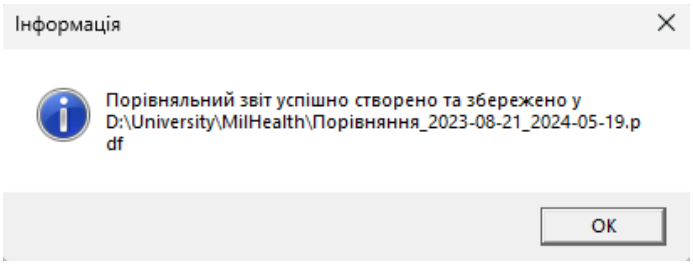

Рис. 4.16 Підтвердження створення порівняльного звіту

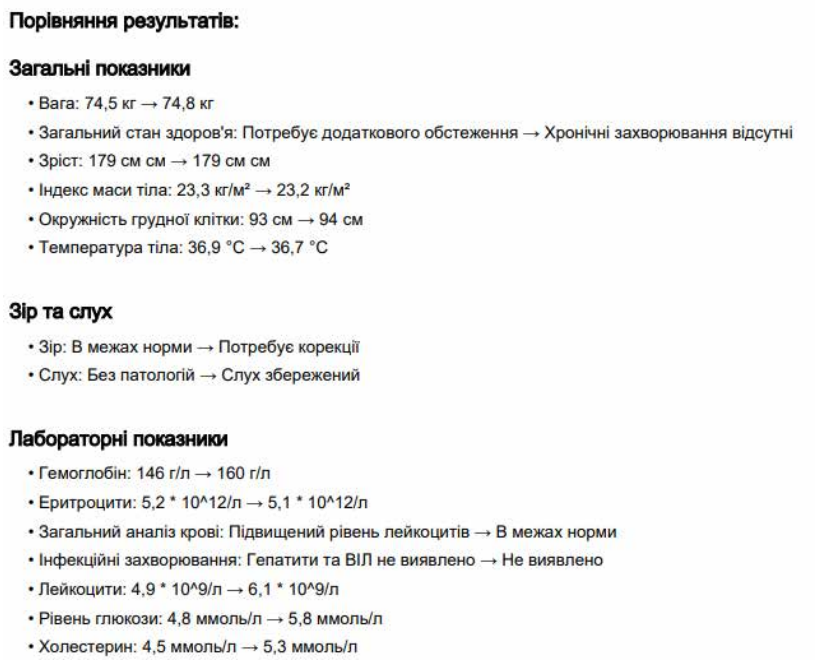

Рис. 4.17 Перегляд порівняльного звіту

– Крок 6. Завершення сеансу роботи

Коли користувач завершує роботу, він може вийти з системи, натиснувши кнопку "Вихід", що призводить до фіксації дії у журналі та повернення до вікна авторизації. Усі введені дані зберігаються у базі, а тимчасові змінні, такі як обраний військовослужбовець чи обстеження, скидаються для забезпечення конфіденційності. Якщо ж користувач залишає програму неактивною протягом 10 хвилин, спрацьовує таймер неактивності, який автоматично виконує вихід із системи, захищаючи дані від несанкціонованого доступу. Завершення роботи програми відбувається також при закритті вікна, коли всі ресурси, включно з підключенням до бази даних, коректно звільняються, що гарантує стабільність системи при наступних запусках.

– Крок 7. Завершення роботи програми

Остаточне завершення роботи програми відбувається, коли користувач закриває додаток або коли система перезапускається, наприклад, після відновлення бази даних. На цьому етапі всі активні процеси, такі як таймер неактивності чи підключення до бази, завершаються. Програма забезпечує коректне збереження всіх змін, внесених під час сеансу, у базі даних, що

виключає втрату інформації. Завдяки архітектурі, побудованій на принципах MVVM, та використанню Entity Framework Core, система залишається стабільною навіть при частих запусках і завершеннях. Після закриття програми користувач може бути впевнений, що всі його дії збережено, а доступ до даних надійно захищено до наступного входу.

4.3 Вимоги до апаратного та програмного забезпечення

Інформаційна система для обліку медичних показників військовослужбовців, реалізована в рамках додатку "MedicalApp", та представляє собою настільне програмне рішення, яке передусім розроблене для використання в медичних установах. Для забезпечення стабільної роботи системи, її ефективного впровадження та експлуатації необхідно чітко визначити вимоги до апаратного та програмного забезпечення. Цей підрозділ розкриває топологію системи, а також надай детальну інформацію про вимоги до апаратного й програмного забезпечення, враховуючи особливості її архітектури та функціоналу.

Система "MedicalApp" побудована за клієнт-серверною архітектурою, де клієнтська частина представлена настільним додатком WPF, а серверна частина може використовуватися в локальному та віддаленому варіантах, для забезпечення роботи з базами даних Microsoft SQL Server, Microsoft SQL Server Local DB. Для наочності топології системи пропонується наступна схема розміщення:

1) Клієнтський вузол:

Тип: персональний комп’ютер або ноутбук медичного персоналу (лікарів або адміністраторів).

Компоненти:

– Настільний додаток "MedicalApp" (MainWindow.xaml та пов’язані файли).

– Локальний кеш даних для тимчасового зберігання введених параметрів.

Місцезнаходження: робочі місця медичних працівників у медичних установах (кабінети лікарів, адміністраторські приміщення).

Функції: взаємодія користувача з інтерфейсом, введення даних, перегляд медичних карток, генерація звітів у PDF, побудова графіків за допомогою бібліотеки LiveCharts.

2) Серверний вузол:

В залежності від масштабів впровадження, серверний вузол може функціонувати в двох варіантах, локально та віддалено.

- Варіант 1. Локальне розгортання (Local DB):

Тип: локальна база даних, розгорнута на тому ж комп’ютері, де встановлено клієнтський додаток.

Компоненти:

– Microsoft SQL Server LocalDB з базою даних "MilHealthDB".

– Схема даних, визначена через Entity Framework Core (Code First).

Місцезнаходження: фізично на клієнтському комп’ютері або на окремому локальному сервері в межах однієї медичної установи (за наявності централізованого розгортання).

- Варіант 2. Віддалене розгортання (SQL Server):

Тип: віддалена база даних розгорнута на окремому сервері.

Компоненти:

– Microsoft SQL Server 2019/2022 Standard або Enterprise з базою даних "MilHealthDB".

– Сервіс синхронізації та реплікації даних.

– Система резервного копіювання та відновлення.

Місцезнаходження: серверна кімната медичного закладу або центральний дата-центр військово-медичної служби.

Функції для обох варіантів залишаються однаковими, серед них: зберігання даних про військовослужбовців, медичні обстеження, призначення, користувачів, журнали системи; обробка запитів від клієнтської частини.

3) Зв’язок між вузлами.

- Локальне розгортання (Local DB):

Тип зв’язку: локальне підключення через ADO.NET та Entity Framework Core.

Протокол: внутрішній протокол SQL Server для доступу до бази даних.

Особливості: оскільки SQL Server LocalDB працює в межах однієї машини, мережевий обмін даними мінімізується, що знижує вимоги до пропускної здатності мережі.

Топологія є децентралізованою, оскільки кожен клієнтський вузол містить власну копію бази даних. У разі централізованого розгортання (наприклад, у великій медичній установі) серверний вузол може бути винесений на окремий фізичний сервер із підключенням через локальну мережу.

- Віддалене розгортання (SQL Server):

Тип зв’язку: мережеве підключення через ADO.NET та Entity Framework Core.

Протокол: TCP/IP-протокол SQL Server для мережевого доступу до централізованої бази даних.

Серверний вузол: окрема фізична або віртуальна машина, на якій встановлений SQL Server. Він зберігає централізовану базу даних MilHealthDB.

Особливості: для забезпечення безпеки з’єднання може використовуватись шифрування та автентифікація SQL Server. Якість мережевого з’єднання впливає на продуктивність доступу до даних, тому рекомендовано використання стабільного Ethernet-з'єднання або швидкої VPN-мережі. Важливо налаштувати правила брандмауера для безпосереднього доступу до порту SQL Server (типово 1433).

Топологія є централізована- усі клієнтські вузли підключаються до єдиного серверного вузла з SQL Server по локальній мережі (LAN) або через VPN.

Для забезпечення стабільної роботи системи необхідно врахувати мінімальні та рекомендовані апаратні характеристики клієнтських і серверних вузлів які наведено у табл. 4.3.

Таблиця 4.3

Апаратні характеристики клієнтських і серверних вузлів

Вузол	Характеристика	Мінімум	Рекомендується
Клієнтський вузол (персональний комп’ ютер/ноутбук)	Процесор	Двоядерний процесор із частотою 2 ГГц (наприклад, Intel Core i3 7-го покоління або аналогічний AMD)	Чотириядерний процесор із частотою 3 ГГц або вище (Intel Core i5 10-го покоління або аналогічний)
	Оперативна пам’ять	4 ГБ для базового функціоналу (автентифікація, введення даних, перегляд)	8 ГБ або більше для комфортної роботи з графіками та генерацією звітів
Клієнтський вузол (персональний комп’ ютер/ноутбук) комп’ ютер/ноутбук)	Накопичувач	10 ГБ вільного місця на HDD для встановлення додатку, бібліотек та бази даних	SSD із 20 ГБ вільного місця для швидшого доступу до даних
	Відеокарта	Інтегрована графіка (Intel HD Graphics 620 або аналогічна)	Дискретна відеокарта для плавного рендерингу графіків LiveCharts
	Монітор	Роздільна здатність 1280x720 пікселі	Full HD (1920x1080) для зручного розташування елементів інтерфейсу
	Периферія	Клавіатура, миша, принтер	

Таблиця 4.3 (закінчення)

Вузол	Характеристика	Мінімум	Рекомендується
Серверний вузол (якщо база даних винесена на окремий сервер)	Процесор	Двоядерний процесор із частотою 2.5 ГГц	Чотириядерний процесор із частотою 3.5 ГГц
	Оперативна пам’ять	8 ГБ	16 ГБ для обробки великих обсягів даних
	Накопичувач	50 ГБ на HDD	SSD із 100 ГБ для швидшого виконання запитів
	Мережевий адаптер (для централізованого розгортання)	100 Мбіт/с	1 Гбіт/с для стабільного доступу кількох клієнтів

Програмне забезпечення, яке потребує система для коректної роботи, включає: операційну систему, фреймворки, бібліотеки та інструменти для роботи з базою даних.

Операційна система:

– Мінімум: Windows 10 (версія 1803 або новіша, 64-біт).

– Рекомендується: Windows 11 (64-біт) для оптимальної підтримки .NET 6.0 та сучасних бібліотек.

– Примітка: Система не підтримує інші операційні системи (Linux, macOS) через залежність від SQL Server LocalDB та WPF.

Фреймворк:

– Обов’язково: .NET 6.0 Runtime (або новіша версія).

– Для розробки/налагодження: .NET 6.0 SDK.

Система керування базами даних:

– Обов’язково: Microsoft SQL Server LocalDB (версія 2019 або новіша).

– Рекомендується: Microsoft SQL Server Management Studio (SSMS) для адміністрування бази даних у разі ручного керування.

Бібліотеки та залежності:

– Entity Framework Core: Для ORM та роботи з базою даних.

– LiveCharts.Wpf: Для побудови графіків (автоматично встановлюється через NuGet).

– iTextSharp: Для генерації PDF-звітів із підтримкою кирилиці.

– Інші залежності: Встановлюються автоматично через NuGet (наприклад, Microsoft.EntityFrameworkCore.SqlServer).

Середовище розробки (для підтримки та модифікації):

– Рекомендується: Visual Studio 2019 (Community, Professional або Enterprise, версія 16.11+) або Visual Studio 2022 із підтримкою WPF та .NET 6.0.

– Інструменти: NuGet Package Manager для встановлення залежностей.

Додаткове ПЗ:

– PDF-переглядач: Наприклад, Adobe Acrobat Reader для відкриття згенерованих звітів.

– Браузер (опціонально): Для документації або доступу до зовнішніх ресурсів.

Для забезпечення безпеки та стабільності роботи системи:

– Регулярно оновлюйте .NET Runtime та SQL Server LocalDB до останніх версій.

– Налаштуйте автоматичне резервне копіювання бази даних через функцію BackupDatabaseButton_Click.

– Використовуйте антивірусне ПЗ для захисту від шкідливих програм.

– Обмежте фізичний доступ до клієнтських комп’ютерів, щоб уникнути несанкціонованого доступу до даних.

Таким чином, дотримання зазначених вимог до апаратного та програмного забезпечення забезпечить надійну роботу системи, її швидкодію та зручність використання для медичного персоналу.

4.4 Склад інсталяційного пакету

Для забезпечення коректного функціонування інформаційної системи обліку медичних показників військовослужбовців необхідно правильно підготувати та розгорнути інсталяційний пакет. Інсталяційний пакет включає набір файлів і компонентів, які забезпечують роботу додатка, враховуючи архітектурні особливості системи та вимоги до її розміщення. Чітке визначення складу інсталяційного пакету дозволяє уникнути помилок під час розгортання та забезпечити стабільну експлуатацію системи. Детальний опис компонентів інсталяційного пакету, сформований на основі структури проєкту, наданий у табл. 4.4.

Таблиця 4.4

Опис компонентів інсталяційного пакету

№	Компоненти	Опис
1	Виконуваний файл програми та основні бібліотеки	Інсталяційний пакет містить основний виконуваний файл програми (MedicalApp.exe), який є точкою входу для запуску додатка. Разом із ним включаються бібліотеки (.dll), що відповідають за базову функціональність, такі як обробка інтерфейсу користувача, логіка програми та взаємодія з базою даних. Ці файли формуються під час компіляції проєкту та розміщуються в директорії bin
2	Файли конфігурації	До складу пакету входять конфігураційні файли, зокрема App.config, які містять налаштування підключення до бази даних (наприклад, рядок підключення до SQL Server LocalDB, ConnectionString.txt) та інші параметри роботи програми. Ці файли дозволяють адаптувати систему до конкретного середовища без необхідності зміни вихідного коду
3	Ресурси інтерфейсу користувача	Пакет включає XAML-файли (наприклад, MainWindow.xaml, ResourceDictionary.xaml) та пов’язані з ними ресурси, такі як стилі, кольорові схеми та шаблони, які визначають вигляд і поведінку графічного інтерфейсу. Ці файли забезпечують адаптивність і зручність взаємодії користувача з системою
4	Залежності від зовнішніх бібліотек	Система використовує зовнішні бібліотеки, такі як Entity Framework Core для роботи з базою даних, LiveCharts для візуалізації графіків і iTextSharp для генерації PDF-звітів. Інсталяційний пакет містить відповідні NuGet-пакети або їх зібрані версії, які автоматично розгортаються разом із програмою. Для уникнення конфліктів версій передбачено використання менеджерів пакетів під час встановлення

Таблиця 4.4 (закінчення)

№	Компоненти	Опис
5	Скрипти ініціалізації бази даних	Для створення структури бази даних і заповнення її початковими даними до пакету включено класи та методи ініціалізації (наприклад, DatabaseSeed.cs). Ці компоненти забезпечують автоматичне створення бази даних при першому запуску, а також можливість генерації демонстраційних даних для тестування
6	Документація та інструкції	Інсталяційний пакет супроводжується документацією, такою як README.md, яка містить інструкції щодо встановлення, конфігурації та запуску програми. Документація включає інформацію про системні вимоги, налаштування середовища та облікові дані для тестового доступу (наприклад, логін і пароль адміністратора)
7	Файли резервного копіювання та відновлення	Для забезпечення безпеки даних пакет може містити інструменти для створення резервних копій бази даних (.bak файли) та їх відновлення. Ці компоненти дозволяють адміністраторам захистити інформацію від втрати та швидко відновити систему у разі збоїв
8	Додаткові утиліти	До пакету можуть входити утиліти для перевірки цілісності встановлення, оновлення бази даних або діагностики проблем. Наприклад, скрипти для перевірки наявності встановленого .NET Framework, SQL Server чи LocalDB

Інсталяційний пакет формується з урахуванням діаграми розміщення (рис. 4.18), яка визначає, де саме компоненти системи мають бути розташовані на клієнтському комп’ютері [38]. Усі файли розміщуються в чітко структурованих директоріях, щоб надати легкий доступ і підтримку. Наприклад, виконувані файли та бібліотеки розташовуються в основній папці програми, тоді як файли бази даних розміщуються в директорії визначеній для SQL Server LocalDB або на окремому сервері, визначеному для SQL Server (рис. 4.19). Такий підхід гарантує портативність і масштабованість системи.

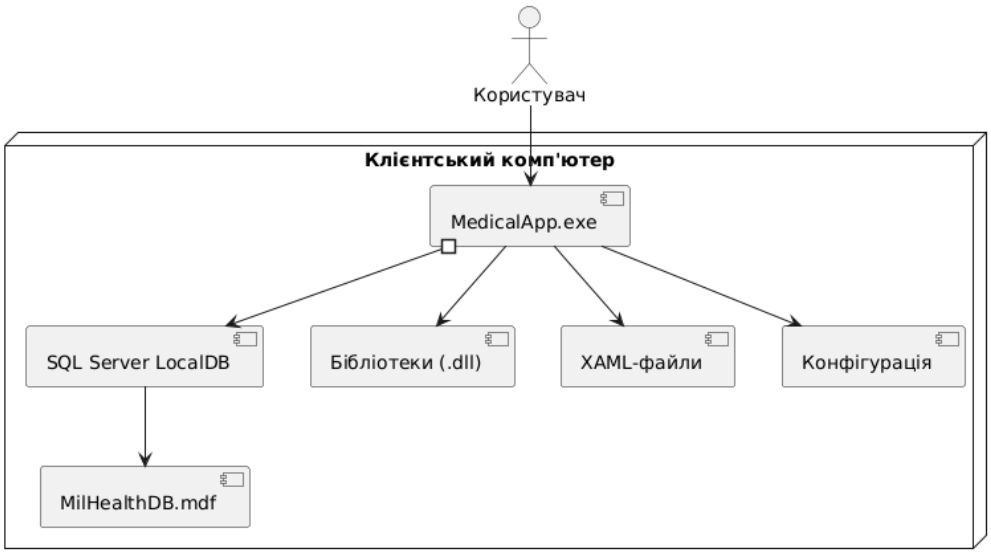

Рис 4.18 Діаграма розміщення (локальне розгортання)

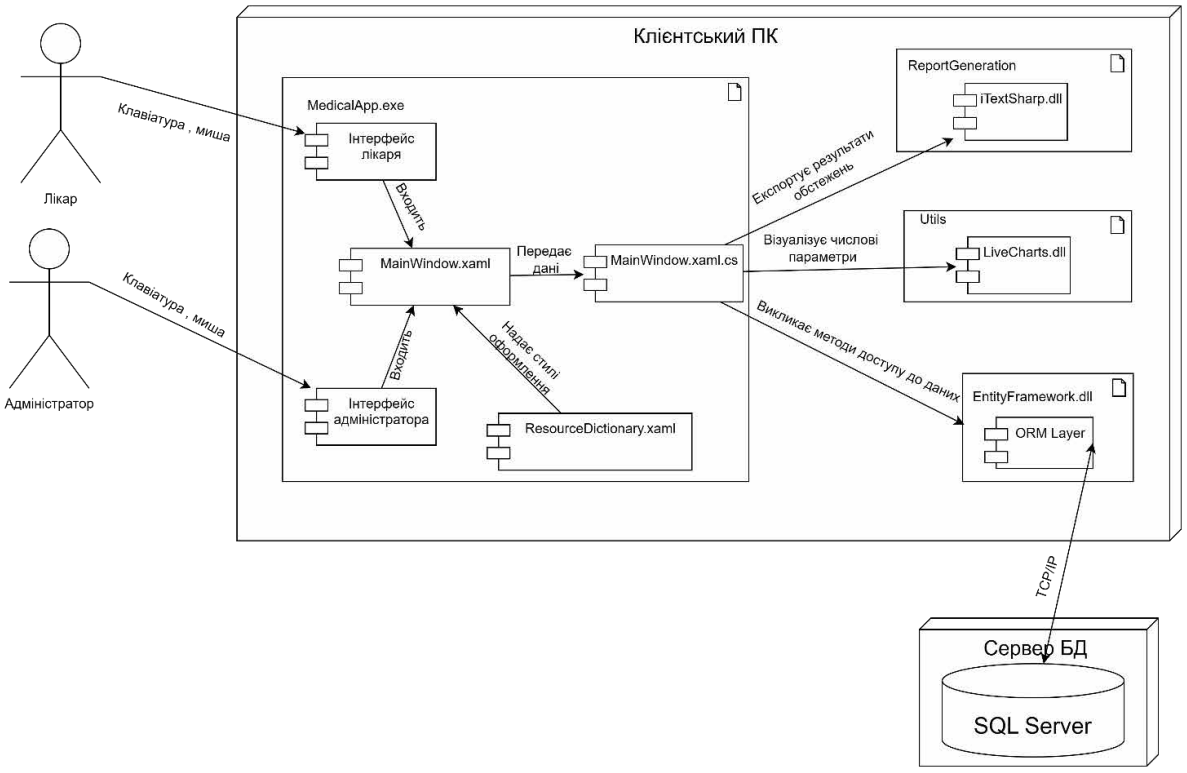

Рис 4.19 Діаграма розміщення (віддалене розгортання)

Діаграми розміщення відображають архітектуру розгортання системи, показуючи взаємодію клієнтського додатка як з локальною, так і з віддаленою базами даних на комп’ютері користувача.

Інсталяційний пакет інформаційної системи обліку медичних показників військовослужбовців включає всі необхідні компоненти для коректного розгортання та функціонування програми, забезпечуючи стабільну роботу та зручність використання. Чітка структура пакету, що охоплює виконувані файли, бібліотеки, конфігурації, ресурси та документацію, гарантує легке встановлення і підтримку системи. Діаграма розміщення відображає оптимальну організацію компонентів для ефективної взаємодії клієнтського додатка з базою даних.

Висновок до розділу 4

Розробка інформаційної системи обліку медичних показників військовослужбовців завершила всі необхідні етапи, включаючи тестування та підготовку до розгортання. Система демонструє високу готовності до експлуатації, що було підтверджено результатами проведеного тестування. Вона відповідає основним вимогам щодо функціональності, продуктивності та безпеки, що робить передусім надійний інструмент для роботи медичного персоналу. Для відображення роботи програми, було продемонстровано основні кроки виконання - від запуску програми до завершення роботи програми.

Важливим аспектом стало визначення оптимальних вимог до апаратного та програмного забезпечення. Дана система демонструє гнучкість у розгортанні, що дозволяє медичним установам використовувати її для різних умов, як локально, так і через мережеві конфігурації.

Останній етап - це підготовка інсталяційного пакету, який включає всі необхідні компоненти для швидкого та безпомилкового розгортання. Насамперед, чітка структура пакету та детальна документація може значно спростити процес впровадження. Система готова до експлуатації та має хороший потенціал для подальшого розвитку.

ВИСНОВКИ

У процесі виконання роботи мені вдалося повністю досягти головної мети-розробити ефективне програмне рішення для автоматизації обліку медичних показників військовослужбовців. Виконавши усі важливі функції, передбачені на етапі планування.

Розробка інформаційної системи для обліку індивмедичних показників військовослужбовців, виконана на базі платформи WPF з використанням C#, .NET Framework та Entity Framework Core, успішно вирішує завдання автоматизації медичного обліку, забезпечуючи централізоване управління даними про стан здоров'я. Система дозволяє вести електронні медичні картки, фіксувати результати обстежень, генерувати PDF-звіти, візуалізувати динаміку показників через графіки та виконувати адміністративні функції, такі як управління користувачами і резервне копіювання даних. Отримані результати відповідають поставленим вимогам, забезпечуючи інтуїтивно зрозумілий інтерфейс, підтримку української локалізації та високу безпеку завдяки автентифікації, розподілу ролей і автоматичному виходу після 10 хвилин бездіяльності.

У порівнянні з аналогами, такими як MHS GENESIS чи OpenEMR, розроблена система є економічно вигіднішою, оскільки базується на безкоштовній SQL Server LocalDB і не потребує значних витрат на розгортання. Водночас система поступається комерційним рішенням за масштабованістю, оскільки орієнтована на локальне використання, що є незначним недоліком у контексті цільового застосування. Однак за необхідності може бути перебазованою на потужнішу централізовану СУБД SQL Server.

Новизна роботи полягає у створенні спеціалізованого рішення, адаптованого до потреб військової медицини України, з урахуванням специфічних параметрів, таких як фізична підготовка та психологічний стан, що рідко зустрічаються в універсальних медичних системах. Я використала

унікальну комбінацію технологій: інтеграцію LiveCharts для динамічної візуалізації показників і iTextSharp для генерації кириличних звітів. Ці рішення полегшують аналіз та підвищують практичну корисність системи. Переваги розробки включають модульну архітектуру MVVM, яка спрощує підтримку та дозволяє безпосереднє розширення функціоналу, а також оптимізовану роботу з базою даних через Entity Framework Core, що гарантує швидкий доступ до інформації.

Техніко-економічна ефективність системи проявляється у зниженні витрат часу на обробку медичних даних, автоматизації рутинних операцій і підвищенні точності моніторингу здоров'я військовослужбовців, що сприяє оперативному прийняттю медичних рішень. Система може використовуватися у військових госпіталях, медичних центрах та інших установах, де потрібен облік здоров'я особового складу. Усі ключові вимоги виконано: від підтримки авторизації до генерації аналітичних звітів, хоча подальші вдосконалення можуть включати інтеграцію з хмарними технологіями для розподіленої роботи.

Рекомендується впровадити систему в тестовому режимі у військових медичних установах для оцінки її ефективності в реальних умовах, а також розглянути можливість додавання модуля інтеграції зі стандартом HL7 FHIR, що зможе підвищити сумісність з іншими медичними платформами. Подальший розвиток може передбачати створення мобільної версії для оперативного доступу до даних у польових умовах, що підвищить універсальність системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Torab-Miandoab A., Basiri M., Moghaddam A. D., Gholamhosseini L. Electronic health record in military healthcare systems: A systematic review // PLOS One. – 2025. – Vol. 20, №2. – DOI: 10.1371/journal.pone.0313641.
2. HL7® is the authority on HL7 standards and the home of the next-generation standard FHIR! – [Електронний ресурс]. – Режим доступу: https://info.hl7.org/learn-more
3. Achanta A., Bo R. The Role of Data Visualization in Healthcare Analytics. – 2023.
4. MHS GENESIS: The Electronic Health Record. – [Електронний ресурс]. – Режим доступу: https://www.health.mil/Military-Health-Topics/Technology/MHS-GENESIS
5. New Facts You Need to Know About Military Electronic Medical Records. – [Електронний ресурс]. – Режим доступу: https://epicgovernment.com/news/military-electronic-medical-records-mhs-genesis-2023-insights/
6. Electronic Health Records: DOD Has Deployed New System but Challenges Remain. – [Електронний ресурс]. – Режим доступу: https://www.gao.gov/products/gao-24-106187
7. Що таке електронна медична картка. – [Електронний ресурс]. – Режим доступу: www.umj.com.ua/uk/novyna-245190-shho-take-elektronna-medichna-kartka
8. Armed Forces Health Longitudinal Technology Application. – [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Armed_Forces_Health_Longitudinal_Technology_Application

9. Electronic Health Record Modernization (EHRM). – [Електронний ресурс]. – Режим доступу: https://www.va.gov/lovell-federal-health-care-va/programs/electronic-health-record-modernization-ehrm/
10. health.mil. – [Електронний ресурс]. – Режим доступу: https://www.health.mil/Military-Health-Topics/Technology/MHS-GENESIS
11. OpenEMR. – [Електронний ресурс]. – Режим доступу: https://www.open-emr.org/
12. MedTrak Systems. – [Електронний ресурс]. – Режим доступу: https://www.medtraksystems.com/main/
13. TrakCare: EHR Built for Evolving Healthcare. – [Електронний ресурс]. – Режим доступу: https://www.intersystems.com/products/trakcare/
14. Уніфікована мова моделювання (Unified Modeling Language – UML). – [Електронний ресурс]. – Режим доступу: https://www.maxzosim.com/unifikovana-mova-modeluvannia/
15. Що таке діаграма класів UML і найкращий творець діаграм класів UML. – [Електронний ресурс]. – Режим доступу: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/
16. What is Use Case Diagram? – [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
17. Дізнайтеся все про діаграму активності UML [з методами]. – [Електронний ресурс]. – Режим доступу: https://www.mindonmap.com/uk/blog/uml-activity-diagram/
18. Знайомство з контекстною діаграмою та чудовим програмним забезпеченням для легкого створення. – [Електронний ресурс]. – Режим доступу: https://www.mindonmap.com/uk/blog/context-diagram/
19. Tworzenie parametrów połączenia i praca z bazą danych SQL Server LocalDB. – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/pl-pl/aspnet/mvc/overview/getting-started/introduction/creating-a-connection-string

20. Przewodnik dotyczący aplikacji klasycznych (WPF .NET). – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/pl-pl/dotnet/desktop/wpf/overview/?view=netdesktop-9.0
21. MVC проти MVVM – різниця між ними. – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/uk/mvc-vs-mvvm.html
22. Модель діаграми зв’язків сутностей (ER) із прикладом СУБД. – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html
23. Модель «сутність-зв'язок». Основні понятння – [Електронний ресурс]. – Режим доступу: https://wiki.cusu.edu.ua/index.php/Модель_«сутність_–_зв’язок»._Основні_поняття_моделі_«сутність_–_зв’язок»:_сутності,_зв’язки,_атрибути_та_їх_класифікація.
24. Нормальні форми бази даних. – [Електронний ресурс]. – Режим доступу: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture02
25. Microsoft SQL Server LocalDB. – [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/help/datagrip/connecting-to-sql-server-express-localdb.html
26. Класифікація баз даних. – [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/10091086/page:2/
27. Типи баз даних. – [Електронний ресурс]. – Режим доступу: https://it-osvita.diia.gov.ua/task/item/2ce93257-df68-4072-bca6-65163b9f4278
28. Entity Framework Core. – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/
29. Що таке файл XAML? – [Електронний ресурс]. – Режим доступу: https://docs.fileformat.com/uk/web/xaml/
30. Як використовувати LINQ у C#: керівництво з прикладами. – [Електронний ресурс]. – Режим доступу: https://foxminded.ua/robota-z-linq-v-c-sharp/
31. Діаграма пакетів. – [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Діаграма_пакетів

32. Про середовище програмування C#. – [Електронний ресурс]. – Режим доступу: https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/
33. Microsoft Visual Studio: що це таке і для чого це потрібно. – [Електронний ресурс]. – Режим доступу: https://macrosoft.store/uk/blog/post/29-для-чого-потрібна-microsoft-visual-studio
34. Windows Presentation Foundation (WPF) Лекція 06. – [Електронний ресурс]. – Режим доступу: https://learn.ztu.edu.ua/pluginfile.php/403111/mod_resource/content/2/ООП.%20Лекція%2006.pdf
35. Entity Framework Core Базовий. – [Електронний ресурс]. – https://itvdn.com/ua/video/ef-core-essential
36. C# Winform LiveCharts does not know how to plot. – [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/68364317/c-sharp-winform-livecharts-does-not-know-how-to-plot
37. ITextSharp 4.1.6 extract PDF content as text. – [Електронний ресурс]. – Режим доступу: https://stackoverflow.com/questions/55391515/itextsharp-4-1-6-extract-pdf-content-as-text
38. Діаграма розгортання: Підручник з UML із ПРИКЛАДОМ. – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/uk/deployment-diagram-uml-example.html

Додаток А

ДІАГРАМА АКТИВНОСТІ

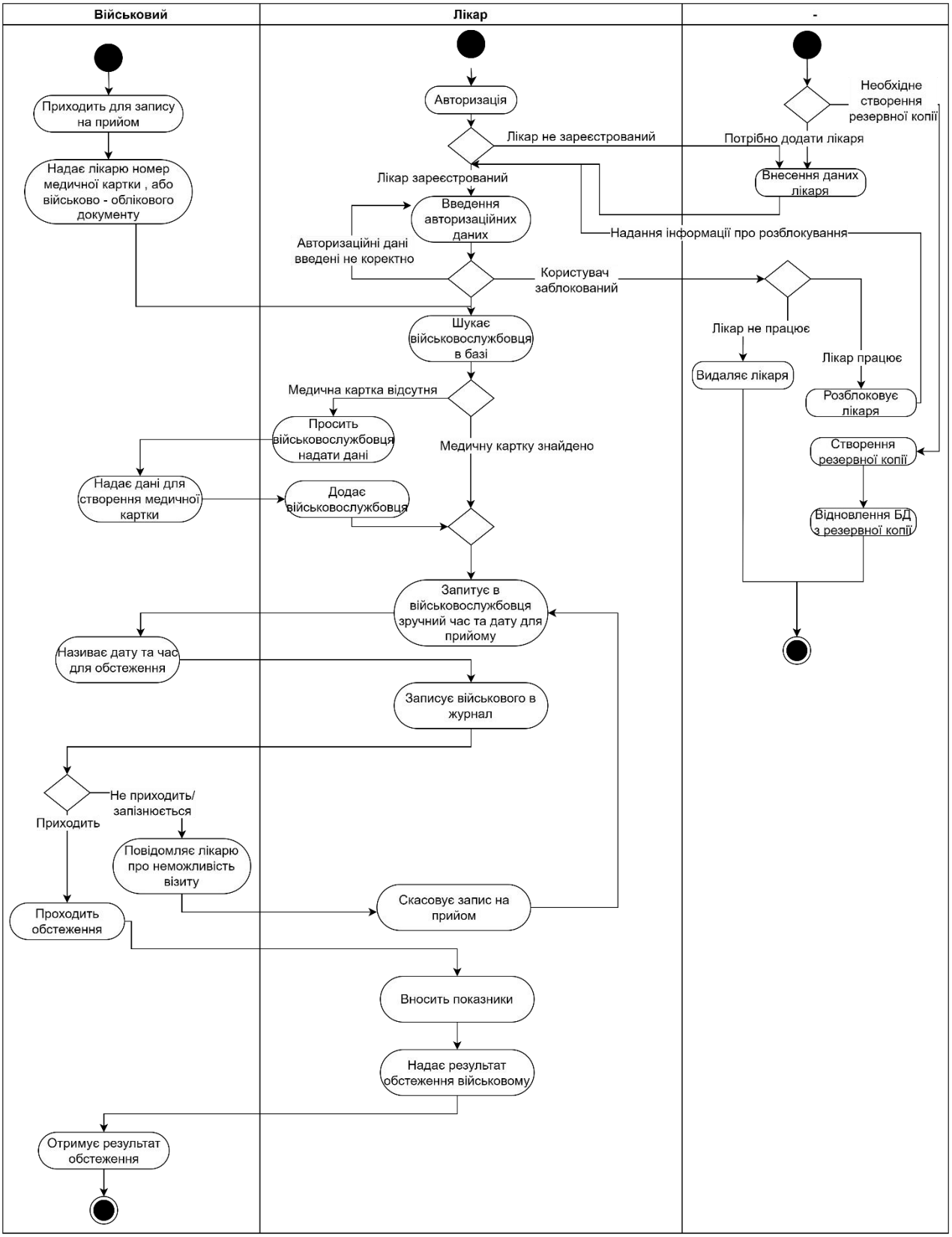

Рис. А.1 Діаграма активності

Додаток Б

ЛІСТИНГИ ПРОГРАМИ

Модуль моделі даних

```
#region Моделі даних
public class User
{
    public int UserID { get; set; }
    public string? FullName { get; set; }
    public string? Login { get; set; }
    public string? Password { get; set; }
    public int UserRoleId { get; set; }
    public UserRole UserRole { get; set; }
    public int UserStatusId { get; set; }
    public UserStatus UserStatus { get; set; }
    public int? MedicalFacilityId { get; set; }
    public virtual MedicalFacility? MedicalFacility { get; set; }
}
public class UserRole
{
    public int UserRoleID { get; set; }
    public string RoleName { get; set; }

    public ICollection<User> Users { get; set; }
}
public class UserStatus
{
    public int UserStatusID { get; set; }
    public string NameUserStatus { get; set; }

    public ICollection<User> Users { get; set; }
}
public class MedicalFacility
{
    public int MedicalFacilityID { get; set; }
    public string? NameMedicalFacility { get; set; }
    public string? Address { get; set; }
}
public class Serviceman
{
    public int ServicemanID { get; set; }
    public string? FullName { get; set; }
    public DateTime BirthDate { get; set; }
    public int GenderId { get; set; }
    public Gender Gender { get; set; }
    public string? MilitaryDocNumber { get; set; }
    public string? MedicalBookNumber { get; set; }
    public virtual List<Examination> Examinations { get; set; } = new
List<Examination>();
    public virtual List<Appointment> Appointments { get; set; } = new
List<Appointment>();
}
```

```
public class Gender
{
    public int GenderID { get; set; }
    public string NameGender { get; set; }

    public ICollection<Serviceman> Servicemen { get; set; }
}
public class Examination
{
    public int ExaminationID { get; set; }
    public int ServicemanId { get; set; }
    public virtual Serviceman? Serviceman { get; set; }
    public int DoctorId { get; set; }
    public virtual User? Doctor { get; set; }
    public int MedicalFacilityId { get; set; }
    public virtual MedicalFacility? MedicalFacility { get; set; }
    public DateTime ExaminationDate { get; set; }
    public virtual List<Result> Results { get; set; } = new List<Result>();
}
public class Parameter
{
    public int ParameterID { get; set; }
    public string? NameParameter { get; set; }
    public string? Category { get; set; }
    public string? DataType { get; set; }
    public string? Unit { get; set; }
}
public class Result
{
    public int ResultID { get; set; }
    public int ExaminationId { get; set; }
    public virtual Examination? Examination { get; set; }
    public int ParameterId { get; set; }
    public virtual Parameter? Parameter { get; set; }
    public string? Value { get; set; }
}
public class SystemLog
{
    public int LogID { get; set; }
    public int UserId { get; set; }
    public virtual User? User { get; set; }
    public string Action { get; set; }
    public DateTime Timestamp { get; set; }
}

public class Appointment
{
    public int AppointmentID { get; set; }
    public int ServicemanId { get; set; }
    public virtual Serviceman? Serviceman { get; set; }
    public int DoctorId { get; set; }
    public virtual User? Doctor { get; set; }
    public int MedicalFacilityId { get; set; }
    public virtual MedicalFacility? MedicalFacility { get; set; }
    public DateTime AppointmentDate { get; set; }
    public string? AppointmentTime { get; set; }
    public string? Description { get; set; }
```

```
    public int AppointmentStatusID { get; set; }
    public AppointmentStatus AppointmentStatus { get; set; }
}
public class AppointmentStatus
{
    public int AppointmentStatusID { get; set; }
    public string NameAppointmentStatus { get; set; }

    public ICollection<Appointment> Appointments { get; set; }
}
```

Модуль контексту бази даних

```
public class MedicalContext : DbContext
{
    public static string _connectionString;
    public DbSet<User> User { get; set; }
    public DbSet<MedicalFacility> MedicalFacility { get; set; }
    public DbSet<Serviceman> Serviceman { get; set; }
    public DbSet<Examination> Examination { get; set; }
    public DbSet<Parameter> Parameter { get; set; }
    public DbSet<Result> Result { get; set; }
    public DbSet<SystemLog> SystemLog { get; set; }
    public DbSet<Appointment> Appointment { get; set; }
    public DbSet<AppointmentStatus> AppointmentStatus { get; set; }
    public DbSet<UserRole> UserRole { get; set; }
    public DbSet<UserStatus> UserStatus { get; set; }
    public DbSet<Gender> Gender { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        LoadConnectionStringFromFile();
        optionsBuilder.UseSqlServer(_connectionString);
    }

    public static void LoadConnectionStringFromFile()
    {
        try
        {
            string projectRoot =
Path.GetFullPath(Path.Combine(AppDomain.CurrentDomain.BaseDirectory,
@"..\..\.."));
            string filePath = Path.Combine(projectRoot, "ConnectionString.txt");

            if (!File.Exists(filePath))
            {
                MessageBox.Show("Файл ConnectionString.txt не знайдено.",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }

            _connectionString = File.ReadAllText(filePath).Trim();

            if (string.IsNullOrWhiteSpace(_connectionString))
            {
```

```
                MessageBox.Show("Файл ConnectionString.txt порожній.",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка при зчитуванні рядка підключення:
{ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    public static string getconnectionstring() { return _connectionString; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<User>().HasKey(e => e.UserID);
        modelBuilder.Entity<UserRole>().HasKey(e => e.UserRoleID);
        modelBuilder.Entity<UserStatus>().HasKey(e => e.UserStatusID);
        modelBuilder.Entity<MedicalFacility>().HasKey(e => e.MedicalFacilityID);
        modelBuilder.Entity<Serviceman>().HasKey(e => e.ServicemanID);
        modelBuilder.Entity<Gender>().HasKey(e => e.GenderID);
        modelBuilder.Entity<Examination>().HasKey(e => e.ExaminationID);
        modelBuilder.Entity<Parameter>().HasKey(e => e.ParameterID);
        modelBuilder.Entity<Result>().HasKey(e => e.ResultID);
        modelBuilder.Entity<SystemLog>().HasKey(e => e.LogID);
        modelBuilder.Entity<Appointment>().HasKey(e => e.AppointmentID);
        modelBuilder.Entity<AppointmentStatus>().HasKey(e =>
e.AppointmentStatusID);

        modelBuilder.Entity<Gender>().HasData(
            new Gender { GenderID = 1, NameGender = "Чоловіча" },
            new Gender { GenderID = 2, NameGender = "Жіноча" }
        );

        modelBuilder.Entity<UserRole>().HasData(
            new UserRole { UserRoleID = 1, RoleName = "Admin" },
            new UserRole { UserRoleID = 2, RoleName = "Doctor" }
        );
        modelBuilder.Entity<UserStatus>().HasData(
            new UserStatus { UserStatusID = 1, NameUserStatus = "Активний" },
            new UserStatus { UserStatusID = 2, NameUserStatus = "Заблокований" }
        );
        modelBuilder.Entity<AppointmentStatus>().HasData(
            new AppointmentStatus { AppointmentStatusID = 1,
NameAppointmentStatus = "Заплановано" },
            new AppointmentStatus { AppointmentStatusID = 2,
NameAppointmentStatus = "Скасовано" }
        );
        modelBuilder.Entity<Serviceman>()
            .HasOne(s => s.Gender)
            .WithMany(g => g.Servicemen)
            .HasForeignKey(s => s.GenderId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<User>()
            .HasOne(u => u.UserRole)
            .WithMany(r => r.Users)
            .HasForeignKey(u => u.UserRoleId)
```

```
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<User>()
            .HasOne(u => u.UserStatus)
            .WithMany(s => s.Users)
            .HasForeignKey(u => u.UserStatusId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<Appointment>()
            .HasOne(a => a.AppointmentStatus)
            .WithMany(s => s.Appointments)
            .HasForeignKey(a => a.AppointmentStatusID)
            .OnDelete(DeleteBehavior.Restrict);

    }
}
```

Модуль генерації звітів

```
private void ExaminationReportButton_Click(object sender, RoutedEventArgs e)
{
    if (ExaminationsDataGrid.SelectedItem == null)
    {
        MessageBox.Show("Виберіть обстеження для створення звіту", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }

    try
    {
        var examination = (Examination)ExaminationsDataGrid.SelectedItem;

        using (var context = new MedicalContext())
        {
            var fullExamination = context.Examination
                .Include(ex => ex.Serviceman)
                .Include(ex => ex.Serviceman.Gender)
                .Include(ex => ex.Doctor)
                .Include(ex => ex.MedicalFacility)
                .FirstOrDefault(ex => ex.ExaminationID ==
examination.ExaminationID);

            var results = context.Result
                .Include(r => r.Parameter)
                .Where(r => r.ExaminationId == examination.ExaminationID)
                .ToList();

            SaveFileDialog saveFileDialog = new SaveFileDialog
            {
                Filter = "PDF файли (*.pdf)|*.pdf",
                FileName =
$"Звіт_{fullExamination.Serviceman.FullName}_{fullExamination.ExaminationDate.To
String("yyyy-MM-dd")}.pdf"
            };

            if (saveFileDialog.ShowDialog() == true)
            {
```

```
                GenerateExaminationReport(fullExamination, results,
saveFileDialog.FileName);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка створення звіту: {ex.Message}", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

private void CompareExaminationsButton_Click(object sender, RoutedEventArgs e)
{
    if (ExaminationsDataGrid.SelectedItem == null)
    {
        MessageBox.Show("Виберіть обстеження для порівняння", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }

    if (selectedExamination == null)
    {
        selectedExamination = (Examination)ExaminationsDataGrid.SelectedItem;
        StatusText.Text = $"Вибрано обстеження від
{selectedExamination.ExaminationDate.ToShortDateString()} для порівняння.
Виберіть інше обстеження.";
        return;
    }

    var secondExamination = (Examination)ExaminationsDataGrid.SelectedItem;

    if (selectedExamination.ExaminationID == secondExamination.ExaminationID)
    {
        MessageBox.Show("Виберіть інше обстеження для порівняння", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }

    try
    {
        using (var context = new MedicalContext())
        {
            var firstExam = context.Examination
                .Include(ex => ex.Serviceman)
                .Include(ex => ex.Serviceman.Gender)
                .Include(ex => ex.Doctor)
                .Include(ex => ex.MedicalFacility)
                .FirstOrDefault(ex => ex.ExaminationID ==
selectedExamination.ExaminationID);

            var secondExam = context.Examination
                .Include(ex => ex.Serviceman)
                .Include(ex => ex.Serviceman.Gender)
                .Include(ex => ex.Doctor)
                .Include(ex => ex.MedicalFacility)
                .FirstOrDefault(ex => ex.ExaminationID ==
secondExamination.ExaminationID);
```

```
            var firstResults = context.Result
                .Include(r => r.Parameter)
                .Where(r => r.ExaminationId == firstExam.ExaminationID)
                .ToList();

            var secondResults = context.Result
                .Include(r => r.Parameter)
                .Where(r => r.ExaminationId == secondExam.ExaminationID)
                .ToList();

            Examination oldExam, newExam;
            List<Result> oldResults, newResults;

            if (firstExam.ExaminationDate < secondExam.ExaminationDate)
            {
                oldExam = firstExam;
                newExam = secondExam;
                oldResults = firstResults;
                newResults = secondResults;
            }
            else
            {
                oldExam = secondExam;
                newExam = firstExam;
                oldResults = secondResults;
                newResults = firstResults;
            }
            SaveFileDialog saveFileDialog = new SaveFileDialog
            {
                Filter = "PDF файли (*.pdf)|*.pdf",
                FileName = $"Порівняння_{oldExam.ExaminationDate.ToString("yyyy-
MM-dd")}_{newExam.ExaminationDate.ToString("yyyy-MM-dd")}.pdf"
            };

            if (saveFileDialog.ShowDialog() == true)
            {
                GenerateComparisonReport(oldExam, newExam, oldResults,
newResults, saveFileDialog.FileName);
            }
            selectedExamination = null;
            StatusText.Text = "Звіт порівняння обстежень створено";
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка створення звіту порівняння: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
        selectedExamination = null;
    }
}
```

Модуль авторизації користувача

```
private void LoginButton_Click(object sender, RoutedEventArgs e)
{
```

```
    string login = LoginTextBox.Text;
    string password = PasswordBox.Password;

    if (string.IsNullOrWhiteSpace(login) || string.IsNullOrWhiteSpace(password))
    {
        LoginErrorText.Text = "Введіть логін та пароль";
        return;
    }

    try
    {
        using (var context = new MedicalContext())
        {
            currentUser = context.User
                .Include(u => u.MedicalFacility)
                .Include(u => u.UserRole)
                .Include(u => u.UserStatus)
                .FirstOrDefault(u => u.Login == login && u.Password ==
password);

            if (currentUser == null)
            {
                LoginErrorText.Text = "Невірний логін або пароль";
                return;
            }

            if (currentUser.UserStatusId == 2)
            {
                LogAction(context, $"Спроба входу під заблокованим користувачем:
{currentUser.Login}");

                LoginErrorText.Text = "Користувача заблоковано. Зверніться до
адміністратора.";
                return;
            }
            else if (currentUser.UserStatusId == 1)
            {
                LogAction(context, $"Вхід в систему");

                LoginPanel.Visibility = Visibility.Collapsed;
                MainTabControl.Visibility = Visibility.Visible;

                MainTabControl.SelectedItem = MainTabControl.Items[0];
                UserInfoText.Text = $"Користувач: {currentUser.FullName}
({currentUser.UserRole.RoleName})";
                LogoutButton.Visibility = Visibility.Visible;

                AdminTab.Visibility = currentUser.UserRoleId == 1 ?
Visibility.Visible : Visibility.Collapsed;

                LoadServicemen();

                if (currentUser.UserRoleId == 1)
                {
                    LoadUsers();
                    LoadMedicalFacilities();
                    LoadParameters();
                    LoadSystemLogs();
```

```
                    DeleteUserButton.Visibility = Visibility.Visible;
                    DeleteFacilityButton.Visibility = Visibility.Visible;
                }
                else
                {
                    DeleteUserButton.Visibility = Visibility.Collapsed;
                    DeleteFacilityButton.Visibility = Visibility.Collapsed;
                }


                StatusText.Text = "Авторизацію успішно виконано";
            }
        }
    }
    catch (Exception ex)
    {
        LoginErrorText.Text = $"Помилка: {ex.Message}";
    }
}
```

Модуль додавання військовослужбовця

```
private void SaveNewServicemanButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (string.IsNullOrWhiteSpace(NewServicemanFullNameTextBox.Text))
        {
            MessageBox.Show("Введіть ПІБ військовослужбовця", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (NewServicemanBirthDatePicker.SelectedDate == null)
        {
            MessageBox.Show("Виберіть дату народження", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (NewServicemanGenderComboBox.SelectedItem == null)
        {
            MessageBox.Show("Виберіть стать", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Warning);
            return;
        }

        if (string.IsNullOrWhiteSpace(NewServicemanMilitaryDocTextBox.Text))
        {
            MessageBox.Show("Введіть номер військового документа", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (string.IsNullOrWhiteSpace(NewServicemanMedicalBookTextBox.Text))
        {
```

```
            MessageBox.Show("Введіть номер медичної книжки", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        using (var context = new MedicalContext())
        {
            if (context.Serviceman.Any(s => s.MilitaryDocNumber ==
NewServicemanMilitaryDocTextBox.Text))
            {
                MessageBox.Show("Військовослужбовець з таким номером військового
документа вже існує", "Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
                return;
            }

            if (context.Serviceman.Any(s => s.MedicalBookNumber ==
NewServicemanMedicalBookTextBox.Text))
            {
                MessageBox.Show("Військовослужбовець з таким номером медичної
книжки вже існує", "Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
                return;
            }

            ComboBoxItem selectedGenderItem =
(ComboBoxItem)NewServicemanGenderComboBox.SelectedItem;


            var gender = context.Gender.FirstOrDefault(g => g.NameGender ==
selectedGenderItem.Content.ToString());
            if (gender == null)
            {
                MessageBox.Show("Обрана стать не знайдена в базі даних",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
            var newServiceman = new Serviceman
            {
                FullName = NewServicemanFullNameTextBox.Text,
                BirthDate = NewServicemanBirthDatePicker.SelectedDate.Value,
                GenderId = gender.GenderID,
                MilitaryDocNumber = NewServicemanMilitaryDocTextBox.Text,
                MedicalBookNumber = NewServicemanMedicalBookTextBox.Text
            };

            context.Serviceman.Add(newServiceman);
            context.SaveChanges();

            LogAction(context, $"Додано нового військовослужбовця:
{newServiceman.FullName}");

            MessageBox.Show("Військовослужбовця успішно додано", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);

            LoadServicemen();

            AddServicemanDialog.Visibility = Visibility.Collapsed;
        }
    }
```

```
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка додавання військовослужбовця: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Модуль додавання обстеження

```
private void SaveExaminationButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (selectedServiceman == null)
        {
            MessageBox.Show("Спочатку потрібно вибрати військовослужбовця",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (ExaminationDatePicker.SelectedDate == null)
        {
            MessageBox.Show("Виберіть дату обстеження", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (MedicalFacilityComboBox.SelectedItem == null)
        {
            MessageBox.Show("Виберіть медичну установу", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }
        using (var context = new MedicalContext())
        {
            var facility =
(MedicalFacility)MedicalFacilityComboBox.SelectedItem;

            var examination = new Examination
            {
                ServicemanId = selectedServiceman.ServicemanID,
                DoctorId = currentUser.UserID,
                MedicalFacilityId = facility.MedicalFacilityID,
                ExaminationDate = ExaminationDatePicker.SelectedDate.Value,
                Results = new List<Result>()
            };

            context.Examination.Add(examination);
            context.SaveChanges();

            List<Result> results = new List<Result>();

            CollectResults(GeneralParametersItemsControl,
examination.ExaminationID, context, results);
            CollectResults(CardioParametersItemsControl,
examination.ExaminationID, context, results);
```

```
            CollectResults(LabParametersItemsControl, examination.ExaminationID,
context, results);
            CollectResults(PsychologyParametersItemsControl,
examination.ExaminationID, context, results);
            CollectResults(PhysicalTrainingItemsControl,
examination.ExaminationID, context, results);
            CollectResults(VisionHearingItemsControl, examination.ExaminationID,
context, results);

            if (results.Count == 0)
            {
                MessageBox.Show("Додайте хоча б один результат вимірювання",
"Увага", MessageBoxButton.OK, MessageBoxImage.Warning);
                context.Examination.Remove(examination);
                context.SaveChanges();
                return;
            }

            context.Result.AddRange(results);
            context.SaveChanges();

            LogAction(context, $"Додано нове обстеження для військовослужбовця:
{selectedServiceman.FullName}");

            MessageBox.Show("Обстеження успішно збережено", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);

            LoadExaminations(selectedServiceman.ServicemanID);

            if (ParameterComboBox.SelectedItem != null)
            {
                UpdateChart();
            }

            AddExaminationDialog.Visibility = Visibility.Collapsed;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка збереження обстеження: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Модуль запису на прийом

```
private void SaveAppointmentButton_Click(object sender, RoutedEventArgs e)
 {
     try
     {
         DatePicker appointmentDatePicker =
AddAppointmentDialog.FindName("AppointmentDatePicker") as DatePicker;
         ComboBox appointmentTimeComboBox =
AddAppointmentDialog.FindName("AppointmentTimeComboBox") as ComboBox;
         ComboBox appointmentFacilityComboBox =
AddAppointmentDialog.FindName("AppointmentFacilityComboBox") as ComboBox;
```

```
        TextBox appointmentDescriptionTextBox =
AddAppointmentDialog.FindName("AppointmentDescriptionTextBox") as TextBox;

        if (appointmentDatePicker == null || appointmentTimeComboBox == null ||
            appointmentFacilityComboBox == null ||
appointmentDescriptionTextBox == null)
        {
            MessageBox.Show("Помилка інтерфейсу", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            return;
        }

        if (selectedServiceman == null)
        {
            MessageBox.Show("Спочатку потрібно вибрати військовослужбовця",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (appointmentDatePicker.SelectedDate == null)
        {
            MessageBox.Show("Виберіть дату прийому", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (appointmentTimeComboBox.SelectedItem == null)
        {
            MessageBox.Show("Виберіть час прийому", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (appointmentFacilityComboBox.SelectedItem == null)
        {
            MessageBox.Show("Виберіть медичну установу", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        if (string.IsNullOrWhiteSpace(appointmentDescriptionTextBox.Text))
        {
            MessageBox.Show("Введіть опис прийому", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        using (var context = new MedicalContext())
        {
            try
            {
                var checkIfTableExists =
context.Model.FindEntityType(typeof(Appointment));
                if (checkIfTableExists == null)
                {
                    MessageBox.Show("Таблиця призначень відсутня в базі даних.
Необхідно оновити базу даних.", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
```

```
                    return;
                }

                int facilityId =
((MedicalFacility)appointmentFacilityComboBox.SelectedItem).MedicalFacilityID;
                var facility = context.MedicalFacility.Find(facilityId);

                if (facility == null)
                {
                    MessageBox.Show("Не вдалося знайти вибрану медичну
установу", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                    return;
                }

                var serviceman =
context.Serviceman.Find(selectedServiceman.ServicemanID);
                if (serviceman == null)
                {
                    MessageBox.Show("Не вдалося знайти вибраного
військовослужбовця", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                    return;
                }

                var doctor = context.User.Find(currentUser.UserID);
                if (doctor == null)
                {
                    MessageBox.Show("Не вдалося знайти інформацію про поточного
лікаря", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                    return;
                }

                ComboBoxItem selectedTimeItem =
(ComboBoxItem)appointmentTimeComboBox.SelectedItem;

                var appointment = new Appointment
                {
                    ServicemanId = serviceman.ServicemanID,
                    DoctorId = doctor.UserID,
                    MedicalFacilityId = facility.MedicalFacilityID,
                    AppointmentDate = appointmentDatePicker.SelectedDate.Value,
                    AppointmentTime = selectedTimeItem.Content.ToString(),
                    Description = appointmentDescriptionTextBox.Text,
                    AppointmentStatusID = 1
                };

                context.Appointment.Add(appointment);
                context.SaveChanges();

                LogAction(context, $"Призначено прийом для військовослужбовця:
{serviceman.FullName}");

                MessageBox.Show("Прийом успішно призначено", "Інформація",
MessageBoxButton.OK, MessageBoxImage.Information);

                LoadAppointments(serviceman.ServicemanID);

                AddAppointmentDialog.Visibility = Visibility.Collapsed;
            }
```

```
            catch (Exception ex)
            {
                MessageBox.Show($"Помилка при роботі з базою даних:
{ex.Message}\n\nДетальніше: {ex.InnerException?.Message}", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка призначення прийому: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Модуль додавання користувача

```
private void AddUserButton_Click(object sender, RoutedEventArgs e)
{
    NewUserFullNameTextBox.Text = "";
    NewUserLoginTextBox.Text = "";
    NewUserPasswordBox.Password = "";
    NewUserRoleComboBox.SelectedIndex = 0;

    using (var context = new MedicalContext())
    {
        var facilities = context.MedicalFacility.ToList();
        NewUserMedicalFacilityComboBox.ItemsSource = facilities;
        if (facilities.Count > 0)
            NewUserMedicalFacilityComboBox.SelectedIndex = 0;
    }


    AddUserDialog.Visibility = Visibility.Visible;
}
```

Модуль журналу системи

```
private void LogAction(MedicalContext context, string action)
{
    if (currentUser != null)
    {
        context.SystemLog.Add(new SystemLog
        {
            UserId = currentUser.UserID,
            Action = action,
            Timestamp = DateTime.Now
        });
        context.SaveChanges();
    }
}
```

Модуль резервного копіювання та відновлення бази даних

```
private void BackupDatabaseButton_Click(object sender, RoutedEventArgs e)
```

```
{
    try
    {
        string databaseName = "MilHealthDB";
        SaveFileDialog saveDialog = new SaveFileDialog
        {
            Filter = "Файли резервної копії SQL (*.bak)|*.bak",
            FileName = $"{databaseName}_Backup_{DateTime.Now:yyyy-MM-dd_HH-mm-
ss}.bak"
        };

        if (saveDialog.ShowDialog() != true)
            return;

        string backupFilePath = saveDialog.FileName;

        using (var connection = new
SqlConnection(MedicalContext._connectionString))
        {
            connection.Open();
            string backupQuery = $"BACKUP DATABASE [{databaseName}] TO DISK =
@BackupPath WITH FORMAT, INIT";
            using (var command = new SqlCommand(backupQuery, connection))
            {
                command.Parameters.AddWithValue("@BackupPath", backupFilePath);
                command.ExecuteNonQuery();
            }
        }

        MessageBox.Show("Резервну копію створено успішно.", "Успіх",
MessageBoxButton.OK, MessageBoxImage.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка резервного копіювання: {ex.Message}",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

private void RestoreDatabaseButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string databaseName = "MilHealthDB";
        OpenFileDialog openFileDialog = new OpenFileDialog
        {
            Filter = "Файли резервної копії SQL (*.bak)|*.bak",
            Title = "Оберіть файл резервної копії"
        };
        if (openFileDialog.ShowDialog() != true)
            return;
        string backupFilePath = openFileDialog.FileName;

        string dataFolder = Path.Combine(

Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
            "MedicalApp", "Database", "Restored");
```

```
        try
        {
            Directory.CreateDirectory(dataFolder);
        }
        catch (UnauthorizedAccessException)
        {
            dataFolder = Path.Combine(
                Path.GetTempPath(),
                "MedicalApp", "Database", "Restored");
            Directory.CreateDirectory(dataFolder);
            MessageBox.Show("Використовується тимчасова директорія: " +
dataFolder, "Інформація", MessageBoxButton.OK, MessageBoxImage.Information);
        }

        string mdfPath = Path.Combine(dataFolder, $"{databaseName}.mdf");
        string ldfPath = Path.Combine(dataFolder, $"{databaseName}_log.ldf");

        EnsureSqlServerHasAccess(dataFolder);

        using (var connection = new
SqlConnection(MedicalContext._connectionString))
        {
            connection.Open();

            using (var command = new SqlCommand("USE [master]", connection))
            {
                command.ExecuteNonQuery();
            }

            bool databaseExists = false;
            using (var command = new SqlCommand("SELECT COUNT(*) FROM
sys.databases WHERE name = @name", connection))
            {
                command.Parameters.AddWithValue("@name", databaseName);
                databaseExists = (int)command.ExecuteScalar() > 0;
            }

            if (databaseExists)
            {
                try
                {
                    string setSingleUser = $"ALTER DATABASE [{databaseName}] SET
SINGLE_USER WITH ROLLBACK IMMEDIATE";
                    using (var command = new SqlCommand(setSingleUser,
connection))
                    {
                        command.ExecuteNonQuery();
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Попередження: Не вдалося встановити режим
SINGLE_USER: {ex.Message}",
                        "Попередження", MessageBoxButton.OK,
MessageBoxImage.Warning);
                }
            }
```

```
            List<string> logicalNames = GetLogicalFileNames(connection,
backupFilePath);
            string dataFileName = logicalNames.Count > 0 ? logicalNames[0] :
databaseName;
            string logFileName = logicalNames.Count > 1 ? logicalNames[1] :
databaseName + "_log";

            string restoreQuery = $@"
        RESTORE DATABASE [{databaseName}]
        FROM DISK = @BackupPath
        WITH REPLACE,
        MOVE '{dataFileName}' TO '{mdfPath}',
        MOVE '{logFileName}' TO '{ldfPath}'";

            using (var command = new SqlCommand(restoreQuery, connection))
            {
                command.Parameters.AddWithValue("@BackupPath", backupFilePath);
                command.CommandTimeout = 300;
                command.ExecuteNonQuery();
            }
            string setMultiUser = $"ALTER DATABASE [{databaseName}] SET
MULTI_USER";
            using (var command = new SqlCommand(setMultiUser, connection))
            {
                command.ExecuteNonQuery();
            }
        }

        MessageBox.Show("Відновлення бази даних завершено. Програма буде
перезапущена.", "Успіх", MessageBoxButton.OK, MessageBoxImage.Information);
        System.Diagnostics.Process.Start(Application.ResourceAssembly.Location);
        Application.Current.Shutdown();
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при відновленні БД: {ex.Message}\n\nДеталі:
{ex.StackTrace}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Модуль генерації демонстраційних даних

```
private void ResetAndCreateDemoData()
{
    try
    {
        using (var context = new MedicalContext())
        {
            context.Result.RemoveRange(context.Result);
            context.Examination.RemoveRange(context.Examination);
            context.Appointment.RemoveRange(context.Appointment);
            context.Serviceman.RemoveRange(context.Serviceman);
            context.Parameter.RemoveRange(context.Parameter);
            context.User.RemoveRange(context.User);
            context.MedicalFacility.RemoveRange(context.MedicalFacility);
            context.SystemLog.RemoveRange(context.SystemLog);
            context.SaveChanges();
```

```
            var facility1 = new MedicalFacility { NameMedicalFacility =
"Військовий госпіталь №1", Address = "м. Київ, вул. Госпітальна, 18" };
            var facility2 = new MedicalFacility { NameMedicalFacility =
"Військово-медичний центр", Address = "м. Львів, вул. Медична, 5" };
            context.MedicalFacility.AddRange(facility1, facility2);
            if (!context.AppointmentStatus.Any())
            {
                context.AppointmentStatus.AddRange(
                    new AppointmentStatus { AppointmentStatusID = 1,
NameAppointmentStatus = "Заплановано" },
                    new AppointmentStatus { AppointmentStatusID = 2,
NameAppointmentStatus = "Виконано" },
                    new AppointmentStatus { AppointmentStatusID = 3,
NameAppointmentStatus = "Скасовано" }
                );
            }
            if (!context.UserRole.Any())
            {
                context.UserRole.AddRange(
                    new UserRole { UserRoleID = 1, RoleName = "Admin" },
                    new UserRole { UserRoleID = 2, RoleName = "Doctor" },
                    new UserRole { UserRoleID = 3, RoleName = "Registrator" }
                );
            }

            if (!context.UserStatus.Any())
            {
                context.UserStatus.AddRange(
                    new UserStatus { UserStatusID = 1, NameUserStatus =
"Активний" },
                    new UserStatus { UserStatusID = 2, NameUserStatus =
"Заблокований" }
                );
            }
            context.SaveChanges();

            var admin = new User
            {
                FullName = "Адміністратор Системи",
                Login = "admin",
                Password = "admin123",
                UserStatusId = 1,
                UserRoleId = 1,
                MedicalFacilityId = facility1.MedicalFacilityID
            };

            var doctor1 = new User
            {
                FullName = "Петренко Іван Васильович",
                Login = "doctor1",
                Password = "doctor123",
                UserRoleId = 2,
                UserStatusId = 1,
                MedicalFacilityId = facility1.MedicalFacilityID
            };

            var doctor2 = new User
```

```
            {
                FullName = "Коваленко Марія Петрівна",
                Login = "doctor2",
                Password = "doctor456",
                UserRoleId = 2,
                UserStatusId =1,
                MedicalFacilityId = facility2.MedicalFacilityID
            };

            context.User.AddRange(admin, doctor1, doctor2);
            context.SaveChanges();

            var parameters = new List<Parameter>
            {
                new Parameter { NameParameter = "Загальний стан здоров'я",
Category = "Загальні показники", DataType = "Text" },
                new Parameter { NameParameter = "Температура тіла", Category =
"Загальні показники", DataType = "Number", Unit = "°C" },
                new Parameter { NameParameter = "Вага", Category = "Загальні
показники", DataType = "Number", Unit = "кг" },
                new Parameter { NameParameter = "Зріст", Category = "Загальні
показники", DataType = "Text", Unit = "см" },
                new Parameter { NameParameter = "Індекс маси тіла", Category =
"Загальні показники", DataType = "Number", Unit = "кг/м²" },

                new Parameter { NameParameter = "Частота серцевих скорочень",
Category = "Серцево-судинна система", DataType = "Number", Unit = "уд/хв" },
                new Parameter { NameParameter = "Артеріальний тиск
(систолічний)", Category = "Серцево-судинна система", DataType = "Number", Unit
= "мм рт.ст." },
                new Parameter { NameParameter = "Артеріальний тиск
(діастолічний)", Category = "Серцево-судинна система", DataType = "Number", Unit
= "мм рт.ст." },
                new Parameter { NameParameter = "Пульсовий тиск", Category =
"Серцево-судинна система", DataType = "Number", Unit = "мм рт.ст." },
                new Parameter { NameParameter = "Серцево-судинні захворювання",
Category = "Серцево-судинна система", DataType = "Text" },
                new Parameter { NameParameter = "Функціональність дихальної
системи", Category = "Серцево-судинна система", DataType = "Text" },

                new Parameter { NameParameter = "Загальний аналіз крові",
Category = "Лабораторні показники", DataType = "Text" },
                new Parameter { NameParameter = "Рівень глюкози", Category =
"Лабораторні показники", DataType = "Number", Unit = "ммоль/л" },
                new Parameter { NameParameter = "Холестерин", Category =
"Лабораторні показники", DataType = "Number", Unit = "ммоль/л" },
                new Parameter { NameParameter = "Інфекційні захворювання",
Category = "Лабораторні показники", DataType = "Text" },

                new Parameter { NameParameter = "Психологічний стан", Category =
"Психологія", DataType = "Text" },
                new Parameter { NameParameter = "Рівень стресу", Category =
"Психологія", DataType = "Number", Unit = "бали" },
                new Parameter { NameParameter = "Рівень тривожності", Category =
"Психологія", DataType = "Number", Unit = "бали" },
                new Parameter { NameParameter = "Відсутність психічних
розладів", Category = "Психологія", DataType = "Text" },
```

```
                new Parameter { NameParameter = "Рівень витривалості та сили",
Category = "Фізична підготовка", DataType = "Text" },
                new Parameter { NameParameter = "Функціональність опорно-
рухового апарату", Category = "Фізична підготовка", DataType = "Text" },

                new Parameter { NameParameter = "Зір", Category = "Зір та слух",
DataType = "Text" },
                new Parameter { NameParameter = "Слух", Category = "Зір та
слух", DataType = "Text" }
            };

            context.Parameter.AddRange(parameters);
            context.SaveChanges();

            var servicemen = new List<Serviceman>
            {
                new Serviceman
                {
                    FullName = "Іваненко Василь Петрович",
                    BirthDate = new DateTime(1992, 5, 15),
                    GenderId = 1,
                    MilitaryDocNumber = "МД-1234567",
                    MedicalBookNumber = "МК-2345678"
                },
                new Serviceman
                {
                    FullName = "Петров Олександр Іванович",
                    BirthDate = new DateTime(1988, 8, 20),
                    GenderId = 1,
                    MilitaryDocNumber = "МД-7654321",
                    MedicalBookNumber = "МК-8765432"
                },
                new Serviceman
                {
                    FullName = "Ковальчук Наталія Михайлівна",
                    BirthDate = new DateTime(1995, 3, 10),
                    GenderId = 2,
                    MilitaryDocNumber = "МД-1122334",
                    MedicalBookNumber = "МК-2233445"
                }
            };

            context.Serviceman.AddRange(servicemen);
            context.SaveChanges();

            Random random = new Random();


                var servicemenWithGender = context.Serviceman.Include(s =>
s.Gender).ToList();
            foreach (var serviceman in servicemenWithGender)
            {
                for (int i = 0; i < 5; i++)
                {
                    var examination = new Examination
                    {
                        ServicemanId = serviceman.ServicemanID,
                        DoctorId = i % 2 == 0 ? doctor1.UserID : doctor2.UserID,
```

```
                        MedicalFacilityId = i % 2 == 0 ?
facility1.MedicalFacilityID : facility2.MedicalFacilityID,
                        ExaminationDate = DateTime.Today.AddDays(-i * 30) //
                    };

                    context.Examination.Add(examination);
                    context.SaveChanges();

                    foreach (var parameter in parameters)
                    {
                        string value;

                        if (parameter.DataType == "Number")
                        {
                            switch (parameter.NameParameter)
                            {
                                case "Частота серцевих скорочень":
                                    value = random.Next(60, 90).ToString();
                                    break;
                                case "Артеріальний тиск (систолічний)":
                                    value = random.Next(110, 140).ToString();
                                    break;
                                case "Артеріальний тиск (діастолічний)":
                                    value = random.Next(70, 90).ToString();
                                    break;
                                case "Пульсовий тиск":

                                    int systolic = random.Next(110, 140);
                                    int diastolic = random.Next(70, 90);
                                    value = (systolic - diastolic).ToString();
                                    break;
                                case "Температура тіла":
                                    value = (36.4 + random.NextDouble() *
0.8).ToString("F1");
                                    break;
                                case "Вага":
                                    if (serviceman.Gender.GenderID == 1)
                                        value = random.Next(65, 90).ToString();
                                    else
                                        value = random.Next(50, 75).ToString();
                                    break;
                                case "Зріст":
                                    if (serviceman.Gender.GenderID == 1)
                                        value = random.Next(165,
190).ToString();
                                    else
                                        value = random.Next(155,
175).ToString();
                                    break;
                                case "Індекс маси тіла":
                                    value = (random.Next(19, 28) +
random.NextDouble()).ToString("F1");
                                    break;
                                case "Рівень глюкози":
                                    value = (4.0 + random.NextDouble() *
2.0).ToString("F1");
                                    break;
                                case "Холестерин":
```

```
                                    value = (3.5 + random.NextDouble() *
2.5).ToString("F1");
                                    break;
                                case "Рівень стресу":
                                    value = random.Next(1, 10).ToString();
                                    break;
                                case "Рівень тривожності":
                                    value = random.Next(1, 10).ToString();
                                    break;
                                default:
                                    value = random.Next(1, 100).ToString();
                                    break;
                            }
                        }
                        else
                        {
                            switch (parameter.NameParameter)
                            {
                                case "Загальний стан здоров'я":
                                    var healthOptions = new[] { "Здоровий",
"Задовільний", "Хронічні захворювання відсутні", "Потребує додаткового
обстеження" };
                                    value =
healthOptions[random.Next(healthOptions.Length)];
                                    break;
                                case "Функціональність опорно-рухового апарату":
                                    var locomotorOptions = new[] { "Рухова
активність збережена", "Патологій не виявлено", "Незначні обмеження рухливості",
"В межах норми" };
                                    value =
locomotorOptions[random.Next(locomotorOptions.Length)];
                                    break;
                                case "Серцево-судинні захворювання":
                                    var heartOptions = new[] { "Захворювання
відсутні", "В межах вікової норми", "Незначні відхилення", "Потребує додаткового
обстеження" };
                                    value =
heartOptions[random.Next(heartOptions.Length)];
                                    break;
                                case "Функціональність дихальної системи":
                                    var respiratoryOptions = new[] { "Дихання
рівномірне", "Патологій не виявлено", "Функція збережена", "В межах норми" };
                                    value =
respiratoryOptions[random.Next(respiratoryOptions.Length)];
                                    break;
                                case "Загальний аналіз крові":
                                    var bloodOptions = new[] { "В межах норми",
"Незначні відхилення", "Потребує додаткового дослідження" };
                                    value =
bloodOptions[random.Next(bloodOptions.Length)];
                                    break;
                                case "Інфекційні захворювання":
                                    var infectionOptions = new[] { "Не
виявлено", "Гепатити та ВІЛ не виявлено", "Інфекційних захворювань не виявлено"
};
                                    value =
infectionOptions[random.Next(infectionOptions.Length)];
                                    break;
```

```
                                case "Психологічний стан":
                                    var psychOptions = new[] { "Стабільний",
"Задовільний", "Напружений", "Потребує консультації психолога" };
                                    value =
psychOptions[random.Next(psychOptions.Length)];
                                    break;
                                case "Відсутність психічних розладів":
                                    var mentalOptions = new[] { "Психічні
розлади не виявлено", "Психоемоційний стан стабільний", "В межах норми" };
                                    value =
mentalOptions[random.Next(mentalOptions.Length)];
                                    break;
                                case "Рівень витривалості та сили":
                                    var enduranceOptions = new[] { "Високий",
"Середній", "Вище середнього", "Достатній" };
                                    value =
enduranceOptions[random.Next(enduranceOptions.Length)];
                                    break;
                                case "Зір":
                                    var visionOptions = new[] { "100%", "З
корекцією до 100%", "В межах норми", "Потребує корекції" };
                                    value =
visionOptions[random.Next(visionOptions.Length)];
                                    break;
                                case "Слух":
                                    var hearingOptions = new[] { "В межах
норми", "Без патологій", "Слух збережений" };
                                    value =
hearingOptions[random.Next(hearingOptions.Length)];
                                    break;
                                default:
                                    value = "Норма";
                                    break;
                            }
                        }

                        var result = new Result
                        {
                            ExaminationId = examination.ExaminationID,
                            ParameterId = parameter.ParameterID,
                            Value = value
                        };

                        context.Result.Add(result);
                    }

                    context.SaveChanges();
                }

                for (int i = 0; i < 3; i++)
                {
                    var appointment = new Appointment
                    {
                        ServicemanId = serviceman.ServicemanID,
                        DoctorId = i % 2 == 0 ? doctor1.UserID : doctor2.UserID,
                        MedicalFacilityId = i % 2 == 0 ?
facility1.MedicalFacilityID : facility2.MedicalFacilityID,
                        AppointmentDate = DateTime.Today.AddDays(i * 7 + 3), //
```

```
                        AppointmentTime = $"{9 + i}:00",
                        Description = i == 0 ? "Планове обстеження" :
                                      i == 1 ? "Консультація стосовно результатів
обстеження" :
                                               "Профілактичний огляд",
                        AppointmentStatusID = 1
                    };

                    context.Appointment.Add(appointment);
                }
                context.SaveChanges();
            }

            var logEntries = new List<SystemLog>
            {
                new SystemLog { UserId = admin.UserID, Action = "Вхід в
систему", Timestamp = DateTime.Now.AddDays(-5) },
                new SystemLog { UserId = admin.UserID, Action = "Додано нового
користувача", Timestamp = DateTime.Now.AddDays(-5).AddMinutes(15) },
                new SystemLog { UserId = admin.UserID, Action = "Вихід із
системи", Timestamp = DateTime.Now.AddDays(-5).AddMinutes(45) },
                new SystemLog { UserId = doctor1.UserID, Action = "Вхід в
систему", Timestamp = DateTime.Now.AddDays(-3) },
                new SystemLog { UserId = doctor1.UserID, Action = "Додано нове
обстеження", Timestamp = DateTime.Now.AddDays(-3).AddMinutes(20) },
                new SystemLog { UserId = doctor1.UserID, Action = "Вихід із
системи", Timestamp = DateTime.Now.AddDays(-3).AddMinutes(60) },
                new SystemLog { UserId = doctor2.UserID, Action = "Вхід в
систему", Timestamp = DateTime.Now.AddDays(-1) },
                new SystemLog { UserId = doctor2.UserID, Action = "Призначено
прийом", Timestamp = DateTime.Now.AddDays(-1).AddMinutes(15) },
                new SystemLog { UserId = doctor2.UserID, Action = "Вихід із
системи", Timestamp = DateTime.Now.AddDays(-1).AddMinutes(30) }
            };

            context.SystemLog.AddRange(logEntries);
            context.SaveChanges();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка створення демонстраційних даних:
{ex.Message}", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```