**НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**УДК 004.9-047.36:613**

| **«ПОГОДЖЕНО»** | **«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»** |
|---|---|
| Декан факультету інформаційних технологій | Завідувач кафедри комп'ютерних наук |
| Болбот І.М., д.т.н., професор | Голуб Б.Л., к.т.н., доцент |
| _____ | _____ |
| ____ _____ **2024 р.** | ____ _____ **2024 р.** |

**МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

**на тему** Аналітична підсистема управління інформаційними потоками телемедичної системи моніторингу стану здоров'я

Спеціальність __121 «Інженерія програмного забезпечення»_____
(код і назва)

Освітня програма _____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

**Гарант освітньої програми**

_____професор, д.т.н._____ _____ ___Семко В. В.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Керівник магістерської кваліфікаційної роботи**

_____професор, д.т.н._____ _____ ___Семко В. В.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

**Виконав** _____ _____Гегера С. Л._____
(підпис) (ПІБ студента)

КИЇВ-2024

**НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет (ННІ)** _____

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**_____
_____
_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
“____” _____20_____року

# З А В Д А Н Н Я
# ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Гегера Станіслав Леонідович_____
(прізвище, ім’я, по батькові)

Спеціальність _____121 «Інженерія програмного забезпечення»_____
(код і назва)

Освітня програма _____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Аналітична підсистема управління інформаційними потоками телемедичної системи моніторингу стану здоров’я
затверджена наказом ректора НУБіП України від “____”_____20___р. №_____

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: наукові публікації та звіти у сфері телемедицини, нормативні документи щодо збереження та обробки медичних даних.

Перелік питань, що підлягають дослідженню:

1. Дослідження наукових джерел про телемедичні системи для моніторингу здоров’я.
2. Аналіз існуючих рішень для управління медичними інформаційними потоками.
3. Розробка архітектури аналітичної підсистеми для моніторингу здоров'я.
4. Реалізація основних модулів аналітичної підсистеми для обробки медичних даних.
5. Функціональне та компонентне тестування підсистеми з використанням MS Test.

Перелік графічного матеріалу (за потреби) _____

**Дата видачі завдання** “_____”_____20___ р.

**Керівник магістерської кваліфікаційної роботи** _____ Семко В. В.
( підпис ) (прізвище та ініціали)

**Завдання прийняв до виконання** _____ Гегера С. Л.
(підпис ) (прізвище та ініціали студента)

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Адміністратор – користувач, відповідальний за управління обліковими записами та налаштування доступу до функцій системи, а також за забезпечення стабільності роботи підсистеми.

Аналітична підсистема – частина телемедичної системи, що виконує обробку та аналіз даних, підтримує процес діагностики та прийняття рішень лікарями на основі вхідних медичних показників пацієнтів.

Лікар – медичний працівник, який має доступ до функцій системи для перегляду медичних даних пацієнтів, здійснення віддалених консультацій, а також прогнозування ризику розвитку захворювань.

Медична карта – сукупність даних про стан здоров’я пацієнта.

Пацієнт – користувач телемедичної системи «Здоров’я+», який використовує її для моніторингу стану свого здоров’я, перегляду діагнозів та отримання консультацій від лікарів.

Прогнозування ризику – функція підсистеми, що дозволяє оцінити ймовірність розвитку захворювання на основі моделей машинного навчання та даних про пацієнта.

Телемедицина – метод надання медичних послуг віддалено, який використовує інформаційні технології для діагностики, моніторингу та лікування пацієнтів через інтернет.

Трирівнева архітектура – модель архітектури програмного забезпечення, яка розділяє систему на рівень представлення, бізнес-логіки та рівень доступу до даних, що забезпечує масштабованість і зручність підтримки.

C# – мова програмування, на якій реалізовано функціональні модулі аналітичної підсистеми.

MS SQL Server – система управління реляційними базами даних.

Visual Studio 2022 – інтегроване середовище розробки (IDE), що застосовується для написання, налагодження та тестування коду аналітичної підсистеми.

# ВСТУП

У сучасному світі постійне зростання поширеності хронічних захворювань та необхідність моніторингу стану здоров'я населення зумовлюють важливість впровадження інноваційних підходів у галузі охорони здоров'я. Зокрема, зростає попит на ефективні та надійні системи моніторингу стану пацієнтів у віддаленому режимі, що дозволяють оперативно реагувати на зміни в їхньому стані та мінімізувати ризики ускладнень. Телемедичні системи, що базуються на обробці великих обсягів даних та сучасних інформаційних технологіях, поступово стають основою медичних сервісів у багатьох країнах світу. Здатність таких систем інтегруватися з іншими медичними додатками та надавати доступ до аналітичних даних у реальному часі створює нові можливості для медичних працівників та пацієнтів.

Однією з найбільших проблем у розвитку телемедичних систем є оптимізація управління інформаційними потоками, що надходять від різноманітних датчиків та пристроїв моніторингу здоров'я пацієнтів. Величезні обсяги даних, які постійно оновлюються та надходять із різних джерел, потребують своєчасної обробки, збереження та аналізу. Недостатня координація інформаційних потоків та відсутність єдиних стандартів обробки даних можуть призводити до розпорошеності даних, що негативно позначається на їхній точності, актуальності та корисності для медичних рішень. Окрім того, необхідність забезпечення конфіденційності та безпеки персональних даних у телемедичних системах додає нових викликів для їхньої аналітичної підсистеми [1].

Сучасні наукові дослідження вказують на важливість створення інтегрованих аналітичних підсистем, здатних забезпечити комплексний підхід до управління інформаційними потоками у телемедичних системах. Попри наявні технологічні розробки, все ще існують значні проблеми, пов'язані з обробкою та аналізом медичних даних у режимі реального часу, зокрема, для пацієнтів з хронічними захворюваннями. Відомі рішення надають певні підходи

до збору й обробки даних, проте їхня здатність до масштабування та інтеграції нових джерел інформації часто обмежена [2]. Критичний аналіз існуючих систем показує, що ефективність обробки даних у телемедичних системах значною мірою залежить від можливості створення оптимізованої аналітичної підсистеми, яка була б здатна забезпечувати автоматичне управління великим обсягом інформації та знижувати навантаження на медичний персонал.

Таким чином, розробка аналітичної підсистеми, здатної до інтелектуальної обробки, агрегування та аналізу інформаційних потоків, є надзвичайно актуальною для забезпечення стабільної роботи телемедичних систем. Вона дозволить не лише підвищити ефективність моніторингу здоров'я пацієнтів, але й створити основу для прийняття обґрунтованих рішень на базі об'єктивних даних, що особливо важливо в умовах швидкого зростання обсягів інформації.

Об'єкт дослідження – процес управління інформаційними потоками у системах телемедичного моніторингу стану здоров'я.

Предметом дослідження є алгоритми та методи реалізації основних функціональних модулів системи, зокрема, прогнозування ризику діабету на основі машинного навчання, управління електронними медичними картами, захищеного доступу для різних груп користувачів (лікарів, пацієнтів, адміністраторів), а також механізми взаємодії між лікарями та пацієнтами через консультування та діагностику.

Мета роботи: розробка аналітичної підсистеми, яка забезпечить управління та аналіз інформаційних потоків у телемедичній системі, покращуючи точність діагностики, підтримуючи прийняття медичних рішень, моніторинг стану здоров'я пацієнтів та оптимізацію взаємодії між лікарями і пацієнтами.

Для виконання поставленої мети, необхідно виконати наступні завдання:

– провести аналіз поточного стану телемедичних систем для моніторингу здоров'я, включаючи характеристику особливостей їх

застосування, потребу в аналітичних підсистемах та огляд існуючих рішень для управління інформаційними потоками в медичній галузі;

– обґрунтувати вибір технологій для реалізації аналітичної системи, включаючи мову програмування, інструменти розробки та систему управління базами даних, а також розробити архітектуру системи з урахуванням вимог до аналітичної підсистеми;

– реалізувати ключові модулі аналітичної системи для обробки, зберігання та передачі медичних даних, а також провести функціональне та компонентне тестування з метою оцінки надійності і точності системи;

– розробити інструкцію для користувача, що включає мінімальні вимоги до запуску системи, процес розгортання та детальний опис використання аналітичної системи для моніторингу здоров'я пацієнтів.

Методи дослідження: аналіз наукових публікацій у галузі телемедицини та аналітичних систем, порівняльний аналіз технологій розробки, проектування архітектури програмного забезпечення, модульне та функціональне тестування, використання автоматизованих засобів тестування.

Наукова новизна одержаних результатів полягає в удосконаленні методів прогнозування медичних ризиків шляхом розробки системи підтримки прийняття рішень, яка інтегрує алгоритми машинного навчання з електронними медичними картами для оцінки ризику діабету. Вперше реалізовано підхід до автоматизованого прогнозування на основі моделей машинного навчання з використанням структурованих медичних даних, що дозволяє не лише оцінювати ризик захворювання, а й забезпечувати зручний доступ до медичних даних для пацієнтів і лікарів.

Кваліфікаційна робота магістра складається з завдання, реферату, змісту, переліку скорочень, вступу, 3 розділів, висновків, переліку посилань із 51 найменування та 2 додатків. Загальний обсяг роботи становить 124 сторінки, з них 86 сторінок основного тексту та 38 сторінок додатків. У роботі наведено 21 рисунок, 37 формул та 9 таблиць.

# 1 АНАЛІЗ ПОТОЧНОГО СТАНУ ТЕЛЕМЕДИЧНИХ СИСТЕМ ДЛЯ МОНИТОРИНГУ ЗДОРОВ'Я

## 1.1 Характеристика телемедичних систем та особливості їх застосування

Телемедичні системи – це сучасні інформаційно-комунікаційні технології, що використовуються для дистанційного надання медичних послуг, включаючи діагностику, моніторинг, консультування та лікування пацієнтів. Завдяки телемедичним системам медичні фахівці можуть взаємодіяти з пацієнтами незалежно від їхнього місцезнаходження, що значно розширює доступність медичних послуг та знижує навантаження на лікарів у віддалених чи важкодоступних регіонах [3]. Розвиток телемедицини також сприяє впровадженню технологій прогнозування та аналізу здоров'я пацієнтів, що є особливо актуальним для діагностики хронічних захворювань, таких як діабет.

Завдяки телемедичним системам, лікарі можуть здійснювати моніторинг пацієнтів у режимі реального часу, що є особливо корисним для хронічних хворих, які потребують постійного спостереження та регулярного оновлення лікувальних рекомендацій. Такі системи забезпечують безперервний доступ до актуальних даних про стан здоров'я пацієнтів, включаючи показники, що змінюються протягом дня, як-от рівень глюкози, артеріальний тиск або частота серцевих скорочень. Це дозволяє лікарям своєчасно виявляти критичні зміни та оперативно реагувати на будь-які відхилення від норми, що особливо важливо для пацієнтів з ризиком раптових ускладнень [4]. Крім того, безперервний моніторинг дає змогу пацієнтам почуватися більш впевнено, знаючи, що їхній стан контролюється медичними фахівцями навіть на відстані. Це не лише покращує якість медичних послуг, але й сприяє профілактиці захворювань, дозволяючи виявити тенденції до погіршення стану здоров'я на ранніх етапах.

Телемедичні системи поділяються на кілька основних видів, які відрізняються за функціональними можливостями та областями застосування. Деякі системи спрямовані на забезпечення консультацій через відеозв'язок, інші надають можливість віддаленого моніторингу фізіологічних показників, а ще інші об'єднують функції дистанційного лікування та профілактики. У табл. 1.1 наведено основні види телемедичних систем, їх призначення, типи даних, що обробляються, та переваги для пацієнтів і медичних працівників.

Таблиця 1.1

Основні види телемедичних систем

| Телемедичні системи | Призначення | Типи даних, що обробляються | Переваги для користувачів |
|---|---|---|---|
| Системи для відео-конференцій | Дистанційне консультування між лікарем і пацієнтом або між лікарями | Відео, аудіо, текст | Доступність спеціалістів незалежно від місця знаходження |
| Системи віддаленого моніторингу | Постійний моніторинг показників здоров'я, таких як артеріальний тиск і рівень глюкози | Дані медичних сенсорів, біометричні показники | Можливість моніторингу в реальному часі |
| Системи зберігання медичних даних | Зберігання електронних медичних записів та обмін ними між медичними установами | Електронні медичні картки, результати аналізів | Швидкий доступ до історії хвороби |
| Системи діагностики на відстані | Виконання діагностики на відстані за допомогою спеціальних приладів | Медичні зображення, дані з діагностичного обладнання | Зниження витрат на діагностичні процедури |
| Мобільні додатки для здоров'я | Самостійне відстеження здоров'я пацієнтом і контроль лікування | Дані про фізичну активність, раціон харчування | Підвищення відповідальності пацієнтів за здоров'я |
| Системи телехірургії | Виконання хірургічних процедур на відстані з використанням робототехніки | Відеозображення високої точності, дані сенсорів | Доступ до високо-кваліфікованих фахівців |

Різноманітність телемедичних систем дозволяє ефективно вирішувати специфічні завдання медичних послуг, починаючи від консультацій та моніторингу, до надзвичайних ситуацій та навіть дистанційних хірургічних процедур. Кожен вид телемедичної системи має свої переваги та можливості, які можуть задовольнити різні потреби як пацієнтів, так і медичних працівників. Такий підхід значно розширює доступ до медичних послуг і робить лікування більш гнучким та адаптивним до сучасних вимог [5].

Телемедичні системи надають широкий спектр функцій для забезпечення дистанційного надання медичних послуг та оптимізації взаємодії між лікарями і пацієнтами. Завдяки таким системам лікарі можуть здійснювати віддалені консультації, моніторинг стану здоров’я пацієнтів, а також діагностику, не обмежуючись фізичною присутністю пацієнта в медичному закладі. Це відкриває нові можливості для медичного обслуговування у віддалених регіонах і полегшує доступ до медичних послуг для людей з обмеженими можливостями чи труднощами в пересуванні.

Основні функції телемедичних систем включають організацію віддалених консультацій, проведення діагностики, зберігання та обмін медичними даними, а також підтримку прийняття рішень лікарями на основі автоматизованого аналізу даних. Ці функції забезпечують високий рівень взаємодії між лікарями і пацієнтами, дозволяючи надавати медичну допомогу в режимі реального часу та зменшуючи залежність від фізичної близькості. Окрім того, телемедицина активно використовується для моніторингу хронічних хворих, що потребують постійного спостереження за станом здоров’я, та управління надзвичайними медичними ситуаціями, що дозволяє своєчасно реагувати на критичні зміни у стані пацієнтів. Такий підхід не тільки покращує доступність медичних послуг, але й підвищує ефективність і безпеку лікування для всіх учасників медичного процесу [6].

Функціонал телемедичних систем охоплює різні аспекти медичного обслуговування, спрямовані на підвищення ефективності лікування, профілактики та моніторингу стану здоров’я пацієнтів. Завдяки різноманітним

функціям, ці системи стають важливим інструментом у наданні медичних послуг високої якості незалежно від географічного розташування пацієнта. У табл. 1.2 наведено основні функції телемедичних систем, їх призначення, приклади використання та переваги для медичних працівників і пацієнтів.

Таблиця 1.2

Основні функції телемедичних систем

| Основні функції | Призначення | Приклади використання | Переваги для користувачів |
|---|---|---|---|
| Віддалені консультації | Забезпечення консультацій між лікарем та пацієнтом через відео або текстовий зв’язок | Консультації в реальному часі | Доступність медичних послуг незалежно від локації |
| Дистанційна діагностика | Проведення діагностики на відстані за допомогою спеціальних пристроїв | Обстеження серця, шкіри, дихання | Зниження витрат на діагностику |
| Віддалений моніторинг пацієнтів | Постійне відстеження стану пацієнта через датчики та мобільні додатки | Моніторинг тиску, рівня глюкози | Виявлення змін у стані здоров’я в реальному часі |
| Зберігання та обмін даними | Створення та підтримка електронних медичних записів і обмін ними між лікарями | Обмін історією хвороби між клініками | Швидкий доступ до повної медичної інформації |
| Підтримка прийняття рішень | Надання рекомендацій лікарю на основі автоматизованого аналізу даних | Прогнозування ризиків, підтримка діагнозу | Підвищення точності діагностики та лікування |
| Управління надзвичайними ситуаціями | Швидка комунікація для надання допомоги у випадку критичних медичних ситуацій | Надання допомоги при серцевих нападах | Оперативний доступ до лікарів в екстрених випадках |
| Телехірургія | Проведення хірургічних процедур на відстані з використанням робототехніки | Хірургічні операції у віддалених районах | Можливість доступу до вузькопрофільних спеціалістів |

Функціональні можливості телемедичних систем дозволяють вирішувати важливі завдання у медичному обслуговуванні, починаючи від консультацій та моніторингу до проведення хірургічних операцій на відстані. Завдяки різноманітності функцій, ці системи роблять медичну допомогу доступнішою і зручнішою для пацієнтів, одночасно покращуючи якість діагностики та лікування. Телемедицина значно підвищує ефективність медичного обслуговування, дозволяючи оптимізувати ресурси і забезпечити кращу взаємодію між медичними працівниками та пацієнтами [7].

Таким чином, телемедичні системи представляють собою перспективний напрямок у розвитку медичних послуг, що дозволяє вдосконалити методи діагностики та моніторингу пацієнтів за допомогою сучасних технологій, зокрема прогнозування ризиків хронічних захворювань. Впровадження телемедицини сприяє покращенню доступу до медичної допомоги, оптимізації робочого процесу медичних працівників та активному залученню пацієнтів у процес догляду за власним здоров’ям. Проте для досягнення цих цілей необхідне вирішення завдань щодо захисту даних, підвищення надійності систем та забезпечення якості даних, які використовуються для аналізу і прогнозування.

## 1.2 Потреба в аналітичних підсистемах для обробки інформаційних потоків

У сучасних телемедичних системах аналітичні підсистеми забезпечують автоматизовану обробку великих обсягів медичних даних та їх трансформацію у цінну інформацію для лікарів і пацієнтів. Попит на такі підсистеми зумовлений необхідністю підвищення якості та ефективності медичних послуг, особливо в умовах зростання кількості хронічних захворювань, таких як діабет, що потребують постійного моніторингу та своєчасної діагностики. Аналітичні підсистеми дозволяють систематизувати та аналізувати інформаційні потоки в

реальному часі, що забезпечує оперативність, точність та підтримку прийняття медичних рішень на різних етапах лікування.

Діаграма на рис. 1.1 відображає ключові потреби аналітичних підсистем у медицині, демонструючи їх значення для покращення медичних послуг. Аналітичні підсистеми використовуються для підтримки лікарів у процесі прийняття рішень, полегшують доступ до медичних даних, підвищують точність діагностики та сприяють оптимізації робочих процесів медичного персоналу [8-9]. Вони також спрямовані на залучення пацієнтів до процесу лікування, зниження витрат на медичні послуги та раннє виявлення ризиків, що дозволяє своєчасно запобігати можливим ускладненням.


Рис. 1.1 Потреби у використанні аналітичних підсистем

До основних потреб у використанні аналітичних підсистем відносяться:

– покращення точності діагностики та прогнозування. Аналітичні підсистеми на основі машинного навчання дозволяють точніше визначати ризик захворювань на основі аналізу комплексних даних, таких як медична історія, спосіб життя та фізіологічні показники пацієнта. Це є особливо важливим для таких захворювань, як діабет, де своєчасне виявлення ризику

може допомогти уникнути серйозних ускладнень та знизити потребу в інтенсивному лікуванні на пізніших етапах;

– своєчасне реагування на зміни у стані здоров'я пацієнтів. Завдяки аналітичним підсистемам телемедичні системи можуть виявляти потенційно небезпечні зміни у стані здоров'я пацієнтів та оперативно реагувати на них. Постійний моніторинг даних у реальному часі дозволяє лікарям отримувати сповіщення про погіршення показників, що сприяє швидкому втручанню і зниженню ризику розвитку критичних станів;

– підтримка прийняття рішень лікарями. Аналітичні підсистеми спрощують доступ до великих обсягів медичної інформації та надають лікарям структуровані дані, які полегшують прийняття рішень щодо діагностики та лікування. Маючи доступ до зведеної аналітичної інформації та прогнозів, медичні працівники можуть швидше обирати відповідні методи лікування, що покращує якість надання медичної допомоги;

– підвищення обізнаності та залучення пацієнтів у процес лікування. Пацієнти можуть отримувати аналітичні звіти про свій стан здоров'я та рівень ризику розвитку захворювань, що дозволяє їм усвідомити необхідність змін у способі життя чи слідуванні медичним рекомендаціям. Це підвищує мотивацію до дотримання режиму лікування, робить процес лікування більш прозорим і посилює довіру до медичної системи;

– оптимізація роботи медичного персоналу. Аналітичні підсистеми автоматизують рутинні процеси, такі як моніторинг показників пацієнтів, обробка даних та підготовка аналітичних звітів, що дозволяє лікарям і медичним працівникам приділяти більше часу безпосередньому лікуванню пацієнтів. Завдяки автоматизованому аналізу інформаційних потоків медичні працівники можуть ефективніше розподіляти свої ресурси та зосереджуватися на складніших випадках;

– зниження витрат на медичне обслуговування. Використання аналітичних підсистем дозволяє знизити потребу в дорогих діагностичних дослідженнях на пізніх стадіях захворювань, сприяючи ранньому виявленню

ризиків і запобіганню ускладненням. Це не лише покращує здоров'я пацієнтів, але й знижує загальні витрати на охорону здоров'я, дозволяючи спрямувати ресурси на інші важливі завдання [10].

Аналітичні підсистеми для обробки інформаційних потоків є важливим інструментом у телемедичних системах, оскільки вони дозволяють підвищити точність діагностики, оперативність реагування на ризики, покращують якість медичних послуг та забезпечують ефективне використання ресурсів. Їх використання сприяє більш зваженому прийняттю рішень лікарями, активнішому залученню пацієнтів до процесу лікування та зниженню витрат на медичні послуги. У контексті прогнозування ризику діабету такі підсистеми стають невід'ємною складовою телемедичних рішень, що підвищують ефективність профілактики та лікування захворювань, адаптуючи медичні послуги до потреб сучасного суспільства.

## 1.3 Огляд існуючих рішень для управління інформаційними потоками в медицині

Сучасні системи для управління інформаційними потоками включають можливості збору, зберігання та аналізу даних про пацієнтів, управління робочими процесами медичних установ та підтримку прийняття рішень на основі доказових даних. Такі системи дозволяють інтегрувати різні медичні дані з різних джерел, забезпечуючи комплексне уявлення про стан здоров'я пацієнтів і оптимізуючи взаємодію між медичними фахівцями. Крім того, вони активно використовуються для дистанційного моніторингу пацієнтів, особливо при хронічних захворюваннях, що вимагають постійного спостереження.

Існуючі рішення охоплюють різноманітні сфери, включаючи телемедичні системи, які дозволяють проводити консультації на відстані, системи для підтримки діагностики та прогнозування, а також платформи для управління медичними записами. Кожне з цих рішень має свої особливості та призначення, залежно від типу даних, що обробляються, функціональних можливостей та

специфіки роботи медичного закладу. Розгляд існуючих рішень є важливим етапом для визначення ефективних підходів до управління інформаційними потоками, що забезпечує прозорість і доступність медичних послуг, а також підвищення якості лікування та безпеки пацієнтів [11].

*Epic Systems* – це одна з найбільших електронних медичних систем, призначена для управління медичними даними та процесами в лікарнях, клініках і великих медичних мережах (рис. 1.2). Система охоплює всі аспекти електронної медичної документації, включаючи історії хвороб, лабораторні результати, планування лікування, відстеження стану здоров'я пацієнтів та взаємодію з медичним персоналом. Вона призначена для покращення координації догляду за пацієнтами, підвищення точності діагностики та лікування, а також для спрощення адміністративних і клінічних процесів у медичних установах.

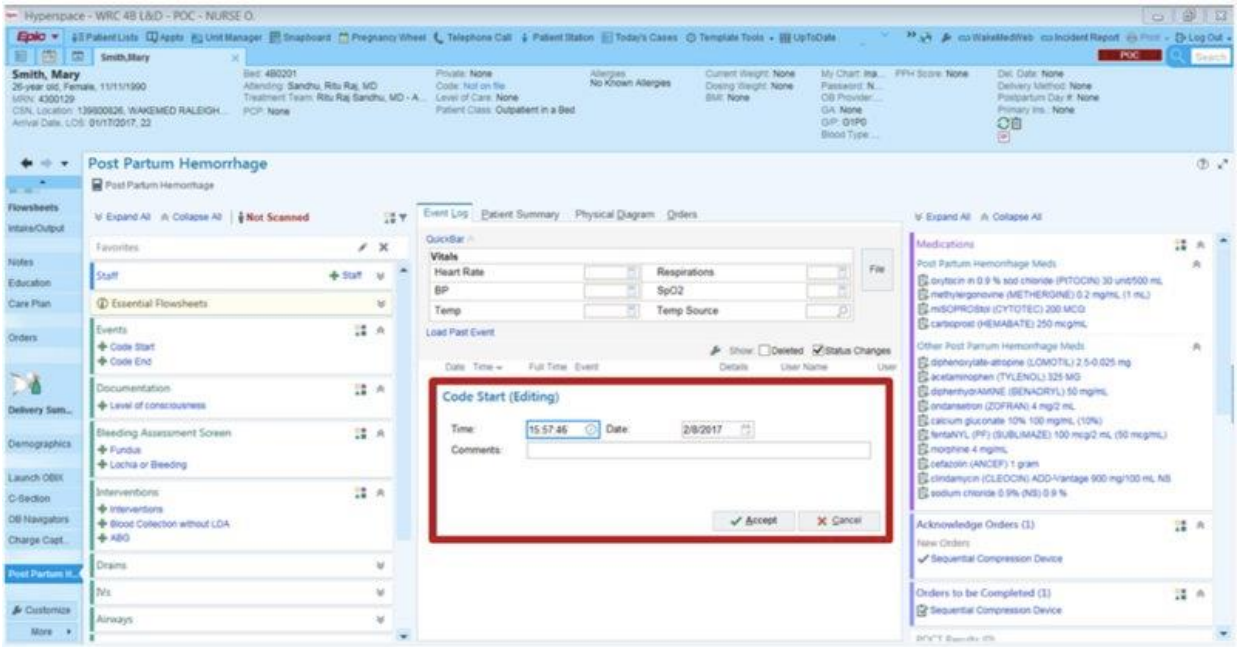


Рис. 1.2 Приклад інтерфейсу системи «Epic Systems» [12]

Архітектура Epic Systems побудована на основі централізованої платформи з великою базою даних, яка дозволяє об'єднувати всі аспекти роботи медичної установи в єдину систему. Це забезпечує доступ до актуальної інформації для лікарів і медичного персоналу, дозволяючи їм швидко

отримувати дані про стан здоров’я пацієнтів, результати аналізів та історію лікування. Епіс використовує клієнт-серверну архітектуру, де серверна частина обробляє і зберігає всі дані, а клієнтські додатки забезпечують інтерфейс для медичного персоналу та пацієнтів. Система також підтримує інтеграцію з іншими медичними додатками та пристроями, що дозволяє забезпечити повну інтероперабельність у рамках медичних мереж.

Переваги Epic Systems:

– цілісність даних. Забезпечує єдиний доступ до повної медичної історії пацієнтів, що покращує координацію та якість медичного обслуговування;

– інтеграція з іншими системами. Підтримує інтеграцію з різними медичними додатками та пристроями, що підвищує інтероперабельність у медичних установах;

– налаштування під потреби закладу. Система є дуже гнучкою та може бути налаштована під специфічні потреби різних лікарень чи клінік;

– широкий спектр функцій. Включає інструменти для клінічної, фінансової та адміністративної діяльності, що робить її універсальним рішенням для великих медичних мереж.

Недоліки Epic Systems:

– висока вартість. Значні витрати на впровадження та підтримку, що робить її недоступною для багатьох малих і середніх медичних закладів;

– складність навчання. Потребує значного часу та ресурсів для навчання персоналу через комплексність і велике число функцій;

– обмежена гнучкість для невеликих установ. Орієнтована на великі організації, через що може бути надмірно складною та дороговартісною для менших закладів;

– закритість системи [13-14]. Обмежений доступ до програмного коду та можливості кастомізації, що може ускладнювати налаштування та інтеграцію.

Отже, Epic Systems – це потужна і комплексна електронна медична система, яка забезпечує централізоване управління медичними даними та

процесами, покращуючи координацію та якість медичних послуг у великих лікарнях і медичних мережах. Хоча система пропонує широкий спектр функцій і можливостей інтеграції, її висока вартість та складність впровадження можуть бути обмежувальними факторами для менших медичних установ. Epic є чудовим вибором для великих медичних центрів, які можуть дозволити собі інвестиції в таку складну систему і мають ресурси для навчання персоналу.

*Cerner Millennium* – це комплексна електронна медична система, призначена для управління всіма аспектами медичних послуг у лікарнях, клініках та великих медичних мережах (рис. 1.3). Вона охоплює процеси ведення медичних записів, планування лікування, управління ресурсами, моніторинг стану здоров'я пацієнтів та взаємодію між різними підрозділами медичного закладу. Cerner Millennium забезпечує інтеграцію клінічних, фінансових та адміністративних даних, що дозволяє покращити координацію догляду за пацієнтами, підвищити ефективність робочих процесів та оптимізувати використання ресурсів у медичних установах.

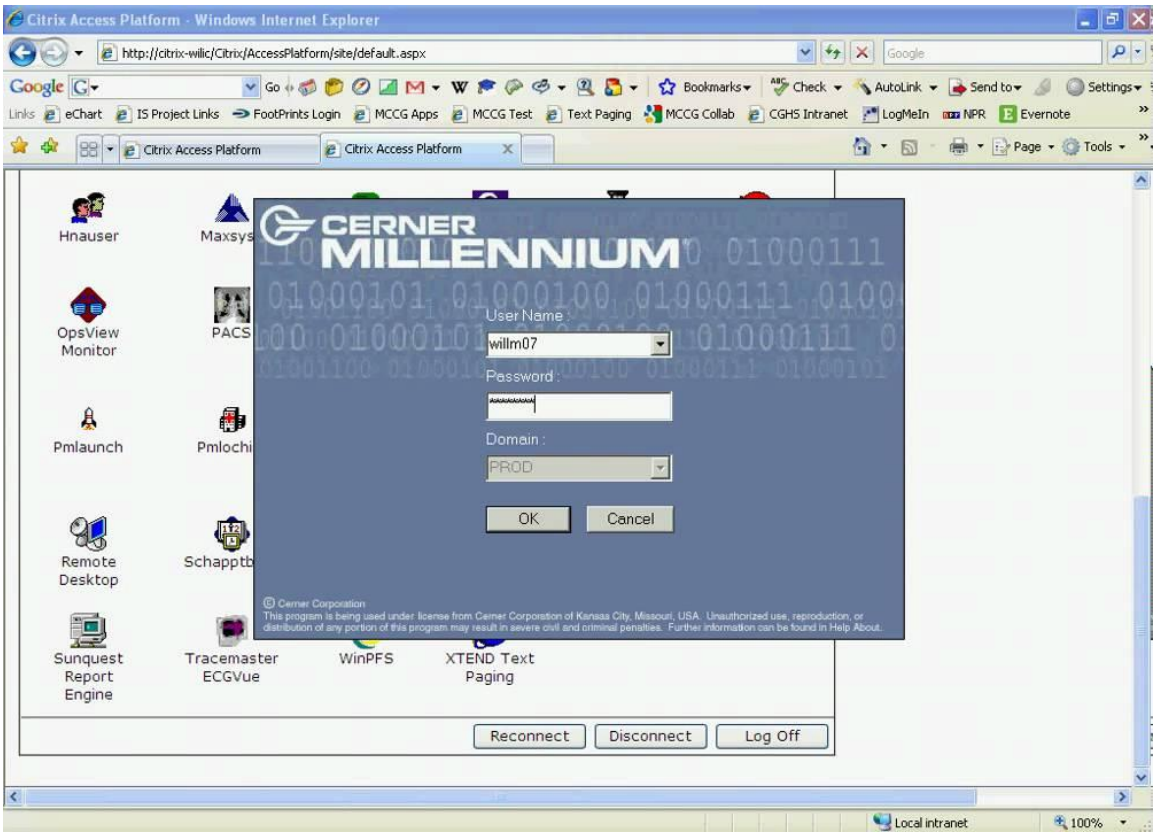


Рис. 1.3 Приклад інтерфейсу системи «Cerner Millennium» [15]

Архітектура Cerner Millennium базується на сервіс-орієнтованій архітектурі з використанням єдиної централізованої бази даних, що об’єднує всі компоненти системи. Це дозволяє забезпечити єдиний доступ до даних про пацієнтів для всіх учасників медичного процесу в реальному часі. Система складається з клієнтських і серверних модулів, де серверна частина виконує функції зберігання та обробки даних, а клієнтські додатки надають користувачам зручний інтерфейс для роботи з інформацією. Cerner Millennium також підтримує інтеграцію з іншими медичними системами та пристроями, що дозволяє забезпечити повноцінний обмін інформацією і забезпечує інтероперабельність у медичних мережах, підвищуючи якість обслуговування пацієнтів.

Переваги Cerner Millennium:

– централізована база даних. Дозволяє зберігати всі дані про пацієнтів у єдиному місці, забезпечуючи зручний доступ до актуальної інформації в режимі реального часу;

– сервіс-орієнтована архітектура. Дозволяє інтегрувати різні модулі і системи, що робить платформу гнучкою та масштабованою для великих медичних закладів;

– інтеграція з іншими системами. Підтримує обмін інформацією з іншими медичними системами та пристроями, що забезпечує повну інтероперабельність;

– підтримка всіх аспектів медичних послуг. Охоплює клінічні, фінансові та адміністративні процеси, що дозволяє централізувати управління медичним закладом.

Недоліки Cerner Millennium:

– висока вартість впровадження та підтримки. Значні витрати можуть бути непосильними для невеликих медичних установ;

– складність навчання персоналу. Потребує значного часу та ресурсів для адаптації медичного персоналу через комплексність системи;

– часом низька продуктивність. Централізована база даних інколи може призводити до уповільнення роботи в пікові моменти або при великих обсягах даних;

– обмежені можливості для кастомізації [16-17]. Незважаючи на модульність, система може бути недостатньо гнучкою для адаптації під специфічні потреби деяких установ.

Отже, Cerner Millennium – це потужна електронна медична система, яка забезпечує централізоване управління клінічними, фінансовими та адміністративними аспектами медичних закладів. Вона відзначається високим рівнем інтеграції та підтримує інтероперабельність з іншими медичними системами, що робить її підходящою для великих медичних мереж і лікарень. Однак, висока вартість і складність впровадження можуть стати перешкодою для менших установ. Загалом, Cerner Millennium є надійним рішенням для медичних закладів, які можуть інвестувати в цю комплексну систему та прагнуть покращити координацію і якість медичних послуг.

Meditech – це електронна медична система, призначена для управління клінічними, адміністративними та фінансовими аспектами медичних закладів (рис. 1.4). Вона призначена для підтримки всіх рівнів медичних послуг і може використовуватися як в окремих клініках, так і у великих медичних мережах, об’єднуючи дані про пацієнтів та забезпечуючи безперервний доступ до медичної інформації.

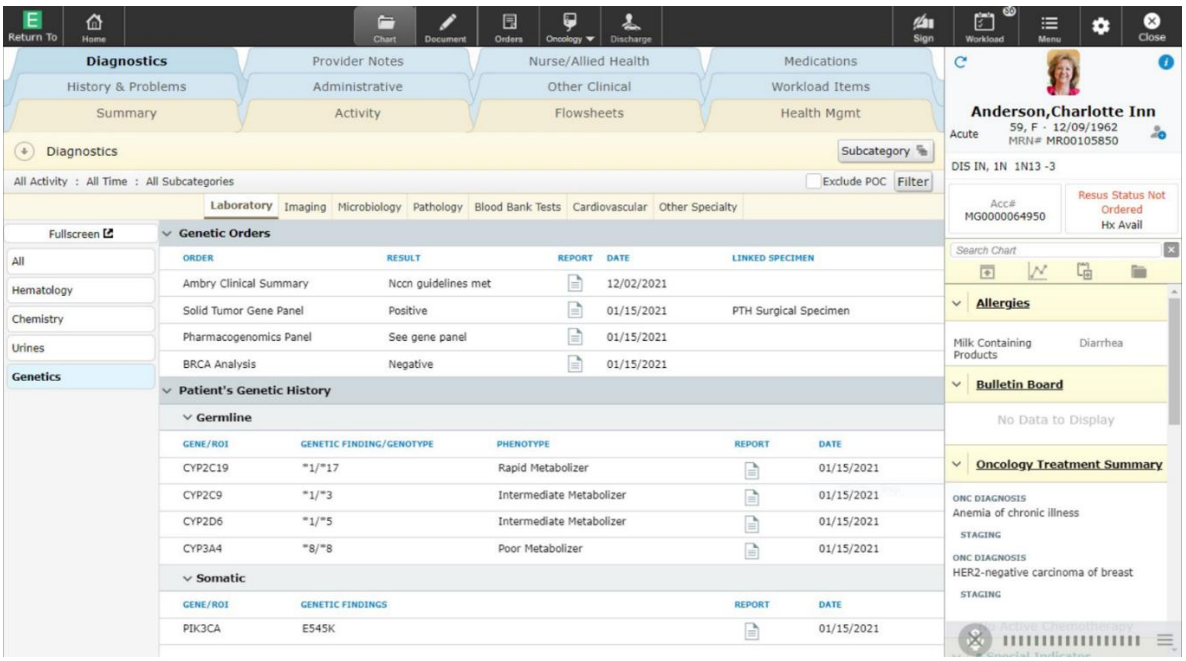


Рис. 1.4 Приклад інтерфейсу системи «Meditech» [18]

Архітектура Meditech побудована на основі централізованої платформи з використанням клієнт-серверного підходу, що дозволяє зберігати дані у єдиній базі та надавати доступ до них через клієнтські додатки. Система інтегрує всі аспекти медичної діяльності, об’єднуючи клінічні, адміністративні та фінансові модулі, що забезпечує єдину точку доступу до медичної інформації. Meditech підтримує інтеграцію з іншими медичними системами та додатками, що дозволяє забезпечити обмін інформацією між різними відділеннями і навіть медичними установами. Це рішення орієнтоване на оптимізацію робочих процесів та підвищення якості обслуговування пацієнтів, роблячи доступ до даних швидким і зручним для всіх учасників медичного процесу.

Переваги Meditech:

– комплексний підхід. Meditech об’єднує клінічні, адміністративні та фінансові функції, забезпечуючи централізоване управління всіма аспектами діяльності медичного закладу;

– зручний доступ до електронних медичних записів. Забезпечує швидкий доступ до історії пацієнта, результатів аналізів та іншої інформації, що підвищує ефективність прийняття рішень;

– інтеграція з іншими системами. Система підтримує обмін інформацією з іншими медичними додатками, що забезпечує інтероперабельність і покращує координацію між медичними установами;

– орієнтованість на оптимізацію робочих процесів. Сприяє підвищенню ефективності роботи медичного персоналу за рахунок автоматизації рутинних завдань.

Недоліки Meditech:

– висока вартість впровадження. Може бути фінансово недоступною для менших медичних закладів через високі витрати на встановлення та підтримку;

– обмежена гнучкість. Система може мати обмежені можливості для налаштування під специфічні потреби медичних закладів, що знижує її універсальність;

– складність навчання персоналу. Через багатофункціональність Meditech потребує часу та ресурсів на навчання персоналу для повноцінного використання всіх можливостей.

– можливі затримки при обробці даних. Централізована база даних інколи може сповільнювати роботу системи, особливо при значних обсягах інформації [19-20].

Отже, Meditech – це надійна електронна медична система, яка надає комплексний набір інструментів для управління клінічними, адміністративними та фінансовими процесами медичних закладів. Система підходить для великих медичних установ, які можуть інвестувати в її впровадження, та забезпечує високий рівень інтеграції і доступності медичних даних. Хоча Meditech забезпечує ефективну оптимізацію робочих процесів, її висока вартість і потреба в навчанні персоналу можуть бути бар’єрами для менших організацій. Загалом, Meditech є потужним рішенням для медичних закладів, що прагнуть підвищити якість надання медичних послуг і покращити управління даними.

Аналіз існуючих програмних рішень для управління медичними інформаційними потоками показав, що більшість систем є складними,

дорогими та часто недоступними для малих і середніх медичних закладів. Крім того, наявні системи потребують значних ресурсів для навчання персоналу і мають обмежені можливості для налаштування під специфічні потреби користувачів. Це створює потребу в розробці нового рішення, яке б поєднувало доступність, простоту використання та можливості штучного інтелекту для підтримки прийняття рішень. Таке рішення має передбачати базову безкоштовну версію з основними функціями, забезпечуючи легкість впровадження та зниження фінансових витрат на медичне обслуговування, а також мати інтуїтивно зрозумілий інтерфейс, що значно скоротить час навчання персоналу і підвищить ефективність його використання.

## 1.4 Формулювання задачі для аналітичної підсистеми моніторингу здоров'я

Основна мета аналітичної підсистеми моніторингу здоров'я в цій роботі – створити програмне забезпечення, яке оптимізує процес взаємодії між пацієнтами та лікарями, забезпечуючи надійну, ефективну та інтуїтивно зрозумілу платформу для спілкування та моніторингу стану здоров'я.

Основні функціональні можливості системи повинні включати:

– реєстрація та авторизація користувачів. Забезпечення безпечних механізмів реєстрації та авторизації для всіх категорій користувачів (пацієнти, лікарі, адміністратори), що дозволяє захистити персональні дані та медичну інформацію;

– підтвердження кваліфікації лікаря. Впровадження процедури верифікації, яка підтверджує кваліфікацію лікарів і забезпечує довіру пацієнтів до медичного персоналу;

– управління медичними картами. Створення, ведення та оновлення електронних медичних записів пацієнтів, що надає лікарям швидкий і зручний доступ до медичної інформації для якісного обслуговування;

– консультації та діагностика. Забезпечення можливості для лікарів вести облік діагностик, консультацій і дистанційно спілкуватися з пацієнтами через чати для обговорення симптомів, результатів лікування та рекомендацій;

– пошук лікарів. Надання пацієнтам функції пошуку лікарів за спеціалізацією та перегляд відгуків інших пацієнтів, що дозволяє приймати інформоване рішення щодо вибору фахівця;

– перегляд медичної інформації пацієнтами. Доступ пацієнтів до власних медичних записів, включаючи діагнози та історію лікування, що підвищує їхню обізнаність та залучення у процес лікування;

– завантаження та обробка даних. Можливість завантаження датасетів з інформацією про пацієнтів у форматі CSV, а також попередня обробка і нормалізація даних для подальшого аналізу;

– навчання моделі для прогнозування діабету. Використання алгоритму FastTree для навчання моделі, яка класифікує ризик розвитку діабету, з подальшим розділенням даних на навчальні та тестові вибірки для оцінки точності;

– збереження та завантаження моделі. Збереження натренованої моделі у файл для подальшого використання та можливість її завантаження для проведення прогнозів на основі нових даних;

– прогнозування ризику діабету. Обчислення ризику діабету на основі введених користувачем параметрів, таких як вік, індекс маси тіла та рівень фізичної активності, з виведенням ймовірності ризику для конкретного пацієнта;

– адміністрування системи. Функції для адміністраторів, що включають управління обліковими записами користувачів, моніторинг активності в системі та контроль доступу до електронних медичних карт.

Нефункціональні вимоги повинні бути:

– продуктивність. Система повинна забезпечувати швидкий відгук на запити користувачів, включаючи доступ до медичних карт та обробку запитів на прогнозування, з часом відгуку не більше 2 секунд для основних операцій;

– безпека даних. Підсистема повинна відповідати високим стандартам безпеки, включаючи шифрування персональних даних, контроль доступу та механізми автентифікації для захисту конфіденційної медичної інформації;

– масштабованість. Система має бути здатною до обробки великої кількості запитів і підтримувати зростаючий обсяг даних без зниження продуктивності, дозволяючи інтегрувати додаткові модулі за необхідності;

– зручність користування. Інтерфейс має бути інтуїтивно зрозумілим, зручним у використанні для всіх категорій користувачів (пацієнтів, лікарів, адміністраторів), з мінімальною потребою в навчанні;

– надійність та відмовостійкість. Система повинна забезпечувати стабільну роботу та швидке відновлення після збоїв, включаючи регулярне резервне копіювання даних для запобігання втратам інформації.

Результатом стане повнофункціональна, зручна у використанні аналітична підсистема для моніторингу здоров’я, яка реалізує викладені завдання, автоматизує рутинні процеси, покращує взаємодію між медичними працівниками та пацієнтами, а також сприяє загальному підвищенню якості медичного обслуговування.

## 1.5 Висновок

У рамках даного розділу проведено детальний аналіз поточного стану телемедичних систем, що використовуються для моніторингу здоров’я, з акцентом на їхню характеристику, основні функції та можливості застосування. Розглянуто основні види телемедичних систем, включаючи системи для відеоконференцій, віддаленого моніторингу, зберігання медичних даних, дистанційної діагностики, мобільні додатки для здоров’я та системи телехірургії, що забезпечують широкий спектр функціоналу для віддалених консультацій, моніторингу пацієнтів, підтримки прийняття рішень та управління надзвичайними ситуаціями. Це дослідження дозволило виокремити

ключові можливості телемедицини та особливості її використання в сучасних умовах.

Виявлено важливу потребу в аналітичних підсистемах, які здатні покращити точність діагностики, забезпечити своєчасне реагування на зміни у стані здоров’я, підтримати лікарів у прийнятті рішень, підвищити обізнаність пацієнтів та оптимізувати роботу медичного персоналу. Аналіз показав, що такі підсистеми також сприяють зниженню витрат на медичне обслуговування, що підкреслює необхідність створення нових рішень для обробки великих обсягів медичних даних і забезпечення їх аналітичною обробкою.

Розглянуто існуючі рішення для управління медичними інформаційними потоками, включаючи системи Epic Systems, Cerner Millennium та Meditech. В результаті виділено основні переваги та недоліки цих платформ, що стало основою для обґрунтування необхідності розробки нового рішення. Враховуючи існуючі обмеження, нова підсистема повинна забезпечувати гнучкість, простоту використання, можливості штучного інтелекту для підтримки рішень, легкість впровадження та доступність базової версії.

Сформульовано задачі для аналітичної підсистеми моніторингу здоров’я з конкретизацією функціональних і нефункціональних вимог. Отримані результати цього аналізу створюють підґрунтя для наступного розділу, де буде здійснено вибір технологій та розроблено архітектуру аналітичної системи.

# 2 ВИБІР ТЕХНОЛОГІЙ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ АНАЛІТИЧНОЇ СИСТЕМИ

## 2.1 Вибір технологій для реалізації системи

Розробка програмного забезпечення для аналітичної підсистеми управління інформаційними потоками телемедичної системи моніторингу стану здоров’я вимагає ретельного вибору технологій, що забезпечать ефективну обробку та аналіз медичних даних у режимі реального часу, можливість масштабування та інтеграцію з різноманітними джерелами медичної інформації. Основні аспекти вибору технологій включають вибір мови програмування, середовища розробки, а також системи управління базами даних (СУБД), яка здатна підтримувати обробку великих масивів чутливих медичних даних.

**2.1.1 Мова програмування для аналітичної системи.** Вибір мови програмування для розробки аналітичної підсистеми управління інформаційними потоками телемедичної системи моніторингу стану здоров’я є критичним аспектом, оскільки від нього залежить ефективність, продуктивність та надійність роботи системи. Основні критерії для вибору включають продуктивність, сумісність із платформами та бібліотеками для роботи з великими обсягами даних, а також підтримку безпеки для захисту конфіденційної медичної інформації. Розглянемо найбільш популярні мови програмування, які часто застосовуються для створення подібних систем.

Мова C++ забезпечує високу продуктивність і контроль над системними ресурсами, що важливо для обробки великих обсягів даних в реальному часі [21]. Вона має широкий спектр бібліотек для високоефективних обчислень, що підходить для ресурсомістких завдань. C++ також підтримує низькорівневу роботу з пам’яттю, що дозволяє оптимізувати використання ресурсів системи.

Java відома своєю портативністю, завдяки можливості працювати на будь-якій платформі, що підтримує Java Virtual Machine (JVM) [22]. Вона добре підходить для мережевих та розподілених додатків, що важливо для телемедичних систем. Java також має сильну підтримку багатопотоковості, що дозволяє ефективно обробляти паралельні потоки даних.

C# оптимізована для розробки додатків під платформи Windows і добре інтегрується з Microsoft .NET, що розширює можливості роботи з веб-сервісами та базами даних [23]. Вона забезпечує високий рівень безпеки та має вбудовану підтримку для роботи з багатопотоковими операціями. C# також зручна у використанні для розробки графічного інтерфейсу, що дозволяє створювати інтуїтивно зрозумілі та зручні для користувача додатки.

Для прийняття рішення щодо вибору мови програмування для розробки аналітичної підсистеми управління інформаційними потоками телемедичної системи доцільно порівняти основні характеристики мов C++, Java та C#. У табл. 2.1 наведено порівняння цих мов за ключовими показниками, які можуть мати значення для розробки телемедичних систем.

Таблиця 2.1

Порівняльні характеристики мов програмування

| Характеристика | C++ | Java | C# |
|---|---|---|---|
| Платформна сумісність | Висока, але потребує компіляції | Висока, через JVM | Висока, оптимальна на Windows |
| Багатопотоковість | Є, потребує додаткового налаштування | Висока, багатопотокові бібліотеки | Висока, спрощена реалізація |
| Продуктивність | Висока, контроль над ресурсами | Висока, але залежить від JVM | Висока, оптимізована під .NET |
| Зручність для розробників | Помірна, складність мови | Висока, простота використання | Висока, фокус на зручності |
| Безпека | Низькорівневі загрози | Захист через JVM | Висока, керування пам’яттю .NET |
| Інтерфейс користувача | Потребує зовнішніх бібліотек | Можливий через JavaFX, Swing | Зручна реалізація через WinForms, WPF |
| Підтримка .NET | Відсутня | Відсутня | Повна інтеграція |

Виходячи з проведеного аналізу, мова програмування C# є оптимальним вибором для розробки аналітичної підсистеми телемедичної системи завдяки своїй високій продуктивності та інтеграції з платформою .NET, що забезпечує ефективну підтримку багатопотокових операцій і дозволяє працювати з великими обсягами даних у реальному часі. C# надає розробникам інструменти для створення інтуїтивно зрозумілого графічного інтерфейсу, що спрощує взаємодію користувачів із системою. Окрім того, мова C# забезпечує високий рівень безпеки та управління пам’яттю, що є критичним для збереження конфіденційної медичної інформації.

**2.1.2 Інструменти та середовище розробки.** Вибір середовища розробки для створення аналітичної підсистеми телемедичної системи моніторингу здоров’я відіграє важливу роль, оскільки зручне та функціональне середовище може значно прискорити розробку та підвищити ефективність команди. Враховуючи особливості мови програмування C# та вимоги до інтеграції з платформою .NET, доцільно розглянути найбільш поширені середовища для розробки – Visual Studio 2022, Visual Studio Code та Rider. У кожному з них є інструменти, що полегшують роботу з мовою C#, однак кожне середовище має унікальні особливості, які слід врахувати при виборі.

Visual Studio 2022 – потужне середовище розробки з широкою підтримкою .NET, яке надає комплексні інструменти для розробки, відлагодження та тестування програм на C# [24]. Visual Studio 2022 підтримує інтеграцію з Azure та іншими сервісами Microsoft, що корисно для створення розподілених систем.

Visual Studio Code – легке середовище з підтримкою великої кількості розширень, яке забезпечує гнучкість для розробників і підтримку мов програмування [25]. Хоча воно не має всіх можливостей Visual Studio 2022, Visual Studio Code дозволяє налаштувати середовище під конкретні задачі, що підходить для проєктів з потребою в швидкій розробці. Завдяки підтримці розширень для C# та інтеграції з Git, це середовище часто обирають для кросплатформенних та відкритих проєктів.

Rider, розроблений JetBrains, є потужним середовищем, оптимізованим для розробки на C# та .NET, яке поєднує в собі швидкодію і підтримку інтелектуального аналізу коду [26]. Це середовище надає інструменти для автоматизації рутинних процесів і покращену інтеграцію з системами контролю версій, що корисно для командної розробки. Rider працює на основі платформи IntelliJ, що забезпечує стабільність та зручний інтерфейс для розробників.

Для кращого розуміння особливостей кожного з розглянутих середовищ розробки доцільно порівняти їх за основними характеристиками, які мають значення для створення аналітичної підсистеми телемедичної системи. У табл. 2.2 наведено порівняння основних характеристик середовищ Visual Studio 2022, Visual Studio Code та Rider, що дасть змогу оцінити їхні можливості та вибрати найбільш відповідне для реалізації проєкту.

Таблиця 2.2

Порівняльні характеристики середовищ розробки

| Характеристика | Visual Studio 2022 | Visual Studio Code | Rider |
|---|---|---|---|
| Підтримка .NET | Повна інтеграція, спеціалізовані інструменти | Через розширення | Добра інтеграція |
| Інструменти для відлагодження | Розширена відлагодка, інтелектуальні підказки | Обмежена відлагодка | Відлагодка на основі IntelliJ |
| Підтримка великих проєктів | Висока, оптимізація для великих рішень | Помірна, залежить від розширень | Помірна |
| Інструменти для тестування | Вбудована підтримка, широкі можливості | Через розширення | Підтримка тестування |
| Інтелектуальні підказки (IntelliSense) | Повна, розширена | Обмежена | Залежить від конфігурації |
| Рефакторинг коду | Потужний, з підтримкою багатьох шаблонів | Через розширення | Добрий, на основі IntelliJ |
| Інтерфейс для командної роботи | Підтримка інструментів для спільної роботи | Через розширення | Інтеграція з VCS |

Visual Studio 2022 є оптимальним вибором для розробки аналітичної підсистеми телемедичної системи завдяки своїй глибокій інтеграції з платформою .NET та повній підтримці мови C#, що дозволяє ефективно працювати з великими обсягами коду й комплексними рішеннями. Це середовище надає розширені інструменти для відлагодження та профілювання, які сприяють швидкій і якісній відладці коду та оптимізації продуктивності, що є критично важливим для роботи з чутливими медичними даними в реальному часі. Visual Studio 2022 також має вбудовану підтримку для розробки графічного інтерфейсу, що спрощує створення зручного та інтуїтивного інтерфейсу для користувачів телемедичної системи.

**2.1.3 Система управління базами даних.** Вибір СУБД для аналітичної підсистеми телемедичної системи є важливим етапом, оскільки від цього залежить надійність зберігання та швидкість обробки великих обсягів медичних даних. Доцільно враховувати критерії, такі як підтримка транзакцій, можливість обробки паралельних запитів, масштабованість, а також засоби безпеки, необхідні для захисту конфіденційної інформації пацієнтів. Розглянемо найбільш популярні СУБД – MS SQL Server, PostgreSQL та SQLite, які відрізняються своїми функціональними можливостями та характеристиками продуктивності.

MS SQL Server – це потужна комерційна СУБД, розроблена Microsoft, яка забезпечує високу продуктивність і безпеку для роботи з великими обсягами даних [27]. MS SQL Server пропонує розширені функції для відновлення даних, управління транзакціями та захисту даних, що особливо важливо для медичних систем.

PostgreSQL – безкоштовна і відкрита СУБД, відома своєю надійністю та підтримкою транзакційності й складних запитів [28]. PostgreSQL має розширені можливості для налаштування, що дозволяє адаптувати її під специфічні потреби проєкту, і підтримує великий набір розширень для аналітики.

SQLite – компактна СУБД, яка не вимагає окремого серверного процесу та

може працювати без інсталяції [29]. SQLite добре підходить для невеликих проєктів і локального зберігання даних, але обмежена в підтримці багатокористувацької роботи та транзакцій. Вона не підходить для великих навантажень, однак забезпечує високу швидкість для локальних операцій і часто використовується в мобільних додатках.

Для обґрунтованого вибору системи управління базами даних, яка відповідатиме вимогам телемедичної аналітичної системи, важливо порівняти ключові характеристики СУБД MS SQL Server, PostgreSQL та SQLite. У табл. 2.3 наведено порівняння цих систем за основними параметрами, що дасть змогу краще оцінити їхні можливості для створення системи.

Таблиця 2.3

Порівняльні характеристики СУБД MS SQL Server, PostgreSQL та SQLite

| Характеристика | MS SQL Server | PostgreSQL | SQLite |
|---|---|---|---|
| Підтримка транзакційності | Повна, з розширеними механізмами | Висока, з підтримкою ACID | Базова підтримка, без розширених функцій |
| Масштабованість | Висока, підтримка великих обсягів даних | Висока, підходить для великих обсягів | Обмежена, не рекомендується для великих обсягів |
| Безпека | Розширені інструменти захисту та аудиту | Добра підтримка безпеки | Базові функції безпеки |
| Інтеграція з хмарними сервісами | Повна інтеграція з Azure | Часткова підтримка | Відсутня |
| Відновлення після збоїв | Надійні механізми резервного копіювання | Вбудовані інструменти | Обмежене, без автоматизованого резервного копіювання |
| Продуктивність на великих обсягах | Висока, оптимізована для великих навантажень | Висока | Оптимізована для невеликих обсягів даних |

MS SQL Server є оптимальним вибором для аналітичної підсистеми телемедичної системи завдяки своїй високій продуктивності та масштабованості, що дозволяє обробляти великі обсяги медичних даних із мінімальними затримками. Ця СУБД надає розширені можливості захисту та аудиту, що є критичним для забезпечення конфіденційності та безпеки медичної інформації пацієнтів. Крім того, MS SQL Server підтримує надійні механізми відновлення після збоїв і резервного копіювання, що підвищує стабільність системи та дозволяє уникати втрати даних у разі непередбачених обставин.

## 2.2 Аналіз вимог до аналітичної підсистеми. Сценарії використання

Початковий етап розробки аналітичної підсистеми включає аналіз вимог до її функціональності та визначення сценаріїв використання, які відповідатимуть потребам кінцевих користувачів. Це дозволяє встановити ключові ролі в системі, що мають доступ до аналітичних даних, та забезпечити ефективну організацію інформаційних потоків [30]. У табл. 2.4 наведено основні ролі, що визначають функціональні можливості аналітичної підсистеми та доступ до її ресурсів.

Таблиця 2.4

Ролі та цілі учасників застосунку

| Актори | Цілі |
|---|---|
| Адміністратор | Відповідає за управління обліковими даними користувачів та контроль функціонування системи, зокрема, моніторинг подій та оперативне втручання при аномаліях. |
| Лікар | Займається медичними консультаціями, веде електронні медичні записи, діагностує та лікує пацієнтів, оновлює дані про здоров’я та планує лікування. |
| Пацієнт | Реєструється у системі, отримує доступ до медичної інформації, стежить за станом свого здоров’я та може генерувати звіти для особистих цілей або страхування. |

Сценарій використання відображає певний вид взаємодії між актором і системою з метою досягнення конкретної цілі. Це описує окремі випадки, коли актор взаємодіє із системою для отримання результату або вирішення завдання. У табл. 2.5 наведено основні сценарії взаємодії з системою, що включають різноманітні дії адміністраторів і користувачів.

Таблиця 2.5

Опис варіантів використання системи

| № | Ім’я | Опис |
|---|---|---|
| 1 | 2 | 3 |
| UC1 | Реєстрація користувачів | Дає змогу зареєструвати нового користувача в системі |
| UC2 | Авторизація у системі | Дозволяє користувачам ідентифікувати себе та отримати доступ до свого облікового запису |
| UC3 | Перегляд зареєстрованих користувачів | Надає адміністраторам можливість переглядати список усіх зареєстрованих у системі користувачів |
| UC4 | Додавання користувачів | Дає адміністраторам можливість додавати нові облікові записи користувачів до системи |
| UC5 | Редагування даних користувача | Дозволяє оновлювати та підтримувати в актуальному стані дані про зареєстрованих користувачів |
| UC6 | Перегляд списку спеціалізацій | Дозволяє користувачам переглядати повний список спеціалізацій медичних фахівців, доступних у системі |
| UC7 | Додавання спеціалізації | Дозволяє додавати нові спеціалізації медичних фахівців у базу даних |
| UC8 | Оновлення спеціалізації | Забезпечує можливість вносити зміни в інформацію про наявні спеціалізації |
| UC9 | Створення чату із лікарем | Дає можливість пацієнтам ініціювати чат із обраним лікарем |
| UC10 | Створення чату із пацієнтом | Дає змогу лікарям розпочати спілкування з вибраним пацієнтом |
| UC11 | Відправлення повідомлення | Дозволяє користувачам надсилати текстові повідомлення в рамках чату |
| UC12 | Перегляд інформації про лікарів | Забезпечує користувачам доступ до інформації про лікарів, зареєстрованих у системі |

Закінчення табл. 2.5

| | | |
|---|---|---|
| UC13 | Зміна інформації медичної карти | Дозволяє лікарям оновлювати записи в медичних картах пацієнтів |
| UC14 | Перегляд діагнозів пацієнта | Дає змогу переглядати історію діагнозів, поставлених пацієнтові |
| UC15 | Додавання нового діагнозу | Дає змогу додавати нові діагнози в систему для конкретного пацієнта |
| UC16 | Оновлення даних діагнозу | Дозволяє лікарям редагувати дані про раніше встановлені діагнози пацієнта |
| UC17 | Вивід моделей | Дозволяє користувачам переглядати доступні в системі моделі для аналізу даних |
| UC18 | Тренування моделей | Надає можливість проводити навчання моделей для підвищення точності прогнозів |
| UC19 | Видалення моделей | Дозволяє видаляти застарілі або непотрібні моделі з бази даних |
| UC20 | Діагностування діабету | Забезпечує використання моделі для діагностики ризику розвитку діабету в пацієнтів |
| UC21 | Перегляд подій | Надає функцію перегляду історії подій у системі |

На основі вимог було розроблено діаграми варіантів використання системи для всіх ролей користувачів системи, які наведено на рис. 2.1–2.3.

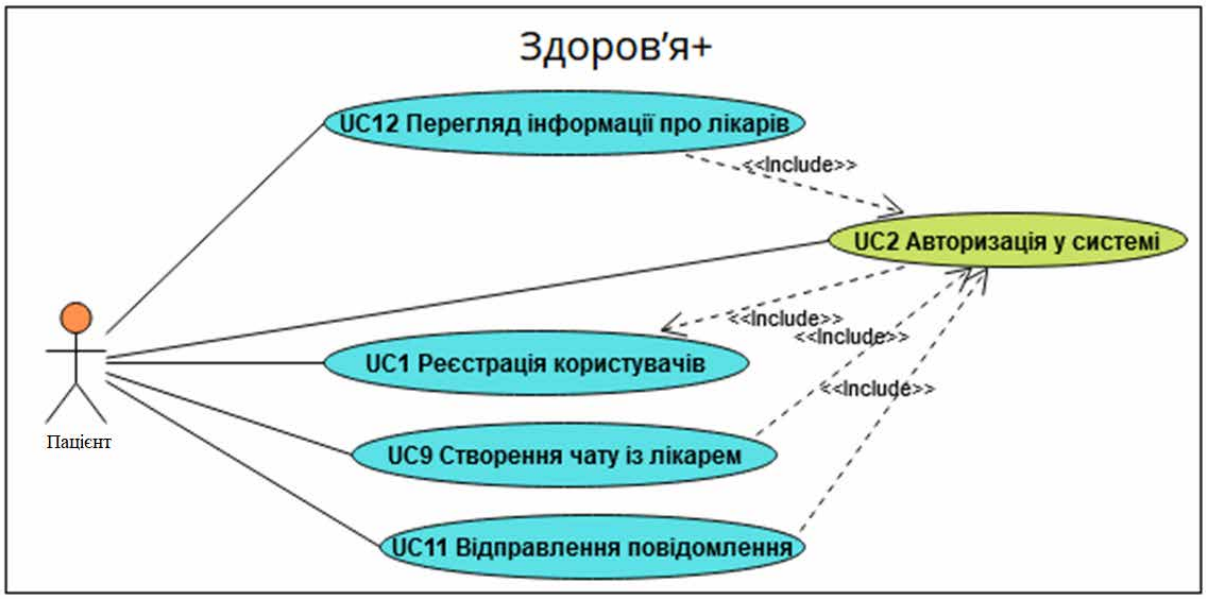


Рис. 2.1 Діаграма варіантів використання системи для ролі «пацієнт»

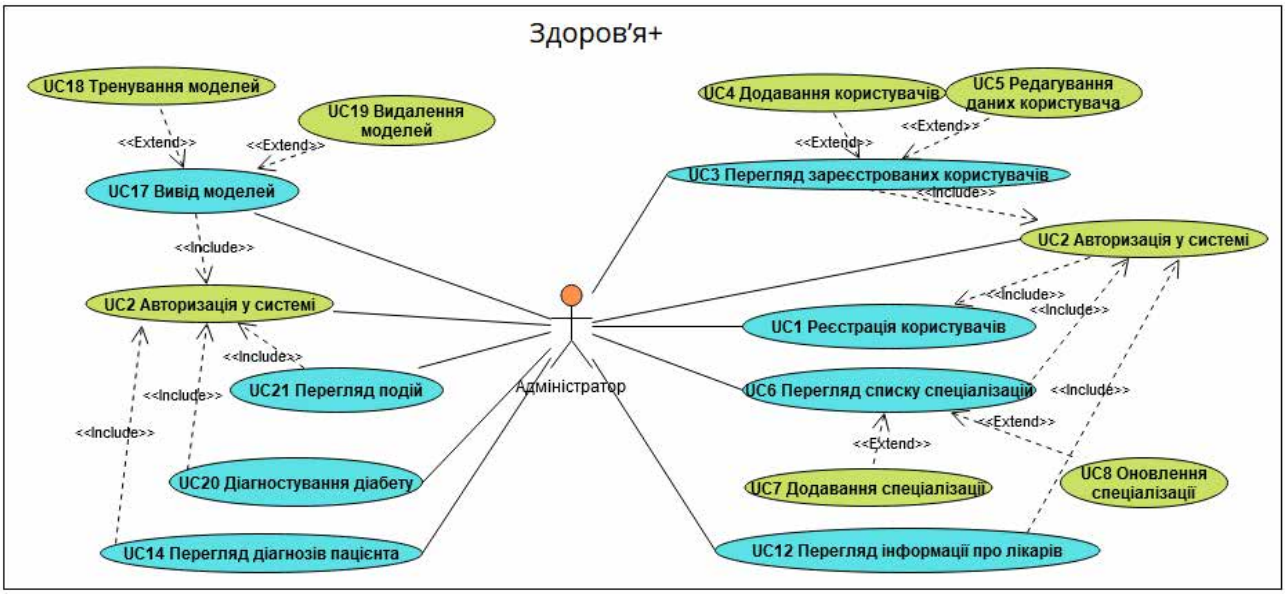


Рис. 2.2 Діаграма варіантів використання системи для ролі «адміністратор»

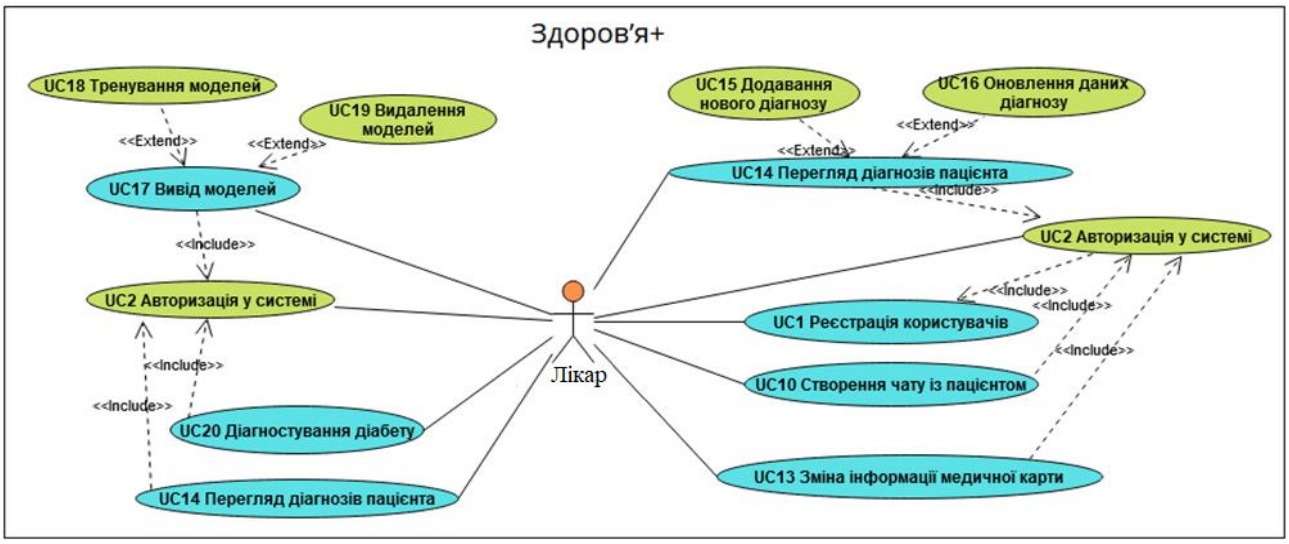


Рис. 2.3 Діаграма варіантів використання системи для ролі «лікар»

На діаграмах варіантів використання системи показано основні сценарії взаємодії для кожної ролі: користувача, адміністратора та лікаря. Діаграма на рис. 2.1 демонструє, як пацієнти взаємодіють із системою, виконуючи такі дії, як реєстрація, перегляд інформації про лікарів, доступ до медичних даних та ініціювання чатів. На рис. 2.2 наведені варіанти використання для адміністратора, який здійснює управління користувачами, оновлення даних та контроль подій у системі. Діаграма на рис. 2.3 відображає взаємодію лікаря із системою, включаючи надання медичних консультацій, внесення змін у

медичні карти пацієнтів, додавання діагнозів і ведення комунікації з пацієнтами.

## 2.3 Проєктування діаграм основних процесів аналітичної системи

Проєктування діаграм основних процесів аналітичної системи є важливим етапом для забезпечення ефективної роботи системи та зручності користувачів. Це дозволяє чітко визначити логіку взаємодії між користувачами та функціональними компонентами системи, описуючи ключові етапи та умови виконання кожного процесу. Одним із важливих процесів є авторизація користувача, яка забезпечує захищений доступ до системи та визначає права користувачів залежно від їх ролі.

На рис. 2.4 представлено діаграму процесу авторизації, що починається з відкриття форми входу.

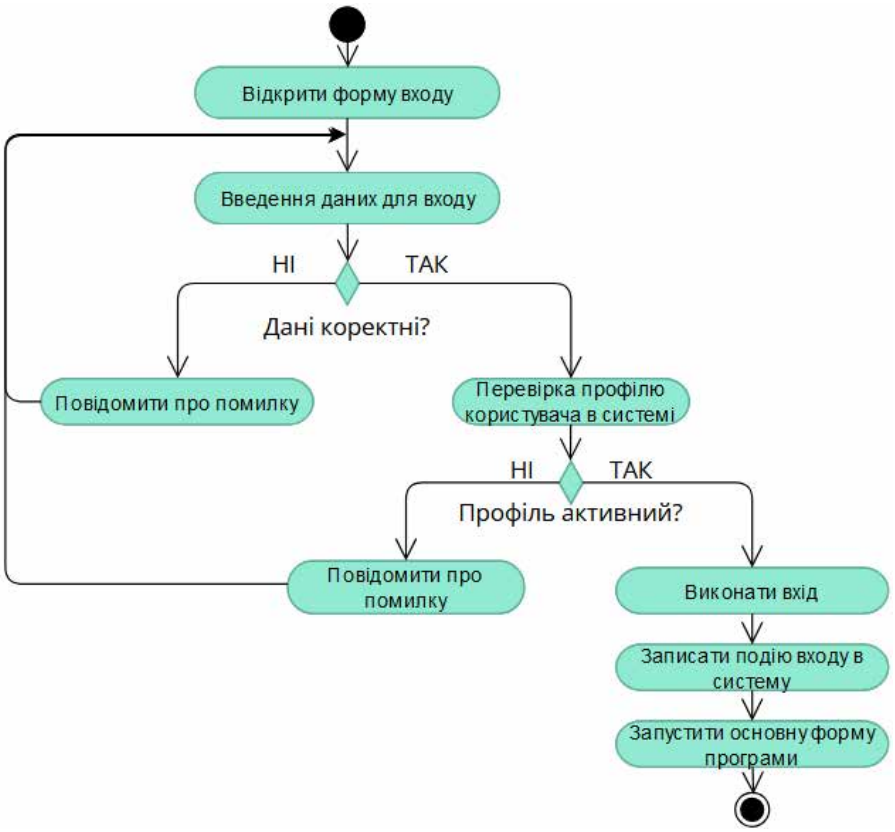


Рис. 2.4 Процес авторизації користувача у системі

Користувач вводить дані для входу, після чого система перевіряє коректність введених даних. У разі некоректності даних користувач отримує повідомлення про помилку та може повторити спробу входу. Якщо дані введені правильно, відбувається перевірка активності профілю користувача в системі. Якщо профіль неактивний, користувач також отримує повідомлення про помилку. У разі коректності даних та активності профілю система виконує вхід, фіксує подію авторизації та запускає основну форму програми для подальшої роботи.

На рис. 2.5 представлено етапи процесу тренування моделей діагностики діабету, починаючи від завантаження необхідних даних і завершуючи збереженням готової моделі.

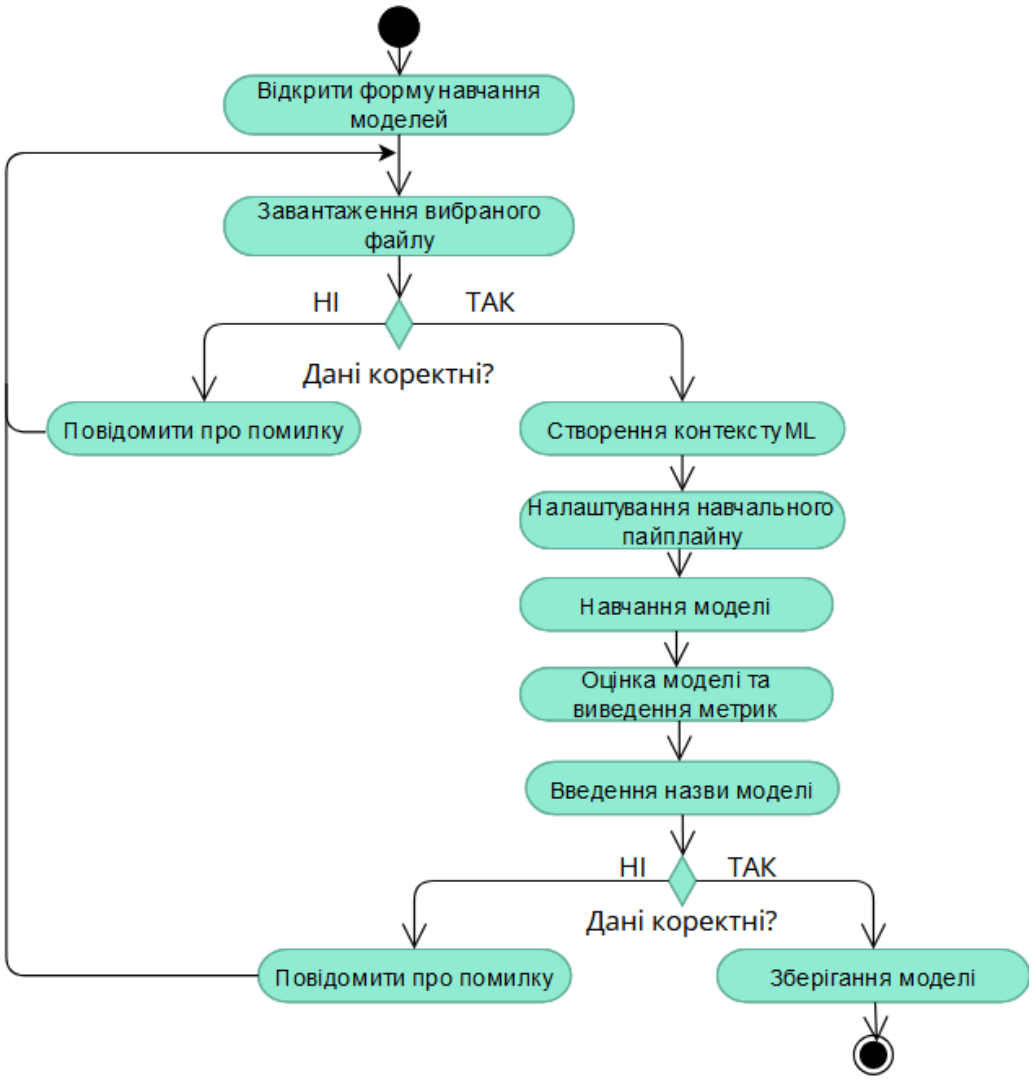


Рис. 2.5 Процес навчання моделей для діагностики діабету

Процес розпочинається з відкриття форми для навчання моделей, де користувач завантажує обраний файл з даними. Система перевіряє коректність завантажених даних: у разі помилки виводиться відповідне повідомлення, і користувач має змогу виправити помилку. Якщо дані коректні, створюється контекст для машинного навчання, налаштовується навчальний пайплайн, після чого відбувається безпосереднє навчання моделі. Після завершення навчання система проводить оцінку моделі та виводить метрики, що характеризують її точність. Далі користувач вводить назву моделі, і система перевіряє коректність введених даних. Якщо немає помилок, модель зберігається для подальшого використання в діагностиці діабету.

Діаграми послідовності є важливим інструментом у моделюванні динамічної поведінки системи, який відображає порядок взаємодій між об’єктами для виконання певного процесу. Вони показують, як об’єкти обмінюються повідомленнями в рамках сценарію використання, чітко демонструючи, в якій послідовності відбуваються дії та як дані передаються між компонентами. Завдяки цьому діаграми послідовності допомагають розробникам і аналітикам краще зрозуміти логіку роботи системи, оптимізувати процеси і виявити потенційні проблеми у взаємодії між компонентами. Вони також полегшують документування складних процесів, що сприяє підвищенню ефективності розробки та підтримки системи.

Обробка медичних карток є ключовим процесом у системі, який дозволяє лікарям переглядати, редагувати та зберігати дані про стан здоров’я пацієнтів. Це важлива частина роботи лікаря з аналітичною підсистемою, яка забезпечує доступ до актуальних медичних записів та спрощує їх управління. На рис. 2.6 зображено діаграму послідовності дій, пов’язаних з обробкою медичних карток лікарем.

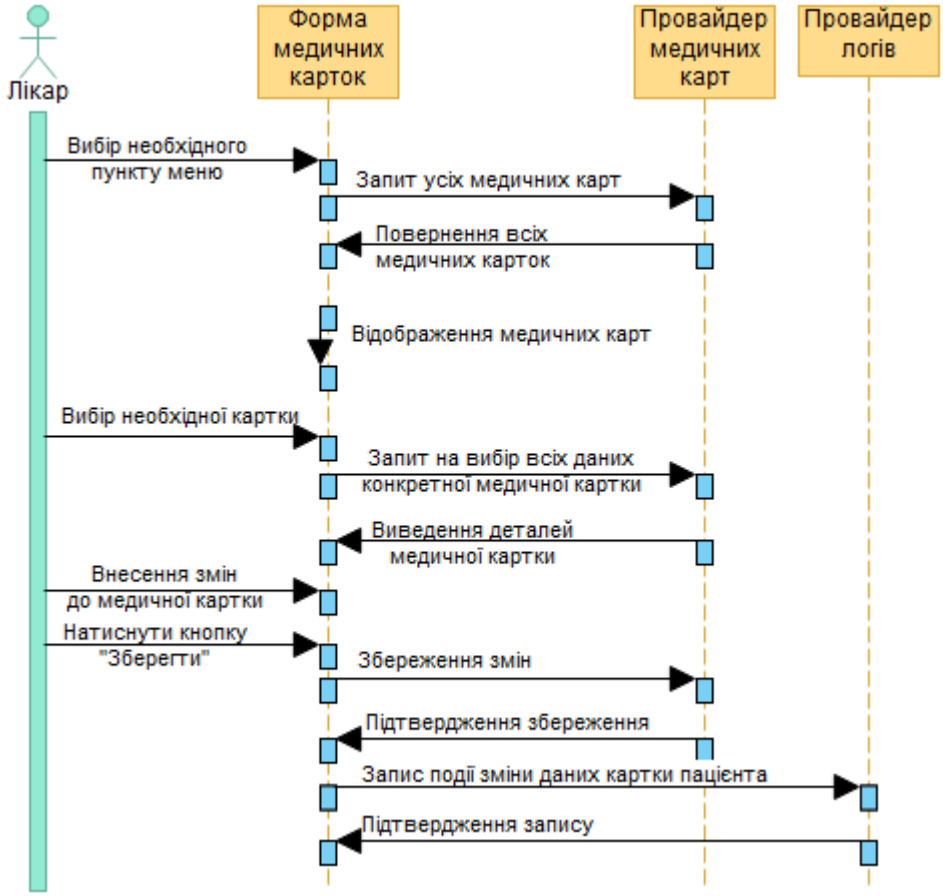


Рис. 2.6 Діаграма послідовності обробки медичних карток лікарем

Процес розпочинається з вибору лікарем необхідного пункту меню у формі медичних карток. Це ініціює запит на отримання всіх наявних медичних карток, після чого система повертає список карток і відображає їх на екрані. Лікар обирає конкретну картку для детального перегляду, і система надсилає запит на отримання повних даних по обраній картці. Після цього відображаються деталі медичної картки, і лікар може вносити необхідні зміни. Для збереження змін лікар натискає кнопку «Зберегти», що викликає процес збереження нових даних і підтвердження успішного збереження. Окрім цього, система реєструє подію змін у медичній картці пацієнта у логах для забезпечення аудиту та відстеження змін.

На діаграмі послідовності перегляду власних діагнозів пацієнтом (рис. 2.7) показано етапи взаємодії користувача з системою для отримання інформації про діагнози. Процес починається з того, що пацієнт обирає потрібний пункт меню на формі оглядів. Це ініціює запит на отримання списку

лікарів, після чого система повертає список доступних лікарів. Далі надсилається запит на отримання всіх оглядів пацієнта, і система повертає повний список оглядів.

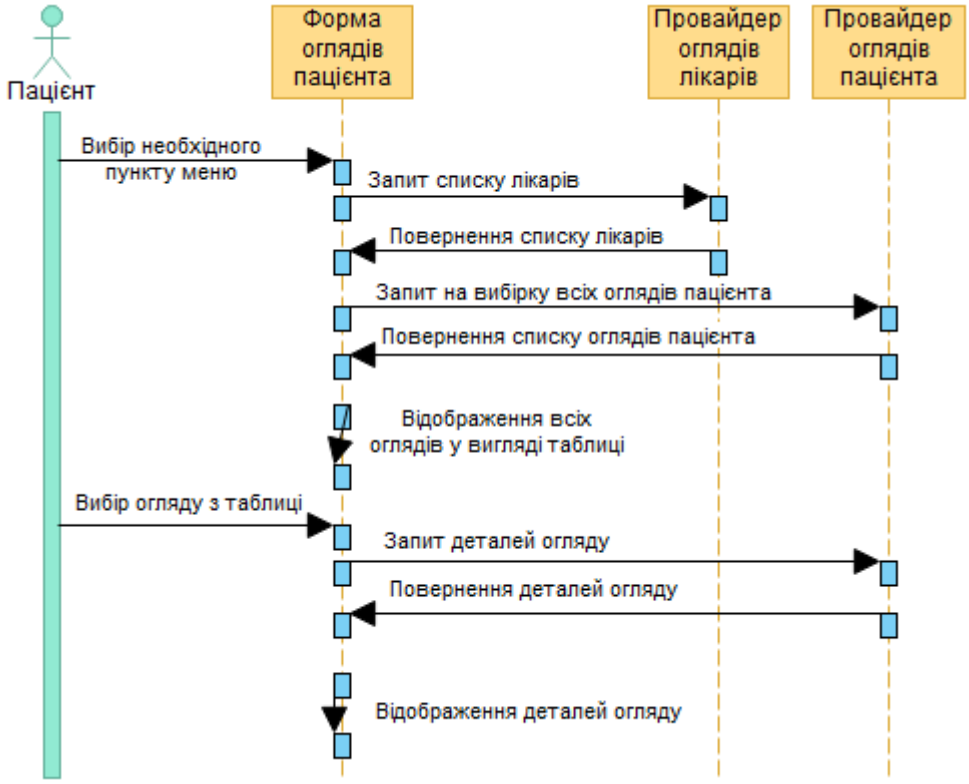


Рис. 2.7 Діаграма перегляду власних діагнозів пацієнтом

Пацієнт обирає конкретний огляд зі списку, відображеного у вигляді таблиці, що призводить до запиту на отримання деталей цього огляду. Система відповідає, повертаючи повну інформацію про обраний огляд, і відображає її пацієнтові. Таким чином, пацієнт може легко отримати доступ до детальної інформації про свої діагнози та результати медичних оглядів.

## 2.4 Проєктування бази даних для зберігання медичних даних

Проєктування бази даних для зберігання медичних даних також є важливим етапом створення аналітичної системи, оскільки від правильної структури бази залежить зручність доступу, швидкість обробки інформації та

забезпечення захисту конфіденційних даних пацієнтів. База даних повинна бути спроектована таким чином, щоб підтримувати різноманітні аспекти функціонування системи, включаючи управління користувачами, зберігання медичних записів, обробку результатів медичних оглядів і моделей машинного навчання. Кожна таблиця бази даних виконує специфічну роль і містить унікальний набір полів, що відповідають за зберігання конкретної інформації, яка буде використовуватися у системі.

Таблиця «Models» розроблена для зберігання даних, пов'язаних із моделями машинного навчання, що застосовуються для аналізу та діагностики (табл. 2.6). У цій таблиці зберігаються ідентифікатор моделі, її назва, дата створення та файли моделі, що представляють собою готові до використання алгоритми для прогнозування.

Таблиця 2.6

Структура таблиці «Models»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| ModelsId | INT | + | - | Ідентифікатор моделі, первинний ключ |
| ModelsName | nvarchar (150) | - | - | Назва моделі, що дозволяє ідентифікувати її в системі |
| CreateDate | datetime | - | - | Дата створення моделі |
| ModelsFileModel | nvarchar (max) | - | - | Файл до моделі прогнозування діабету |

Таблиця «Chats» створена для зберігання інформації про чати між лікарями та пацієнтами в системі, дозволяючи відстежувати час початку розмови, останнє повідомлення та ін. дані, пов'язані з комунікацією (табл 2.7).

Таблиця 2.7

Структура таблиці «Chats»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| ChatsId | int | + | - | Ідентифікатор чату, первинний ключ |
| PatientId | int | - | - | Ідентифікатор пацієнта, що бере участь у чаті |
| DoctorId | int | - | - | Ідентифікатор лікаря, що бере участь у чаті |
| StartTime | datetime | - | - | Час початку чату, що дозволяє відстежувати тривалість спілкування |
| LastMessageTime | datetime | - | - | Час останнього повідомлення |
| LastMessage | nvarchar (max) | - | - | Текст останнього повідомлення, що полегшує відновлення контексту розмови |

Таблиця «DoctorProfile» створена для зберігання інформації про профілі лікарів, включаючи їх спеціалізацію, досвід, кваліфікацію та біографічні дані, які допомагають у підборі лікарів відповідно до потреб пацієнтів (табл 2.8).

Таблиця 2.8

Структура таблиці «DoctorProfile»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| DoctorProfileId | int | + | - | Унікальний ідентифікатор профілю лікаря, первинний ключ |
| UsersId | int | - | - | Ідентифікатор користувача, що відповідає лікарю |
| SpecializationsId | int | - | - | Ідентифікатор спеціалізації лікаря |
| YearsOfExperience | int | - | - | Кількість років досвіду лікаря, що дозволяє оцінити його кваліфікацію |
| Qualifications | Nvarchar (max) | - | - | Опис кваліфікацій лікаря, який детально вказує його професійні навички |
| Bio | Nvarchar (max) | - | - | Біографічні дані лікаря, що надають пацієнтам інформацію про спеціаліста |

Таблиця «Logs» призначена для зберігання записів про події, що відбулися в системі, дозволяючи відстежувати дії користувачів і вести аудит їхньої активності (табл. 2.9). Вона містить ідентифікатор події, ідентифікатор

користувача, тип події, дату події та ім’я користувача, що полегшує аналіз історії активності в системі.

Таблиця 2.9

Структура таблиці «Logs»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| LogsId | int | + | - | Ідентифікатор події, первинний ключ |
| UsersId | int | - | - | Ідентифікатор користувача, який ініціював подію |
| EventNameShow | Nvarchar (max) | - | - | Назва події, яка описує тип дії, що була виконана |
| EventDate | datetime | - | - | Дата і час події |
| UsersName | nvarchar (250) | - | - | Ім’я користувача, що дозволяє ідентифікувати ініціатора події |

Таблиця «MedicalCards» призначена для зберігання інформації про медичні картки пацієнтів, яка включає діагноз, стадію захворювання, алергії, стан здоров’я, протипоказання, фактори ризику, прогноз лікування, складність і результати лікування. Це дозволяє забезпечити зручний доступ до повних медичних даних пацієнтів, що необхідно для ефективного надання медичних послуг.

Таблиця 2.10

Структура таблиці «MedicalCards»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| MedicalCardsId | int | + | - | Ідентифікатор медичної картки |
| UsersId | int | - | - | Ідентифікатор користувача, до якого належить медична картка |
| Diagnosis | nvarchar (500) | - | - | Діагноз пацієнта, що описує основне захворювання |
| Stage | nvarchar (500) | - | - | Стадія захворювання, що дозволяє відстежувати його розвиток |

Закінчення табл. 2.10

| | | | | |
|---|---|---|---|---|
| Allergy | nvarchar (500) | - | - | Інформація про алергії пацієнта, що є важливою для призначення лікування |
| Condition | nvarchar (500) | - | - | Стан пацієнта на поточний момент |
| Contraindications | nvarchar (500) | - | - | Протипоказання для лікування, що враховуються при призначенні процедур або ліків |
| RiskFactor | nvarchar (500) | - | - | Фактори ризику, що можуть впливати на прогноз і перебіг захворювання |
| TreatmentForecast | nvarchar (500) | - | - | Прогноз лікування, що відображає ймовірність успішного лікування |
| Complexity | nvarchar (500) | - | - | Складність випадку, що враховується для планування лікувальних заходів |
| TreatmentResult | nvarchar (max) | - | - | Результат лікування, що містить інформацію про завершений курс терапії або її ефект |

Таблиця «PatientExaminations» створена для зберігання інформації про медичні огляди пацієнтів, що включає дані про дату огляду, симптоми, поставлений діагноз, план лікування, примітки лікаря та дату наступного прийому. Ця таблиця дозволяє системі зберігати історію оглядів кожного пацієнта, що є важливим для підтримки безперервного медичного спостереження.

Таблиця 2.11

Структура таблиці «PatientExaminations»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| PatientExaminationsId | int | + | - | Унікальний ідентифікатор огляду пацієнта, первинний ключ |
| PatientId | int | - | - | Ідентифікатор пацієнта, якому належить огляд |
| DoctorId | int | - | - | Ідентифікатор лікаря, який провів огляд |

Закінчення табл. 2.11

| | | | | |
|---|---|---|---|---|
| ExaminationDate | datetime | - | - | Дата огляду, що дозволяє відстежувати послідовність подій |
| Symptoms | nvarchar(500) | - | - | Симптоми, що були зафіксовані під час огляду |
| Diagnosis | nvarchar(max) | - | - | Поставлений діагноз, що описує стан пацієнта |
| TreatmentPlan | nvarchar(500) | - | - | План лікування, розроблений для пацієнта |
| Notes | nvarchar(500) | - | - | Примітки лікаря, які можуть містити додаткові зауваження або рекомендації |
| NextAppointment | datetime | - | - | Дата наступного прийому, що допомагає планувати подальше спостереження |

Таблиця «Messages» призначена для зберігання повідомлень, що надсилаються в межах чатів між користувачами, такими як пацієнти та лікарі. Вона містить інформацію про унікальний ідентифікатор повідомлення, ідентифікатор чату, ідентифікатор користувача, текст повідомлення та час його відправлення, що дозволяє підтримувати повний архів переписок.

Таблиця 2.12

Структура таблиці «Messages»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| MessagesId | int | + | - | Ідентифікатор повідомлення, первинний ключ |
| ChatsId | int | - | - | Ідентифікатор чату, до якого належить повідомлення |
| UsersId | int | - | - | Ідентифікатор користувача, який надіслав повідомлення |
| MessageText | nvarchar (max) | - | - | Текст повідомлення, що відображає зміст переписки |
| MessageTime | datetime | - | - | Час надсилання повідомлення, що дозволяє відстежувати хронологію спілкування |

Таблиця «Specializations» призначена для зберігання інформації про спеціалізації медичних фахівців, що доступні в системі. Вона містить унікальний ідентифікатор спеціалізації, назву спеціалізації та її опис, що дозволяє лікарям та пацієнтам орієнтуватися в доступних напрямках медицини.

Таблиця 2.13

Структура таблиці «Specializations»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| SpecializationsId | int | + | - | Унікальний ідентифікатор спеціалізації, первинний ключ |
| SpecializationsName | nvarchar(250) | - | - | Назва спеціалізації, яка вказує на область медичної практики |
| Description | nvarchar(max) | - | - | Опис спеціалізації, що допомагає деталізувати профіль та сферу застосування |

Таблиця «Users» створена для зберігання даних про користувачів системи, включаючи особисту інформацію, логін, пароль, роль і статус користувача. Вона забезпечує управління доступом до системи, зберігаючи необхідні дані для ідентифікації та авторизації користувачів різних ролей.

Таблиця 2.14

Структура таблиці «Users»

| Назва поля | Тип даних | ПК | ЗК | Опис поля |
|---|---|---|---|---|
| UsersId | int | + | - | Унікальний ідентифікатор користувача, первинний ключ |
| FirstName | nvarchar(200) | - | - | Ім’я користувача |
| LastName | nvarchar(200) | - | - | Прізвище користувача |
| UsersName | nvarchar(50) | - | - | Логін користувача для входу в систему |
| UsersPassword | nvarchar(250) | - | - | Пароль користувача |

Закінчення табл. 2.11

| | | | | |
|---|---|---|---|---|
| RoleId | int | - | - | Ідентифікатор ролі, що визначає рівень доступу користувача |
| Description | nvarchar(max) | - | - | Додатковий опис або примітки про користувача |
| Email | nvarchar(200) | - | - | Електронна пошта користувача |
| UserStatus | bit | - | - | Статус користувача (активний чи неактивний), що визначає доступ до системи |

Побудова фізичної моделі бази даних для аналітичної підсистеми управління інформаційними потоками телемедичної системи моніторингу здоров’я дозволяє детально спроєктувати структуру медичних даних та взаємозв’язки, необхідні для ефективного функціонування системи. Така модель містить визначення таблиць, атрибутів та зв’язків між ними, що забезпечує цілісне уявлення про організацію даних, необхідних для моніторингу стану здоров’я пацієнтів та управління інформаційними потоками між лікарями, пацієнтами та адміністраторами. Це дозволяє спроєктувати архітектуру бази даних, яка підтримує високу продуктивність, безпеку даних і надійність обміну медичною інформацією.

На рис. 2.8 зображена ER-діаграма, що демонструє основні сутності системи, такі як користувачі, медичні карти, профілі лікарів, повідомлення та спеціалізації, а також зв’язки між ними, які забезпечують інтегроване управління інформаційними потоками для надання якісної телемедичної підтримки.

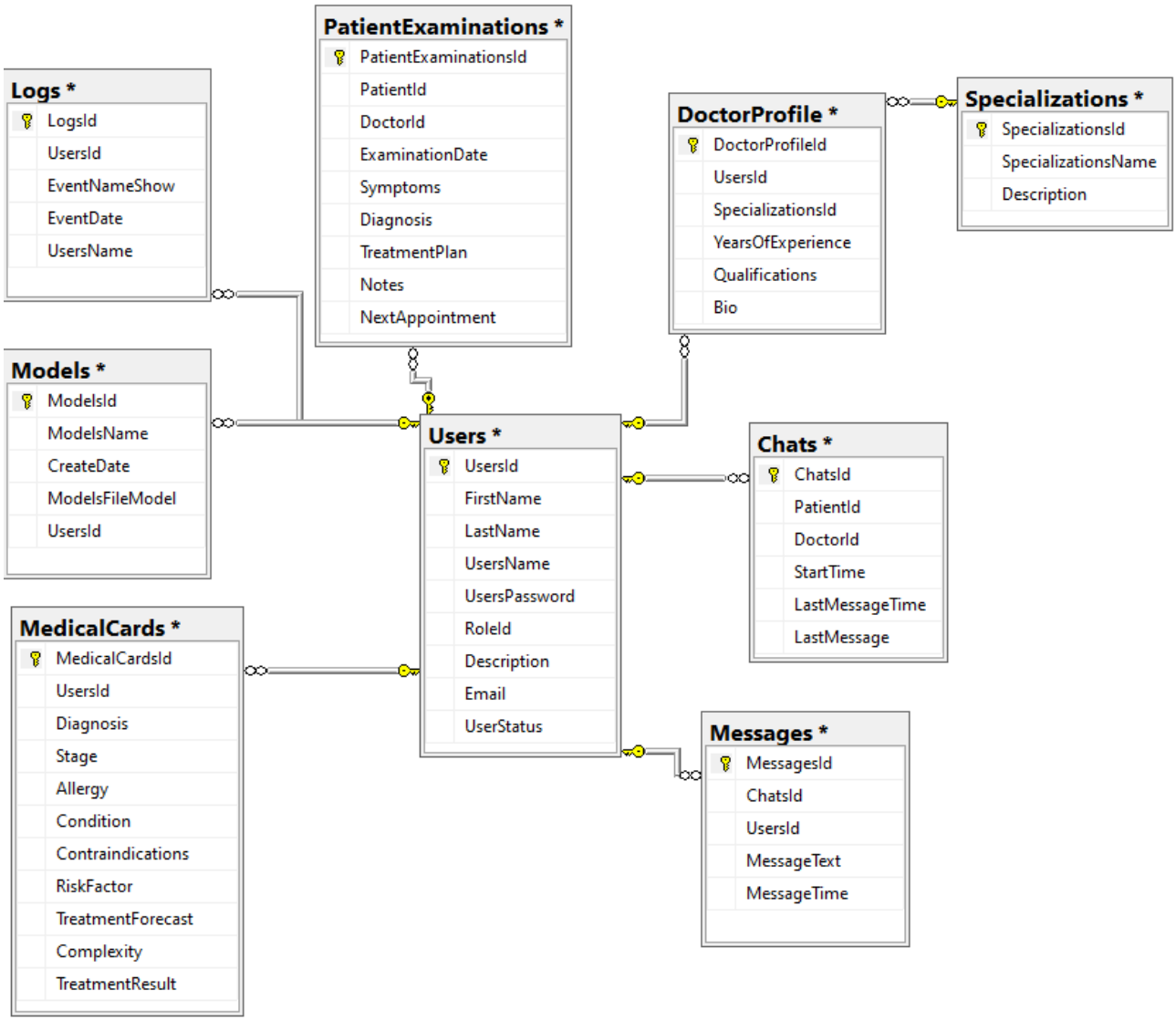


Рис. 2.8 ER діаграма бази даних

ER-діаграма є важливим інструментом для візуалізації структури бази даних аналітичної підсистеми телемедичної системи, що допомагає уникнути помилок під час розробки завдяки чіткому представленню даних і їх взаємозв'язків. Ця діаграма слугує надійною основою для проектування бази даних, забезпечуючи структурований підхід до зберігання та управління медичною інформацією, що є критично важливим для стабільної роботи системи. ER-діаграма сприяє реалізації надійної архітектури бази даних, яка дозволяє ефективно обробляти великі обсяги даних і підтримувати ключові функції моніторингу та управління станом здоров'я пацієнтів у телемедичній системі.

## 2.5 Архітектура аналітичної системи

Побудова архітектури аналітичної підсистеми забезпечує структурований підхід до обробки великих обсягів медичних даних, управління інформаційними потоками між користувачами системи (пацієнтами, лікарями та адміністраторами) та надає інтегровану платформу для аналітичних і моніторингових процесів. Вибір трьохрівневої архітектури для розробки зумовлений необхідністю забезпечити масштабованість системи для обробки різнорідних потоків даних у реальному часі та захисту конфіденційної медичної інформації. Архітектура сприяє безпечному та зручному доступу до даних, дозволяючи легко інтегрувати додаткові функції та підтримувати високу продуктивність у багатокористувацькому середовищі.

Рівень бізнес-логіки відповідає за реалізацію основної функціональності аналітичної системи, що включає обробку даних та виконання бізнес-правил, необхідних для підтримки процесів телемедичного моніторингу. На цьому рівні зосереджені класи, які забезпечують обробку та захист даних, управління доступом до різних функцій системи та валідацію даних. Діаграма класів, що відображає цей рівень, наведена на рис. 2.9.

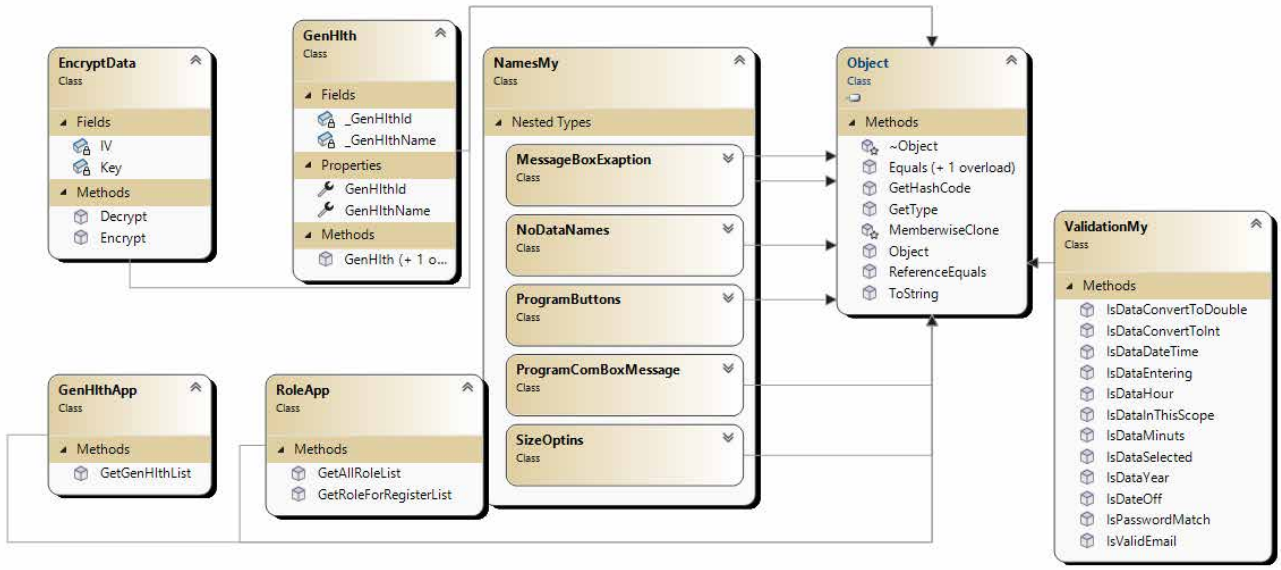


Рис. 2.9 Діаграма класів рівня бізнес-логіки

Рівень бізнес-логіки включає наступні основні класи:

– EncryptData відповідає за шифрування та розшифрування даних, що забезпечує захист конфіденційної інформації під час її передачі та зберігання в системі. Клас містить ключ для шифрування та методи для виконання основних криптографічних операцій;

– GenHilth зберігає ідентифікатор та ім’я користувача, що допомагає забезпечити доступ до профільної інформації користувачів у системі. Цей клас також має методи для генерації та зберігання даних, пов’язаних з ідентифікацією користувача;

– RoleApp забезпечує доступ до списку ролей у системі, включаючи методи для отримання переліку ролей, що використовуються при налаштуванні доступу та дозволів для різних категорій користувачів;

– NamesMy містить вкладені типи та константи, які використовуються для спрощення обробки помилок, відображення повідомлень та управління розмірами компонентів інтерфейсу. Це допомагає структурувати інформацію про повідомлення та забезпечити узгоджене відображення в системі;

– ValidationMy забезпечує функції валідації даних, такі як перевірка формату електронної пошти, коректності введення числових даних та інші методи перевірки. Це дозволяє контролювати правильність даних, введених користувачами, і забезпечити їх відповідність встановленим стандартам.

Ці класи надають необхідні функції для підтримки бізнес-логіки телемедичної системи, зокрема для управління безпекою, доступом і валідацією даних, що є критичними для забезпечення стабільної та захищеної роботи системи.

Рівень даних забезпечує доступ до інформації, що зберігається в базі даних, та надає засоби для роботи з медичними даними, інформацією про користувачів і повідомленнями. Цей рівень виконує функції взаємодії з базою даних, включаючи виконання запитів на вибірку, вставку, оновлення та

видалення даних, а також забезпечує централізоване управління цими операціями. Використання класів цього рівня дозволяє інкапсулювати логіку роботи з базою даних, підвищуючи гнучкість і безпеку системи. Діаграма класів, що представляє рівень даних, наведена на рис. 2.10.

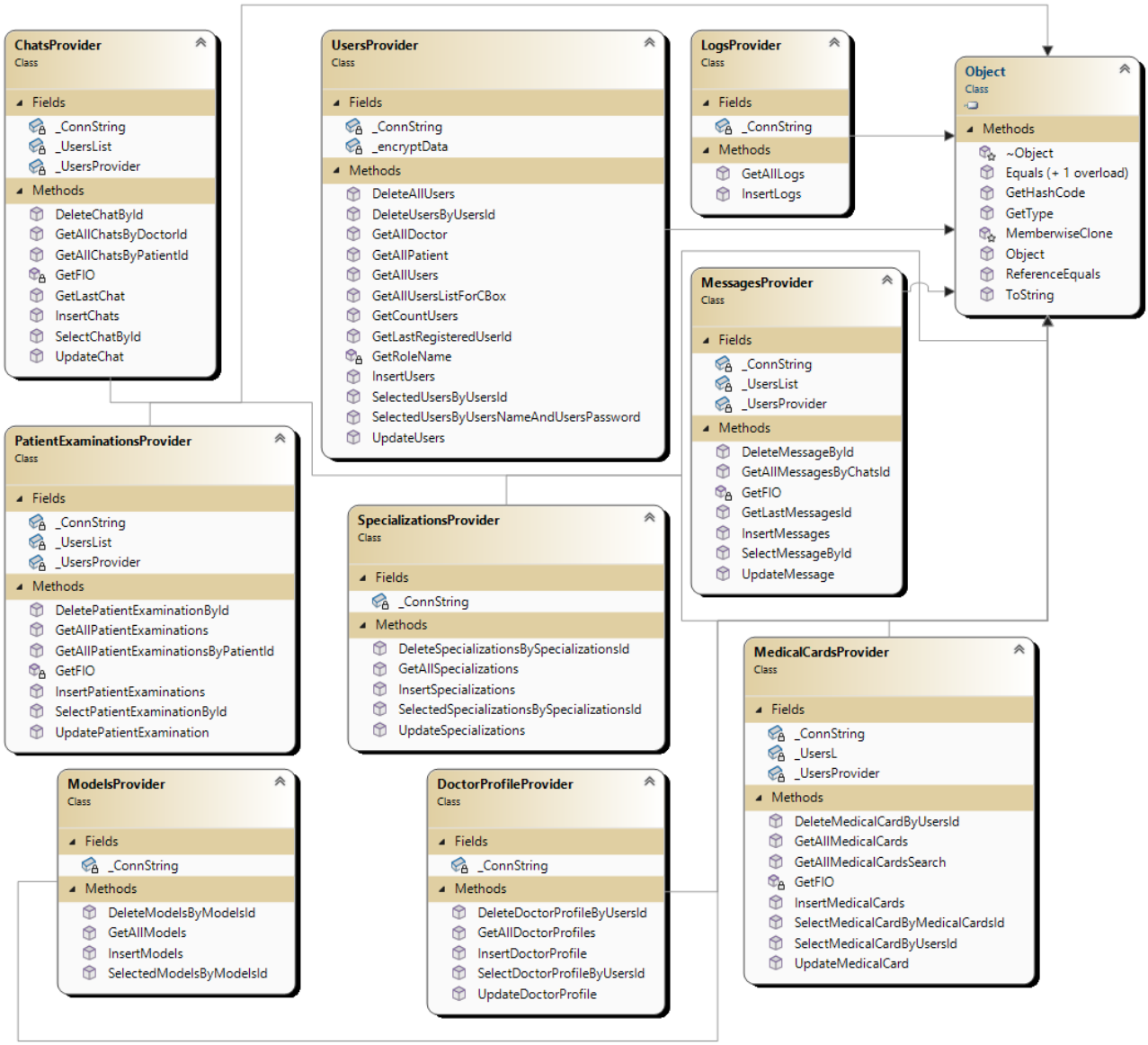


Рис. 2.10 Діаграма класів рівня даних

Даний рівень включає наступні класи:

– UsersProvider відповідає за обробку даних користувачів, надаючи методи для створення, редагування, видалення та вибірки користувачів з бази

даних. Він також забезпечує функції для отримання користувачів відповідно до різних критеріїв;

– LogsProvider призначений для обробки записів журналу подій, включаючи отримання й вставку нових записів, що забезпечує можливість відстежувати активність користувачів і системні події;

– MessagesProvider забезпечує функціональність для роботи з повідомленнями, надаючи методи для вставки, вибірки та оновлення повідомлень, що дозволяє підтримувати повний архів переписок у системі;

– PatientExaminationsProvider керує даними про медичні огляди пацієнтів, надаючи можливість отримувати, додавати та оновлювати записи про огляди, що дозволяє підтримувати медичну історію пацієнтів;

– MedicalCardsProvider відповідає за роботу з медичними картками, надаючи методи для вставки, оновлення та вибірки медичних даних пацієнтів, що є важливим для ефективного зберігання медичної інформації;

– SpecializationsProvider забезпечує обробку даних про спеціалізації лікарів, дозволяючи додавати, оновлювати та вибирати спеціалізації, що полегшує адміністрування медичних напрямків у системі;

– ModelsProvider керує даними про моделі машинного навчання, забезпечуючи можливість додавання, видалення та вибірки моделей, що використовуються для аналізу та прогнозування в системі;

– DoctorProfileProvider забезпечує доступ до профілів лікарів, дозволяючи додавати та оновлювати інформацію про лікарів, включаючи їх досвід, кваліфікації та спеціалізації.

Ці класи є основою для доступу до бази даних, забезпечуючи централізоване управління всіма операціями з даними в телемедичній аналітичній системі та підтримуючи необхідний рівень безпеки та ефективності роботи з інформаційними потоками.

Рівень користувацького інтерфейсу забезпечує зручну взаємодію користувачів із системою, надаючи доступ до функціональності через різноманітні форми та елементи управління. Цей рівень об’єднує класи, що

відповідають за відображення інформації, введення даних, а також за виконання основних операцій, доступних лікарям, адміністраторам та пацієнтам. Кожен клас інтерфейсу відповідає за окремий функціональний модуль системи, дозволяючи користувачам легко взаємодіяти з медичними картками, переглядати діагнози, спілкуватися в чаті та здійснювати інші дії. Діаграма класів, що представляє рівень користувацького інтерфейсу, наведена на рис. 2.11.

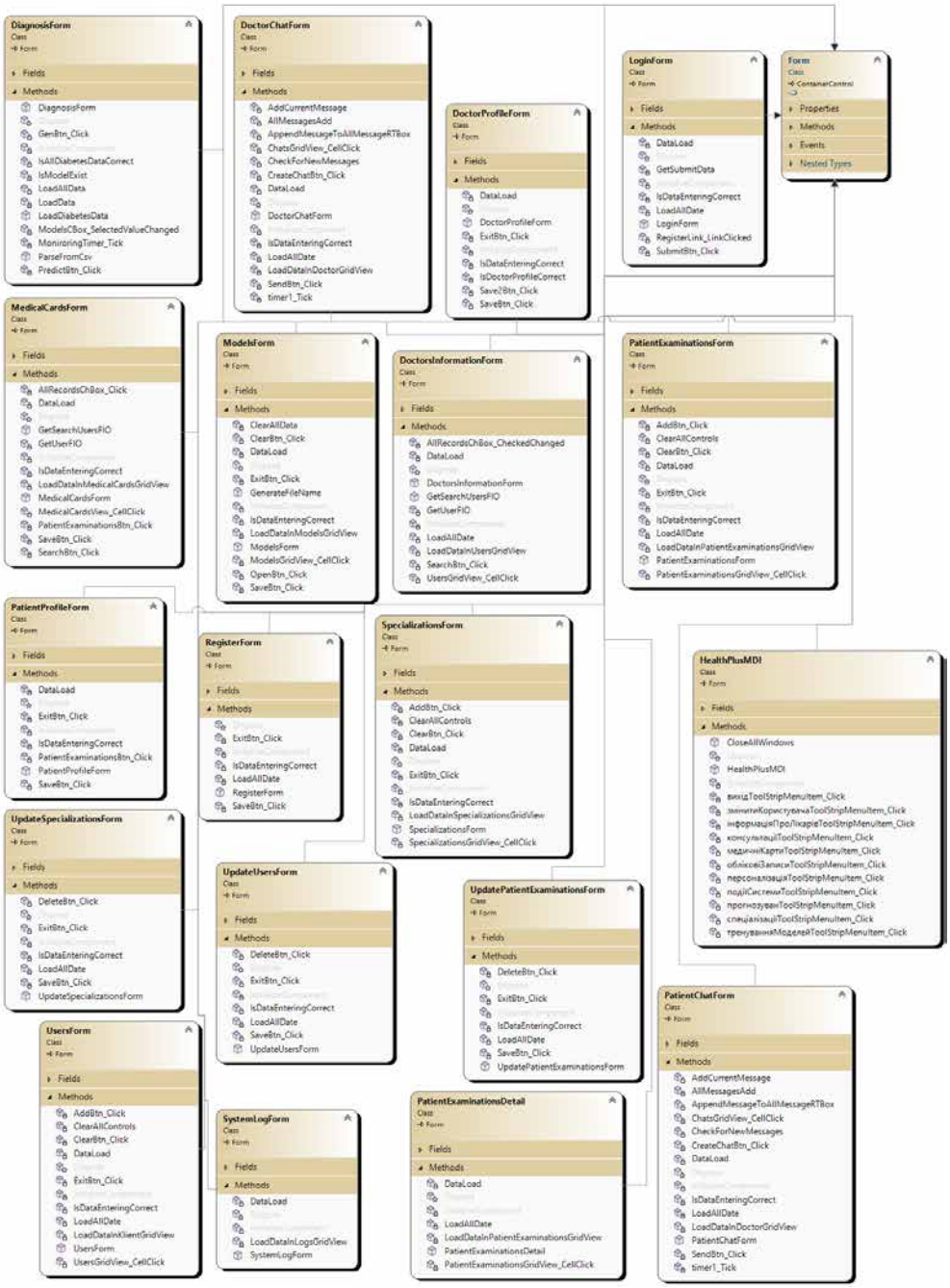


Рис. 2.11 Діаграма класів рівня користувацького інтерфейсу

Даний рівень включає наступні класи:

– DiagnosisForm – форма, що забезпечує доступ до інформації про діагнози пацієнтів, включаючи методи для завантаження, редагування та пошуку даних;

– DoctorChatForm – клас, що відповідає за інтерфейс чату лікаря з пацієнтами, дозволяючи лікарям переглядати повідомлення, відповідати на запити та керувати спілкуванням;

– DoctorProfileForm – форма для перегляду та редагування профілю лікаря, яка включає функції для оновлення інформації про спеціалізації та кваліфікації лікаря;

– LogForm – форма журналу, що надає можливість перегляду записів системних подій, забезпечуючи адміністраторам інструменти для моніторингу активності в системі;

– MedicalCardForm – клас для управління медичними картками пацієнтів, що дозволяє завантажувати, редагувати та зберігати медичні записи;

– HealthPlusMDI – головне вікно системи, яке надає користувачам можливість навігації між основними модулями інтерфейсу;

– PatientExaminationForm та PatientExaminationDetail – форми для роботи з оглядами пацієнтів, які забезпечують доступ до загальної інформації та деталей оглядів;

– SpecializationsForm – форма для управління спеціалізаціями, яка дозволяє адміністраторам додавати та редагувати напрями медицини в системі;

– RegisterForm та UpdateUserForm – форми для реєстрації нових користувачів та оновлення їхніх даних, що дозволяють ефективно керувати обліковими записами;

– SystemLogForm – клас, який надає інтерфейс для перегляду системних логів, полегшуючи контроль за подіями та діями користувачів.

Перелічені класи надають функціональний та інтуїтивний інтерфейс, що дозволяє користувачам виконувати свої завдання з моніторингу, діагностики та

управління медичною інформацією, забезпечуючи цілісну взаємодію з аналітичною підсистемою телемедичної системи.

## 2.6 Висновок

У рамках даного розділу проведено комплексний аналіз і обґрунтування вибору технологій та архітектури для розробки аналітичної підсистеми телемедичної системи моніторингу стану здоров’я. Було проаналізовано різні мови програмування, серед яких обрано C# як оптимальний вибір для створення надійного та продуктивного програмного забезпечення. Для реалізації проєкту обрано середовище розробки Visual Studio 2022, що забезпечує максимальну інтеграцію з C# та підтримку платформи .NET. Серед систем управління базами даних розглянуто MS SQL Server, PostgreSQL та SQLite, з яких обрано MS SQL Server для надійного зберігання медичних даних, зручної інтеграції з іншими компонентами системи та забезпечення безпеки даних.

Проведено аналіз вимог до аналітичної підсистеми та розроблено сценарії використання, які детально описують взаємодію користувачів системи – адміністраторів, лікарів і пацієнтів. На основі цього було створено діаграми діяльності та послідовності, що демонструють основні процеси системи, такі як авторизація, навчання моделей для діагностики діабету, обробка медичних карток лікарем та перегляд діагнозів пацієнтом. Також здійснено проєктування бази даних для зберігання медичних даних, описано структуру таблиць і побудовано ER-діаграму, яка відображає зв’язки між сутностями. На завершення розглянуто трьохрівневу архітектуру системи та детально описано кожен рівень (даних, бізнес-логіки та інтерфейсу), що дозволяє забезпечити гнучкість, масштабованість і продуктивність системи.

Отримані результати, включаючи обґрунтований вибір технологій, розроблені діаграми процесів, структуру бази даних та архітектуру, створюють

надійний фундамент для подальшої реалізації, використання та тестування аналітичної системи.

# 3 РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ АНАЛІТИЧНОЇ СИСТЕМИ

## 3.1 Розробка ключових модулів аналітичної системи

Аналітична підсистема управління інформаційними потоками телемедичної системи моніторингу стану здоров'я складається з ряду функціональних модулів, які забезпечують збирання, обробку, зберігання та аналіз медичних даних. Для забезпечення коректної роботи з даними розроблені класи та методи, що відповідають за різні аспекти обробки інформації, особливо на рівні взаємодії з базою даних. Кожен метод виконує чітко визначену функцію, що дозволяє ефективно обробляти великі обсяги даних і забезпечує високу швидкість виконання операцій. Серед реалізованих методів особливу роль відіграють ті, що забезпечують маніпуляції з медичними картками пацієнтів – ключовим елементом телемедичної системи.

Одним із таких методів є метод додавання нових записів у таблицю медичних карток (рис. 3.1).

```
public void InsertMedicalCards(int UsersId, string Diagnosis, string Stage, string Allergy,
  string Condition, string Contraindications,
  string RiskFactor, string TreatmentForecast, string Complexity, string TreatmentResult) {
  string sqlString = "INSERT INTO MedicalCards (UsersId, Diagnosis, Stage, Allergy, Condition, " +
    "Contraindications, RiskFactor, TreatmentForecast, Complexity, TreatmentResult) " +
    "Values (@UsersId, @Diagnosis, @Stage, @Allergy, @Condition, @Contraindications, " +
    "@RiskFactor, @TreatmentForecast, @Complexity, @TreatmentResult)";

  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@UsersId", UsersId);
      cmd.Parameters.AddWithValue("@Diagnosis", Diagnosis);
      cmd.Parameters.AddWithValue("@Stage", Stage);
      cmd.Parameters.AddWithValue("@Allergy", Allergy);
      cmd.Parameters.AddWithValue("@Condition", Condition);
      cmd.Parameters.AddWithValue("@Contraindications", Contraindications);
      cmd.Parameters.AddWithValue("@RiskFactor", RiskFactor);
      cmd.Parameters.AddWithValue("@TreatmentForecast", TreatmentForecast);
      cmd.Parameters.AddWithValue("@Complexity", Complexity);
      cmd.Parameters.AddWithValue("@TreatmentResult", TreatmentResult);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис. 3.1 Код методу «InsertMedicalCards»

Метод реалізовано для забезпечення інтеграції даних, що зберігаються в системі, з телемедичними показниками пацієнтів, відстеження історії хвороб та урахування індивідуальних факторів ризику. Метод приймає параметри, які включають унікальний ідентифікатор пацієнта, діагноз, стадію захворювання, алергічні реакції, стан здоров'я, протипоказання, фактори ризику, прогноз лікування, рівень складності випадку та результат лікування.

У рамках функціонування методу відбувається побудова SQL-запиту для вставки нових даних у відповідну таблицю бази даних. Метод налаштовує зв'язок із базою даних, передає параметри запиту, відкриває з'єднання, виконує SQL-команду, після чого завершує роботу з'єднання, забезпечуючи таким чином безпеку і цілісність даних. Такий підхід дозволяє уникнути помилок та забезпечити високу надійність обробки даних, що є критично важливим в медичній сфері.

Метод отримання всіх повідомлень за певним ідентифікатором чату виконує вибірку з бази даних для збору інформації про повідомлення, упорядкованої за часом (рис. 3.2).

```
public List<Messages> GetAllMessagesByChatsId(int ChatsId) {
  string sqlString = "SELECT * FROM Messages WHERE ChatsId=" + ChatsId + " ORDER BY MessageTime ASC";
  _UsersList = _UsersProvider.GetAllUsers();
  List<Messages> listAllMessages = new List<Messages>();
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
      conn.Open();
      using (SqlDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Messages oneMessage = new Messages();
          oneMessage.MessagesId = Convert.ToInt32(reader["MessagesId"]);
          oneMessage.ChatsId = Convert.ToInt32(reader["ChatsId"]);
          oneMessage.UsersId = Convert.ToInt32(reader["UsersId"]);
          oneMessage.MessageText = reader["MessageText"].ToString();
          oneMessage.MessageTime = Convert.ToDateTime(reader["MessageTime"]);
          oneMessage.FIO = GetFIO(oneMessage.UsersId, _UsersList);
          listAllMessages.Add(oneMessage);
        }
      }
      conn.Close();
    }
  }
```

Рис. 3.2 Код методу «GetAllMessagesByChatsId»

Після формування SQL-запиту для вибірки повідомлень за переданим параметром ChatsId, метод ініціалізує список користувачів за допомогою іншого класу, який забезпечує доступ до даних користувачів. Потім створюється порожній список для зберігання об’єктів повідомлень.

Відкривши підключення до бази даних, метод виконує запит і обробляє отримані результати за допомогою SqlDataReader. Під час ітерації через отримані рядки, кожен запис про повідомлення створюється як окремий об’єкт класу Messages. Параметри об’єкта, як-от MessagesId, ChatsId, UsersId, текст повідомлення та час надсилання, витягуються з результатів запиту і зберігаються в полях цього об’єкта. Додатково для кожного повідомлення викликається метод GetFIO, який визначає ініціали користувача на основі його ідентифікатора та списку користувачів. Після завершення обробки всіх записів метод перевіряє, чи список повідомлень залишився порожнім.

Метод на рис. 3.3, призначений для оновлення даних обстеження пацієнта реалізований для редагування запису в базі даних про медичний огляд.

```
public void UpdatePatientExamination(int PatientId, int DoctorId, DateTime ExaminationDate, string Symptoms,
  string Diagnosis, string TreatmentPlan, string Notes, DateTime NextAppointment,
  int PatientExaminationsId) {
  string sqlString = "UPDATE PatientExaminations SET " +
                     "PatientId=@PatientId, " +
                     "DoctorId=@DoctorId, " +
                     "ExaminationDate=@ExaminationDate, " +
                     "Symptoms=@Symptoms, " +
                     "Diagnosis=@Diagnosis, " +
                     "TreatmentPlan=@TreatmentPlan, " +
                     "Notes=@Notes, " +
                     "NextAppointment=@NextAppointment " +
                     "WHERE PatientExaminationsId=@PatientExaminationsId";
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@PatientId", PatientId);
      cmd.Parameters.AddWithValue("@DoctorId", DoctorId);
      cmd.Parameters.AddWithValue("@ExaminationDate", ExaminationDate);
      cmd.Parameters.AddWithValue("@Symptoms", Symptoms);
      cmd.Parameters.AddWithValue("@Diagnosis", Diagnosis);
      cmd.Parameters.AddWithValue("@TreatmentPlan", TreatmentPlan);
      cmd.Parameters.AddWithValue("@Notes", Notes);
      cmd.Parameters.AddWithValue("@NextAppointment", NextAppointment);
      cmd.Parameters.AddWithValue("@PatientExaminationsId", PatientExaminationsId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис. 3.3 Код методу «UpdatePatientExamination»

Він приймає параметри, які містять ідентифікатори пацієнта та лікаря, дату обстеження, симптоми, діагноз, план лікування, додаткові нотатки, наступну дату прийому та ідентифікатор обстеження для оновлення конкретного запису. SQL-запит формує оновлення, вказуючи всі відповідні поля для внесення змін.

Метод використовує об’єкт SqlConnection для підключення до бази даних, а також SqlCommand для виконання SQL-запиту. Кожен параметр, який передається у метод, додається до SQL-команди за допомогою AddWithValue, що зменшує ризик SQL-ін’єкцій і дозволяє безпечно передавати значення в запит. Після налаштування всіх параметрів метод відкриває підключення до бази даних і виконує команду ExecuteNonQuery, яка вносить зміни до відповідного запису обстеження пацієнта. Після завершення операції з’єднання з базою даних закривається, забезпечуючи цілісність і завершеність транзакції.

На рис. 3.4 приведено метод, який реалізований для оновлення інформації про профіль лікаря у базі даних, надаючи можливість редагувати такі дані, як спеціалізація, стаж роботи, кваліфікація та коротка біографія. Метод приймає відповідні параметри, включаючи ідентифікатор спеціалізації, кількість років досвіду, кваліфікацію, описову інформацію про лікаря та унікальний ідентифікатор користувача, за яким ідентифікується запис у базі даних.

```
public void UpdateDoctorProfile(int SpecializationsId, int YearsOfExperience,
  string Qualifications, string Bio, int UsersId) {
  string sqlString = "UPDATE DoctorProfile SET SpecializationsId=@SpecializationsId, " +
    "YearsOfExperience=@YearsOfExperience, Qualifications=@Qualifications," +
    " Bio=@Bio WHERE UsersId=@UsersId";
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@SpecializationsId", SpecializationsId);
      cmd.Parameters.AddWithValue("@YearsOfExperience", YearsOfExperience);
      cmd.Parameters.AddWithValue("@Qualifications", Qualifications);
      cmd.Parameters.AddWithValue("@Bio", Bio);
      cmd.Parameters.AddWithValue("@UsersId", UsersId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис. 3.4 Код методу «UpdateDoctorProfile»

Запит на оновлення створюється з використанням параметризованої SQL-команди, що мінімізує ризики SQL-ін’єкцій та забезпечує безпеку даних. Спочатку створюється підключення до бази даних через SqlConnection, і SQL-команда конфігурується із вказівкою на тип команди як текстовий запит. До запиту додаються параметри за допомогою методу AddWithValue, що дозволяє передати конкретні значення, які потрібно внести в базу.

Після налаштування всіх значень метод відкриває з’єднання з базою даних, виконує запит на оновлення запису через ExecuteNonQuery, і, після завершення транзакції, з’єднання закривається. Цей підхід забезпечує точне і надійне оновлення інформації про лікаря, дозволяючи легко змінювати ключові професійні атрибути лікарського профілю в межах бази даних.

Для зручного управління інформацією про пацієнтів у телемедичній системі розроблена форма «Медичні карти», яка приведена на рис. 3.5.

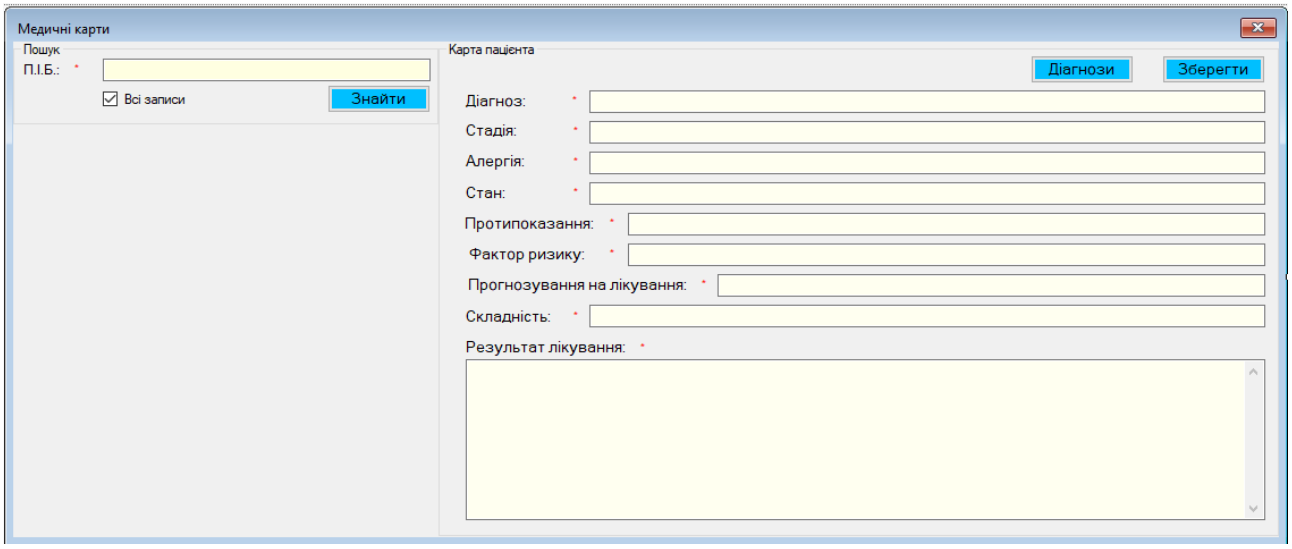


Рис. 3.5 Форма «Медичні карти»

Форма включає функцію пошуку за П.І.Б. пацієнта з можливістю перегляду всіх записів, що дозволяє лікарю швидко знайти потрібну картку. У правій частині форми містяться поля для введення та редагування детальної інформації про пацієнта, зокрема діагнозу, стадії захворювання, алергії, стану здоров’я, протипоказань, факторів ризику, прогнозу лікування, складності

випадку та результату лікування. Для збереження змін або вибору діагнозу передбачені відповідні кнопки, що забезпечує зручний інтерфейс для управління медичними даними.

Метод на рис. 3.6 виконує пошук медичних карток на основі введених користувачем даних у поле пошуку. Після натискання кнопки «Знайти» список _MedicalCardsList очищається, щоб видалити попередні результати, а потім заповнюється новими даними, отриманими з функції GetSearchUsersFIO, яка здійснює пошук за П.І.Б. введеним у текстове поле SearchFIOTBox.

```
private void SearchBtn_Click(object sender, EventArgs e) {
  _MedicalCardsList.Clear();
  _MedicalCardsList = GetSearchUsersFIO(SearchFIOTBox.Text);
  DataLoad();
}
```

Рис. 3.6 Код методу «SearchBtn_Click»

У кінці, метод викликає функцію DataLoad, що відповідає за відображення оновленого списку медичних карток у формі, забезпечуючи актуалізацію інформації для користувача.

Рис. 3.7 відображає метод, який активується під час вибору клітинки в таблиці медичних карток і призначений для відображення детальної інформації про вибрану картку. Спочатку перевіряється, чи перший запис у списку _MedicalCardsList не містить повідомлення про відсутність даних. Якщо дані дійсно присутні, метод отримує MedicalCardsId вибраного рядка і за цим ідентифікатором завантажує відповідну медичну картку за допомогою SelectMedicalCardByMedicalCardsId.

```
private void MedicalCardsView_CellClick(object sender, DataGridViewCellEventArgs e) {
  if (_MedicalCardsList[0].Message != NamesMy.NoDataNames.NoDataInMedicalCards) {
    int SelectedMedicalCardsId =
      Convert.ToInt32(MedicalCardsGridView[0, e.RowIndex].Value.ToString());
    _SelectedMedicalCards =
      _MedicalCardsProvider.SelectMedicalCardByMedicalCardsId(SelectedMedicalCardsId);
    DiagnosisTBox.Text = _SelectedMedicalCards.Diagnosis;
    StageTBox.Text = _SelectedMedicalCards.Stage;
    AllergyTBox.Text = _SelectedMedicalCards.Allergy;
    ConditionTBox.Text = _SelectedMedicalCards.Condition;
    ContraindicationsTBox.Text = _SelectedMedicalCards.Contraindications;
    RiskFactorTBox.Text = _SelectedMedicalCards.RiskFactor;
    TreatmentForecastTBox.Text = _SelectedMedicalCards.TreatmentForecast;
    ComplexityTBox.Text = _SelectedMedicalCards.Complexity;
    TreatmentResultTBox.Text = _SelectedMedicalCards.TreatmentResult;
  }
}
```

Рис. 3.7 Код методу «MedicalCardsView_CellClick»

Після цього інформація з вибраної картки, зокрема діагноз, стадія захворювання, алергії, стан, протипоказання, фактори ризику, прогноз лікування, рівень складності та результат лікування, відображається у відповідних текстових полях форми. Це забезпечує швидкий доступ до детальної інформації про пацієнта для подальшої роботи з його медичною карткою.

Метод «SaveBtn_Click» виконується при натисканні кнопки "Зберегти" і призначений для оновлення даних медичної картки пацієнта (рис. 3.8).

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    _MedicalCardsProvider.UpdateMedicalCard(DiagnosisTBox.Text, StageTBox.Text, AllergyTBox.Text,
      ConditionTBox.Text, ContraindicationsTBox.Text,
      RiskFactorTBox.Text, TreatmentForecastTBox.Text, ComplexityTBox.Text,
      TreatmentResultTBox.Text, _SelectedMedicalCards.MedicalCardsId);
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Було змінено інформацію у медичній карті пацієнта "
      + GetUserFIO(_SelectedMedicalCards.UsersId), DateTime.Now);
    MessageBox.Show("Дані успішно збережено!");
  }
}
```

Рис. 3.8 Код методу «SaveBtn_Click»

На початку він перевіряє правильність введених даних за допомогою методу IsDataEnteringCorrect. Якщо всі дані введено коректно, метод викликає функцію UpdateMedicalCard з провайдера медичних карток

_MedicalCardsProvider, передаючи всі необхідні поля, заповнені у відповідних текстових полях форми, разом з ідентифікатором вибраної медичної картки. Крім цього, в логах створюється запис про зміну інформації, в якому фіксується ідентифікатор користувача, що виконав дію, його П.І.Б. та час внесення змін, використовуючи метод InsertLogs з провайдера логів _LogsProvider. Після успішного збереження даних відображається повідомлення, яке інформує користувача про успішне збереження змін.

Форма «Діагностування діабету» призначена для комплексного збору даних, які можуть впливати на ризик розвитку діабету (рис. 3.9). Інтерфейс дозволяє користувачеві ввести різноманітну інформацію, включаючи фактори ризику, такі як артеріальний тиск, рівень холестерину, індекс маси тіла, наявність шкідливих звичок (куріння, вживання алкоголю) та рівень фізичної активності. Окрім того, форма враховує додаткові параметри, такі як частота споживання фруктів та овочів, наявність медичного страхування, частота звернень до лікаря та витрати на медичні послуги, що надає комплексне розуміння стану пацієнта.

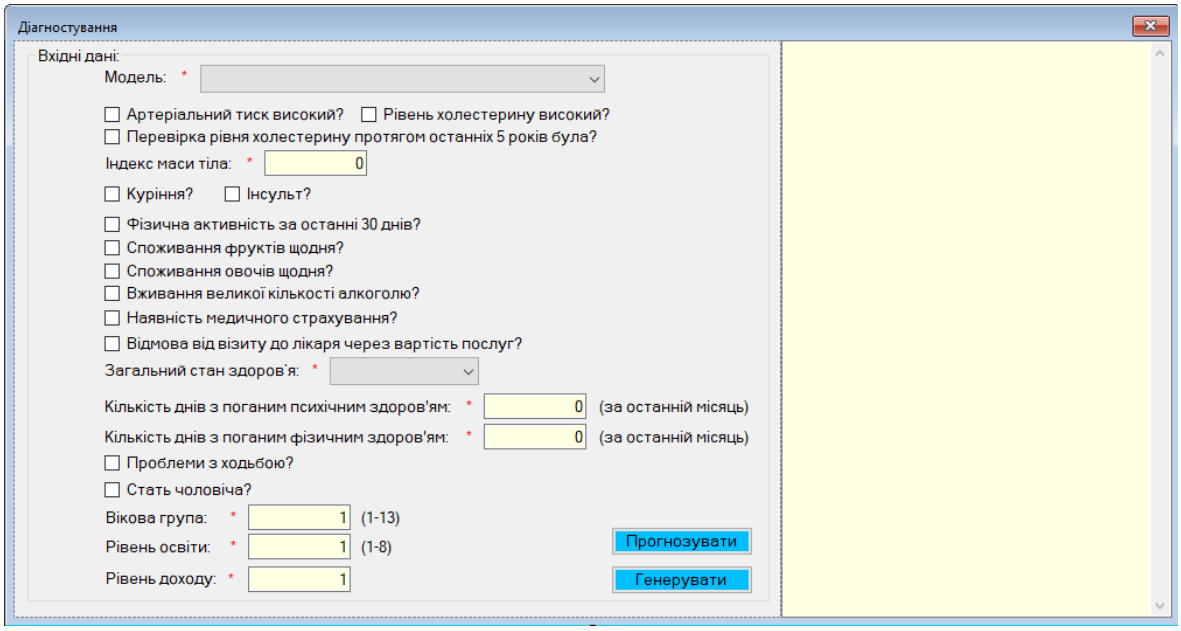


Рис. 3.9 Форма «Діагностування діабету»

Особливістю форми є кнопка «Прогнозувати», яка, на основі введених даних, дозволяє тестувати модель машинного навчання. Функціональність цієї кнопки включає можливість генерації випадкових тестових прикладів для перевірки моделі, що дозволяє оцінити ймовірність розвитку діабету на основі заданих параметрів. Такий підхід сприяє перевірці точності моделі в різних умовах, підвищуючи її надійність і адаптивність до різноманітних сценаріїв. Форма також містить кнопку «Генерувати» для створення звітів чи подальшого аналізу на основі отриманих результатів, що робить її ефективним інструментом як для діагностики, так і для тестування машинного навчання.

Фрагмент коду на рис. 3.10 виконує прогнозування стану здоров'я пацієнта на основі введених даних, використовуючи модель машинного навчання.

```
// Прогнозування на основі введених даних
var prediction = predictionEngine.Predict(testDiabetesData);
// Формування та виведення результату прогнозування
var answer = new StringBuilder();
answer.AppendLine("\r\n--- Прогнозування ---");
answer.AppendLine($"\r\nРекомендований клас: {(prediction.Prediction ? "Є ризик" : "Немає ризику")}");
answer.AppendLine($"\r\nЙмовірність наявності ризику: {prediction.Probability:P2}");
// Логування прогнозу для обліку
_LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
    "Було проведено прогнозування стану здоров'я для моделі " +
    ModelsCBox.Text, DateTime.Now);
// Додавання результату прогнозування до MonitoringTBox
MonitoringTBox.Text += answer.ToString();
```

Рис. 3.10 Фрагмент коду для прогнозування ліабету

На початку запускається процес прогнозування за допомогою predictionEngine, який отримує на вхід тестові дані testDiabetesData і формує передбачений результат. Далі, для відображення прогнозу створюється об'єкт StringBuilder, в який послідовно додаються рядки з інформацією про результати: зокрема, чи є ризик розвитку діабету та ймовірність цього ризику, форматована у відсотках.

Окрім цього, результат прогнозування логуються для обліку. За допомогою _LogsProvider здійснюється запис у журнал дій із зазначенням користувача, який ініціював прогнозування, моделі, що використовувалася, і

часу виконання операції. Нарешті, згенерований текстовий результат додається до поля MonitoringTBox, що дозволяє користувачеві переглянути прогноз та відповідні ймовірності ризику безпосередньо в інтерфейсі програми.

## 3.2 Аналіз результатів функціонального та компонентного тестування

Аналіз результатів функціонального та компонентного тестування є важливим етапом у забезпеченні якості програмного забезпечення, оскільки він дозволяє не лише виявити потенційні помилки, але й оцінити, наскільки продукт відповідає встановленим функціональним та нефункціональним вимогам. У процесі тестування для розробленої системи було зосереджено увагу на перевірці точності обробки введених користувачем медичних даних, коректності виконання основних операцій з інформаційними потоками, а також оцінці стійкості системи до підвищених навантажень і виникнення виняткових ситуацій. Окремий акцент зроблено на тестуванні компонентів, пов’язаних з машинним навчанням, де особливу увагу приділено точності прогнозування та стабільності роботи алгоритмів при роботі з випадковими тестовими даними.

Тестування форми «Авторизація» є важливим кроком для перевірки коректності функціонування системи доступу до телемедичної платформи, оскільки воно забезпечує точність і надійність обробки даних користувачів при вході в систему. Наприклад, у таблиці 3.1 представлені тестові сценарії, які охоплюють перевірку основних елементів інтерфейсу форми авторизації та дій користувача під час введення даних, включаючи перевірку обов’язкових полів «Логін» та «Пароль», коректність обробки кнопки «Підтвердити», а також перехід на форму реєстрації.

Таблиця 3.1

Тестові сценарії для форми «Авторизація»

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка проходження |
|---|---|---|---|---|
| 1 | Ввести правильні значення для полів «Логін» і «Пароль» та натиснути «Підтвердити» | Користувача успішно авторизовано, відкрито головне меню системи | Успішно авторизовано | Так |
| 2 | Не ввести значення в обов’язкові поля «Логін» або «Пароль» та натиснути «Підтвердити» | Виведено повідомлення про помилку, авторизацію не виконано | Повідомлення виведено | Так |
| 3 | Ввести некоректний логін або пароль та натиснути «Підтвердити» | Виведено повідомлення про невірні дані авторизації | Повідомлення виведено | Так |
| 4 | Натиснути «Підтвердити» без заповнення жодного з полів | Виведено повідомлення про обов’язковість заповнення полів | Повідомлення виведено | Так |
| 5 | Натиснути «Реєстрація» | Відкрито форму для реєстрації нового користувача | Форма реєстрації відкрилася | Так |

На рис. 3.12 зображена форма «Авторизація», яка демонструє приклад тестування для забезпечення надійності доступу користувачів до системи.

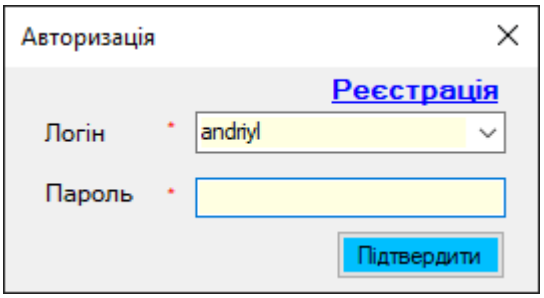


Рис. 3.11 Приклад тестування форми «Авторизації»

У табл. 3.2 наведено тестові сценарії, які охоплюють перевірку основних елементів інтерфейсу форми «Консультації з лікарями» та функцій, пов’язаних

з обробкою і відображенням повідомлень, а також роботою з чатами. Це включає перевірку коректності створення нового чату, надсилання повідомлень, відображення історії чату та функції пошуку лікаря.

Таблиця 3.2

Тестові сценарії для форми «Консультації з лікарями»

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка проходження |
|---|---|---|---|---|
| 1 | Вибрати лікаря зі списку та натиснути "Створити чат" | Новий чат створено, відображено в списку чатів | Чат створено | Так |
| 2 | Ввести текст повідомлення та натиснути "Відправити" | Повідомлення відправлено, відображено в полі чату | Повідомлення відправлено | Так |
| 3 | Натиснути "Відправити" без введення тексту | Виведено повідомлення про обов’язковість введення тексту | Повідомлення виведено | Так |
| 4 | Відкрити існуючий чат з лікарем | Відображено історію повідомлень у відповідному чаті | Історію відображено | Так |
| 5 | Використати функцію пошуку лікаря за іменем | У списку лікарів відображаються лише ті, що відповідають запиту | Результат коректний | Так |
| 6 | Надіслати повідомлення у чат і перевірити, чи з’явилося воно у хронологічному порядку | Повідомлення додано в кінець чату в хронологічному порядку | Повідомлення додано | Так |
| 7 | Перейти до іншого чату під час активної сесії з лікарем | Відображено історію повідомлень іншого чату | Історія відображена | Так |

На рис. 3.12 зображено інтерфейс форми «Консультації з лікарями», яка демонструє процес тестування різних сценаріїв для перевірки коректності функцій чату та забезпечення стабільності комунікації між пацієнтом і лікарем.

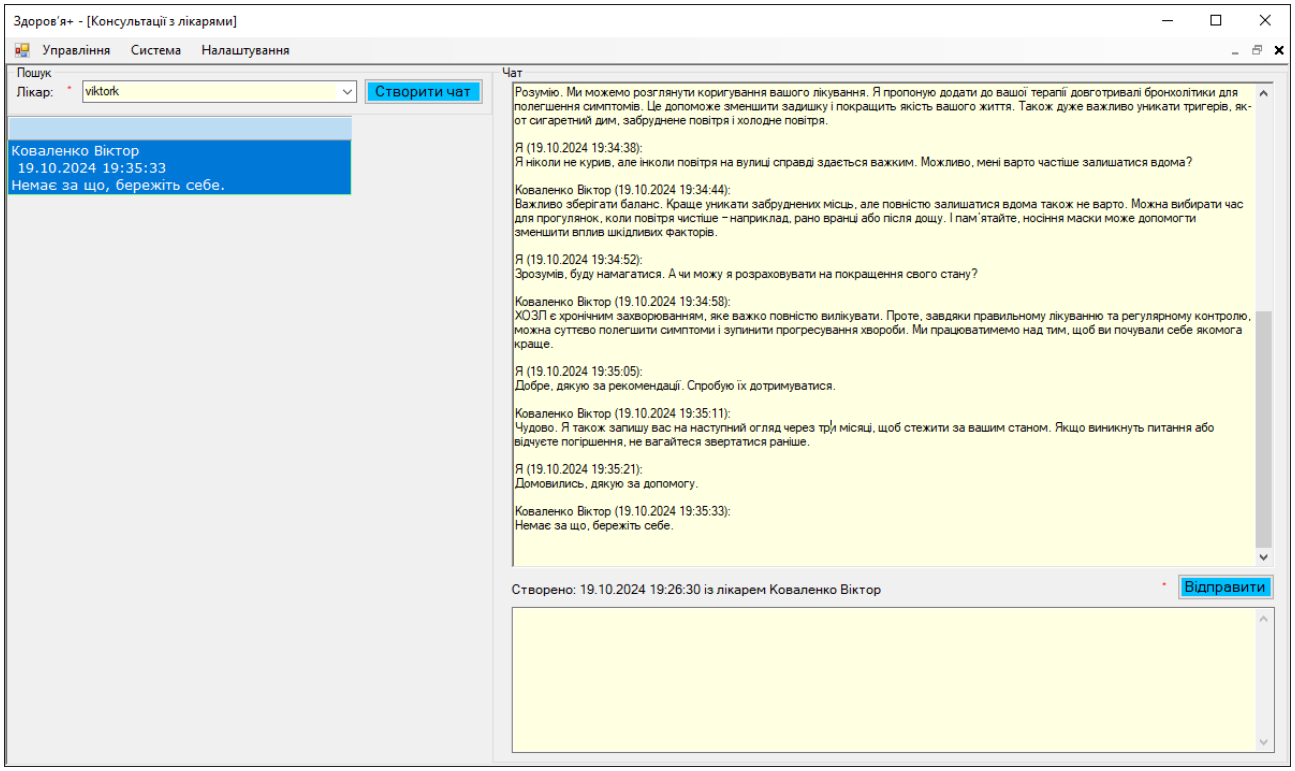


Рис. 3.12 Приклад тестування форми «Авторизації»

Тестування форми «Персоналізація» для ролі "пацієнт" є важливим етапом для забезпечення коректності введення та обробки персональних даних користувача, а також відповідності записів у медичній картці пацієнта. У табл. 3.3 наведені тестові сценарії, які охоплюють перевірку основних елементів інтерфейсу форми та взаємодії з ключовими полями.

Таблиця 3.3

Тестові сценарії для форми «Персоналізація»

| № з/п | Крок сценарію | Очікуваний результат | Отриманий результат | Відмітка проходження |
|---|---|---|---|---|
| 1 | Ввести коректні дані у всі поля форми та натиснути «Зберегти» | Дані успішно збережено, відображено повідомлення про успішне оновлення профілю | Профіль оновлено | Так |
| 2 | Залишити обов’язкові поля «Прізвище», «Ім’я» або «Пароль» порожніми | Виведено повідомлення про необхідність заповнення обов’язкових полів | Повідомлення виведено | Так |
| 3 | Ввести некоректний формат електронної пошти (наприклад, без символу «@») | Виведено повідомлення про некоректний формат електронної пошти | Повідомлення виведено | Так |
| 4 | Ввести різні значення у поля «Пароль» та "Підтвердити" | Виведено повідомлення про невідповідність паролів | Повідомлення виведено | Так |
| 5 | Натиснути «Зберегти» без змін у полях | Виведено повідомлення про відсутність змін для збереження | Повідомлення виведено | Так |
| 6 | Відредагувати дані в секції «Карта пацієнта» і натиснути «Зберегти» | Дані медичної картки успішно збережено, відображено повідомлення про оновлення | Картку оновлено | Так |
| 7 | Натиснути «Вийти» | Відбувається повернення на головний екран системи | Повернення на головний екран | Так |

На рис. 3.13 зображено інтерфейс форми «Персоналізація», який демонструє приклади тестових сценаріїв, що забезпечують коректність

введення персональних даних і підтримку точності записів медичної інформації.

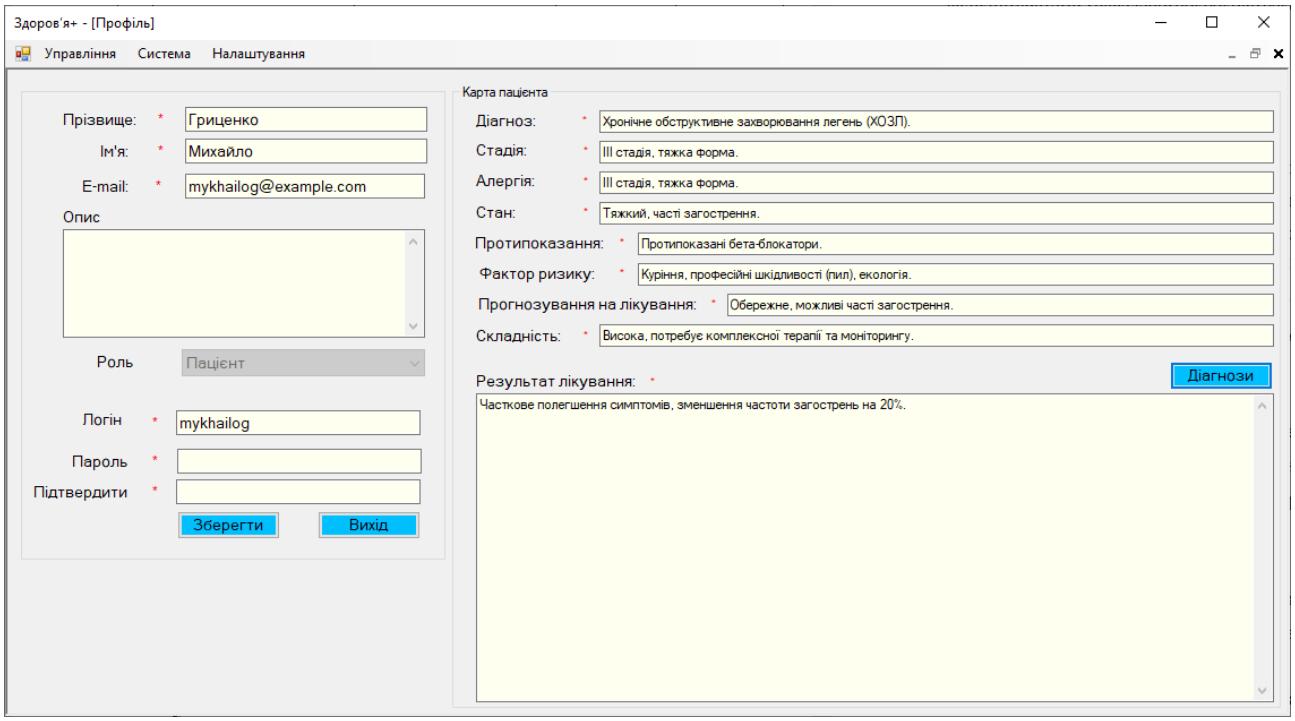


Рис. 3.13 Приклад тестування форми «Персоналізація»

Після завершення розробки та первинного тестування основних форм і функцій системи, наступним кроком є проведення автоматизованого тестування для перевірки її коректності, стабільності та відповідності вимогам. Одним із ключових інструментів, використаних для цього, є MS Test — потужна платформа для створення і виконання юніт-тестів у середовищі .NET, яка забезпечує автоматизацію тестових процесів і підвищення надійності результатів.

MS Test дозволяє створювати тестові методи для різних функцій системи, перевіряючи роботу як окремих компонентів, так і інтегрованих модулів. Це особливо корисно для тестування обробки даних в системі та роботи з базою даних, оскільки MS Test забезпечує можливість відтворення стандартних і крайніх сценаріїв взаємодії з даними. Зокрема, були розроблені тести для перевірки коректності методів збереження, оновлення та видалення інформації, що є критично важливим для медичної інформації, яку обробляє система.

Окрім цього, MS Test дозволяє легко виявляти та ізолювати помилки на ранніх етапах тестування, що знижує ризик появи непередбачуваних помилок на рівні кінцевого продукту. Завдяки можливості задавати очікувані результати та порівнювати їх з фактичними, MS Test допомагає впевнитися в тому, що система відповідає встановленим вимогам і надає стабільні результати в різних сценаріях використання. Це робить MS Test важливим інструментом для досягнення високої якості системи перед її передачею кінцевим користувачам.

Результати проведення автоматизованого тестування з використанням MS Test наведено на рис. 3.14. Як видно з представлених даних, було виконано 24 тестових сценарії, які охоплюють основні методи та функції системи, зокрема роботу з медичними картками, профілями лікарів та пацієнтів, функції створення та видалення чатів, а також інші ключові операції. Усі тести були успішно пройдені, що підтверджує коректність роботи системи: жодних помилок чи попереджень не виявлено, і кожен тестовий метод завершився з очікуваним результатом.

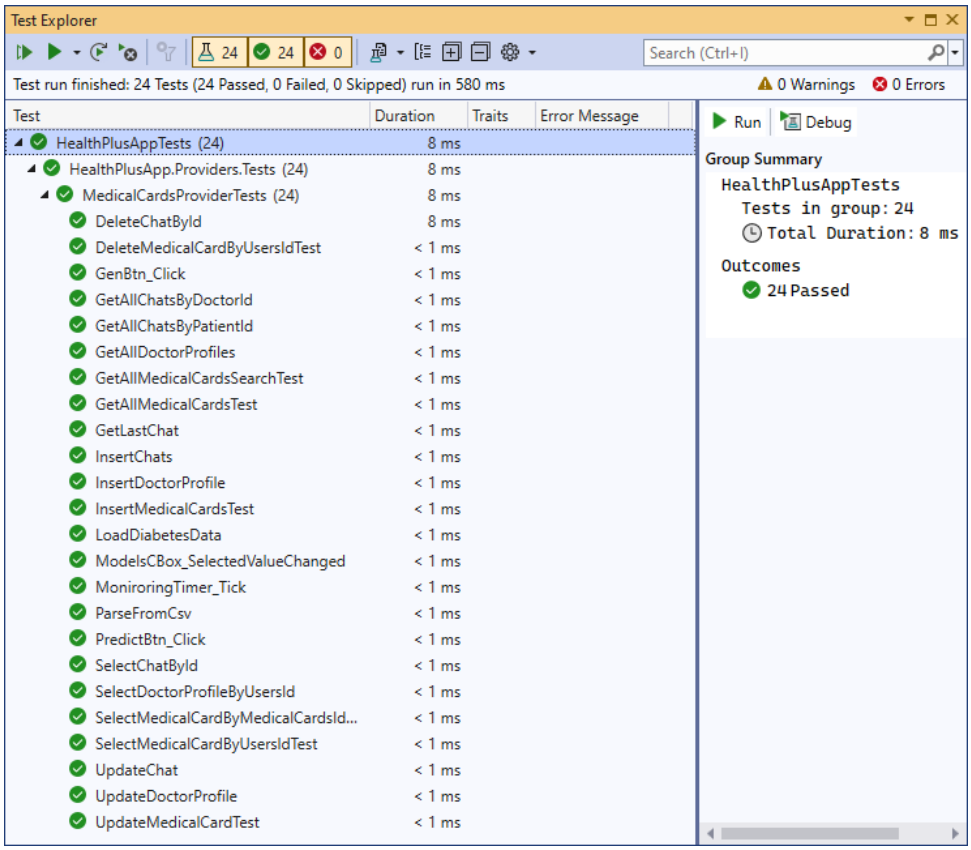


Рис. 3.14 Тестування системи з допомогою MS Test

Це підтверджує стабільність та функціональність системи в умовах різних сценаріїв використання, включаючи обробку даних і взаємодію між компонентами. Завдяки проведеному тестуванню можна впевнено стверджувати, що система готова до експлуатації та забезпечує необхідний рівень надійності для кінцевих користувачів. Автоматизація тестування сприяла швидкій перевірці всіх функцій, що значно оптимізує процес підтримки та розвитку системи в майбутньому.

## 3.3 Інструкція користувача

Інструкція користувача призначена для спрощення процесу взаємодії з інформаційною системою «Здоров'я+» та забезпечення її максимально ефективного використання. У цьому підрозділі детально описано основні функції системи, а також наведено покрокові рекомендації для виконання ключових операцій. Інструкція допоможе користувачам швидко освоїти інтерфейс і функціонал системи, забезпечуючи зручний доступ до інструментів моніторингу здоров'я, консультацій з лікарями та управління особистими даними.

**3.3.1 Мінімальні вимоги для запуску системи.** Для успішного запуску та стабільної роботи інформаційної системи «Здоров'я+» необхідно враховувати мінімальні вимоги до апаратного та програмного забезпечення. Дотримання цих вимог дозволить забезпечити належну продуктивність, стабільність та коректне функціонування всіх компонентів системи.

Мінімальні апаратні вимоги для запуску системи:

процесор з частотою не менше 2.5 ГГц (рекомендовано Intel Core i3 або аналогічний AMD Ryzen 3);

оперативна пам'ять: щонайменше 4 ГБ для базової роботи, рекомендовано 8 ГБ для більш стабільної роботи;

вільний дисковий простір: не менше 1 ГБ для зберігання даних та логів

системи;

стандартні периферійні пристрої, такі як клавіатура, миша та монітор з роздільною здатністю не менше 1024x768.

Програмні вимоги:

операційна система Windows 10 або новіша версія (рекомендовано Windows 11 для кращої сумісності);

встановлений .NET Framework версії 4.8 або вище для забезпечення повної функціональності всіх модулів системи.

Для встановлення системи «Здоров'я+» необхідно створити окрему папку на жорсткому диску, куди потрібно скопіювати всі файли з каталогу з інсталяційними файлами. Запуск програми здійснюється шляхом подвійного клацання на виконуваному файлі, що дозволить користувачу отримати доступ до інтерфейсу системи та розпочати роботу.

**3.3.2 Процес розгортання аналітичної системи.** Для успішного розгортання аналітичної системи «Здоров'я+» рекомендується виконати наступні кроки, щоб забезпечити стабільну роботу системи та доступ до всіх її функцій:

переконатись, що на комп'ютері встановлено актуальну версію .NET Framework, що забезпечує повну сумісність з додатком, розробленим на C#;

встановити MS SQL Server і налаштувати його для роботи з базою даних системи. Після встановлення необхідно запустити сервер і перевірити, що він активний і готовий до підключення;

розгорнути базу даних, створивши нову базу за допомогою SQL Server Management Studio (SSMS). Виконати надані SQL-скрипти для створення необхідних таблиць, індексів і залежностей. Параметри підключення, такі як ім'я сервера, назва бази даних та облікові дані, знадобляться для подальшої конфігурації системи;

скопіювати файли аналітичної системи до обраної директорії на комп'ютері, наприклад, «HealthPlusSystem». Для запуску програми достатньо двічі клацнути на виконуваному файлі (.exe), що дозволить перевірити доступ до інтерфейсу

системи;

налаштувати підключення до бази даних у конфігураційному файлі програми (app.config), де вказати рядок підключення із зазначенням адреси сервера, імені бази, облікових даних користувача та пароля. Це забезпечить коректну інтеграцію системи з базою даних;

провести тестування аналітичної системи, запустивши програму та перевіривши основні функції, включаючи введення, обробку та збереження медичних даних, а також доступ до аналітичних інструментів системи.

Виконавши всі вказані кроки, система «Здоров’я+» буде повністю готова до експлуатації та може бути передана кінцевим користувачам для ефективного моніторингу стану здоров’я.

**3.3.3 Використання системи для моніторингу здоров’я.** Для початку роботи з інформаційною системою «Здоров’я+», призначеною для моніторингу стану здоров’я користувачів, необхідно виконати кілька кроків, які забезпечать доступ до всіх її функцій. Система надає можливість не лише контролювати особисті медичні дані, але й отримувати консультації від лікарів, а також здійснювати аналіз стану здоров’я на основі наданих показників. Важливим етапом є процес авторизації, який забезпечує захист персональних даних та обмежує доступ до інформації лише авторизованим користувачам.

Форма авторизації є першим екраном, з яким взаємодіє користувач під час входу в систему «Здоров’я+» (рис. 3.15). Для входу користувач повинен вибрати свій логін із випадаючого списку або ввести його вручну, а також вказати пароль у відповідному полі. Поля "Логін" і "Пароль" є обов’язковими для заповнення, що відзначено червоними позначками. Після введення даних користувач натискає кнопку «Підтвердити», яка виконує перевірку введених даних на коректність та, у випадку успішної авторизації, відкриває доступ до основного інтерфейсу системи.

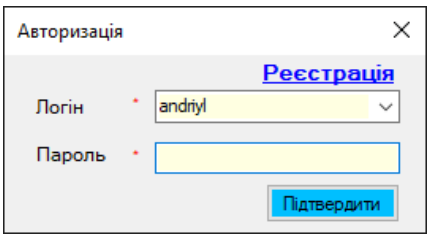


Рис. 3.15 Форма авторизації

У разі відсутності облікового запису, користувач може скористатися посиланням «Реєстрація», розташованим у верхній частині форми, що дозволить йому створити новий акаунт у системі.

Після успішної авторизації користувач, який виконує роль пацієнта, має можливість налаштувати свій профіль у системі «Здоров'я+» за допомогою спеціальної форми профілю (рис. 3.16). Форма «Профіль пацієнта» призначена для введення та редагування особистих даних, а також інформації, що стосується медичної картки пацієнта, що є ключовим елементом для ефективного моніторингу здоров'я.

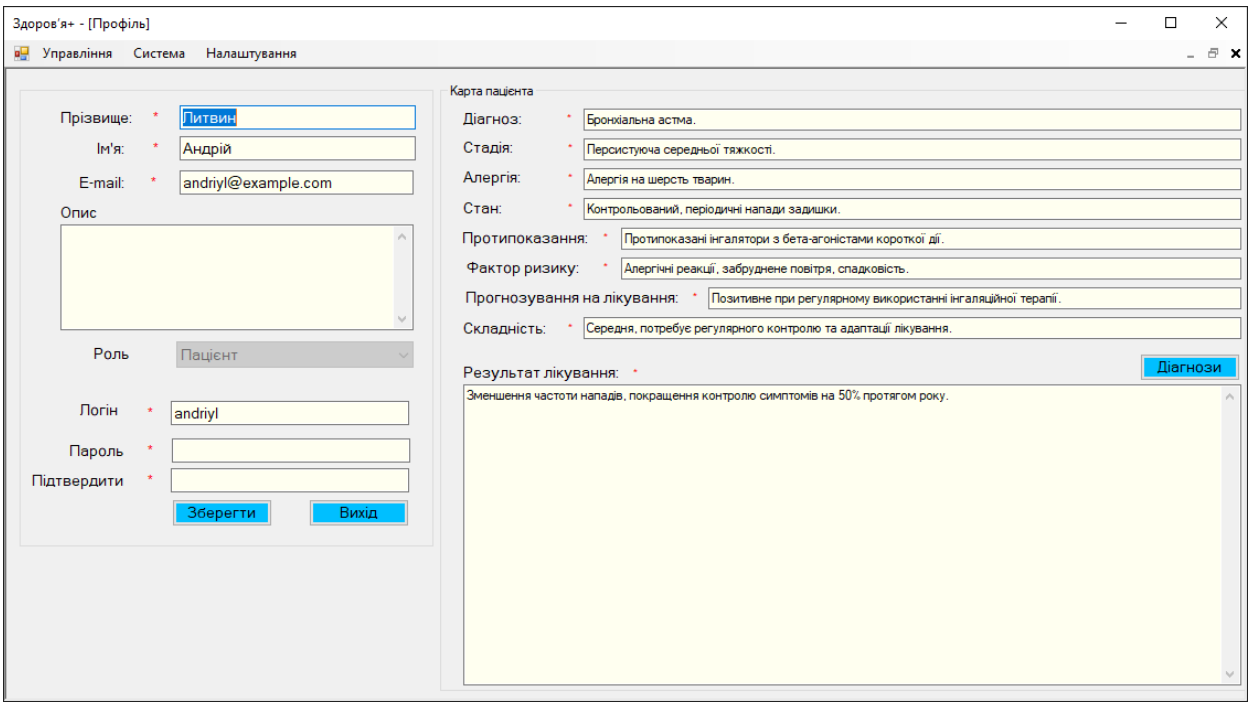


Рис. 3.16 Форма «Профіль пацієнта»

У лівій частині форми користувач може заповнити особисті дані: прізвище, ім’я, адресу електронної пошти, опис профілю, роль (автоматично визначена як «Пацієнт»), логін та пароль. Поля «Логін», «Пароль» і «Підтвердити» є обов’язковими для заповнення і позначені червоними позначками, що підкреслює їхню важливість для безпеки доступу. Після внесення змін користувач може зберегти інформацію, натиснувши кнопку «Зберегти», або вийти з профілю, натиснувши кнопку «Вихід». У правій частині форми розміщена секція «Карта пацієнта», яка містить детальну медичну інформацію.

Форма «Інформація про лікарів» призначена для надання користувачу детальних відомостей про кваліфікацію, спеціалізацію, досвід та біографічну інформацію лікарів, доступних у системі «Здоров’я+» (рис. 3.17). У лівій частині форми знаходиться поле для пошуку лікаря за прізвищем, що дозволяє користувачу швидко знайти потрібного спеціаліста за введеними даними. Після виконання пошуку в нижній частині лівої колонки відображається список лікарів, що відповідають запиту, з основною інформацією, такою як прізвище, ім’я та спеціальність.

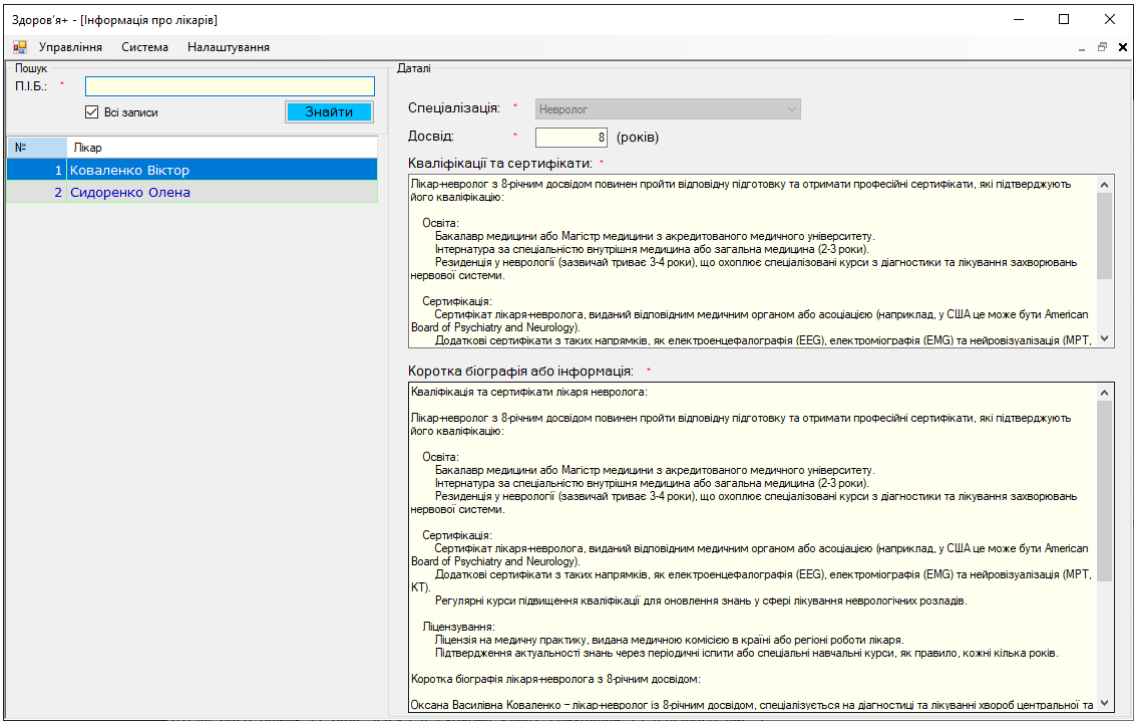


Рис. 3.17 Форма «Інформація про лікарів»

У правій частині форми відображаються деталі обраного лікаря. Тут користувач може ознайомитися з інформацією про спеціалізацію лікаря, його досвід у роках, кваліфікацію та сертифікати, отримані лікарем, а також із короткою біографією. Поля «Спеціалізація» і «Досвід» дозволяють зручно структурувати інформацію про профіль лікаря, а текстові поля для кваліфікацій і біографії містять детальний опис освіти, ліцензій, сертифікацій та досвіду лікаря у відповідній галузі медицини.

Форма «Профіль лікаря» надає можливість редагування та перегляду основних даних про лікаря в системі «Здоров'я+», що важливо для забезпечення актуальної інформації про медичних спеціалістів (рис. 3.18). У лівій частині форми користувач має можливість ввести та оновити базові дані: прізвище, ім'я, адресу електронної пошти, опис, логін і пароль. Поля «Логін», «Пароль» і «Підтвердити» є обов'язковими для заповнення, що забезпечує безпечний доступ до профілю лікаря. Кнопки «Зберегти» та «Вийти» дозволяють зберегти зміни або вийти з форми відповідно.

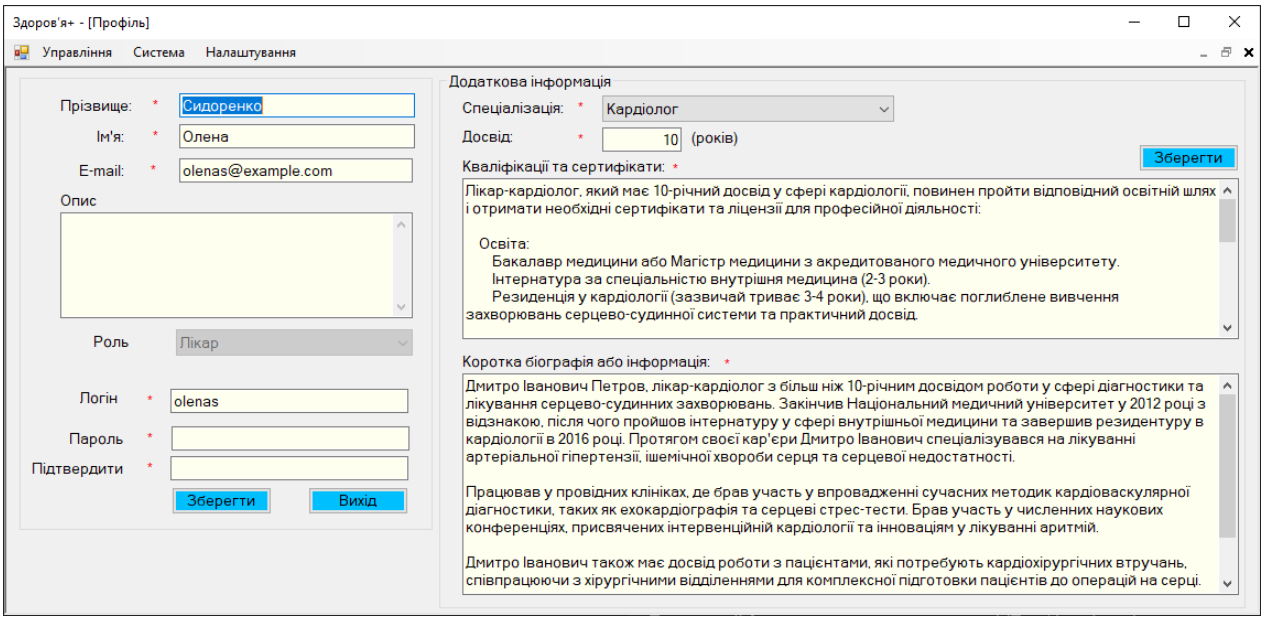


Рис. 3.18 Форма «Профіль лікара»

Форма «Медичні карти» надає лікарям зручний доступ до інформації про стан здоров'я пацієнтів у системі «Здоров'я+» (рис. 3.19). У лівій частині

форми розташоване поле пошуку, яке дозволяє швидко знайти пацієнта за прізвищем або переглянути всі записи, відмітивши відповідний чекбокс. Після виконання пошуку відображається список пацієнтів, де лікар може вибрати потрібного пацієнта для перегляду детальної інформації.

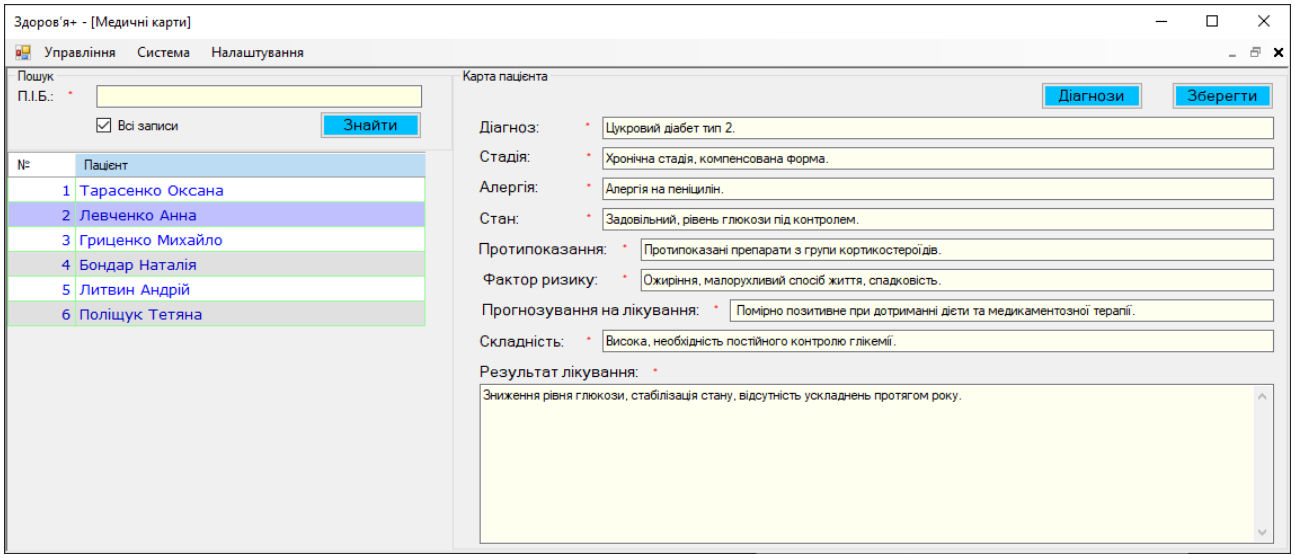


Рис. 3.19 Форма «Медичні карти»

Форма «Медичні карти» надає лікарям зручний доступ до інформації про стан здоров’я пацієнтів у системі «Здоров’я+». У лівій частині форми розташоване поле пошуку, яке дозволяє швидко знайти пацієнта за прізвищем або переглянути всі записи, відмітивши відповідний чекбокс. Після виконання пошуку відображається список пацієнтів, де лікар може вибрати потрібного пацієнта для перегляду детальної інформації.

Права частина форми містить картку пацієнта, де відображено медичні дані, такі як діагноз, стадія захворювання, алергічні реакції, поточний стан, протипоказання, фактори ризику, прогнозування на лікування, складність випадку та результат лікування. Поля заповнені детальною інформацією, що дозволяє лікарю мати повне уявлення про стан пацієнта та планувати подальші лікувальні заходи. Кнопка «Діагнози» надає можливість перегляду чи внесення додаткових діагнозів, а кнопка «Зберегти» дозволяє оновити медичні дані в системі після внесення змін.

Форма «Консультації з пацієнтами» у системі «Здоров’я+» дозволяє лікарям ефективно взаємодіяти з пацієнтами, надаючи можливість для текстових консультацій у режимі реального часу (рис. 3.20). У лівій частині форми розташоване поле пошуку, де лікар може ввести ім’я пацієнта для швидкого доступу до потрібного чату. Після пошуку в нижній частині лівої колонки відображається список активних чатів з пацієнтами, включаючи ім’я пацієнта, дату і час останнього повідомлення, що забезпечує швидку навігацію та доступ до історії взаємодії.

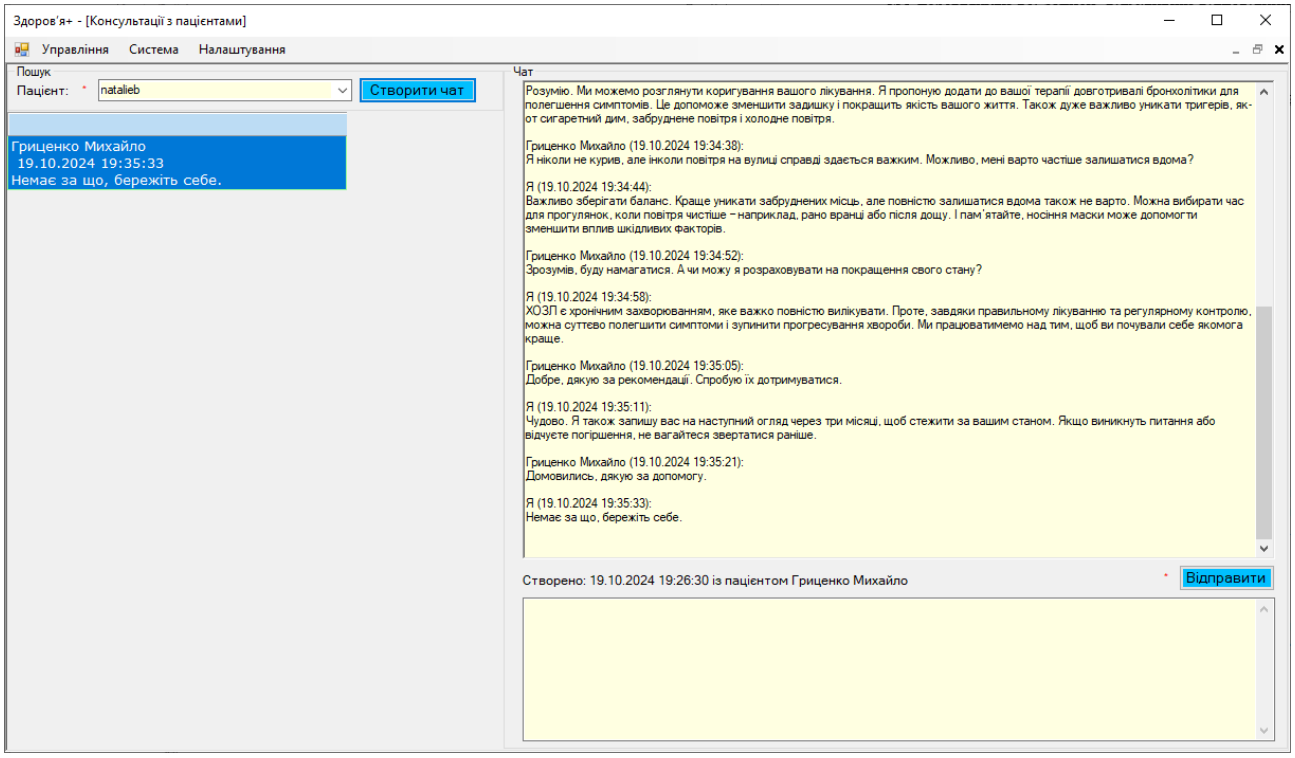


Рис. 3.20 Форма «Консультації з пацієнтами»

Права частина форми представляє собою поле чату, де відображається історія переписки з обраним пацієнтом у хронологічному порядку. Тут лікар може бачити як свої повідомлення, так і відповіді пацієнта, що дозволяє зберігати безперервний контекст консультацій. Це особливо корисно для надання рекомендацій, відповіді на запитання пацієнта та моніторингу виконання медичних приписів.

У нижній частині форми розміщене поле введення повідомлення, де лікар може надрукувати нове повідомлення та відправити його, натиснувши кнопку «Відправити». Такий підхід забезпечує оперативний зв’язок між лікарем і пацієнтом, сприяючи безперервному супроводу лікування та підтримці здоров’я пацієнта на належному рівні.

Форма «Тренування моделей» призначена для створення, налаштування та оцінки моделей машинного навчання, які використовуються для прогнозування ризику розвитку діабету у пацієнтів (рис. 3.21). У верхній частині форми знаходиться кнопка «Відкрити», яка дозволяє завантажити файл з набором даних у форматі CSV для тренування моделі. Після вибору файлу в полі «Файл» відображається шлях до завантаженого файлу, що забезпечує зручну перевірку використаного датасету.

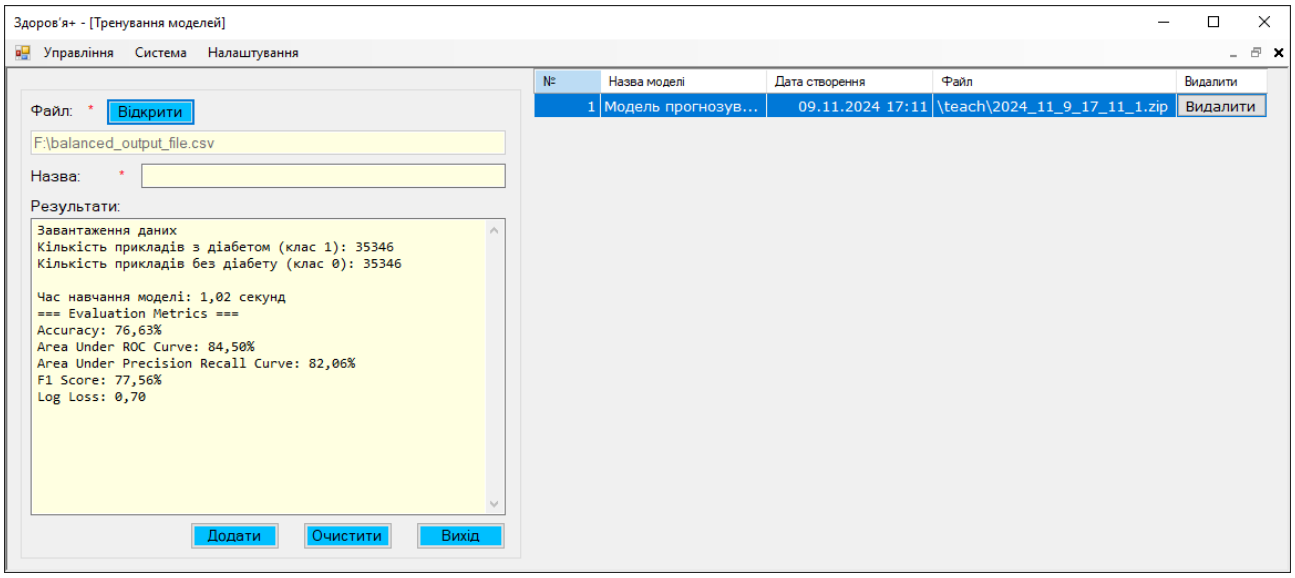


Рис. 3.21 Форма «Тренування моделей»

Після завантаження даних у полі «Результати» відображається інформація про кількість прикладів з діабетом та без діабету, час навчання моделі, а також ключові метрики оцінки, такі як точність (Accuracy), площа під кривою ROC (Area Under ROC Curve), точність прогнозування (Precision), повнота (Recall), F1 Score та Log Loss. Ці метрики дозволяють лікарям та

адміністраторам аналізувати якість моделі та приймати рішення про її готовність до використання в реальних умовах.

У правій частині форми розміщується список збережених моделей з інформацією про їх назву, дату створення, шлях до файлу з навченою моделлю та можливістю видалення. Для додавання нової моделі користувач вводить її назву в відповідне поле та натискає кнопку «Додати». Додатково передбачені кнопки «Очистити» для скидання полів та «Вийти» для завершення роботи з формою. Ця функціональність надає користувачам інструменти для підготовки та аналізу моделей, що допомагає забезпечити точність прогнозування діабету та підтримує клінічні рішення на основі машинного навчання.

Форма «Прогнозування діабету» призначена для прогнозування ризику розвитку діабету у пацієнтів та доступна для використання лікарями (рис. 3.22). Ліва частина форми містить розширений набір полів для введення медичних показників пацієнта, таких як індекс маси тіла, наявність високого артеріального тиску, високого рівня холестерину, шкідливих звичок, фізичної активності, споживання овочів і фруктів, та інших факторів ризику, які можуть впливати на ймовірність розвитку діабету. Поля позначені відповідними позначками для обов’язкових даних, що забезпечує точність введеної інформації.

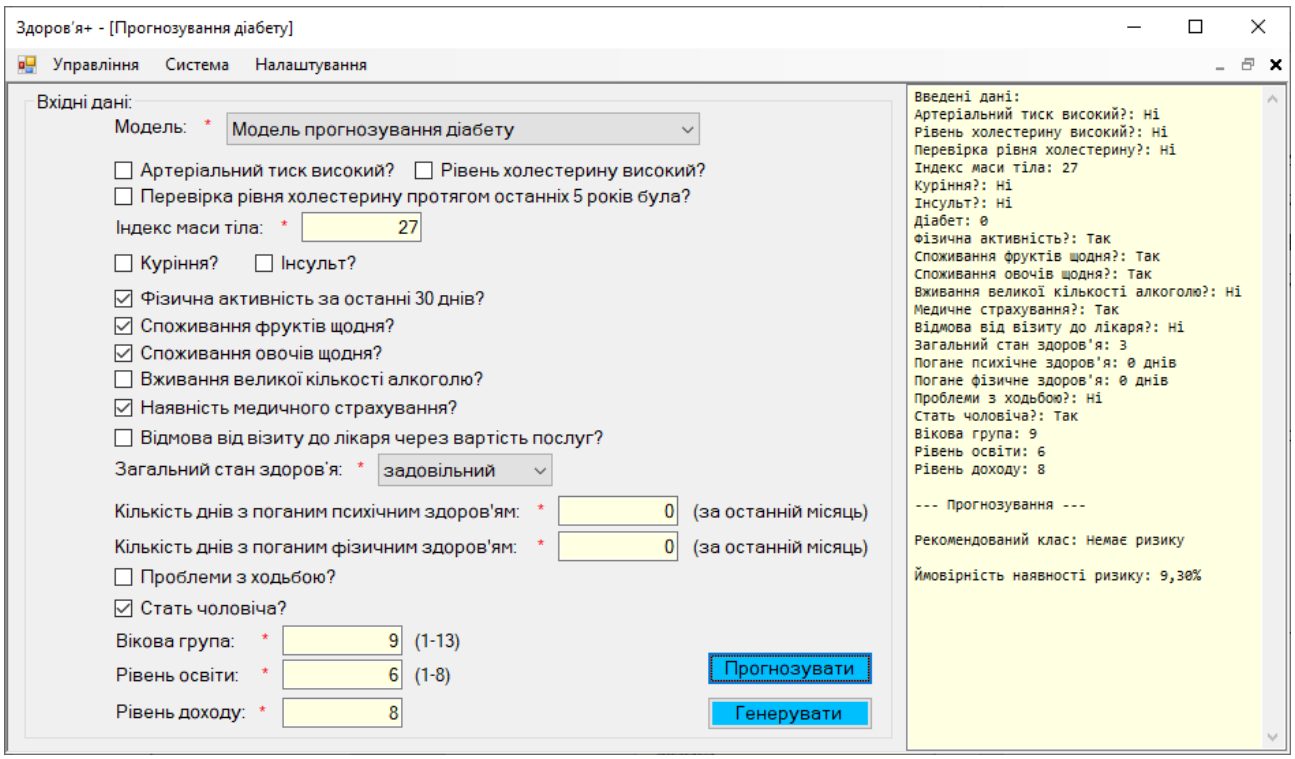


Рис. 3.22 Форма «Прогнозування діабету»

Для прогнозування лікар обирає модель із випадаючого списку «Модель», що дозволяє використовувати різні алгоритми машинного навчання, навчання яких відбувалося на основі історичних даних пацієнтів з діабетом. Після введення всіх необхідних параметрів лікар натискає кнопку «Прогнозувати», яка запускає алгоритм прогнозування, а результати автоматично відображаються у правій частині форми.

Результати включають вичерпну інформацію про ймовірність ризику розвитку діабету у пацієнта. Тут зазначено, чи є ризик (рекомендований клас), а також відображається точний відсоток ймовірності ризику. Ця функціональність дозволяє лікарям приймати обґрунтовані рішення на основі прогнозів, що допомагає вчасно виявляти пацієнтів з підвищеним ризиком та пропонувати профілактичні заходи. Кнопка «Генерувати» дозволяє проводити тестування моделі на випадково згенерованих даних.

Форма «Користувачі системи» призначена для адміністраторів системи, які відповідають за управління обліковими записами користувачів, зокрема пацієнтів, лікарів та інших адміністраторів (рис. 2.23). Ліва частина форми

містить поля для введення даних нового користувача, таких як прізвище, ім’я, електронна пошта, опис профілю, роль користувача, логін, пароль та підтвердження пароля. Обов’язкові поля позначені червоними зірочками, що вказує на необхідність їх заповнення для створення облікового запису. Адміністратор також може встановити статус користувача як активний або неактивний, позначивши чекбокс «Статус (акт./не акт.)».

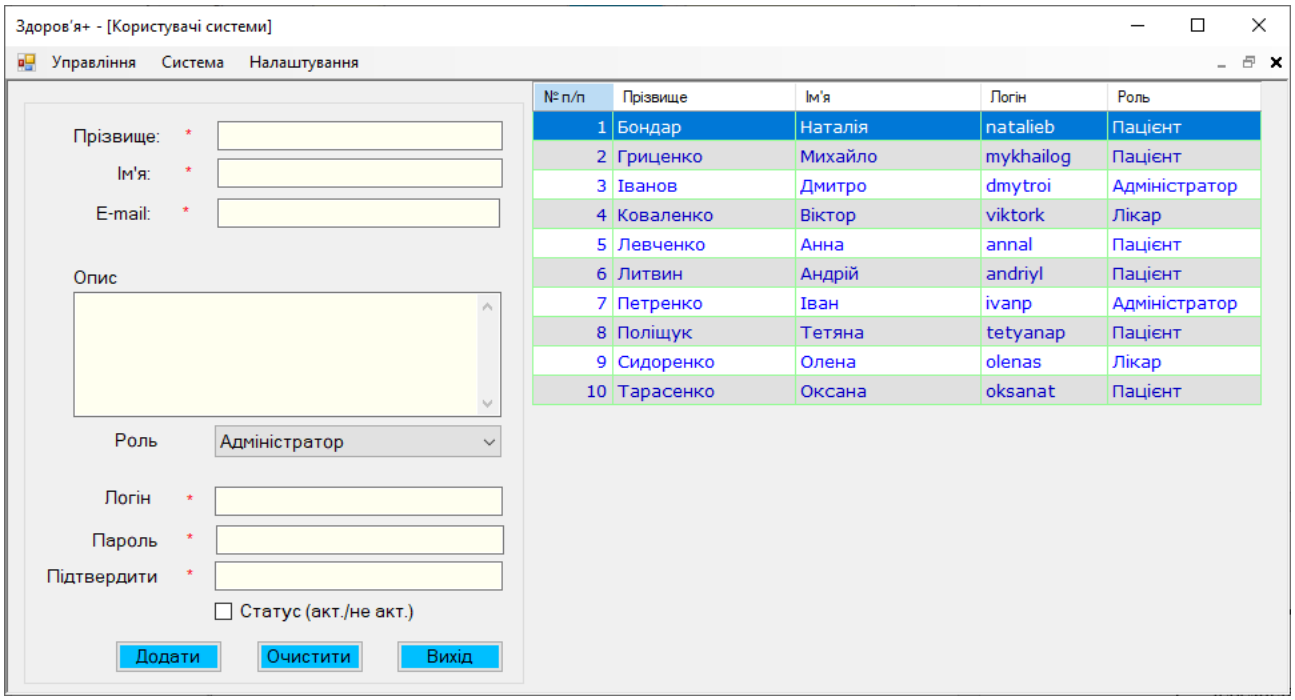


Рис. 3.23 Форма «Користувачі системи»

Форма «Користувачі системи» є ключовим інструментом для адміністратора, що надає можливість централізовано керувати обліковими записами, контролювати доступ до системи, а також підтримувати актуальність та безпеку даних користувачів у системі «Здоров’я+».

Для виходу з системи «Здоров’я+» користувачеві необхідно скористатися пунктом меню «Управління». При виборі опції «Управління» у верхній панелі, відкривається випадаючий список, де доступний пункт «Вихід». Натискання на пункт «Вихід» завершує поточну сесію роботи з системою та закриває програму.

## 3.4 Висновок

У рамках даного розділу було розроблено, протестовано та детально описано аналітичну систему «Здоров’я+», призначену для моніторингу стану здоров’я пацієнтів і підтримки медичних процесів. Зокрема, було реалізовано ключові модулі системи, серед яких основні методи для обробки та управління даними пацієнтів, що забезпечує функціональність, необхідну для лікарів і користувачів. Для забезпечення надійності та стабільної роботи системи проведено функціональне та компонентне тестування, зокрема для форм «Авторизація», «Консультації з лікарями», «Персоналізація». Крім того, за допомогою MS Test виконано автоматизоване тестування, що дозволило підтвердити коректність роботи основних компонентів і виявити можливі проблеми на ранніх етапах.

Також розроблено інструкцію користувача, яка включає мінімальні вимоги для запуску, покроковий процес розгортання системи, а також детальний опис використання основних функцій для моніторингу здоров’я пацієнтів. Це дозволяє користувачам легко ознайомитися з роботою системи та ефективно використовувати її можливості.

Отримані результати та проведені тести підтвердили, що аналітична система «Здоров’я+» відповідає поставленим вимогам, забезпечує зручність і надійність у роботі, а також надає інтуїтивно зрозумілий інтерфейс для моніторингу та підтримки медичних даних.

# ВИСНОВКИ

Кваліфікаційна робота присвячена розробці аналітичної підсистеми управління інформаційними потоками телемедичної системи моніторингу стану здоров'я «Здоров'я+», що призначена для покращення точності діагностики, підвищення ефективності роботи медичного персоналу та підтримки прийняття рішень лікарями.

Перший розділ роботи зосереджений на аналізі поточного стану телемедичних систем, характеристиці їх основних видів та функцій, а також обґрунтуванні необхідності аналітичних підсистем для обробки великих обсягів даних. Також було проведено огляд існуючих рішень для управління інформаційними потоками в медицині, зокрема розглянуто системи Epic Systems, Cerner Millennium та Meditech, що дозволило виділити переваги і недоліки цих систем і обґрунтувати розробку нового рішення. На основі проведеного аналізу були сформульовані функціональні та нефункціональні вимоги до аналітичної підсистеми.

У другому розділі висвітлено вибір технологій для реалізації системи, де розглянуто альтернативні варіанти мов програмування, середовищ розробки та систем управління базами даних. У результаті аналізу було обрано мову програмування C#, середовище розробки Visual Studio 2022 та систему управління базами даних MS SQL Server, що забезпечує надійність і сумісність для реалізації поставлених завдань. Також у цьому розділі було проєктовано основні сценарії використання системи, включаючи діаграми діяльності та послідовності, що охоплюють ключові процеси системи, зокрема авторизацію користувачів, навчання моделей діагностики діабету, обробку медичних карток лікарями та перегляд власних діагнозів пацієнтами. Для зберігання даних розроблено структуру бази даних, представлено таблиці та побудовано ER-діаграму, що дозволяє оптимально організувати зберігання медичної інформації. Крім того, розроблено трьохрівневу архітектуру системи, яка

забезпечує розподіл функцій на рівні даних, бізнес-логіки та представлення, що сприяє покращенню масштабованості та зручності підтримки системи.

Третій розділ був присвячений безпосередній розробці, тестуванню та інструктажу користувачів аналітичної системи «Здоров'я+». У рамках цього розділу було описано ключові модулі системи, зокрема методи для роботи з медичними картками, профілями пацієнтів та лікарів, а також реалізацію інструментів для консультацій і прогнозування діабету. Проведено функціональне та компонентне тестування основних форм системи, таких як «Авторизація», «Консультації з лікарями» та «Персоналізація», а також автоматизоване тестування з використанням MS Test, що дозволило перевірити коректність роботи системи та виявити можливі помилки. Крім цього, було створено інструкцію користувача, яка містить мінімальні вимоги для запуску системи, опис процесу її розгортання, а також детальні інструкції для роботи з основними функціями, такими як моніторинг здоров'я, ведення медичних карток, тренування моделей для прогнозування та управління користувачами.

Проведена робота демонструє застосування сучасних технологій програмування та баз даних для створення аналітичної підсистеми в телемедичній системі моніторингу стану здоров'я. Розроблений додаток «Здоров'я+» дозволяє лікарям ефективно контролювати та аналізувати стан здоров'я пацієнтів, надає можливості для підтримки клінічних рішень на основі прогнозування ризиків розвитку діабету, а також забезпечує зручний інтерфейс для зберігання та обробки медичної інформації. Отримані результати свідчать про відповідність розробленої системи поставленим вимогам та її готовність до використання у клінічній практиці.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черемісіна В. В.; Снісаренко П. І. Телемедицина та її роль у реформуванні системи охорони здоров'я. Наукові праці [Чорноморського державного університету імені Петра Могили]. Сер.: Техногенна безпека, 2012, 203, Вип. 191: 141 с.

2. Дубчак Л. О. Телемедицина: сучасний стан та перспективи розвитку. Системи обробки інформації, 2017, 1: 146 с.

3. Поліщук Г.; Лупенко С. Загальна архітектура телемедичних систем діагностики та моніторингу. Збірник тез доповідей XVI наукової конференції Тернопільського національного технічного університету імені Івана Пулюя, 2012, – 59 с.

4. В Україні стартує пілотний проект телемедицини: які можливості відкриває. URL: https://www.5.ua/suspilstvo/v-ukraini-startuie-pilotnyi-proekt-telemedytsyny-iaki-mozhlyvosti-vidkryvaie-162172.html (дата звернення 09.11.2024).

5. Табунщик Г. В. Проєктування інформаційної інфраструктури медичних та телемедичних систем. 2021.– 70с.

6. Владзимирський А. В.; Дорохова О. Т. Телемедицина в управлінні охороною здоров'я. Медична освіта, 2002, 2: 17с.

7. Ярославський Я. І. Особливості побудови телемедичних мереж і систем. In: Сучасні технології біомедичної інженерії. Національний університет «Одеська політехніка», 2023.– 91 С.

8. Кулик А.Я., Мотигін В.В., Кулик Я.А., Книш Б.П. Телемедицина. Комп'ютерні системи та інформаційні технології : монографія. Вінниця : ВНМУ, 2020. 293 с.

9. Медичні інформаційні системи: огляд можливостей і приклади використання. URL: https://emci.ua/statti/iak-vybraty-mis/ (дата звернення 09.11.2024).

10. Пікуш В. В. Телемедичні технології в діяльності закладів охорони здоров’я в умовах цифровізації. 2023. –12 с.

11. Програмні рішення для телемедицини: область застосування і розробка. URL: https://evergreens.com.ua/ua/articles/telemedicine-vstelehealth.html (дата звернення 09.11.2024).

12. Epic EHR. URL: https://www.ehrinpractice.com/epic-ehr-software-profile-119.html (дата звернення 09.11.2024).

13. Epic: Employee Benefits and Perks URL: https://www.glassdoor.com/Benefits/Epic-US-Benefits-EI_IE35163.0,4_IL.5,7_IN1.htm (дата звернення 09.11.2024).

14. Epic Systems Corporation Benefits URL: https://www.comparably.com/companies/epic-systems-corporation/perks-and-benefits (дата звернення 09.11.2024).

15. About Cerner Millennium. URL: https://www.softwareadvice.com/medical/cerner-millennium-profile/ (дата звернення 09.11.2024).

16. Cerner Reviews & Pros and Cons URL: https://ehrguide.org/ehr-reviews/cerner-reviews/(дата звернення 09.11.2024).

17. Cerner Millennium Code Upgrade Services URL: https://www.applytosupply.digitalmarketplace.service.gov.uk/g-cloud/services/617347205384087 (дата звернення 09.11.2024).

18. What makes MEDITECH remarkable. URL: https://ehr.meditech.com/ (дата звернення 09.11.2024).

19. 20 Pros and Cons of Meditech URL: https://uk.educationalwave.com/pros-and-cons-of-meditech/(дата звернення 09.11.2024).

20. Benefits & Perks URL: https://ehr.meditech.com/careers/benefits-perks (дата звернення 09.11.2024).

21. Жуковський С.С. Об’єктно-орієнтоване програмування мовою С++ Житомир:ЖДУ, 2016. – 100 с.

22. Карапецький В. П. Побудова графічного контенту додатків з використанням JavaFX і Swing компонентів і даних, взятих із баз даних / В. П. Карапецький. – Науковий вісник НЛТУ. – 2015. – 418 с.

23. Kozma, M., Vincur, J., & Kapec, P. CollaVRation: An Immersive Virtual Environment for Collaborative Software Development. In Science and Information Conference. Cham: Springer Nature Switzerland. 2024. pp. 280-298.

24. Коноваленко І.В., Марущак П.О. Платформа .NET та мова програмування C# 8.0: навчальний посібник. Тернопіль: ФОП Паляниця В. А., 2020. 320 с.

25. Kahlert T., Giza K. Visual Studio Code Tips & Tricks. Microsoft Deutschland GmbH. 2016. Vol. 1. 26 p.

26. Most Helpful Rider Reviews. URL: https://www.gartner.com/reviews/market/integrated-development-environment-ide-software/vendor/jetbrains/product/rider (дата звернення 11.11.2024).

27. Microsoft SQL Server : веб-сайт. URL: https://www.metabase.com/data_sources/microsoft-sql-server (дата звернення 11.11.2024).

28. PostgreSQL Tutorial. URL: https://neon.tech/postgresql/tutorial (дата звернення 11.11.2024).

29. DB Browser for SQLite. URL: https://sqlitebrowser.org/ (дата звернення 11.11.2024).

30. Iqbal, S., Al-Azzoni, I., Allen, G., & Khan, H. U. Extending UML use case diagrams to represent non-interactive functional requirements. E-Informatica Software Engineering Journal. 2020. Vol. 14 №1. pp. 97-115.

ДОДАТОК А

# ЛІСТИНГИ ПРОГРАМНОГО КОДУ ПІДСИСТЕМИ

Сторінок 12

Лістинг 1. Код класу «DiabetesDataProvider»

```
using Microsoft.ML.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HealthPlusApp.Providers {
  internal class DiabetesDataProvider {

  }
}

public class DiabetesData {
  /// <summary> Наявність діабету: 1 для позитивного випадку, 0 для негативного.
</summary>
  [LoadColumn(0)]
  public float DiabetesBinary;

  /// <summary> Високий кров'яний тиск: 1, якщо у пацієнта високий тиск, 0 інакше.
</summary>
  [LoadColumn(1)]
  public float HighBP;

  /// <summary> Високий рівень холестерину: 1, якщо холестерин високий, 0 інакше.
</summary>
  [LoadColumn(2)]
  public float HighChol;

  /// <summary> Перевірка рівня холестерину: 1, якщо пацієнт перевіряв рівень холестерину, 0
інакше. </summary>
  [LoadColumn(3)]
  public float CholCheck;

  /// <summary> Індекс маси тіла (BMI) пацієнта. </summary>
  [LoadColumn(4)]
  public float BMI;

  /// <summary> Куріння: 1, якщо пацієнт курить, 0 інакше. </summary>
  [LoadColumn(5)]
  public float Smoker;

  /// <summary> Історія інсульту: 1, якщо пацієнт мав інсульт, 0 інакше. </summary>
  [LoadColumn(6)]
  public float Stroke;

  /// <summary> Історія серцевого нападу чи хвороби: 1 для позитивного випадку, 0 інакше.
</summary>
  [LoadColumn(7)]
  public float HeartDiseaseorAttack;
```

```
  /// <summary> Фізична активність: 1, якщо пацієнт веде активний спосіб життя, 0 інакше.
</summary>
  [LoadColumn(8)]
  public float PhysActivity;

  /// <summary> Споживання фруктів: 1, якщо пацієнт регулярно споживає фрукти, 0 інакше.
</summary>
  [LoadColumn(9)]
  public float Fruits;

  /// <summary> Споживання овочів: 1, якщо пацієнт регулярно споживає овочі, 0 інакше.
</summary>
  [LoadColumn(10)]
  public float Veggies;

  /// <summary> Високе споживання алкоголю: 1, якщо пацієнт часто вживає алкоголь, 0
інакше. </summary>
  [LoadColumn(11)]
  public float HvyAlcoholConsump;

  /// <summary> Наявність медичного обслуговування: 1, якщо пацієнт має доступ до
медичних послуг, 0 інакше. </summary>
  [LoadColumn(12)]
  public float AnyHealthcare;

  /// <summary> Відсутність можливості звернутися до лікаря через високу вартість: 1, якщо
була така ситуація, 0 інакше. </summary>
  [LoadColumn(13)]
  public float NoDocbcCost;

  /// <summary> Загальне здоров'я (за шкалою від 1 до 5): 1 — відмінне, 5 — погане.
</summary>
  [LoadColumn(14)]
  public float GenHlth;

  /// <summary> Кількість днів із поганим психічним здоров'ям за останній місяць.
</summary>
  [LoadColumn(15)]
  public float MentHlth;

  /// <summary> Кількість днів із поганим фізичним здоров'ям за останній місяць. </summary>
  [LoadColumn(16)]
  public float PhysHlth;

  /// <summary> Труднощі в пересуванні: 1, якщо пацієнт відчуває труднощі, 0 інакше.
</summary>
  [LoadColumn(17)]
  public float DiffWalk;

  /// <summary> Стать пацієнта: 1 для чоловіків, 0 для жінок. </summary>
  [LoadColumn(18)]
  public float Sex;
```

```
  /// <summary> Вік пацієнта (категоризоване значення). </summary>
  [LoadColumn(19)]
  public float Age;

  /// <summary> Рівень освіти пацієнта (категоризоване значення). </summary>
  [LoadColumn(20)]
  public float Education;

  /// <summary> Рівень доходу пацієнта (категоризоване значення). </summary>
  [LoadColumn(21)]
  public float Income;
}

public class DiabetesPredictionResult {
  /// <summary> Прогноз наявності діабету: true, якщо діабет прогнозується, false інакше.
</summary>
  [ColumnName("PredictedLabel")]
  public bool Prediction { get; set; }

  /// <summary> Ймовірність, що пацієнт належить до групи ризику діабету. </summary>
  public float Probability { get; set; }

  /// <summary> Оцінка моделі щодо наявності діабету. </summary>
  public float Score { get; set; }
}
```

Лістинг 2. Код класу «PatientExaminationsProvider»

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using HealthPlusApp.AppCode;

namespace HealthPlusApp.Providers {
  internal class PatientExaminationsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    private UsersProvider _UsersProvider = new UsersProvider();
    private List<Users> _UsersList = new List<Users>();


    public void InsertPatientExaminations(int PatientId, int DoctorId, DateTime ExaminationDate,
string Symptoms, string Diagnosis,
      string TreatmentPlan, string Notes, DateTime NextAppointment) {
      string sqlString = "INSERT INTO PatientExaminations (PatientId, DoctorId, ExaminationDate,
Symptoms, Diagnosis, TreatmentPlan, Notes, NextAppointment) " +
        "Values(@PatientId, @DoctorId, @ExaminationDate, @Symptoms, @Diagnosis,
@TreatmentPlan, @Notes, @NextAppointment)";
```

```
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@PatientId", PatientId);
        cmd.Parameters.AddWithValue("@DoctorId", DoctorId);
        cmd.Parameters.AddWithValue("@ExaminationDate", ExaminationDate);
        cmd.Parameters.AddWithValue("@Symptoms", Symptoms);
        cmd.Parameters.AddWithValue("@Diagnosis", Diagnosis);
        cmd.Parameters.AddWithValue("@TreatmentPlan", TreatmentPlan);
        cmd.Parameters.AddWithValue("@Notes", Notes);
        cmd.Parameters.AddWithValue("@NextAppointment", NextAppointment);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public List<PatientExaminations> GetAllPatientExaminations() {
    int i = 0;
    string sqlString = "SELECT * FROM PatientExaminations ORDER BY ExaminationDate
DESC";
    _UsersList = _UsersProvider.GetAllPatient();
    List<PatientExaminations> listAllPatientExaminations = new List<PatientExaminations>();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            PatientExaminations oneExamination = new PatientExaminations();
            oneExamination.Number = ++i;
            oneExamination.PatientExaminationsId =
Convert.ToInt32(reader["PatientExaminationsId"]);
            oneExamination.PatientId = Convert.ToInt32(reader["PatientId"]);
            oneExamination.PatientFIO = GetFIO(oneExamination.PatientId, _UsersList);
            oneExamination.DoctorId = Convert.ToInt32(reader["DoctorId"]);
            oneExamination.ExaminationDate = Convert.ToDateTime(reader["ExaminationDate"]);
            oneExamination.Symptoms = reader["Symptoms"].ToString();
            oneExamination.Diagnosis = reader["Diagnosis"].ToString();
            oneExamination.TreatmentPlan = reader["TreatmentPlan"].ToString();
            oneExamination.Notes = reader["Notes"].ToString();
            oneExamination.NextAppointment = Convert.ToDateTime(reader["NextAppointment"]);

            listAllPatientExaminations.Add(oneExamination);
          }
        }
        conn.Close();
      }
    }

    if (listAllPatientExaminations.Count == 0) {
```

```
      PatientExaminations noPatientExaminations = new PatientExaminations();
      noPatientExaminations.PatientExaminationsId = 0;
      noPatientExaminations.Message = NamesMy.NoDataNames.NoDataInPatientExaminations;
      listAllPatientExaminations.Add(noPatientExaminations);
    }

    return listAllPatientExaminations;
  }

  public List<PatientExaminations> GetAllPatientExaminationsByPatientId(int PatientId) {
    int i = 0;
    string sqlString = "SELECT * FROM PatientExaminations WHERE PatientId=" + PatientId + "
ORDER BY ExaminationDate DESC";

    List<PatientExaminations> listAllPatientExaminations = new List<PatientExaminations>();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            PatientExaminations oneExamination = new PatientExaminations();
            oneExamination.Number = ++i;
            oneExamination.PatientExaminationsId =
Convert.ToInt32(reader["PatientExaminationsId"]);
            oneExamination.PatientId = Convert.ToInt32(reader["PatientId"]);
            oneExamination.DoctorId = Convert.ToInt32(reader["DoctorId"]);
            oneExamination.ExaminationDate = Convert.ToDateTime(reader["ExaminationDate"]);
            oneExamination.Symptoms = reader["Symptoms"].ToString();
            oneExamination.Diagnosis = reader["Diagnosis"].ToString();
            oneExamination.TreatmentPlan = reader["TreatmentPlan"].ToString();
            oneExamination.Notes = reader["Notes"].ToString();
            oneExamination.NextAppointment = Convert.ToDateTime(reader["NextAppointment"]);

            listAllPatientExaminations.Add(oneExamination);
          }
        }
        conn.Close();
      }
    }

    if (listAllPatientExaminations.Count == 0) {
      PatientExaminations noPatientExaminations = new PatientExaminations();
      noPatientExaminations.PatientExaminationsId = 0;
      noPatientExaminations.Message = NamesMy.NoDataNames.NoDataInPatientExaminations;
      listAllPatientExaminations.Add(noPatientExaminations);
    }

    return listAllPatientExaminations;
  }

  private string GetFIO(int UsersId, List<Users> UsersL) {
    for (int i = 0; i < UsersL.Count; i++) {
```

```
      if (UsersId == UsersL[i].UsersId) {
        return UsersL[i].FIO;
      }
    }
    return "";
  }

  public PatientExaminations SelectPatientExaminationById(int PatientExaminationsId) {
    string sqlString = "SELECT * FROM PatientExaminations WHERE
PatientExaminationsId=@PatientExaminationsId";
    PatientExaminations oneExamination = new PatientExaminations();

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
        cmd.Parameters.AddWithValue("@PatientExaminationsId", PatientExaminationsId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          if (reader.Read()) {
            oneExamination.PatientExaminationsId =
Convert.ToInt32(reader["PatientExaminationsId"]);
            oneExamination.PatientId = Convert.ToInt32(reader["PatientId"]);
            oneExamination.DoctorId = Convert.ToInt32(reader["DoctorId"]);
            oneExamination.ExaminationDate = Convert.ToDateTime(reader["ExaminationDate"]);
            oneExamination.Symptoms = reader["Symptoms"].ToString();
            oneExamination.Diagnosis = reader["Diagnosis"].ToString();
            oneExamination.TreatmentPlan = reader["TreatmentPlan"].ToString();
            oneExamination.Notes = reader["Notes"].ToString();
            oneExamination.NextAppointment = Convert.ToDateTime(reader["NextAppointment"]);
          }
        }
        conn.Close();
      }
    }
    return oneExamination;
  }

  public void UpdatePatientExamination(int PatientId, int DoctorId, DateTime ExaminationDate,
string Symptoms,
    string Diagnosis, string TreatmentPlan, string Notes, DateTime NextAppointment,
    int PatientExaminationsId) {
    string sqlString = "UPDATE PatientExaminations SET " +
                  "PatientId=@PatientId, " +
                  "DoctorId=@DoctorId, " +
                  "ExaminationDate=@ExaminationDate, " +
                  "Symptoms=@Symptoms, " +
                  "Diagnosis=@Diagnosis, " +
                  "TreatmentPlan=@TreatmentPlan, " +
                  "Notes=@Notes, " +
                  "NextAppointment=@NextAppointment " +
                  "WHERE PatientExaminationsId=@PatientExaminationsId";
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
```

```
            cmd.CommandType = CommandType.Text;
            cmd.Parameters.AddWithValue("@PatientId", PatientId);
            cmd.Parameters.AddWithValue("@DoctorId", DoctorId);
            cmd.Parameters.AddWithValue("@ExaminationDate", ExaminationDate);
            cmd.Parameters.AddWithValue("@Symptoms", Symptoms);
            cmd.Parameters.AddWithValue("@Diagnosis", Diagnosis);
            cmd.Parameters.AddWithValue("@TreatmentPlan", TreatmentPlan);
            cmd.Parameters.AddWithValue("@Notes", Notes);
            cmd.Parameters.AddWithValue("@NextAppointment", NextAppointment);
            cmd.Parameters.AddWithValue("@PatientExaminationsId", PatientExaminationsId);
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
          }
        }
      }
       public void DeletePatientExaminationById(int PatientExaminationsId) {
       string sqlString = "DELETE FROM PatientExaminations WHERE
PatientExaminationsId=@PatientExaminationsId";
       using (SqlConnection conn = new SqlConnection(_ConnString)) {
         using (SqlCommand cmd = new SqlCommand(sqlString, conn)) {
           cmd.Parameters.AddWithValue("@PatientExaminationsId", PatientExaminationsId);
           conn.Open();
           cmd.ExecuteNonQuery();
           conn.Close();
         }
       }
     }


  }
}


public class PatientExaminations {
  private int _Number; // Номер запису огляду
  private int _PatientExaminationsId; // Унікальний ідентифікатор огляду
  private int _PatientId; // Ідентифікатор пацієнта
  private string _PatientFIO;
  private int _DoctorId; // Ідентифікатор лікаря
  private DateTime _ExaminationDate; // Дата проведення огляду
  private string _Symptoms; // Опис симптомів
  private string _Diagnosis; // Поставлений діагноз
  private string _TreatmentPlan; // Рекомендації щодо лікування
  private string _Notes; // Додаткові нотатки лікаря
  private DateTime _NextAppointment; // Дата наступного призначеного візиту
  private string _Message; // Повідомлення

  public PatientExaminations() {
    _Number = 0;
    _PatientExaminationsId = 0;
    _PatientId = 0;
```

```
  _PatientFIO = String.Empty;
  _DoctorId = 0;
  _ExaminationDate = DateTime.MinValue;
  _Symptoms = String.Empty;
  _Diagnosis = String.Empty;
  _TreatmentPlan = String.Empty;
  _Notes = String.Empty;
  _NextAppointment = DateTime.MinValue;
  _Message = String.Empty;
}

// Властивості (Properties)
public int Number {
  set { _Number = value; }
  get { return _Number; }
}
public int PatientExaminationsId {
  set { _PatientExaminationsId = value; }
  get { return _PatientExaminationsId; }
}
public int PatientId {
  set { _PatientId = value; }
  get { return _PatientId; }
}
public string PatientFIO {
  set { _PatientFIO = value; }
  get { return _PatientFIO; }
}
public int DoctorId {
  set { _DoctorId = value; }
  get { return _DoctorId; }
}
public DateTime ExaminationDate {
  set { _ExaminationDate = value; }
  get { return _ExaminationDate; }
}
public string Symptoms {
  set { _Symptoms = value; }
  get { return _Symptoms; }
}
public string Diagnosis {
  set { _Diagnosis = value; }
  get { return _Diagnosis; }
}
public string TreatmentPlan {
  set { _TreatmentPlan = value; }
  get { return _TreatmentPlan; }
}
public string Notes {
  set { _Notes = value; }
  get { return _Notes; }
}
```

```
  public DateTime NextAppointment {
    set { _NextAppointment = value; }
    get { return _NextAppointment; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 3. Код класу «PatientChatForm»

```
using HealthPlusApp.AppCode;
using HealthPlusApp.Forms.Systems;
using HealthPlusApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace HealthPlusApp.Forms.Controls {
  public partial class PatientChatForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private UsersProvider _UsersProvider = new UsersProvider();
    private List<Users> _DoctorsList = new List<Users>();
    private Users _selectedDoctor = new Users();

    private ChatsProvider _ChatsProvider = new ChatsProvider();
    private List<Chats> _ChatsList = new List<Chats>();
    private Chats _selectedChats = new Chats();

    private MessagesProvider _MessagesProvider = new MessagesProvider();
    private List<Messages> _MessagesList = new List<Messages>();

    private bool _IsChatCreate = false;

    public PatientChatForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
      ChatsGridView_CellClick(ChatsGridView, new DataGridViewCellEventArgs(1, 0));
    }

    private void CreateChatBtn_Click(object sender, EventArgs e) {
      _selectedDoctor =
_UsersProvider.SelectedUsersByUsersId(Convert.ToInt32(UserNameCBox.SelectedValue));
      AllMessageRTBox.Text = String.Empty;
```

```
      _IsChatCreate = true;
      ChatGBox.Visible = true;
    }

    private void SendBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        if (_IsChatCreate) {
          _ChatsProvider.InsertChats(LoginForm.CurrentUser.UsersId, _selectedDoctor.UsersId,
DateTime.Now, DateTime.Now, LastMessageTBox.Text);
          _selectedChats = _ChatsProvider.SelectChatById(_ChatsProvider.GetLastChat());
          _MessagesProvider.InsertMessages(_selectedChats.ChatsId,
LoginForm.CurrentUser.UsersId, LastMessageTBox.Text, DateTime.Now);
          _IsChatCreate = false;
          DataLoad();
        } else {
          _ChatsProvider.UpdateChat(DateTime.Now, LastMessageTBox.Text,
_selectedChats.ChatsId);
          _MessagesProvider.InsertMessages(_selectedChats.ChatsId,
LoginForm.CurrentUser.UsersId, LastMessageTBox.Text, DateTime.Now);
        }
        AddCurrentMessage();
        ChatInfoLbl.Text = "Створено: " + _selectedChats.StartTime + " із лікарем " +
_selectedChats.DoctorFIO;
      }
    }



    private void ChatsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && ChatsGridView[0, e.RowIndex].Value.ToString() !=
_ChatsList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        int SelectedChatsId = Convert.ToInt32(ChatsGridView[0, e.RowIndex].Value.ToString());
        _selectedChats = _ChatsProvider.SelectChatById(SelectedChatsId);
        _MessagesList = _MessagesProvider.GetAllMessagesByChatsId(SelectedChatsId);
        AllMessagesAdd(_MessagesList);
        ChatInfoLbl.Text = "Створено: " + _selectedChats.StartTime + " із лікарем " +
_selectedChats.DoctorFIO;
        ChatGBox.Visible = true;
      } else {
        ChatGBox.Visible = false;
      }
    }

    private void LoadAllDate() {
      _DoctorsList = _UsersProvider.GetAllDoctor();
      UserNameCBox.DataSource = _DoctorsList;
      UserNameCBox.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
      UserNameCBox.AutoCompleteSource = AutoCompleteSource.ListItems;
      UserNameCBox.ValueMember = "UsersId";
      UserNameCBox.DisplayMember = "UsersName";
    }
```

```
  private void DataLoad() {
    int firstRowIndex = 0;
    if (ChatsGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = ChatsGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _ChatsList = _ChatsProvider.GetAllChatsByPatientId(LoginForm.CurrentUser.UsersId);
      LoadDataInDoctorGridView(_ChatsList);
      if (_selectedRowIndex == ChatsGridView.Rows.Count) {
        _selectedRowIndex = ChatsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        ChatsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        ChatsGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInDoctorGridView(List<Chats> ChatsList) {
    ChatsGridView.DataSource = null;
    ChatsGridView.Columns.Clear();
    ChatsGridView.AutoGenerateColumns = false;
    ChatsGridView.RowHeadersVisible = false;
    ChatsGridView.DefaultCellStyle.WrapMode = DataGridViewTriState.True;
    ChatsGridView.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;

    ChatsGridView.DataSource = ChatsList;

    if (ChatsList.Count > 0) {
      if (ChatsList[0].Message == NamesMy.NoDataNames.NoDataInChats) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = ChatsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
        ChatsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn EncyclopediOnlineIdColumn = new
DataGridViewTextBoxColumn();
        EncyclopediOnlineIdColumn.DataPropertyName = "ChatsId";
        ChatsGridView.Columns.Add(EncyclopediOnlineIdColumn);
        ChatsGridView.Columns[0].Visible = false;

        DataGridViewColumn ProgrammNameColumn = new DataGridViewTextBoxColumn();
        ProgrammNameColumn.DataPropertyName = "Tille";
        ProgrammNameColumn.Width = 320;
        ChatsGridView.Columns.Add(ProgrammNameColumn);
      }
      for (int i = 0; i < ChatsGridView.Columns.Count; i++) {
        ChatsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }
```

```
    private void AllMessagesAdd(List<Messages> MessagesL) {
      AllMessageRTBox.Text = "";
      foreach (var message in MessagesL) {
        if (message.UsersId == LoginForm.CurrentUser.UsersId) {
          AllMessageRTBox.AppendText($"Я ({message.MessageTime}):\r\n");
        } else {
          AllMessageRTBox.AppendText($"{message.FIO} ({message.MessageTime}):\r\n");
        }
        AllMessageRTBox.AppendText($"{message.MessageText}\r\n\r\n");
      }
      // Автопрокрутка до останнього повідомлення
      AllMessageRTBox.SelectionStart = AllMessageRTBox.Text.Length;
      AllMessageRTBox.ScrollToCaret();
    }



    private void AddCurrentMessage() {
      AllMessageRTBox.AppendText($"Я
({DateTime.Now}):\r\n{LastMessageTBox.Text}\r\n\r\n");
      LastMessageTBox.Text = String.Empty;
      // Автопрокрутка до останнього повідомлення
      AllMessageRTBox.SelectionStart = AllMessageRTBox.Text.Length;
      AllMessageRTBox.ScrollToCaret();
    }


    private void CheckForNewMessages() {
      List<Messages> updatedMessages =
_MessagesProvider.GetAllMessagesByChatsId(_selectedChats.ChatsId);
      // Знаходимо нові повідомлення, які ще не відображені в AllMessageRTBox
      foreach (var message in updatedMessages) {
        // Перевіряємо, чи повідомлення є новим (тобто відсутнє в _MessagesList)
        bool isNewMessage = !_MessagesList.Any(m => m.MessagesId == message.MessagesId);

        // Додаємо повідомлення, якщо воно нове і від іншого користувача
        if (isNewMessage && message.UsersId != LoginForm.CurrentUser.UsersId) {
          AppendMessageToAllMessageRTBox(message);
          _MessagesList.Add(message); // Додавання повідомлення до поточного списку, щоб
уникнути повторного додавання
        }
      }
    }

    private void AppendMessageToAllMessageRTBox(Messages message) {
      string messageToAdd = $"{message.FIO}
({message.MessageTime}):\r\n{message.MessageText}\r\n\r\n";
      AllMessageRTBox.AppendText(messageToAdd);
    }

    private void timer1_Tick(object sender, EventArgs e) {
```

```
      CheckForNewMessages();
    }


    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(LastMessageTBox.Text)) {
        LastMessageValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        LastMessageValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

ДОДАТОК Б

# СКРИПТИ БАЗИ ДАНИХ ПІДСИСТЕМИ

Сторінок 5

```
USE [master]
GO
/****** Object:  Database [HealthPlus]    Script Date: 11.11.2024 14:41:02 ******/
CREATE DATABASE [HealthPlus]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'HealthPlus', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\HealthPlus.mdf' , SIZE = 8192KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'HealthPlus_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\HealthPlus_log.ldf' , SIZE = 8192KB ,
MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
GO
ALTER DATABASE [HealthPlus] SET COMPATIBILITY_LEVEL = 160
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [HealthPlus].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [HealthPlus] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [HealthPlus] SET QUERY_STORE (OPERATION_MODE =
READ_WRITE, CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60,
MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200,
WAIT_STATS_CAPTURE_MODE = ON)
GO
USE [HealthPlus]
GO
/****** Object:  Table [dbo].[Chats]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Chats](
	[ChatsId] [int] IDENTITY(1,1) NOT NULL,
	[PatientId] [int] NULL,
	[DoctorId] [int] NULL,
	[StartTime] [datetime] NULL,
	[LastMessageTime] [datetime] NULL,
	[LastMessage] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ChatsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

```
GO
/****** Object:  Table [dbo].[DoctorProfile]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[DoctorProfile](
	[DoctorProfileId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[SpecializationsId] [int] NULL,
	[YearsOfExperience] [int] NULL,
	[Qualifications] [nvarchar](max) NULL,
	[Bio] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[DoctorProfileId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Logs]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Logs](
	[LogsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[EventNameShow] [nvarchar](max) NULL,
	[EventDate] [datetime] NULL,
	[UsersName] [nvarchar](250) NULL,
PRIMARY KEY CLUSTERED
(
	[LogsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[MedicalCards]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MedicalCards](
	[MedicalCardsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[Diagnosis] [nvarchar](500) NULL,
	[Stage] [nvarchar](500) NULL,
	[Allergy] [nvarchar](500) NULL,
	[Condition] [nvarchar](500) NULL,
```

```
	[Contraindications] [nvarchar](500) NULL,
	[RiskFactor] [nvarchar](500) NULL,
	[TreatmentForecast] [nvarchar](500) NULL,
	[Complexity] [nvarchar](500) NULL,
	[TreatmentResult] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[MedicalCardsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Messages]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Messages](
	[MessagesId] [int] IDENTITY(1,1) NOT NULL,
	[ChatsId] [int] NULL,
	[UsersId] [int] NULL,
	[MessageText] [nvarchar](max) NULL,
	[MessageTime] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[MessagesId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Models]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Models](
	[ModelsId] [int] IDENTITY(1,1) NOT NULL,
	[ModelsName] [nvarchar](150) NULL,
	[CreateDate] [datetime] NULL,
	[ModelsFileModel] [nvarchar](max) NULL,
	[UsersId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[ModelsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[PatientExaminations]    Script Date: 11.11.2024 14:41:03 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PatientExaminations](
	[PatientExaminationsId] [int] IDENTITY(1,1) NOT NULL,
	[PatientId] [int] NULL,
	[DoctorId] [int] NULL,
	[ExaminationDate] [datetime] NULL,
	[Symptoms] [nvarchar](500) NULL,
	[Diagnosis] [nvarchar](max) NULL,
	[TreatmentPlan] [nvarchar](500) NULL,
	[Notes] [nvarchar](500) NULL,
	[NextAppointment] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[PatientExaminationsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Specializations]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Specializations](
	[SpecializationsId] [int] IDENTITY(1,1) NOT NULL,
	[SpecializationsName] [nvarchar](250) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[SpecializationsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]    Script Date: 11.11.2024 14:41:03 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
	[UsersId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](200) NULL,
	[LastName] [nvarchar](200) NULL,
	[UsersName] [nvarchar](50) NULL,
	[UsersPassword] [nvarchar](250) NULL,
	[RoleId] [int] NULL,
	[Description] [nvarchar](max) NULL,
```

```
	[Email] [nvarchar](200) NULL,
	[UserStatus] [bit] NULL,
PRIMARY KEY CLUSTERED
(
	[UsersId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
USE [master]
GO
ALTER DATABASE [HealthPlus] SET  READ_WRITE
GO
```