

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультет)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВЕДЕННЯ І АНАЛІЗУ МЕДИЧНИХ
КАРТОК ПАЦІЄНТІВ. SOFTWARE FOR KEEPING TRACK OF AND
ANALYZING HEALTH RECORD

Виконав: студент(ка) 4 курсу, групи КНТ-228
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

Лактіонов В.В.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 «_____» _____ 20__ року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Лактіонова Валерія Віталійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення ведення і аналізу медичних карток пацієнтів. Software for Keeping Track of and Analyzing Health Record.
 керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102
2. Строк подання студентом проєкту (роботи) червень 2022 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Розробка програми. 4. Експлуатація та тестування програми.
5. Перелік графічного матеріалу Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Льовкін В.М., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання «4» квітня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Проектування програмного забезпечення.	3 тиждень	Розділ 2
4	Розробка програми.	4–5 тижні	Розділ 3
5	Експлуатація та тестування програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи.	8 тиждень	

Студент (ка) _____ Лактіонов В.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Льовкін В.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
146 с., 4 табл., 68 рис., 3 дод., 13 джерел.

ВЕДЕННЯ МЕДИЧНИХ КАРТОК, АНАЛІЗ МЕДИЧНИХ КАРТОК,
АНАЛІТИКА, МЕДИЧНИЙ ЗАКЛАД, ПАЦІЄНТ, ПРИЙОМ ПАЦІЄНТА,
ДЕКЛАРАЦІЯ, РЕЦЕПТ, ЛІКАРНЯНИЙ, АНАЛІЗ, БАЗА ДАНИХ.

Об'єкт дослідження – процес ведення і аналізу медичних карток пацієнтів. Предмет дослідження – програмне забезпечення ведення і аналізу медичних карток пацієнтів для медичних закладів.

Мета роботи – розробка програмного забезпечення ведення і аналізу медичних карток пацієнтів для забезпечення інформаційної підтримки прийняття рішень лікарем.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування C#, базове апаратне забезпечення – ноутбук Asus X509h (процесор Intel Core I5, 20 ГБ RAM, 256 ГБ SSD, Nvidia MX250, під керуванням операційної системи Microsoft Windows 11 Pro).

Результати. Розроблено програмне забезпечення для ведення і аналізу медичних карток пацієнтів. Програма побудована на архітектурі клієнт-сервер. Клієнт розроблений за патерном MVVM. Клієнт та сервер комунікують за допомогою технології Rest API. Програмне забезпечення має додаткові можливості для обліку декларацій пацієнтів, виписування рецептів та лікарняних.

Висновки. Розроблене програмне забезпечення дозволяє спростити та підвищити ефективність і працездатність лікарів медичних закладів шляхом забезпечення інформаційної підтримки прийняття рішень.

Галузь використання – медичні заклади різних спрямувань та спеціалізацій.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 146 pages, 68 figures, 4 tables, 3 appendixes, 13 sources.

MAINTENANCE OF MEDICAL CARDS, ANALYSIS OF MEDICAL CARDS, ANALYTICS, MEDICAL INSTITUTION, PATIENT, PATIENT RECEPTION, DECLARATION, RECIPE, HOSPITAL, HOSPITAL.

The object of research is the process of maintaining and analyzing medical records of patients.

The subject of research is software for maintaining and analyzing medical records of patients for medical institutions.

The purpose of the work is to develop software for maintaining and analyzing medical records of patients to provide information support for doctor's decision-making.

Materials, methods and technical means: object-oriented programming, C# programming language, basic hardware – laptop Asus X509h (Intel Core I5 processor, 20 GB RAM, 256 GB SSD, Nvidia MX250, running Microsoft Windows 11 Pro).

Results. Software for maintaining and analyzing medical records of patients has been developed. The program is based on client-server architecture. The client is designed according to the MVVM pattern. The client and server communicate using Rest API technology. The software has additional capabilities for accounting for patient declarations, prescriptions and hospitalizations.

Conclusions. The developed software allows to simplify and increase efficiency and efficiency of doctors of medical institutions, by providing information support of decision-making.

Field of use – medical institutions of various directions and specializations.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Постановка проблеми.....	10
1.2 Огляд аналогів.....	12
1.3 Вимоги до програмного забезпечення	13
1.4 Висновки за розділом 1	20
2 Проєктування програмного забезпечення.....	21
2.1 Вибір мови програмування.....	21
2.2 Проєктування архітектури програмного забезпечення	23
2.3 Проєктування структури бази даних	25
2.3.1 Аналіз сутностей.....	27
2.3.2 Аналіз атрибутів	28
2.4 Структурна схема програмного проєкту.....	29
2.5 Висновки за розділом 2	30
3 Розробка програми.....	31
3.1 Алгоритм функціонування програмного забезпечення.....	31
3.2 Опис модулів реалізованої програми	33
3.3 Висновки за розділом 3	45
4 Експлуатація та тестування програми.....	46
4.1 Призначення програми.....	46
4.2 Умови виконання програми.....	46
4.3 Виконання програми	46
4.4 Повідомлення користувачу.....	64
4.5 Тестування програми	65
Висновки.....	68
Перелік джерел посилання	70
Додаток А Технічне завдання.....	71

Додаток Б Текст програми.....	75
Додаток В Слайди презентації	138

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CLR	– Common Language Runtime;
IDE	– Integrated Development Environment;
MVVM	– Model-View-ViewModel;
SQL	– Structured Query Language;
WPF	– Windows Presentation Foundation.
СКБД	– система керування базами даних;

ВСТУП

Вкрай важливою сферою для людини була та є медицина, а в майбутньому її значимість тільки зростатиме. Навіть в наш час спостерігається збільшення тривалості життя, що, звісно, означає також зростаючу увагу до здоров'я людини від самого початку і протягом усього життя.

Важливою та невід'ємною частиною будь-якої медичної інформаційної системи є електронна медична картка пацієнта. Вона має в своєму розпорядженні зручну та зрозумілу функціональність для зберігання та управління даними про пацієнтів і істотно полегшує роботу лікарів усіх рівнів.

За відсутністю електронних медичних карт, лікарі з високою кваліфікацією змушені приділяти більше часу не прийому пацієнта, а введенню інформації про нього, чи її пошуку. Подібне адміністративне навантаження не повинно обтяжувати лікаря. Краще витратити цей час на більш якісний прийом хворих і уважне ставлення до них.

Метою дипломної роботи є створення програмного забезпечення ведення і аналізу медичних карток пацієнтів для забезпечення інформаційної підтримки прийняття рішень лікарем.

Актуальність роботи полягає у тому, що багато років робота лікарів була тривіальною, тобто був великий обсяг паперової роботи, який ускладнював роботу лікарів у медичних закладах. Задача дипломної роботи – створити застосунок зі зручним, інтуїтивно зрозумілим інтерфейсом, що дозволить в зручній формі виконувати роботу лікарям та використовувати засоби аналітики для будування оптимізованих методів лікування.

Окрім того функціональність аналітики ставить електронні медичні системи на зовсім новий рівень, адже завдяки цьому лікарі отримують дуже важливий інструмент, який структурує та візуалізує інформацію про пацієнта, щоб було простіше оцінити стан хворого та прийняти грамотні клінічні рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка проблеми

Інформаційні технології відіграють все більш важливу роль у всіх секторах економіки, і сфера охорони здоров'я не є винятком. Фактично, підвищення ролі ІТ у секторі охорони здоров'я призвело до створення нової термінології «Інформатика здоров'я», яка стосується використання ІТ для покращення медичних послуг. Додатки медичної інформатики часто мають справу з медичною картою осіб у цифровій формі, яка називається електронною медичною картою [1].

Обраною темою для дипломного проєкту є розробка програмного забезпечення для ведення та аналізу медичних карток пацієнтів. Програмне забезпечення електронних медичних записів дозволяє медичним закладам та закладам охорони здоров'я записувати, систематизувати та керувати медичними записами пацієнтів. Електронні медичні записи – це цифрові відбитки амбулаторної картки, рецептів, лікарняних, аналізів, що являють собою повні відомості про лікування пацієнтів.

До головних переваг програмного забезпечення для електронних медичних записів належить те, що воно допомагає лікарям і медичним працівникам заощадити їх час і зусилля, записуючи дані пацієнта в цифровому вигляді та зменшуючи ручне введення даних. Багато даних підтягується з баз даних, що також автоматизує завдання заповнення форм, надаючи практикуючим лікарям більше часу для огляду та спілкування з пацієнтами замість того, щоб тонути в паперовій роботі.

Медичні записи – це повна історія хвороби пацієнта, яка включає його особисту інформацію, алергії та симптоми, деталі лікування, рецепти та історію лікувань. Ці записи накопичуються в міру прогресування лікування. Стає важко керувати всією цією інформацією сотень і тисяч пацієнтів, особливо коли записи зберігаються на папері. Програмне забезпечення електронних медичних записів допомагає пацієнтам і лікарям записувати інформацію в електронному

вигляді, що не займає жодного фізичного місця. Це вбереже записи від фізичного пошкодження або втрати. Також, не менш важливим є те, що зупиняється використання сотень кілограмів паперу, що в наш час є актуальним з огляду на екологічні проблеми [1].

Зазвичай медичні записи передаються між відділами для обговорення або подальших тестів. Але паперові файли мають ризик загубитися під час транспортування або переплутатися. За допомогою цифрових систем можна легко передавати медичні записи. Крім того, виключається ймовірність втрати сторінок. До записів можна отримати навіть дистанційний доступ, якщо лікар або пацієнт, наприклад, подорожують [2].

Логічно припустити, що розробленою програмою мають користуватися в медичних закладах різних спеціалізацій та спрямувань. Програмне забезпечення повинно спростити та покращити роботу медиків завдяки переходу від паперової роботи до електронної. Розроблена програма має виконувати усі необхідні дії, які виконує лікар чи медсестра на прийомі пацієнтів, а саме: перегляд та ведення медичних карток пацієнтів, перегляд та створення рецептів для лікарських засобів, а також лікарняних; пошук пацієнтів серед пацієнтів з укладеними деклараціями та пошук серед усіх зареєстрованих пацієнтів, а також перегляд та створення декларацій з пацієнтами, іншими словами – укладення декларацій.

Невід’ємною частиною функціональності програмного забезпечення є перегляд зведених даних та графіків для забезпечення інформаційної підтримки прийняття рішень лікарем. Дана можливість дозволить лікарям більш кваліфіковано виконувати лікування пацієнтів.

Також варто зазначити, що з програмою повинен мати змогу працювати не тільки лікарі первинної ланки – сімейні лікарі, а також лікарі вторинної ланки – лікарі вузької спеціалізації, наприклад, кардіолог, офтальмолог, невролог та інші. Аналізи, в свою чергу, робить лабораторія і заносить дані аналізів в електронному вигляді.

Досить важливо, щоб пацієнт мав доступ до перегляду своїх даних та інформації. Тобто він повинен мати доступ до своєї електронної медичної картки, рецептів, лікарняних, декларації, а також до результатів аналізів.

1.2 Огляд аналогів

В Україні вже декілька років працює електронна система охорони здоров'я, що забезпечує обмін медичною інформацією та реалізацію програми медичних гарантій населення для лікарів та медсестер, – Helsi.me [3]. В цій системі лікар має свій власний електронний кабінет, де він має можливість вести облік пацієнтів, виписувати електронні рецепти, а також укладати декларації. Інтерфейс кабінету лікаря наведений на рисунку 1.1.

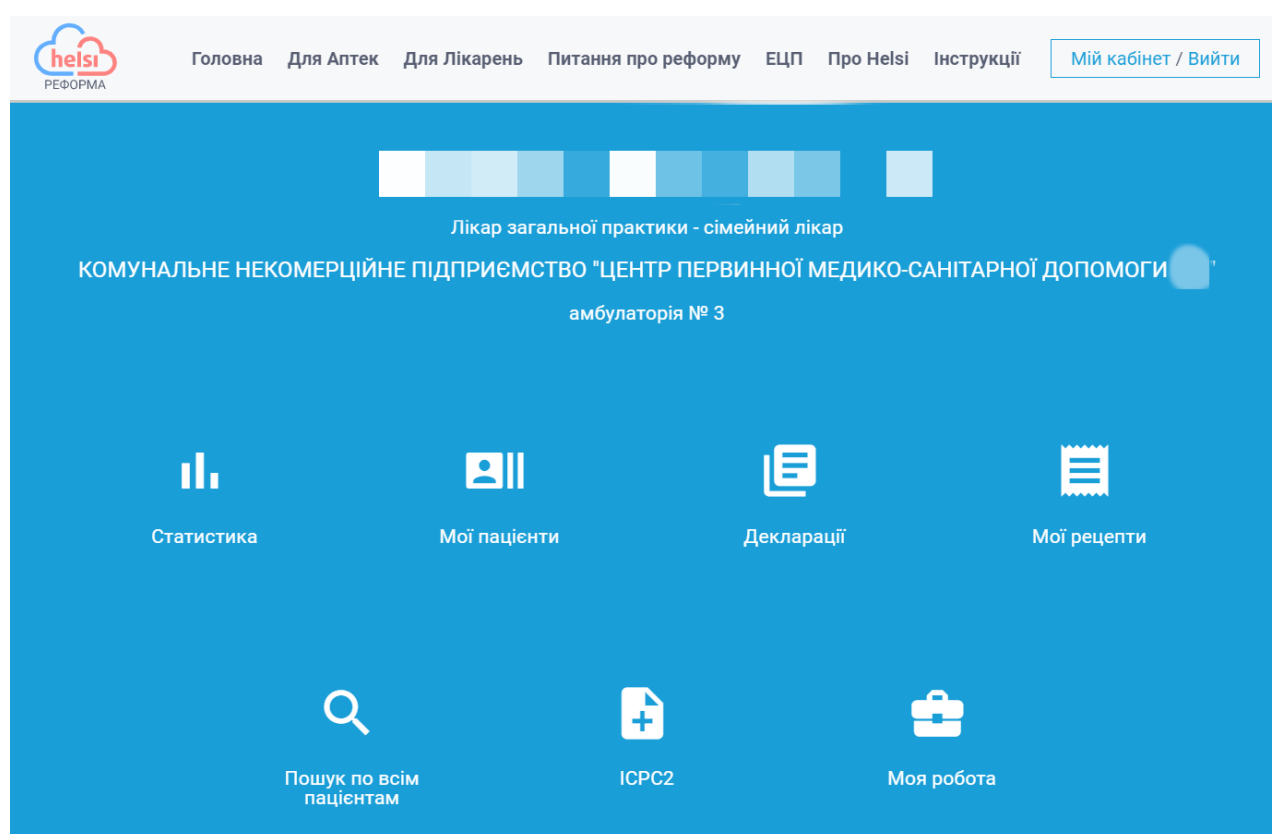


Рисунок 1.1 – Інтерфейс кабінету лікаря в системі Helsi.me

Також досить нещодавно з'явилась електронна система Helsi.Pro [4], що є складовою частиною системи eHealth та є логічним продовженням кабінету

лікаря. В Helsi.Pro лікар має можливість перегляду свого графіку, запису пацієнта на прийом, випису направлень, наприклад, на флюорографію та аналізи, та найголовніше – ведення прийому пацієнтів. Також ця система включає можливості для роботи кадровика, реєстратора та інших. Інтерфейс кабінету лікаря у системі Helsi.Pro наведений на рисунку 1.2.

Так, наявні електроні медичні системи мають широкую функціональність, але в них відсутня можливість переглядати аналітику, таку як зведену інформацію та дані для одного пацієнта, чи групи пацієнтів, що дозволяло би покращувати та оптимізувати лікувальні процеси.

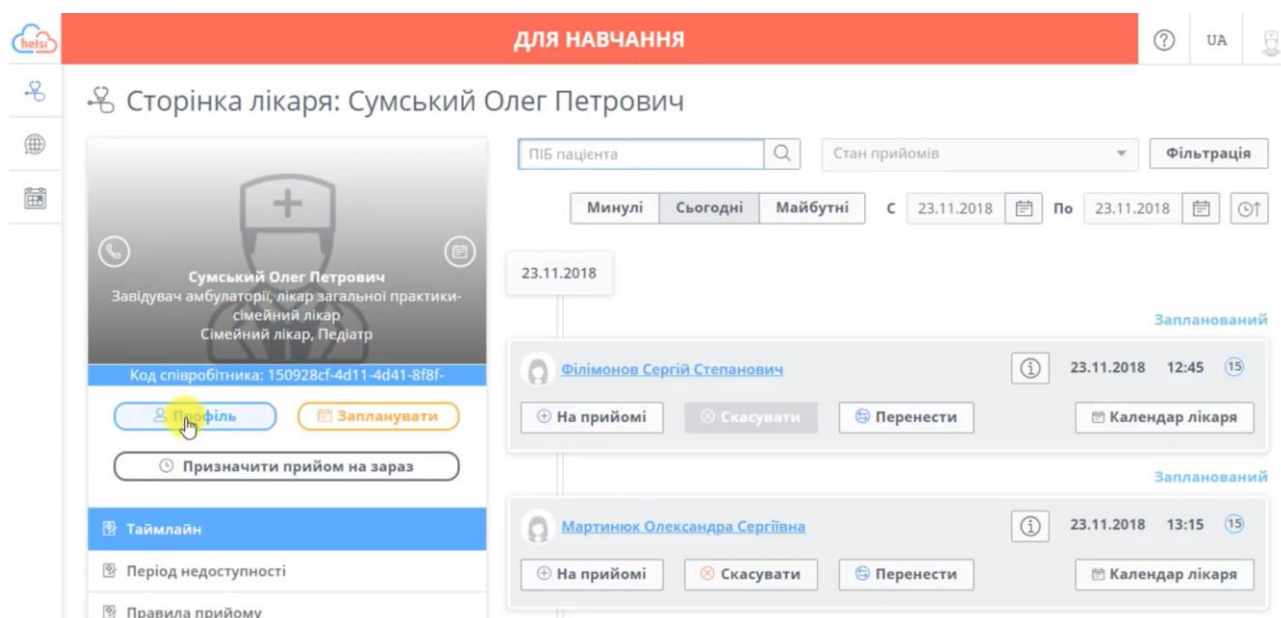


Рисунок 1.2 – Інтерфейс кабінету лікаря в системі Helsi.Pro

1.3 Вимоги до програмного забезпечення

Аналіз вимог перед розробкою програмного забезпечення є дуже важливим кроком для успішного виконання проєкту. Насамперед, вимоги мають бути описані з достатнім рівнем деталізації для створення системи, а також пов'язані з бізнес–потребами.

З головних бізнес правил можна виділити наступні:

- кожен користувач повинен мати логін та пароль для доступу в систему;
- лікар належить до однієї і тільки однієї амбулаторії;
- пацієнт має унікальний реєстраційний номер облікової картки платника податків;
- номери кожної декларації та кожного аналізу повинні бути унікальними;
- номери кожного рецепту та лікарняного повинні бути унікальними;
- лікар вторинної ланки не має можливості переглядати та створювати декларації;
- пацієнт не може редагувати свої дані;
- пацієнт не може редагувати записи лікаря в медичній картці, аналізи, рецепти та лікарняні.

Серед функціональних вимог до програмного забезпечення можна наступні:

а) для лікаря:

- 1) внесення даних щодо прийому: робота з електронною картою;
- 2) перегляд інформації щодо пацієнта: виписані рецепти, лікарняні, результати аналізів;
- 3) створення та перегляд декларацій;
- 4) пошук пацієнтів серед своїх та чужих (декларація з іншим лікарем або без декларації);
- 5) виписування рецептів та лікарняних;
- 6) перегляд аналітики;
- 7) генерація документу «Декларація пацієнта»;

б) для лаборанта: внесення результатів аналізів;

в) для пацієнта: перегляд медичної картки, рецептів, аналізів, лікарняних та декларацій.

До нефункціональних вимог можна віднести наступні:

- операційна система Windows 7 та вище;

- інтуїтивно зрозумілий інтерфейс;
- надійність збереження даних;
- висока продуктивність;
- гнучкість для додавання нових функцій.

До програмного забезпечення поставлено такі вимоги:

- повинно безперебійно працювати на комп'ютерах з встановленою операційною системою Windows;
- опрацьовувати дані без значних затримок;
- мати максимальний обсяг обробки виключних ситуацій;
- при виникненні помилки в програмі або при зверненні до бази даних, користувач повинен бути чітко повідомлений про суть помилки та шлях її усунення, а програмне забезпечення, в цей час, повинно не втрачати працездатності.

Для розроблюваної системи можна виділити наступних акторів: лікар, лаборант та пацієнт. Можна виділити такі основні прецеденти: «Внесення запису до електронної карти» (рисунок 1.3), «Створення декларації» та «Внесення результатів аналізів».

Специфікація для прецеденту «Внесення запису до електронної карти»:

- основний виконавець – лікар;
- зацікавлені особи – пацієнт;
- передумова – пацієнт наявний у системі;
- постумова – запис в електронній картці збережено в базу даних, пацієнт отримує рекомендації.

Основний успішний сценарій:

- лікар відкриває основне вікно програми, в якому необхідно ввести логін та пароль. Лікар вводить дані. Система перевіряє дані та надає доступ на рівні лікаря;
- лікар відкриває вікно пошуку пацієнтів і вводить інформацію про пацієнта. Система виконує пошук пацієнта в базі даних та виводить перелік пацієнтів, що відповідають пошуковій інформації. Лікар обирає відповідного

пацієнта та система виводить картку пацієнта, можливий альтернативний сценарій A1;

– лікар створює запис поточного прийому та вводить дані. Після того як обирає "Зберегти", запис зберігається в базі даних.

Альтернативні сценарії.

A1. Немає електронної карти пацієнта в системі. Створюється нова електронна картка Пацієнта.

Вимоги до даних – всі поля зберігаються за типом string, за винятком дати – тип datetime

Вимоги до технологій – для доступу до бази даних – технологія EntityFramework, для комунікації клієнтського застосунку з сервером – RestAPI.

Відкриті питання – лікар може редагувати внесені записи?

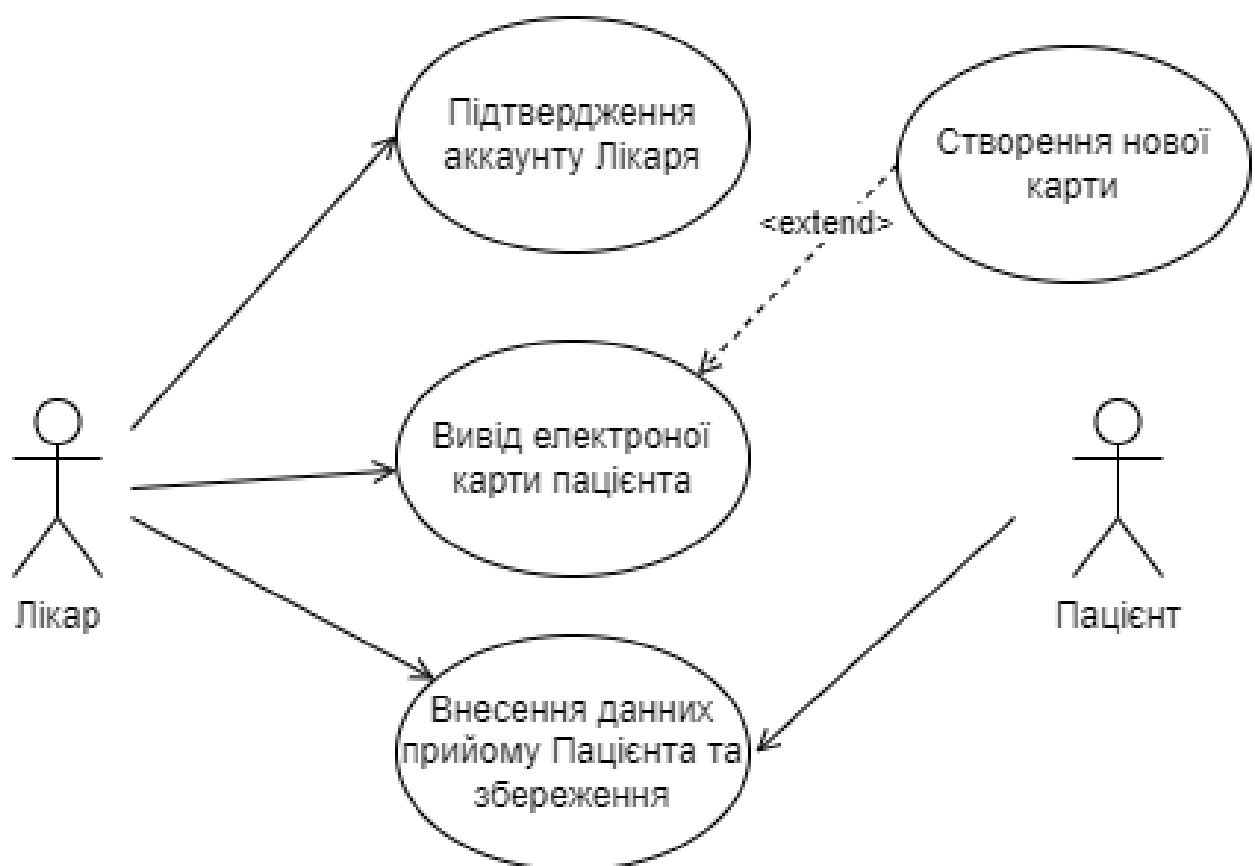


Рисунок 1.3 – Діаграма use case для прецеденту «Внесення запису до електронної карти»

Специфікація для прецеденту «Створення декларації» (рисунок 1.4):

- основний виконавець – лікар;
- зацікавлені особи – пацієнт;
- передумова – лікар пройшов авторизацію, особа підтверджена;
- постумова – інформація декларації внесена до бази даних.

Основний успішний сценарій:

– лікар відкриває вікно пошуку пацієнтів і вводить інформацію про пацієнта. Система виконує пошук пацієнта в базі даних та виводить перелік пацієнтів, що відповідають пошуковій інформації. Лікар обирає відповідного пацієнта та система відкриває сторінку з даними Пацієнта, можливий альтернативний сценарій A1;

– лікар вносить дані пацієнта до електронної декларації та після того, як обирає "Зберегти", запис зберігається в базі даних;

– лікар натискає кнопку “Повна інформація” та система формує декларацію для друку.

Альтернативні сценарії.

A1. Пацієнта немає в системі, система генерує сторінку з початковими даними пацієнта.

Вимоги до даних – всі поля зберігаються за типом string, за винятком дати – тип datetime.

Вимоги до технологій – для доступу до бази даних – технологія EntityFramework, для комунікації клієнтського застосунку з сервером – RestAPI.

Відкриті питання – Чи потрібен дозвіл лікаря для розірвання декларації?

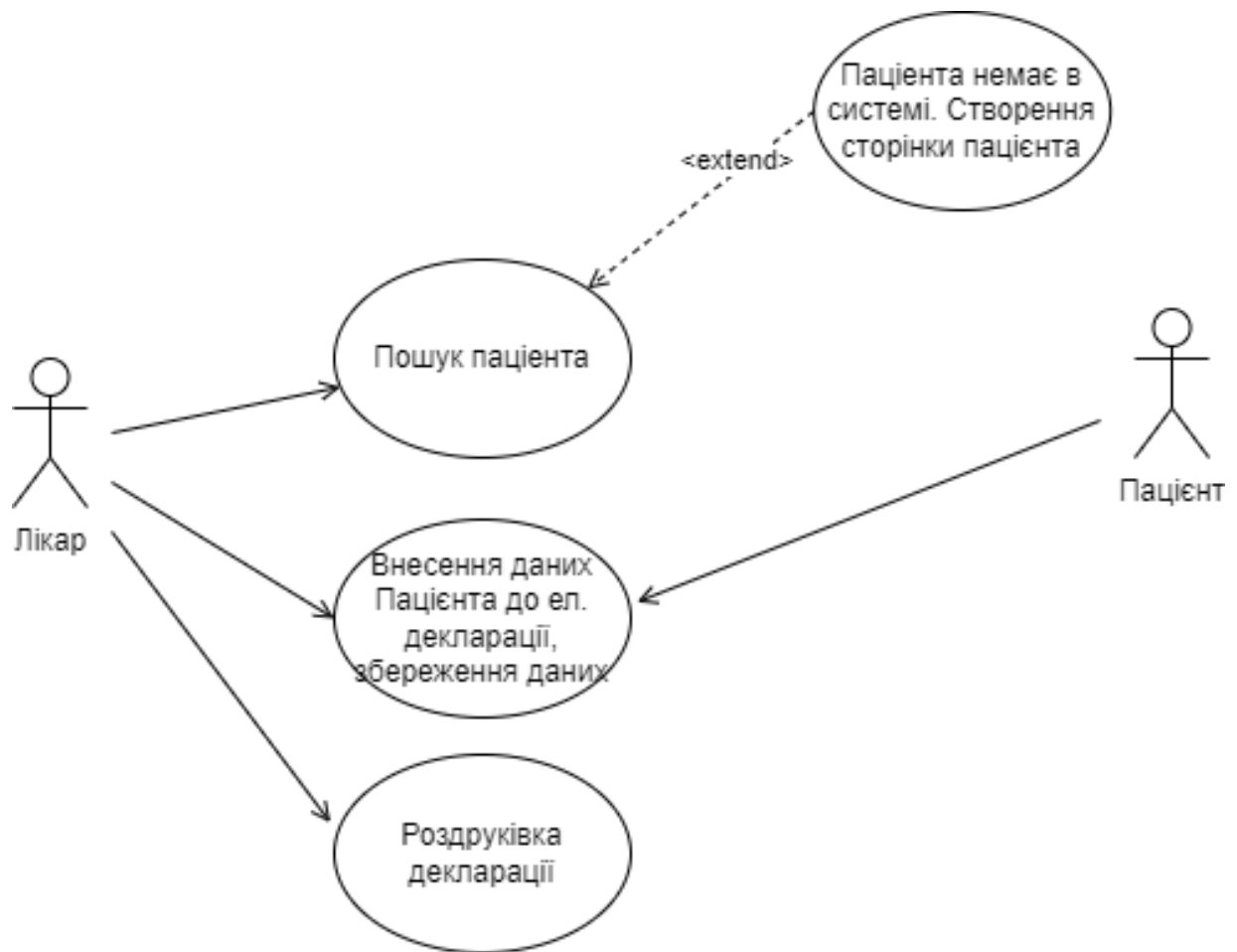


Рисунок 1.4 – Діаграма use case для прецеденту «Створення декларації»

Специфікація для «Внесення результатів аналізів» (рисунок 1.5):

- основний виконавець – лаборант;
- зацікавлені особи – пацієнт;
- передумова – лаборант прийняв та дослідив аналізи. Лаборант авторизувався в системі, особа підтверджена;
- постумова – створений запис аналізу в системі.

Основний успішний сценарій:

- лаборант відкриває вікно пошуку пацієнтів і вводить інформацію про пацієнта. Система виконує пошук пацієнта в базі даних та виводить перелік пацієнтів, що відповідають пошуковій інформації. Лаборант обирає відповідного пацієнта та система відкриває сторінку внесення даних аналізу, можливий альтернативний сценарій A1;

– лаборант вносить дані аналізу до системи та після того, як обирає "Зберегти", запис зберігається в базі даних. Пацієнт отримує результати.

Альтернативні сценарії.

A1. Немає пацієнта в системі, внесення ФІО та дати народження Пацієнта у систему адміністратором.

Вимоги до даних – всі поля зберігаються за типом string, за винятком дати – тип datetime.

Вимоги до технологій – для доступу до бази даних – технологія EntityFramework, для комунікації клієнтського застосунку з сервером – RestAPI.

Відкриті питання – лаборант може редагувати внесені дані?

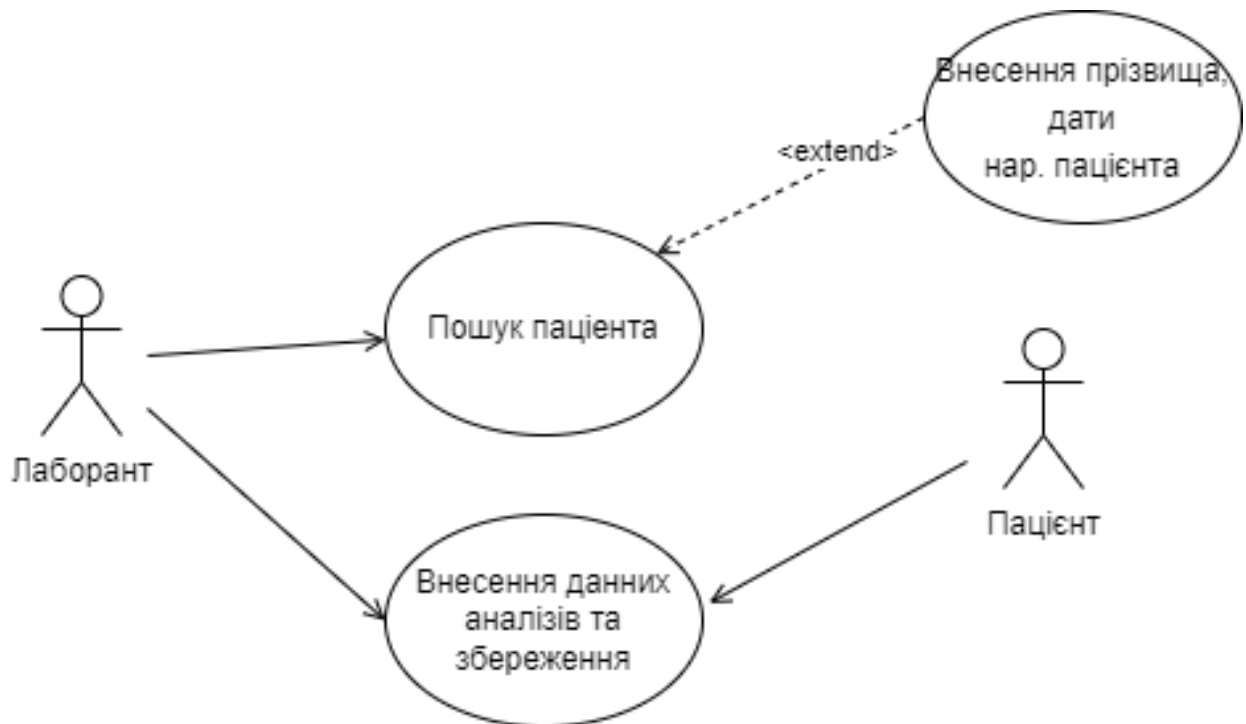


Рисунок 1.5 – Діаграма use case для прецеденту «Внесення результатів аналізів»

На рисунку 1.6 зображена узагальнююча діаграма прецедентів для розроблюваної програмної системи, яка відображає всі основні функціональні вимоги, які сформовані для майбутнього програмного забезпечення.

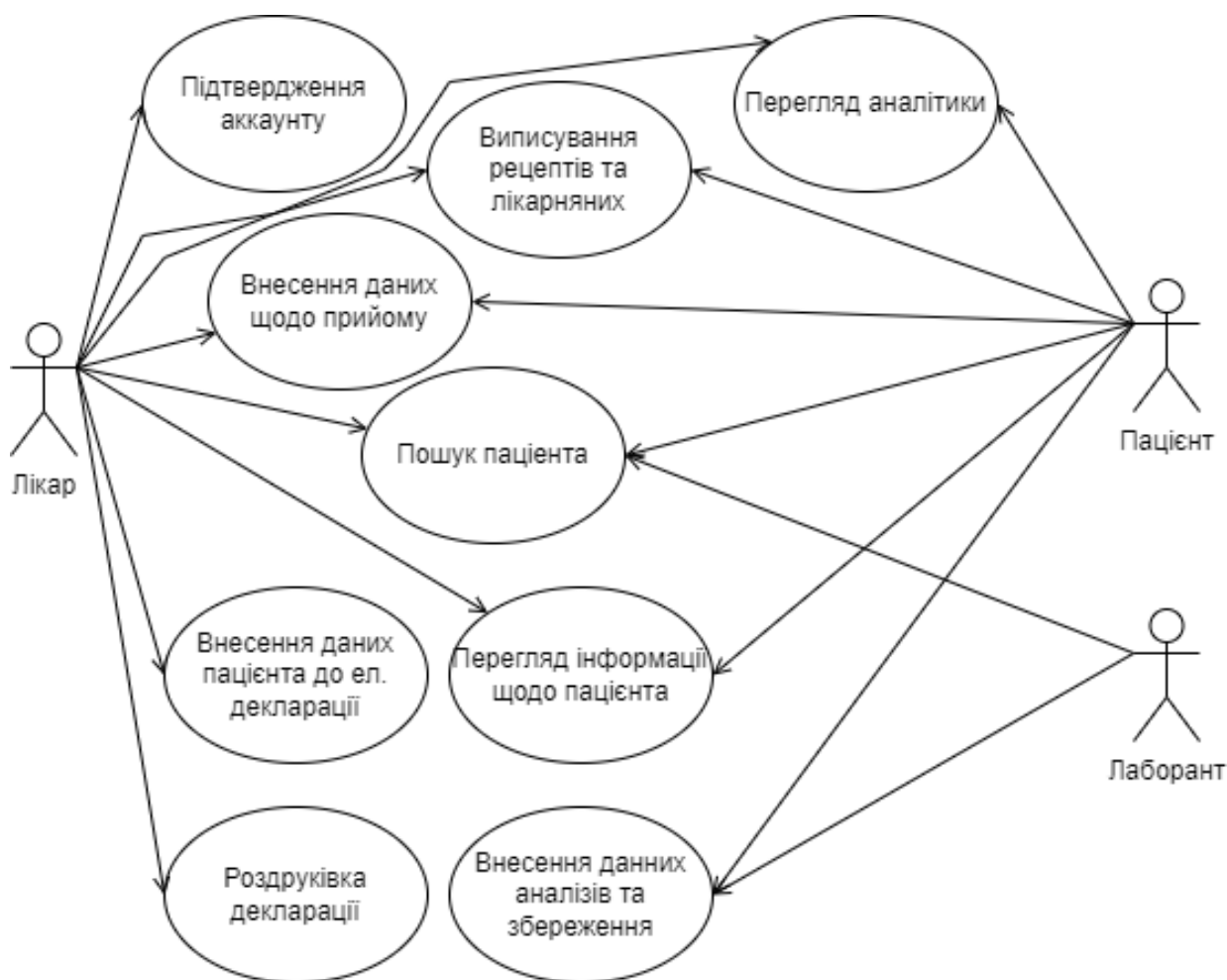


Рисунок 1.6 – Загальна діаграма use case

1.4 Висновки за розділом 1

Відповідно до сформульованої теми дипломного проєкту, було проаналізовано предметну область, розглянуто два програмні аналоги та сформовано вимоги для подальшої розробки програмного забезпечення для ведення та аналізу медичних карток пацієнтів у медичних закладах.

Визначено завдання роботи:

- визначення вимог до програмного забезпечення;
- проєктування програмного забезпечення;
- реалізація і налагодження програмного забезпечення;
- тестування програмного забезпечення.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування

Для реалізації програмного рішення обрано мову програмування C#, оскільки вона має дуже добру інтеграцію з Microsoft SQL Server, а також зручну технологію розробки графічних застосунків – WPF та ряд інших переваг. Найчастіше для розробки графічних застосунків використовують мови програмування C# та C++.

C++ – це статично типізована, скомпільована, універсальна, чутлива до регістру, мова програмування вільної форми, яка підтримує процедурне, об'єктно-орієнтоване та загальне програмування [5].

C++ вважається мовою середнього рівня, оскільки містить комбінацію мов високого та низького рівня [5].

C# – це об'єктно-орієнтована мова програмування від Microsoft, яка має на меті поєднати обчислювальну потужність C++ із легкістю програмування Visual Basic. C# заснований на C++ і містить функції, подібні до Java [5].

C# розроблено для загальномовної інфраструктури (CLI), яка складається з виконуваного коду та середовища виконання, що дозволяє використовувати різні мови високого рівня на різних комп'ютерних платформах та архітектурах [6].

Наступні причини роблять C# широко вживаною професійною мовою:

- сучасна мова програмування загального призначення;
- об'єктно-орієнтованість;
- компонентно-орієнтованість;
- легка до вивчення;
- структурована;
- дозволяє створювати ефективні програми;
- компілюється на різних комп'ютерних платформах;
- є частиною .Net Framework.

Серед основних переваг мови C# можна виділити наступні:

- чистий синтаксис;
- велика кількість корисних модулів;
- C-подібний синтаксис мови;
- уніфікована система типізації;
- C# створювався паралельно з каркасом Framework .Net і повною мірою враховує всі його можливості – як FCL, так й CLR;
- C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами;
- C# є потужною об'єктною мовою з можливостями спадкування й універсализації;
- C# є спадкоємицею мов C/C++, зберігаючи кращі риси цих популярних мов програмування;
- потужна бібліотека каркасів додатків дозволяє легко та швидко будувати вебслужби та інші компоненти, також це досить спрощує роботу з базами даних;
- реалізація, що сполучає побудову надійного й ефективного коду, є важливим чинником, що сприяє успіху C# [7].

Також значним плюсом є дуже потужне середовище програмування для мови C# є IDE під назвою Microsoft Visual Studio, яке в собі також містить компоненти для роботи з базами даних. Для написання програми було використано середовище Visual Studio 2022.

Порівняння обраної мови програмування із іншими популярними мовами програмування наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерії порівняння	Мови програмування		
	C#	C++	Python
Наявність літератури та простота вивчення	++	+	++

Продовження таблиці 2.1

Критерії порівняння	Мови програмування		
	C#	C++	Python
Підтримка об'єктно-орієнтованої парадигми програмування	+	+	+
Зручність використання для створення Windows-застосунків	++	+	-
Зручність використання засобів розробки	++	+	+
Сприйняття коду	++	+	+
Стандартні бібліотеки шаблонів	++	+	+
Кросплатформність	+	+	+
Продуктивність роботи	++	+	+

За результатами порівняння популярних мов програмування для створення графічних застосунків було обрано мову C# через ряд переваг, таких як зручність використання для створення Windows-застосунків, зручність використання засобів розробки, добре сприйняття коду, наявність широкого спектру стандартних бібліотек та високу продуктивність роботи.

2.2 Проєктування архітектури програмного забезпечення

Для розробки програмного забезпечення ведення та аналізу медичних карток пацієнтів було обрано використовувати клієнт-серверну архітектуру.

Архітектура клієнт-сервер [8] – це обчислювальна модель, в якій кілька компонентів працюють у чітко визначених ролях для комунікації. Сервер розміщує, доставляє та керує більшістю ресурсів і послуг, які споживає клієнт. Цей тип архітектури спільних ресурсів має один або кілька клієнтських комп'ютерів, під'єднаних до центрального сервера через мережу або інтернет-з'єднання.

Клієнтські комп'ютери забезпечують інтерфейс, який дозволяє користувачеві комп'ютера запитувати послуги сервера та відображати результати, які повертає сервер. Сервери чекають надходження запитів від клієнтів, а потім відповідають на них.

Клієнти часто знаходяться на робочих станціях або на персональних комп'ютерах, тоді як сервери розташовані в інших місцях мережі, як правило, на більш потужних машинах. Ця обчислювальна модель особливо ефективна, коли клієнти і сервер мають різні завдання, які вони регулярно виконують. При обробці лікарняних даних, наприклад, на клієнтському комп'ютері може бути запущена прикладна програма для введення інформації про пацієнта, тоді як на серверному комп'ютері запущена інша програма, яка керує базою даних, в якій постійно зберігається інформація [9].

Після вибору клієнт-серверної архітектури постає питання визначення способу комунікації між сервером та клієнтом. Сучасним способом є використання HTTP запитів. Реалізувати це допоможе технологія REST API.

REST API – це інтерфейс програмування прикладних програм (API або веб-API), який відповідає обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful. REST розшифровується як репрезентаційна передача стану і був створений комп'ютерним науковцем Роем Філдінгом.

API – це набір визначень і протоколів для створення та інтеграції прикладного програмного забезпечення [10].

Для розробки клієнту було обрано архітектуру MVVM. Model-View-ViewModel – це шаблон проєктування програмного забезпечення, структурований для розділення логіки програми та елементів керування інтерфейсом користувача. Як і багато інших шаблонів проєктування, MVVM допомагає організувати код і розбити програми на модулі, щоб зробити розробку, оновлення та повторне використання коду простішими та швидшими.

Поділ коду в MVVM поділяється на View, ViewModel і Model [11]:

- View – це набір видимих елементів, який також отримує введення користувача. Це включає в себе інтерфейси користувача (UI), анімацію та текст;
- ViewModel розташована між шарами View і Model. Тут розміщуються елементи керування для взаємодії з View, тоді як прив'язування використовується для з'єднання елементів інтерфейсу користувача у View з елементами керування ViewModel;
- Model містить логіку програми, яку витягує ViewModel після власного отримання введених даних від користувача через View.

Для розробки View буде використовуватися фреймворк WPF. Windows Presentation Foundation використовується для створення клієнтських програм Windows, які працюють в операційній системі Windows. WPF використовує XAML (декларативна мова розмітки) як інтерфейс мови, а C# – як внутрішню мову.

Для доступу до бази даних буде використовуватися Entity Framework. Структура Entity – це структура ORM (об'єктно-реляційне відображення), яку Microsoft робить доступною як частину розробки .NET. Його мета – абстрагувати зв'язки з реляційною базою даних таким чином, щоб розробник міг зв'язатися з сутністю бази даних як з набором об'єктів, а потім із класами на додаток до їхніх властивостей [12].

Наразі EF в основному допускає два типи підходів, пов'язаних із цим використанням – Database-First та Code-First. Для зручності розробки використовується Code-First, адже цей підхід дозволяє створювати та модифікувати таблиці бази даних напряму з програмного коду. Невід'ємною функцією такого підходу є міграції. Вони забезпечують спосіб поступового оновлення схеми бази даних, щоб підтримувати її синхронізацію з моделлю даних програми, зберігаючи наявні дані в базі даних.

2.3 Проєктування структури бази даних

Схема баз даних – це структура системи баз даних, описана формальною

Для розроблюваної бази даних зв'язки між атрибутами таблиць були створені при створенні бази даних, адже вона була створена за допомогою скрипту на мові T-SQL. Зв'язки, в свою чергу, використовуються “один до багатьох”. Схема даних зображена на рисунку 2.1.

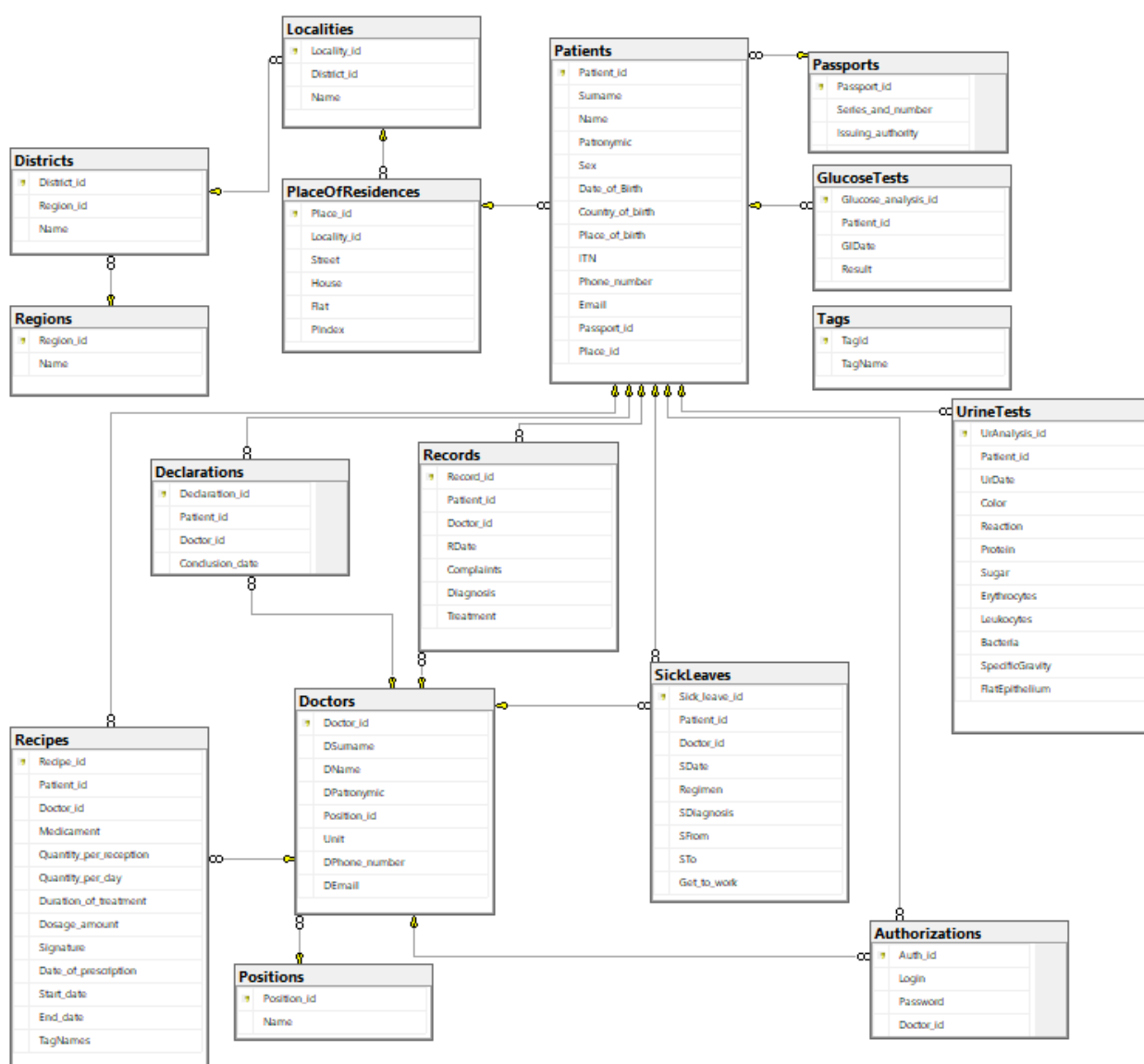


Рисунок 2.1 – Структура бази даних

2.3.1 Аналіз сутностей

Сутність – це будь-який об'єкт в базі даних, який можна виділити виходячи з суті предметної області, для якої розробляється ця база даних.

Виділено наступні сутності:

- лікарі – містить інформацію про лікаря, тобто його дані, посаду та номер амбулаторії;
- пацієнти – включає інформацію про пацієнта, а саме: особисті дані, дані паспорту та місця проживання;
- декларації – містить інформацію про укладені декларації, тобто дані конкретного пацієнта та лікаря, а також дату заключення декларації;
- записи – включає інформацію медичної картки пацієнтів, яка заповнюється на прийомі;
- рецепти – містить інформацію про виписуваний лікарський засіб та дані для прийому;
- лікарняні – включає інформацію про хворого пацієнта, лікаря, який виписав лікарняний та дати початку та кінця роботи;
- аналізи крові на глюкозу – включає інформацію про пацієнта, який проходить аналіз та данні аналізу;
- аналізи сечі – включає інформацію про пацієнта, який проходить аналіз та данні аналізу;
- паспорти – містить інформацію про дані паспортів пацієнтів;
- місце проживання – включає інформацію про фактичне місце проживання пацієнтів;
- посади – містить інформацію про назву посади лікарів у медичному закладі;
- області – включає інформацію про назви областей країни;
- райони – містить інформацію про назви районів країни;

- населені пункти – містить інформацію про назви населених пунктів країни;
- авторизація – включає інформацію для входу користувачів в систему;
- теги – містить інформацію про назви тегів.

2.3.2 Аналіз атрибутів

Перераховані вище сутності містять різні атрибути. Атрибут – властивість сутності (заголовок стовпчика таблиці).

Роздивимось атрибути вищеназваних сутностей:

- лікарі – Doctor_id, DSurname, DName, DPatronymic, Position_id, Unit, DPhone_number та DEmail;
- пацієнти – Patient_id, Surname, Name, Patronymic, Sex, Date_of_Birth, Country_of_birth, Place_of_birth, ITN, Phone_number, Email, Passport_id та Place_id;
- декларації – Declaration_id, Patient_id, Doctor_id, Conclusion_date;
- записи – Record_id, Patient_id, Doctor_id, RDate, Complaints, Diagnosis та Treatment;
- рецепти – Recipe_id, Patient_id, Doctor_id, Medicament, Quantity_per_reception, Quantity_per_day, Duration_of_treatment, Dosage_amount, Signature, Date_of_prescription, Start_date, End_date та TagNames;
- лікарняні – Sick_leave_id, Patient_id, Doctor_id, SDate, Regimen, SDiagnosis, SFrom, STo і Get_to_work;
- аналізи крові на глюкозу – Glucose_analysis_id, Patient_id, GIDate та Result;
- аналізи сечі – UrAnalysis_id, Patient_id, UrDate, Color, Reaction, Specific_gravity, Protein, Sugar, Erythrocytes, Leukocytes, Bacteria та Flat_epithelium;

- паспорти – Passport_id, Series_and_number, Issuing_authority та Date_of_issue;
- місце фактичного проживання – Place_id, Locality_id, Street, House, Flat та PIndex;
- посади – Position_id, Position;
- області – Region_id, Region;
- райони – District_id, Region_id та District;
- населені пункти – Locality_id, District_id та Locality;
- авторизація – Auth_id, Login, Password, Doctor_id та Patient_id;
- теги – TagId, TagName;

2.4 Структурна схема програмного проєкту

На рисунку 2.2 зображена структурна схема розроблюваної програмної системи ведення та аналізу медичних карток пацієнтів. Як видно, елементи знаходяться в загальному просторі програмного забезпечення «ProHealth» та являють комплексну систему.



Рисунок 2.2 – Структурна схема програмного проєкту

База даних є сховищем даних для всієї системи, яке взаємодіє з модулем для управління базою даних за допомогою технології Entity Framework. Зв'язок між ними є двохсторонній, тому що ведеться робота на збереження та отримання даних.

Сервер є ядром системи, тобто є головним елементом системи. Він має двосторонні зв'язки з модулем управління базою даних та клієнтом системи. Сервер отримує запити від клієнта на отримання або збереження даних, оброблює їх та робить відповідні дії з базою даних через модуль «DB.DAL».

Клієнт являє собою графічний застосунок, за допомогою якого користувачі системи отримують інтерфейс для роботи в програмному забезпеченні.

2.5 Висновки за розділом 2

У цьому розділі було проаналізовано мови програмування C# та C++. Для реалізації програмного продукту було обрано використовувати мову C#. Після вибору мови було виконано проєктування архітектури програмного забезпечення та структури бази даних. Також сформовано структурну схему програмного проєкту.

Виконана розробка архітектури програми є твердим підґрунтям для наступного кроку – розробки програми.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програмного забезпечення

На рисунку 3.1 зображена укрупнена функціональна схема програмної системи, на якій видно основні етапи роботи програмного забезпечення.

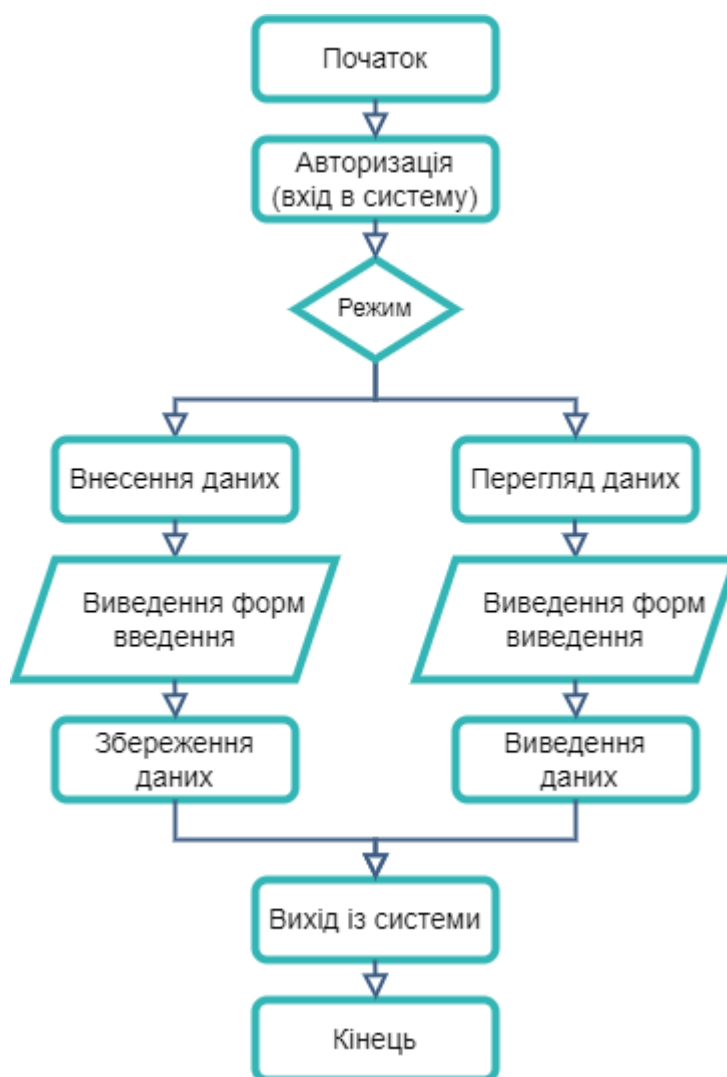


Рисунок 3.1 – Функціональна схема програмного забезпечення

Для більш детального розуміння процесу функціонування програмного забезпечення нижче описані дві діаграми послідовності, які відображають внесення запису до електронної карти лікарем (рисунок 3.2) та внесення результатів аналізів лаборантом.

Основний успішний сценарій для прецеденту «Внесення запису до електронної карти»:

- лікар відкриває основне вікно програми, в якому необхідно ввести логін та пароль, лікар вводить дані, система перевіряє дані та надає доступ на рівні лікаря;
- лікар відкриває вікно пошуку пацієнтів і вводить інформацію про пацієнта, система виконує пошук пацієнта в базі даних та виводить перелік пацієнтів, що відповідають пошуковій інформації, лікар обирає відповідного пацієнта та система виводить картку пацієнта;
- лікар створює запис поточного прийому та вводить дані, після того як обирає "Зберегти" запис зберігається в базі даних.

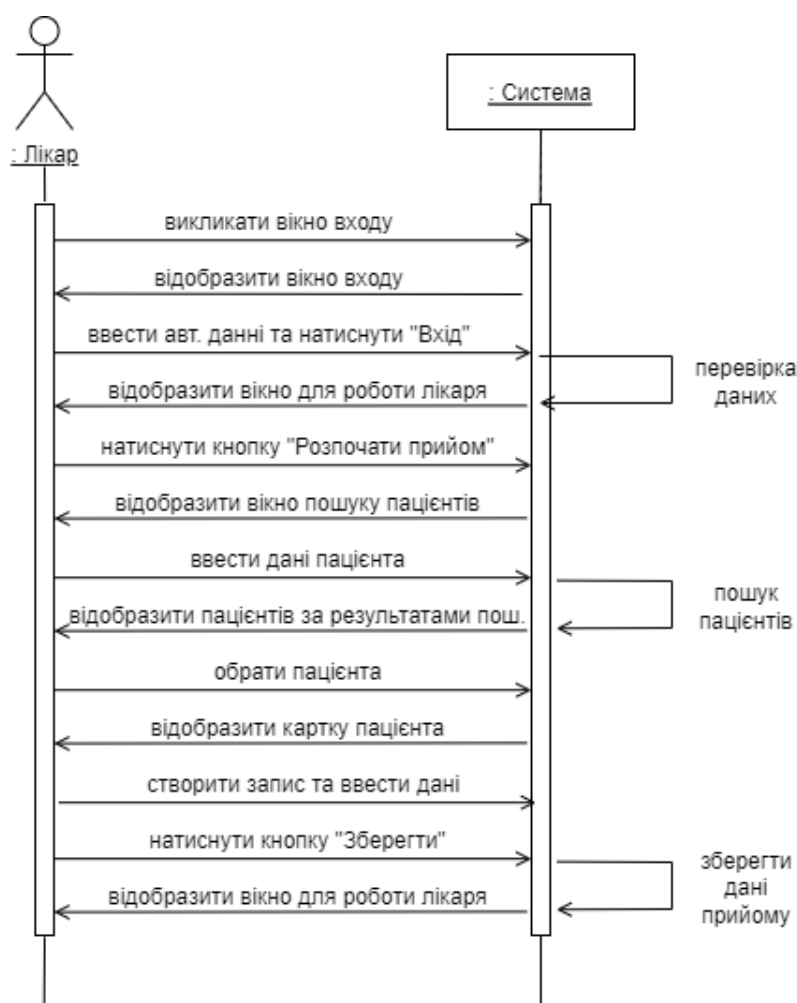


Рисунок 3.2 – Діаграма послідовності для прецеденту «Внесення запису до електронної карти»

Основний успішний сценарій для прецеденту «Внесення результатів аналізів» (діаграма на рисунку 3.3):

- лаборант відкриває вікно пошуку пацієнтів і вводить інформацію про пацієнта, система виконує пошук пацієнта в базі даних та виводить перелік пацієнтів, що відповідають пошуковій інформації, лаборант обирає відповідного пацієнта та система відкриває сторінку внесення даних аналізу;
- лаборант вносить дані аналізу до системи, а після того, як обирає "Зберегти", запис зберігається в базі даних.

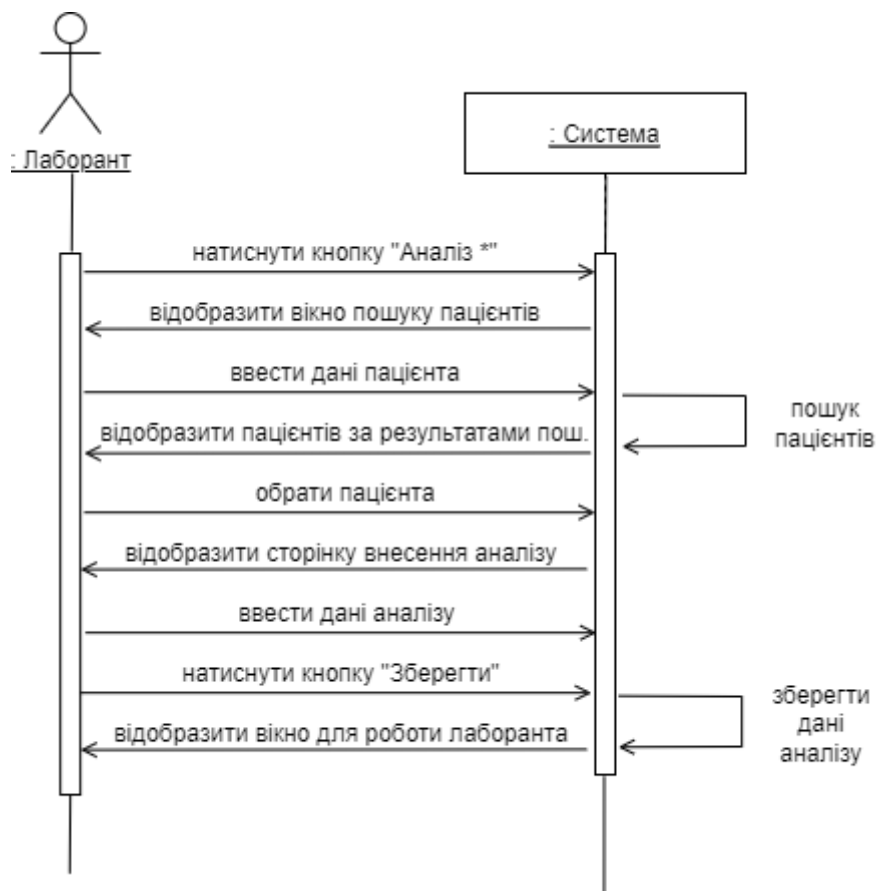


Рисунок 3.3 – Діаграма послідовності для прецеденту «Внесення результатів аналізів»

3.2 Опис модулів реалізованої програми

Програмне забезпечення побудоване на основі клієнт-серверної архітектури та являє собою модульний проєкт. Модульна структура полегшує

сприйняття програмної системи та дозволяє легко модифікувати та додавати нові функції.

Усі модулі, а саме сервер, клієнт та модуль роботи з базою даних, написані на мові програмування C# у середовищі розробки Microsoft Visual Studio 2022. В якості серверу бази даних використовується Microsoft SQL Server, а для налаштування – Microsoft SQL Server Management Studio 18.

Комунікація між сервером програмної системи та клієнтом відбувається за допомогою технології Rest API, а доступ до бази даних забезпечується технологією Entity Framework. Підходом до роботи з базою даних є Code First, завдяки якому є можливість повністю маніпулювати з базою даних напряду з коду програми. Різноманітні запити на отримання, збереження та модифікування даних дозволило зробити використання технології LINQ.

Клієнт розроблений за допомогою технології WPF, яка дозволяє зручно створювати графічні застосунки, завдяки можливості розроблення інтерфейсу користувача на мові розмітки XAML.

Програмна система побудована з акцентом на асинхронне програмування, за допомогою ключових слів `async` та `await`. Такий підхід має багато переваг таких як, наприклад, паралельна обробка декількох запитів від клієнтів сервером та підтримування чутливості інтерфейсу.

Велика увага приділена обробці нестандартних ситуацій та їх логуванню до консолі серверу (для швидкого виявлення та реагування) та до файлів (для помилок та трасування). Логування ведеться за допомогою `nLog` – продуктивною та простою у використанні платформи.

Рішення програмної системи складається з трьох модулів. Перший – це модуль для роботи з базою даних, другий – це клієнт, а третій – це ядро системи, тобто сервер. Програмне забезпечення було створене за допомогою мови програмування C# у середовищі розробки Microsoft Visual Studio 2022. Структура рішення наведена на рисунку 3.4.

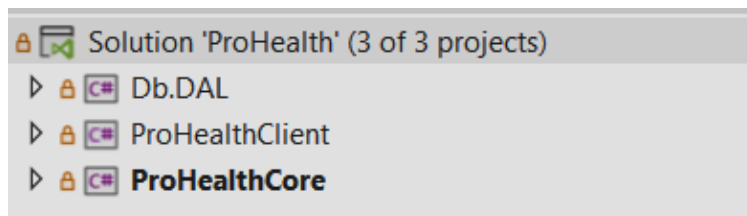


Рисунок 3.4 – Структура рішення

Почнемо огляд модулю для роботи з базою даних «Db.DAL». Структура проєкту наведена на рисунку 3.5. Всі наявні класи – це моделі.

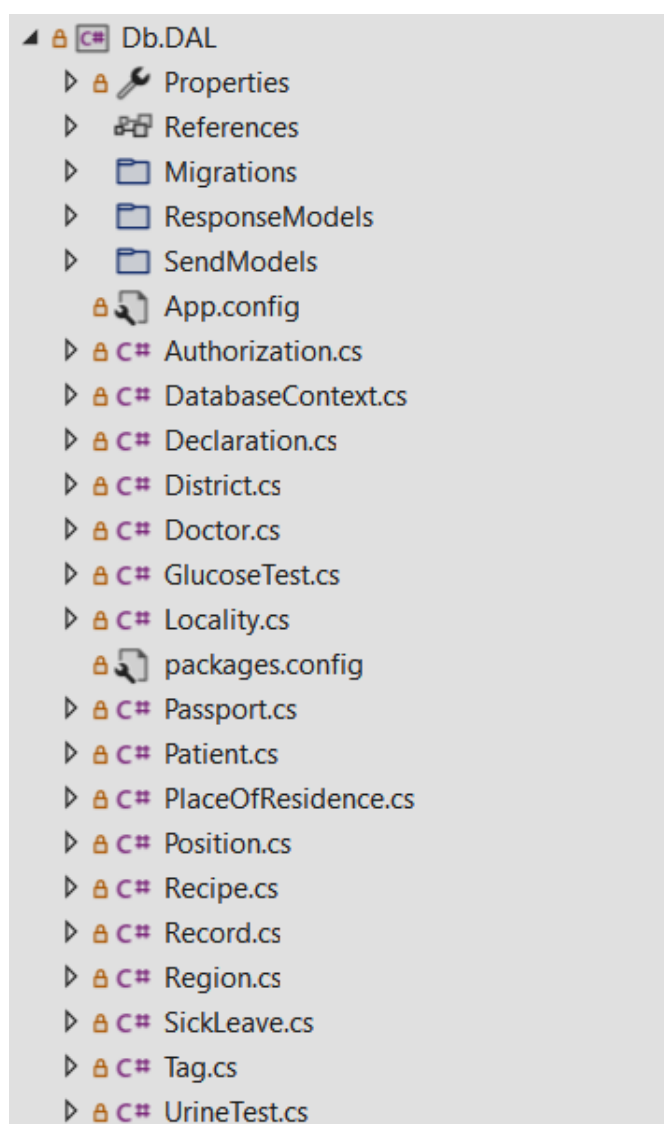


Рисунок 3.5 – Структура проєкту «Db.DAL»

В корні проєкту наявні класи, які відповідають таблицям у базі даних. У папці «Migrations» знаходяться класи для роботи міграцій у проєкті. Функція міграції в Entity Framework забезпечує спосіб поступового оновлення схеми бази даних, щоб підтримувати її синхронізацію з моделлю даних програми, зберігаючи наявні дані в базі даних. У папках «ResponseModels» та «SendModels» знаходяться моделі для комунікації клієнта та сервера. Структура папок наведена на рисунку 3.6

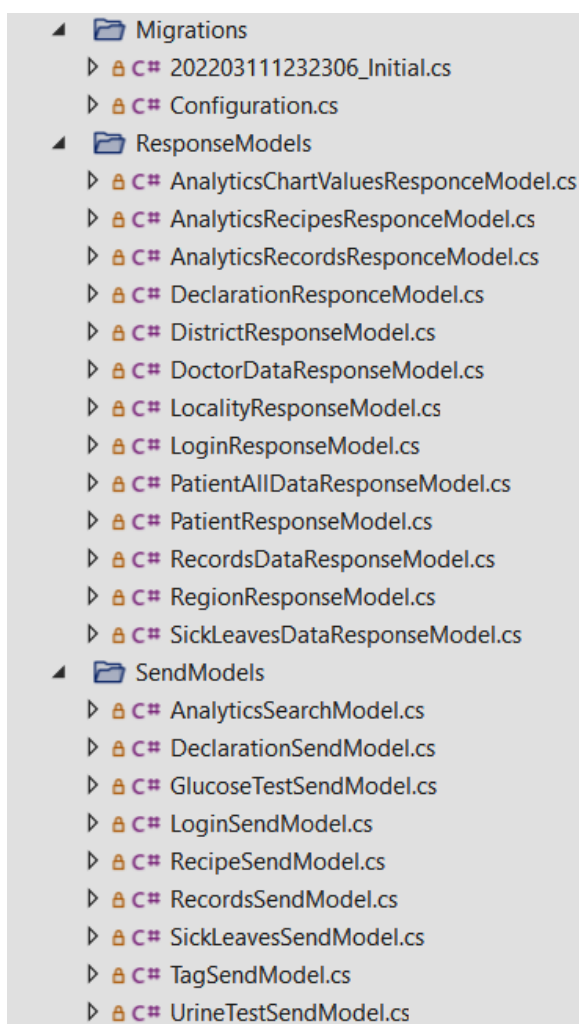


Рисунок 3.6 – Структура папок проєкту «Db.DAL»

Розглянемо наступним модуль ядра системи «ProHealthCore». Структура проєкту наведена на рисунку 3.7. Він містить три класи. Клас «CoreLauncher» відповідає за запуск серверу за допомогою фреймворку Nancy, який відповідає

за HTTP запити в ядрі системи. Клас «MainModule» містить функції для прийому HTTP запитів від клієнта та виконує необхідні дії у класі «Core».

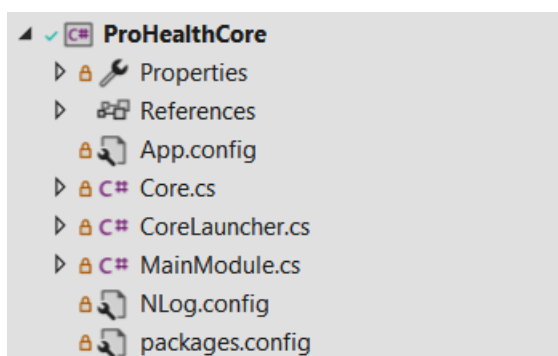


Рисунок 3.7 – Структура проєкту «ProHealthCore»

Останнім є модуль, який є клієнтом програмної системи – «ProHealthClient». Структура проєкту наведена на рисунку 3.8.

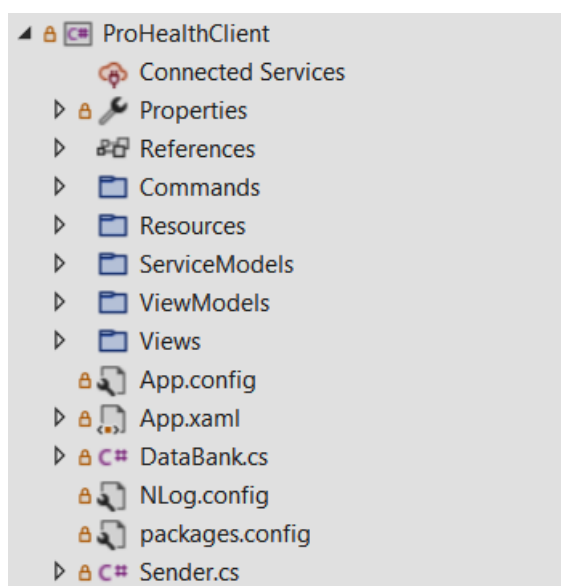


Рисунок 3.8 – Структура проєкту «ProHealthClient»

У корені проєкту наявні два класи – «DataBank» та «Sender». Перший являє собою статичний клас. В ньому є статичні змінні, наприклад, рядок підключення до бази даних, ідентифікатор лікаря та інші. За допомогою статичних змінних можливо отримати доступ до них з будь-якого місця

проекту та легко змінити в разі потреби. «Sender» – це клас, який містить функції для відправки запитів до серверу системи.

Головною папкою в проєкті клієнта – є папка «Views», у якій лежать класи вікон графічного інтерфейсу користувача. Структура папки наведена на рисунку 3.9.

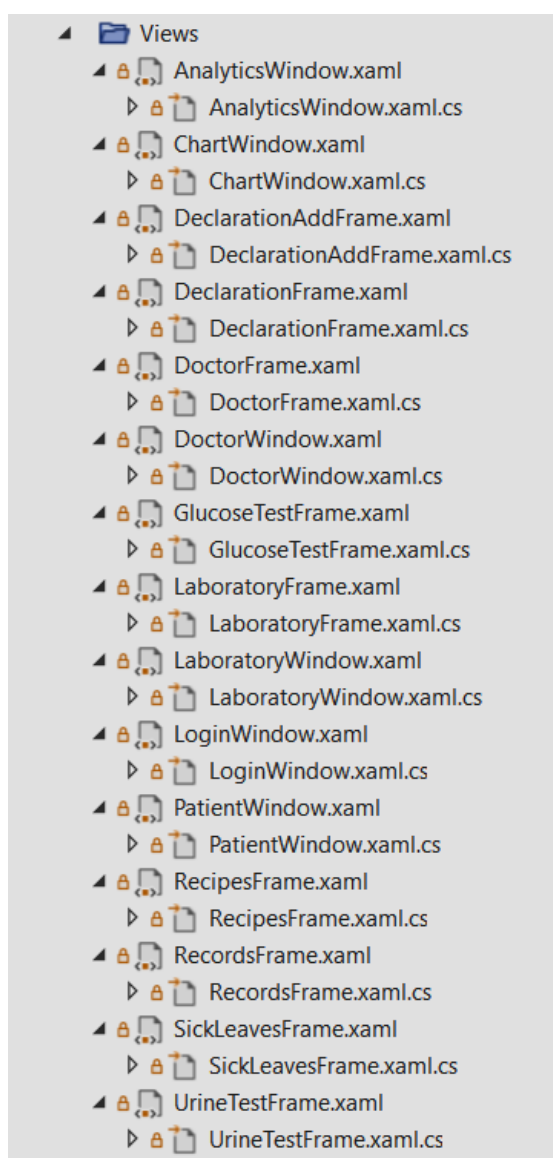


Рисунок 3.9 – Структура папки «Views»

Розглянемо детально файли графічного інтерфейсу з папки «Views».

AnalyticsWindow – це вікно з аналітичними даними.

ChartWindow – це вікно для графіків, на основі даних аналізів.

DeclarationAddFrame – це сторінка створення декларації.

DeclarationFrame – сторінка для перегляду декларацій.

DoctorFrame – головна сторінка для лікаря.

DoctorWindow – вікно лікаря.

GlucoseAnalysFrame – сторінка для створення аналізу крові на глюкозу.

LaboratoryFrame – головна сторінка для лабораторії.

LaboratoryWindow – вікно лабораторії.

MainWindow – вікно авторизації.

PatientWindow – вікно пацієнта.

RecipesFrame – сторінка для створення рецептів лікарських засобів.

RecordsFrame – сторінка перегляду картки пацієнта та ведення прийому.

SickLeavesFrame – сторінка перегляду та створення лікарняних пацієнтів.

UrineAnalysFrame – сторінка для створення аналізу сечі.

Розберемо останні папки у проєкті клієнта. Структура наведена на рисунку 3.10.

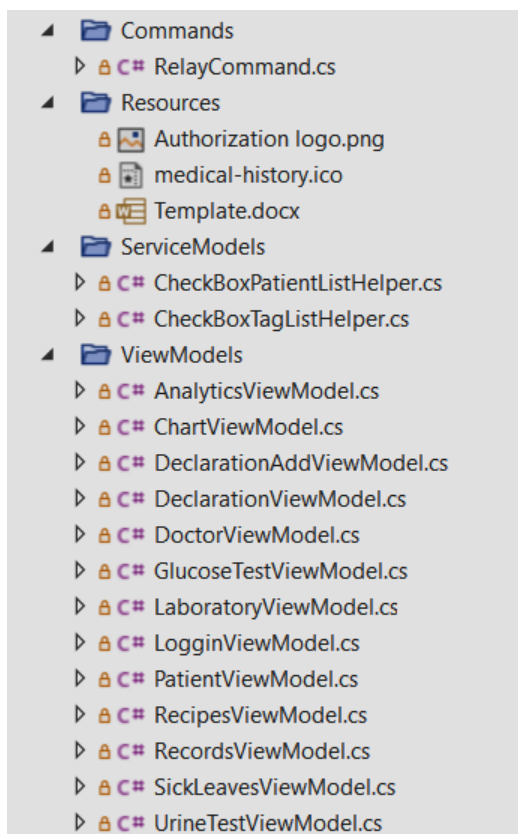


Рисунок 3.10 – Структура папок проєкту «ProHealthClient»

Невід’ємною частиною файлів графічного інтерфейсу – є класи ViewModel, які знаходяться в однойменній папці. В цих класах знаходяться функції, які забезпечують графічні форми логікою та даними.

Опис методів класів ViewModel наведений у таблиці 3.1.

Таблиця 3.1 – Методи класів ViewModel

Назва класу	Метод класу	Опис
AnalyticsViewModel	AnalyticsViewModel	Конструктор
	AddTagClick	Додавання тегу
	FindDataClick	Пошук даних за фільтрами пошуку
	ShowChartClick	Показ графіку за даними аналізів
	LoadTagsList	Завантаження списку тегів
	LoadPatientListAsync	Завантаження списку пацієнтів
ChartViewModel	MakeChart	Обробка даних для графіку
DeclarationAddViewModel	DeclarationAddViewModel	Конструктор
	CreateClick	Створення декларації
	BackClick	Повернення на попередню сторінку
	ShowDataAsync	Завантаження даних для показу
	GetDistricts	Завантаження списку районів

Продовження таблиці 3.1

Назва класу	Метод класу	Опис
	GetLocalities	Завантаження списку регіонів
	SaveDeclaration	Збереження декларацій
DeclarationViewModel	DeclarationViewModel	Конструктор
	SearchClick	Пошук своїх пацієнтів за фільтрами пошуку
	SearchAllClick	Пошук серед всіх пацієнтів за фільтрами пошуку
	ResaveClick	Перезаключення декларації
	AllInfoClick	Відкриття декларації в Word
	AddClick	Відкриття форми для додавання декларації
	BackClick	Повернення на попередню сторінку
	ShowDataAsync	Завантаження даних для форми
	DeclarationReport	Створення декларації в Word
	ReplaceWordStrub	Допоміжна функція для формування декларації в Word
DoctorViewModel	DoctorViewModel	Конструктор
	RecordsClick	Перехід на сторінку з електронними картами

Продовження таблиці 3.1

Назва класу	Метод класу	Опис
	DeclarationClick	Перехід на сторінку з деклараціями
	RecipesClick	Перехід на сторінку з рецептами
	SickLeavesClick	Перехід на сторінку з лікарняними
	AnalyticsClick	Перехід на сторінку з аналітикою
	ShowDataAsync	Завантаження даних для форми
GlucoseTestViewModel	GlucoseTestViewModel	Конструктор
	CreateClick	Додавання аналізу
	BackClick	Повернення на попередню сторінку
	ShowDataAsync	Завантаження даних для форми
	SaveTest	Збереження аналізу
LaboratoryViewModel	LaboratoryViewModel	Конструктор
	GlucoseTestClick	Відкриття форми додавання аналізу
	UrineTestClick	Відкриття форми додавання аналізу сечі
LoginViewModel	LoginViewModel	Конструктор
	Authorize	Авторизація користувача
PatientViewModel	PatientViewModel	Конструктор
	AllInfoClick	Декларація в Word

Продовження таблиці 3.1

Назва класу	Метод класу	Опис
	LoadData	Завантаження даних для форми
	DeclarationReport	Створення декларації в Word
	ReplaceWordStrub	Допоміжна функція для формування декларації в Word
RecipesViewModel	RecipesViewModel	Конструктор
	CreateClick	Додавання рецепту
	SaveRecipe	Збереження рецепту
	BackClick	Повернення на попередню сторінку
	AddTagClick	Додавання тегу
	ShowDataAsync	Завантаження даних для форми
	LoadTags	Завантаження тегів
RecordsViewModel	RecordsViewModel	Конструктор
	SaveClick	Збереження даних
	BackClick	Повернення на попередню сторінку
	InfoClick	Відкриття вікна з інформацією про пацієнта
	AddTagClick	Додавання тегу
	LoadTags	Завантаження тегів
	LoadPatientList	Завантаження списку пацієнтів

Продовження таблиці 3.1

Назва класу	Метод класу	Опис
	ShowDataAsync	Завантаження даних для форми
	SaveRecords	Збереження записів
UrineTestViewModel	UrineTestViewModel	Конструктор
	CreateClick	Додавання аналізу сечі
	BackClick	Повернення на попередню сторінку
	ShowDataAsync	Завантаження даних для форми
	SaveTest	Збереження аналізу сечі
SickLeavesViewModel	SickLeavesViewModel	Конструктор
	SaveClick	Додавання лікарняного
	BackClick	Повернення на попередню сторінку
	LoadPatientList	Завантаження списку пацієнтів
	ShowDataAsync	Завантаження даних для форми
	SaveSickLeaves	Збереження лікарняного

У папці «ServiceModels» знаходяться сервісні моделі для забезпечення роботи певних елементів на формі.

У папці «Commands» знаходиться клас для забезпечення роботи подій екранних форм.

У папці «Resources» знаходяться значки та шаблон для створення декларацій.

3.3 Висновки за розділом 3

В третьому розділі було розроблено алгоритм функціонування програмного забезпечення, що стало основою до розробки. Також описано модулі, з яких складається розроблювана система ведення та аналізу карток пацієнтів.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена, насамперед, для медичних закладів різних спрямувань та спеціалізацій. В ній можуть працювати лікарі первинної та вторинної ланки, персонал лабораторії та пацієнти.

4.2 Умови виконання програми

Програма має коректно працювати за умов:

- наявності та запуску останньої версії програми;
- наявності СУБД MS SQL Server Management Studio;
- наявності налаштованої бази даних;
- наявності MS Word для створення декларацій;
- збереження наборів інформації у директорії, в яких є права на читання та запис файлів, що не є заблокованими;
- правильного введення даних в програму, згідно із вимогами.

4.3 Виконання програми

Для роботи у програмному забезпеченні необхідно, перш за все, запустити програму-сервер на серверному комп'ютері. Вигляд консолі серверу програми ProHealth після запуску зображений на рисунку 4.1.

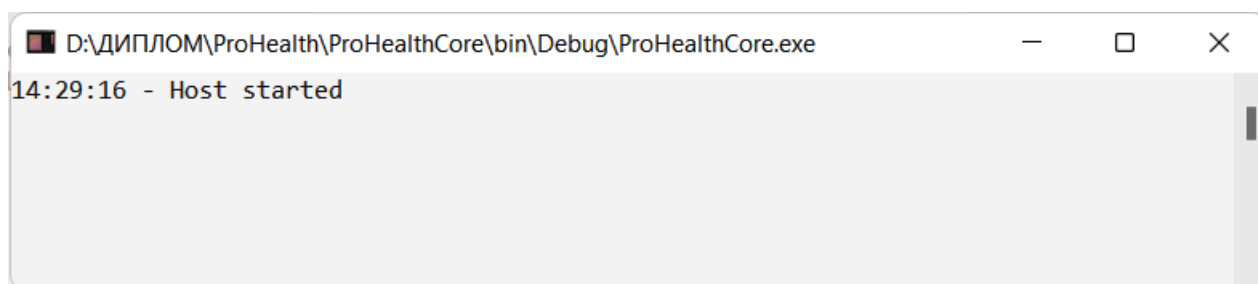


Рисунок 4.1 – Консоль серверу програми ProHealth

Для запуску клієнтського застосунку потрібно запустити файл програми ProHealth.exe, значок якої зображений на рисунку 4.2.

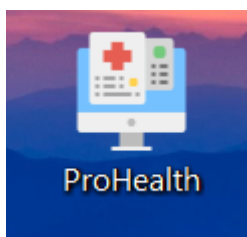


Рисунок 4.2 – Значок програми ProHealth

Після того, як програму було запущено, користувача зустрічає вікно авторизації, яке зображено на рисунку 4.3. У випадку вводу некоректних даних, програма сповіщає про це користувачу відповідним повідомленням (рисунок 4.4).

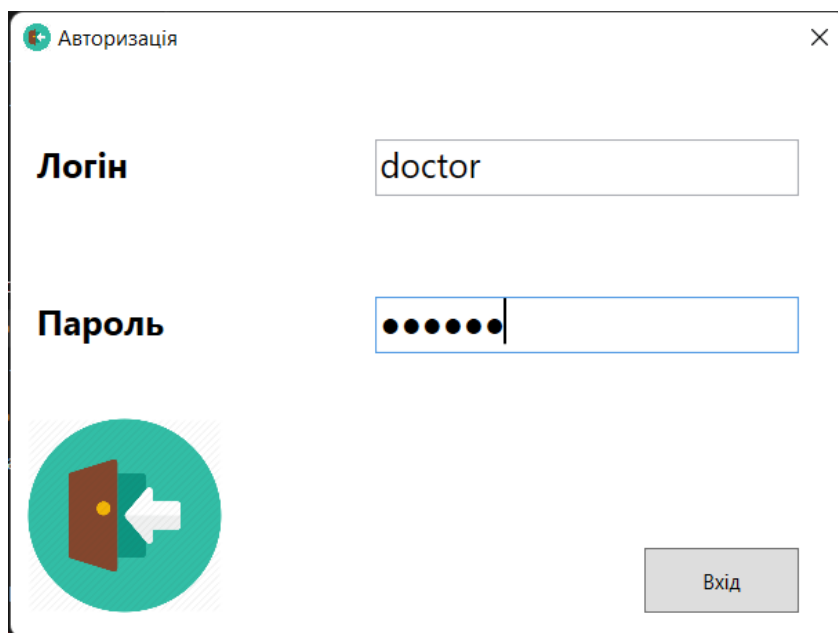


Рисунок 4.3 – Вікно авторизації

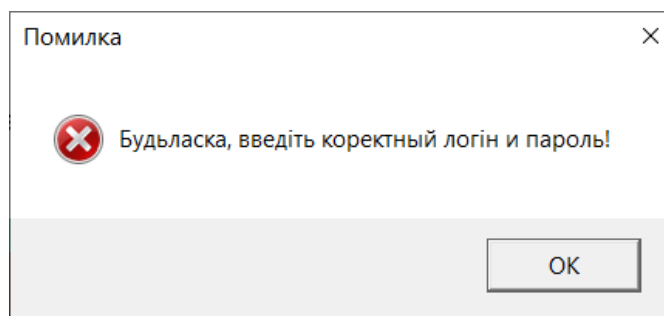


Рисунок 4.4 – Вікно помилки

Вигляд консолі серверу програми ProHealth після запуску та авторизації лікаря зображений на рисунку 4.5. На ньому видно, що ведеться логування кожного кроку з наявними даними для активного моніторингу роботи системи.

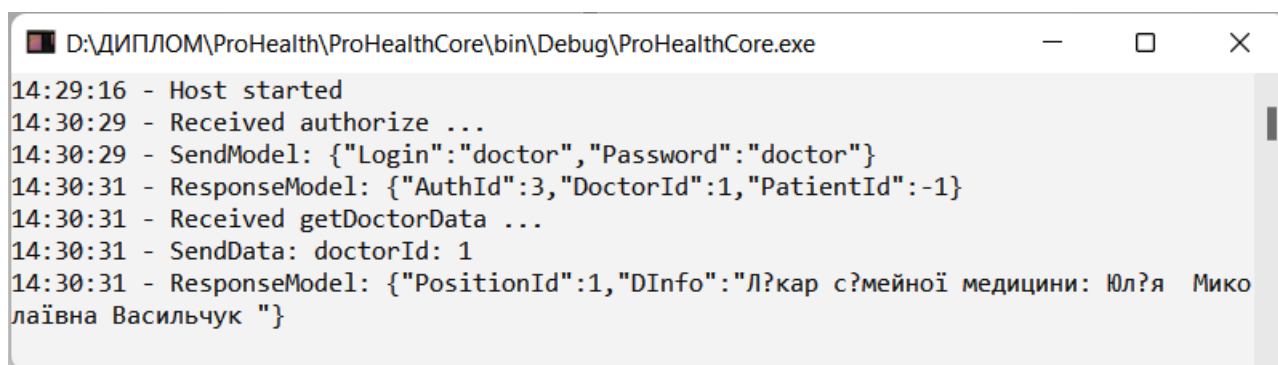


Рисунок 4.5 – Консоль серверу програми ProHealth

Роздивимося вікно лікаря. Після вводу коректних авторизаційних даних лікаря він потрапляє на головну сторінку лікаря первинної ланки (рисунок 4.6) чи вторинної ланки (рисунок 4.7). Відмінність між ними в тому, що лікар вторинної ланки не має можливості створювати та переглядати укладенні декларації, адже він за визначенням не може цього робити.



Рисунок 4.6 – Вікно лікаря первинної ланки

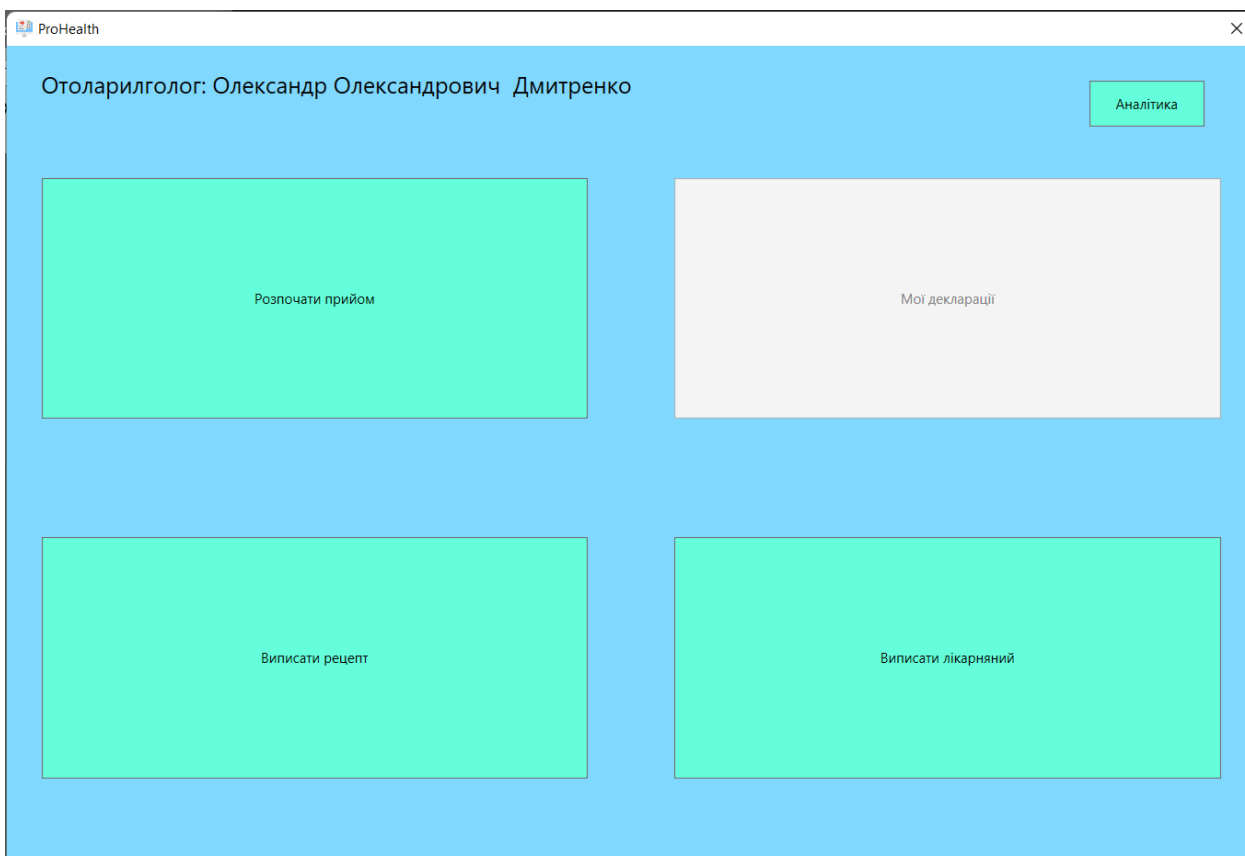


Рисунок 4.7 – Вікно лікаря вторинної ланки

На головній сторінці розміщується інформація про посаду та ПІБ лікаря. Також є чотири кнопки: «Розпочати прийом», «Мої декларації», «Виписати рецепт», «Виписати лікарняний» та «Аналітика». Почнемо розгляд з першої. Сторінка прийому пацієнта зображена на рисунку 4.8.

ProHealth

Пациєнт: Архипов Максим Борисович Декларація укладена Інформація Поточна дата: 03.04.2022

Дата народження: 30-09-1979 Номер телефону: +380503626500 м. Запоріжжя, Запорізький район, Запорізька обл., в. Нова, буд. 75, кв. 45

Дата запису	Скарги	Діагноз	Лікування
01-03-2022	Нежить, підвищена температура, кашель #ГрупаРизику1	Гострий трахеїт	Приймати Сінупред по 1 табл. 3 р. в день, сироп Лазолван по 5 мл 3 р. в день
05-03-2022	Підвищений артеріальний тиск, прискорене серцебиття #ГрупаРизику1	Гіпертонія	Приймати Лозартан 50 мг по 1 табл. в день
10-03-2022	Підвищена температура, кашель	Трахеїт #ГрупаРизику1	Сироп Лазолван по 5 мл 3 р. в
16-03-2022	Прискорене серцебиття	Гіпертонія	Приймати Лозартан 50 мг по 2 табл. в день #ГрупаРизику1

Доступні теги: #ГрупаРизику1, #ГрупаРизику2, #ГрупаРизику3, #ГрупаРизику4

Додати тег Повернутися Закінчити прийом

Рисунок 4.8 – Сторінка прийому пацієнта

На цій сторінці лікар може вибрати пацієнта для прийому, подивитися основну інформацію про пацієнта, чи укладена декларація, подивитися інформацію про виписані рецепти, лікарняні та результати аналізів у відповідному вікні (рисунок 4.9) та, звичайно, провести прийом пацієнта, вписавши скарги, діагноз та лікування (дату можна не вписувати, так як вона проставляється автоматично). По закінченню прийому потрібно натиснути на кнопку «Закінчити прийом», про що буде проінформовано повідомленням, що показане на рисунку 4.10.

ProHealth

Архипов Максим Борисович, лікар: Васильчук Юлія Миколаївна

Декларація

Медична карта Рецепти Аналізи крові Аналізи сечі Лікарняні

Лікар	Препарат	Дозування	Сігнатура	З	По
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	Симвастатин	40	По 1 табл. 1 р. в день	01-03-2022	01-05-2022
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	Гліклязід	60	По 1 табл. 2 р. в день	05-03-2022	01-05-2022
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	Беродуал	20	По 1 т. на день	07-03-2022	01-05-2022
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	Гліклязід	60	По 1 табл. 2 р. в день	08-03-2022	01-05-2022

Рисунок 4.9 – Вікно інформації пацієнта

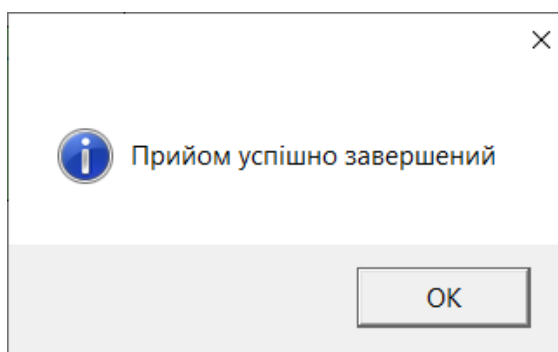


Рисунок 4.10 – Вікно повідомлення про успішне закінчення прийому

Також у вікні наявний список доступних тегів, які будуть використовуватися для аналітики. За потреби, лікар має можливість додати нові теги (вікно додавання зображено на рисунку 4.11). Для їхнього використання потрібно скопіювати тег зі списку чи ввести вручну до таких полів запису електронної книги (на вибір): «Скарги», «Діагноз» або «Лікування».

Рисунок 4.11 – Вікно додавання тегу

Далі розглянемо вікно декларацій (рисунок 4.12). Тут лікар має можливість переглядати свої декларації, укладені з пацієнтами, шукати пацієнтів серед своїх або серед всіх. При виборі свого пацієнта, можливо переглянути декларацію у документі Word (рисунок 4.13).

Прізвище	Ім'я	По батькові	Дата народження	Номер телефону	Дата укладення	Лікар
Архипов	Максим	Борисович	30-09-1979	+380503626500	01-05-2019	Васильчук Юлія Миколаївна
Бурова	Зоя	Григоріївна	07-09-1968	+380973626500	26-01-2020	Васильчук Юлія Миколаївна
Волкова	Тамара	Борисовна	18-09-1973	+380984624464	01-05-2019	Васильчук Юлія Миколаївна
Горбунов	Григорій	Федорович	27-05-1992	+380984600041	01-05-2019	Васильчук Юлія Миколаївна
Городецка	Олена	Даниловна	23-06-1978	+380984615724	01-05-2019	Васильчук Юлія Миколаївна
Ершова	Олена	Львовна	24-05-1975	+380978148467	26-01-2020	Васильчук Юлія Миколаївна
Євсейва	Інна	Ярославовна	10-09-1980	+380978141478	26-01-2020	Васильчук Юлія Миколаївна
Жданова	Марія	Борисовна	11-04-1996	+380984611478	01-05-2019	Васильчук Юлія Миколаївна
Забужко	Петр	Олександрович	05-12-1985	+380503624464	01-05-2019	Васильчук Юлія Миколаївна
Іванов	Іван	Іванович	01-01-2000	+380982311234	15-05-2020	Васильчук Юлія Миколаївна
Ковальова	Лідія	Віталіївна	15-01-1979	+380984618467	01-05-2019	Васильчук Юлія Миколаївна
Кошелев	Ян	Ярославович	16-06-1973	+380984626500	26-01-2020	Васильчук Юлія Миколаївна
Лактіонов	Валерій	В	01-01-2001	79676	07-05-2020	Васильчук Юлія Миколаївна

Рисунок 4.12 – Сторінка декларацій

ДЕКЛАРАЦІЯ № 28

Пацієнт	
1. Прізвище	Архипов
2. Ім'я	Максим
3. По батькові	Борисович
4. Дата народження	30-09-1979
5. Місто, країна народження	м. Запоріжжя, Запорізький район, Запорізька обл., в. Нова, буд. 75, кв. 45
6. Стать	чоловіча
Контактні дані	
7. Номер телефону	+380503626500
8. Адреса електронної пошти (за наявності)	
Документ, що посвідчує особу	
9. Серія (за наявності), номер, орган видачі та дата	862771678, 7187, від 25-06-2017
10. РНОКПП	9449894899
Адреса місця фактичного проживання	
11. Адреса проживання	м. Запоріжжя, Запорізький район, Запорізька обл., в. Нова, буд. 75, кв. 45
Лікар	
1. Прізвище	Васильчук
2. Ім'я	Юлія
3. По батькові	Миколаївна
4. Номер телефону	+380979216334
5. Адреса електронної пошти (за наявності)	
6. Посада	Лікар сімейної медицини
7. Амбулаторія	№ 1
Дата укладення декларації	01-05-2019

Рисунок 4.13 – Декларація у документі Word

При виборі пацієнта, з яким не укладена інформація – можливо перезаключити її на себе, при цьому буде виведено вікно підтвердження (рисунок 4.14) та сповіщення про успішне переукладення декларації (рисунок 4.15).

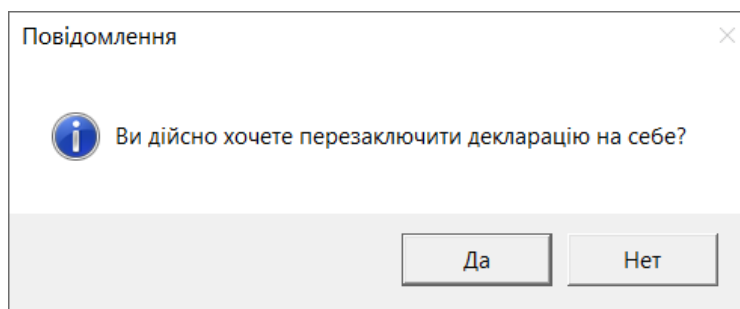


Рисунок 4.14 – Вікно підтвердження переукладення декларації

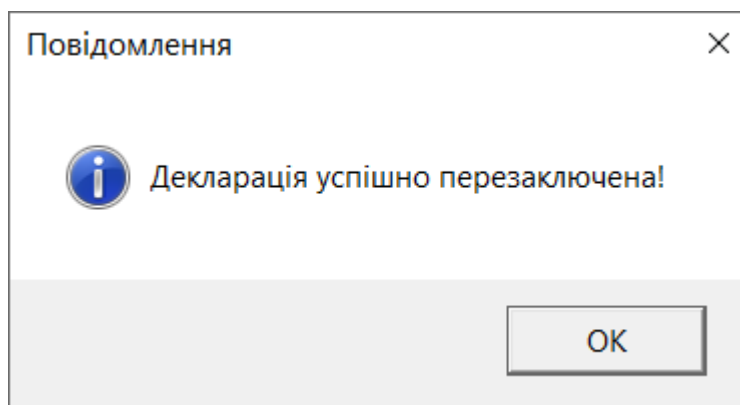


Рисунок 4.15 – Вікно інформування про успішне переукладення декларації

При пошуку серед свої пацієнтів може бути виведено повідомлення, що з цим пацієнтом декларація не укладена (рисунок 4.16), а при глобальному пошуку – що пацієнт не зареєстрований в системі (рисунок 4.17).

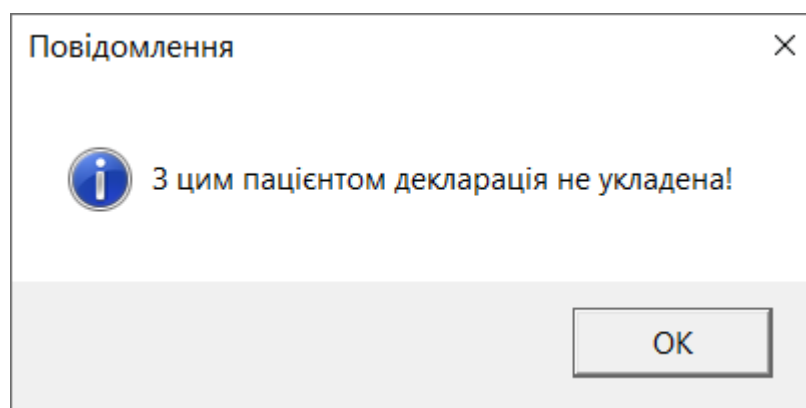


Рисунок 4.16 – Вікно інформування

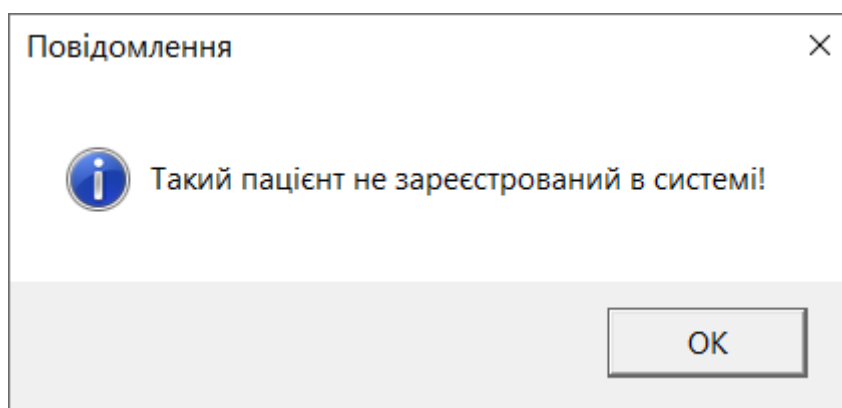


Рисунок 4.17 – Вікно інформування

У випадку, коли шуканий пацієнт не зареєстрований в системі, можна натиснути кнопку «Нова декларація» та заповнити усі потрібні поля на сторінці укладення декларації (рисунок 4.18).

Дані пацієнта		Документ, що посвідчує особу	
Прізвище*	Ім'я*	Серія (за наявності), номер*	
Іванов	Іван	767678678	
По батькові*	Дата народження*	Ким видано*	Коли видано*
Іванович	01.01.2000	1452	01.01.2015
РНОКПП (ІПН)*	Стать*	Адреса	
4125789537	Чоловіча		
Країна народження*	Місце народження*	Область*	Район
Україна	м. Запоріжжя	Запорізька	Василівський
Номер телефону*	Електронної пошти	Населений пункт*	Вулиця*
+380982311234		м. Василівка	Музична
		Будинок*	Квартира
		37	5
		Індекс	
☒ Пацієнт дав згоду на обробку персональних даних*			
* - поля, обов'язкові до заповнення			
		Повернутися	Створити

Рисунок 4.18 – Сторінка створення декларації

Після коректного заповнення всіх обов'язкових полів, повідомляється про успішне створення декларації (рисунки 4.19), або, у випадку, якщо не заповнені обов'язкові поля, – вікно повідомлення про це (рисунки 4.20).

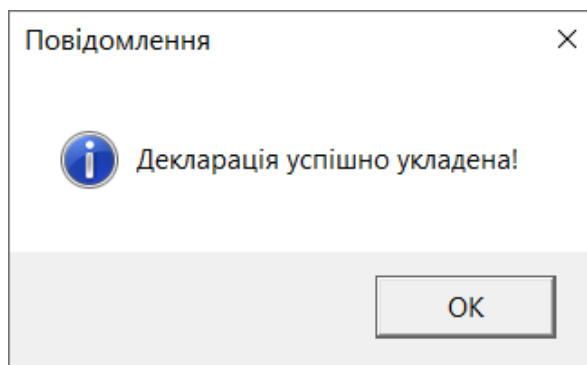


Рисунок 4.19 – Вікно інформування

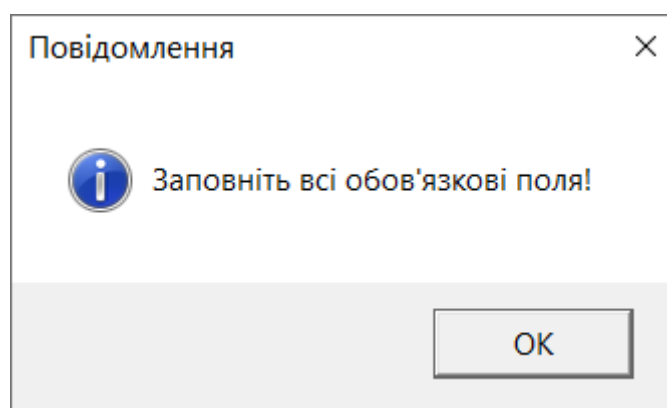


Рисунок 4.20 – Вікно інформування

Після успішного укладення декларації з'являється сторінка декларацій, з якої можна вийти на головну сторінку, натиснувши кнопку «Повернутися на головну».

Далі розглянемо сторінку створення рецептів (рисунки 4.21). Тут можна вибрати пацієнта, для якого буде створений рецепт, та заповнити поля для самого рецепту. У цьому вікні також наявний список доступних тегів, які будуть використовуватися для аналітики. Їх можна додати до відповідного поля за потреби.

ProHealth

Пацієнт
Архипов Максим Борисович

Поточна дата: 03.04.2022

Лікарський засіб
Гліклазід

Кількість дозування
60

Кількість за день
2 р/д

Кількість за прийом
1

Тривалість лікування
30 днів

Сигнатура
По 1 т. 2 р. на день

Дата початку лікування за рецептом
03.04.2022

Дата закінчення лікування за рецептом
03.05.2022

Теги (за потреби)
#ГрупаРизику1

Доступні теги: #ГрупаРизику1, #ГрупаРизику2, #ГрупаРизику3, #ГрупаРизику4

Додати тег

Повернутися

Створити

Рисунок 4.21 – Сторінка створення рецептів

По закінченню створення рецепту, треба натиснути кнопку «Створити», або можна повернутися за допомогою кнопки «Повернутися», про що будуть інформувати повідомлення на рисунках 4.22 та 4.23.

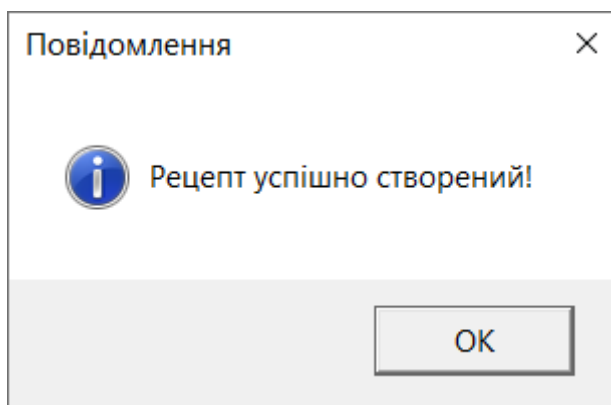


Рисунок 4.22 – Вікно інформування

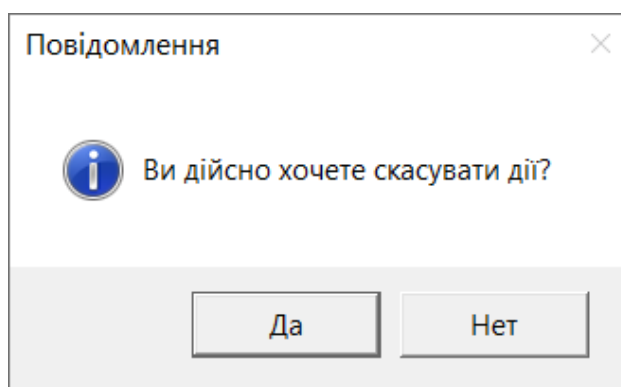


Рисунок 4.23 – Вікно інформування

Потім з'являється головна сторінка, на якій ще можна виписати лікарняний, натиснувши кнопку «Виписати лікарняний». На сторінці виписування лікар обирає пацієнта та може подивитися стислу інформацію про нього, історію лікарняних, продовжити лікарняний, чи виписати новий (рисунок 4.24). Після дій потрібно натиснути кнопку «Зберегти», або можна повернутися за допомогою кнопки «Повернутися».

ProHealth

Пацієнт: Архипов Максим Борисович Декларація укладена Поточна дата: 09.04.2022

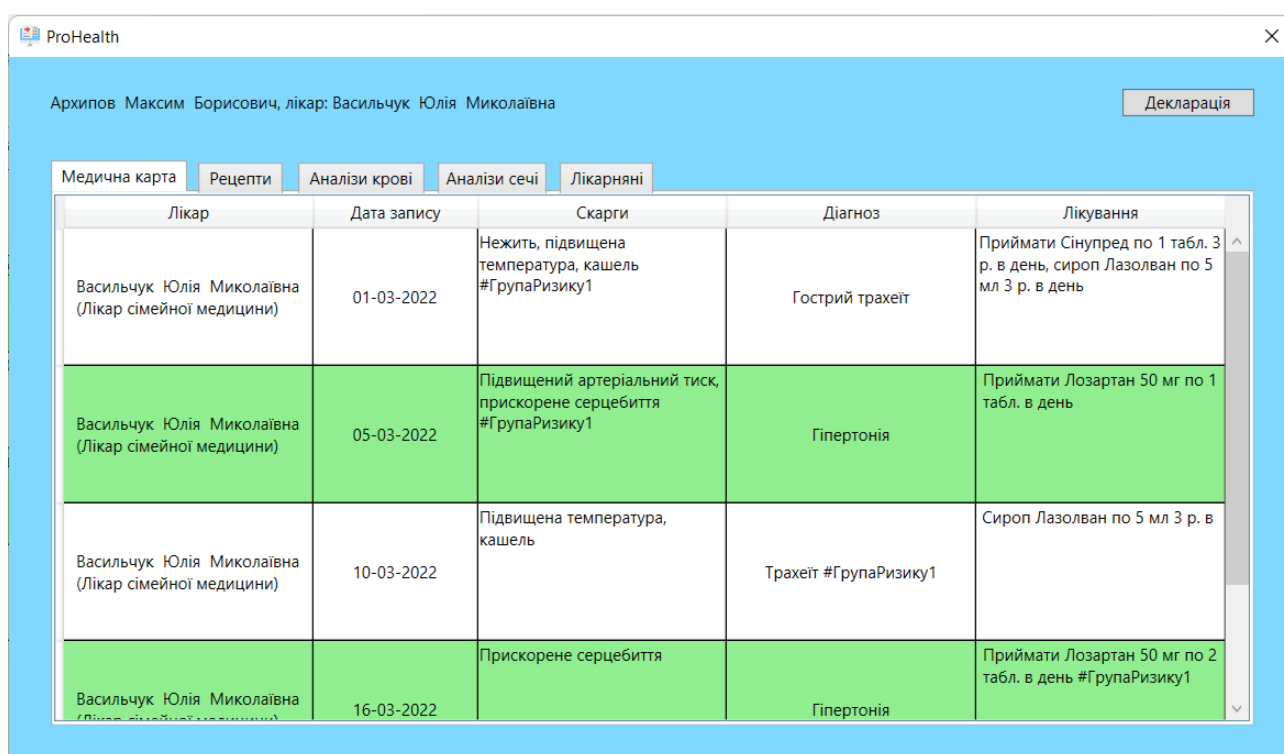
Дата народження: 30-09-1979 Номер телефону: +380503626500 м. Запоріжжя, Запорізький район, Запорізька обл., в. Нова, буд. 75, кв. 45

Дата випису	Режим	Діагноз	З	По	Стати до роботи
01-01-2022	Амбулаторний	ГРВІ	01-01-2022	05-01-2022	06-01-2022
19-03-2022	Амбулаторний	ГРВІ	19-03-2022	20-03-2022	21-03-2022

Повернутися Зберегти

Рисунок 4.24 – Сторінка лікарняних

Тепер розглянемо вікно для пацієнта. Після вводу коректних авторизаційних даних пацієнта він попадає на сторінку для пацієнта (рисунок 4.25). Тут він може подивитися свою декларацію у документі Word, свою медичну картку, рецепти, результати аналізів крові на глюкозу та сечі, а також лікарняні. Зрозуміло, що будь-яку інформацію пацієнт не може змінювати.



ProHealth

Архипов Максим Борисович, лікар: Васильчук Юлія Миколаївна

Декларація

Медична карта | Рецепти | Аналізи крові | Аналізи сечі | Лікарняні

Лікар	Дата запису	Скарги	Діагноз	Лікування
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	01-03-2022	Нежить, підвищена температура, кашель #ГрупаРизику1	Гострий трахеїт	Приймати Сінупред по 1 табл. 3 р. в день, сироп Лазолван по 5 мл 3 р. в день
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	05-03-2022	Підвищений артеріальний тиск, прискорене серцебиття #ГрупаРизику1	Гіпертонія	Приймати Лозартан 50 мг по 1 табл. в день
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	10-03-2022	Підвищена температура, кашель	Трахеїт #ГрупаРизику1	Сироп Лазолван по 5 мл 3 р. в
Васильчук Юлія Миколаївна (Лікар сімейної медицини)	16-03-2022	Прискорене серцебиття	Гіпертонія	Приймати Лозартан 50 мг по 2 табл. в день #ГрупаРизику1

Рисунок 4.25 – Вікно пацієнта

Наступне вікно – це вікно лабораторії. Після вводу коректних авторизаційних даних лабораторії, користувач попадає на сторінку для лабораторії (рисунок 4.26).

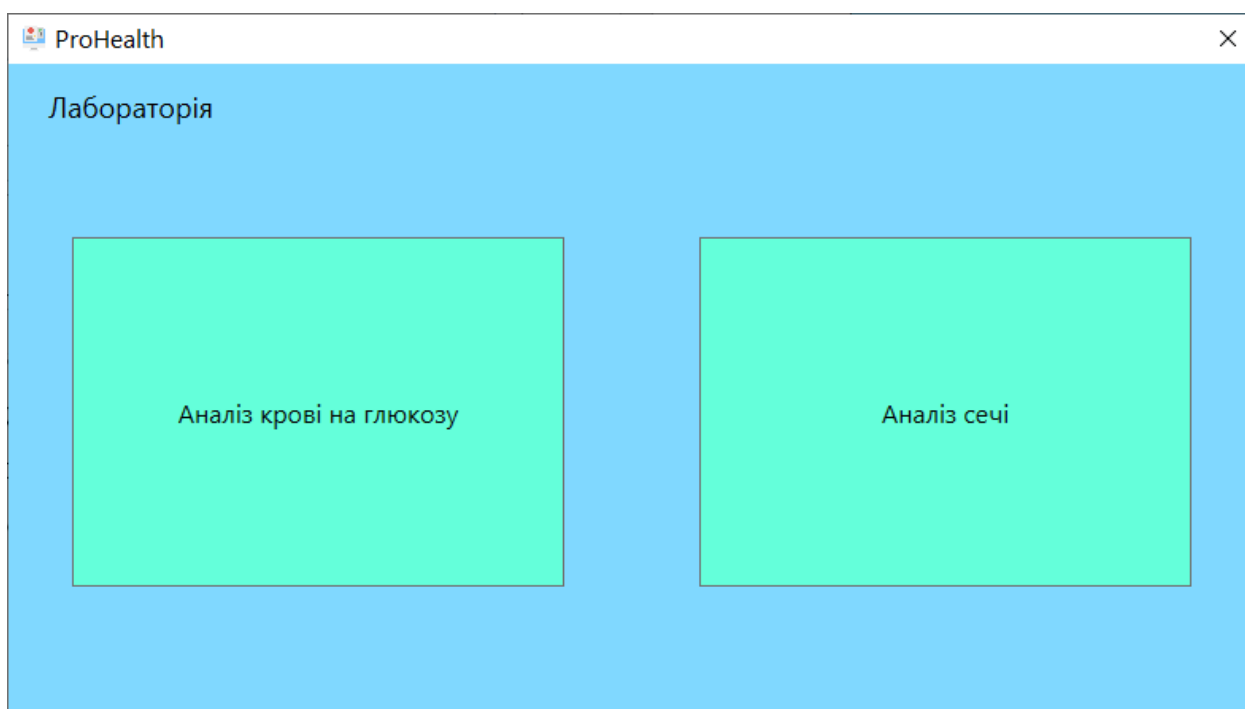


Рисунок 4.26 – Вікно лабораторії

Працівники лабораторії можуть внести дані аналізу крові на глюкозу (рисунок 4.27) та сечі (рисунок 4.28) відповідного пацієнта та зберегти зміни чи повернутися на головну сторінку.

The image shows a software window titled "ProHealth" with a close button (X) in the top right corner. The main area has a light blue background. At the top left, it says "Пацієнт" (Patient) above a dropdown menu showing "Архипов Максим Борисович". To the right, it says "Поточна дата: 03.04.2022". In the center, there is a label "Глюкоза:" followed by an empty text input field and the unit "мл/моль". At the bottom right, there are two buttons: "Повернутися" (Return) and "Створити" (Create).

Рисунок 4.27 – Сторінка аналізу крові

The screenshot shows a web application window titled "ProHealth". At the top left is a small icon and the title "ProHealth". At the top right is a close button "X". Below the title bar, on the left, is a label "Пацієнт" above a dropdown menu containing the text "Архипов Максим Борисович". To the right of this is the text "Поточна дата: 03.04.2022". Below these are nine input fields arranged in a 3x3 grid. The labels for the fields are: "Колір" (Color), "Реакція" (Reaction), "Питома вага" (Specific gravity) in the first row; "Білки" (Proteins), "Цукор" (Sugar), "Еритроцити" (Erythrocytes) in the second row; and "Лейкоцити" (Leukocytes), "Бактерії" (Bacteria), "Плоский епітелій" (Flat epithelium) in the third row. Each label is above its corresponding empty input field. At the bottom right of the form are two buttons: "Повернутися" (Return) and "Створити" (Create).

Рисунок 4.28 – Сторінка аналізу сечі

Останнє вікно, яке не було розглянуто, – це вікно аналітики (рисунок 4.29). Тут можливо подивитися зведену статистику по прийомам та рецептам за обраними тегамі (яку було попередньо вписано до записів прийомів або рецептів) для пацієнта або групи пацієнтів. Лікар може додавати нові теги також і в цьому вікні, натиснувши кнопку «Додати тег».

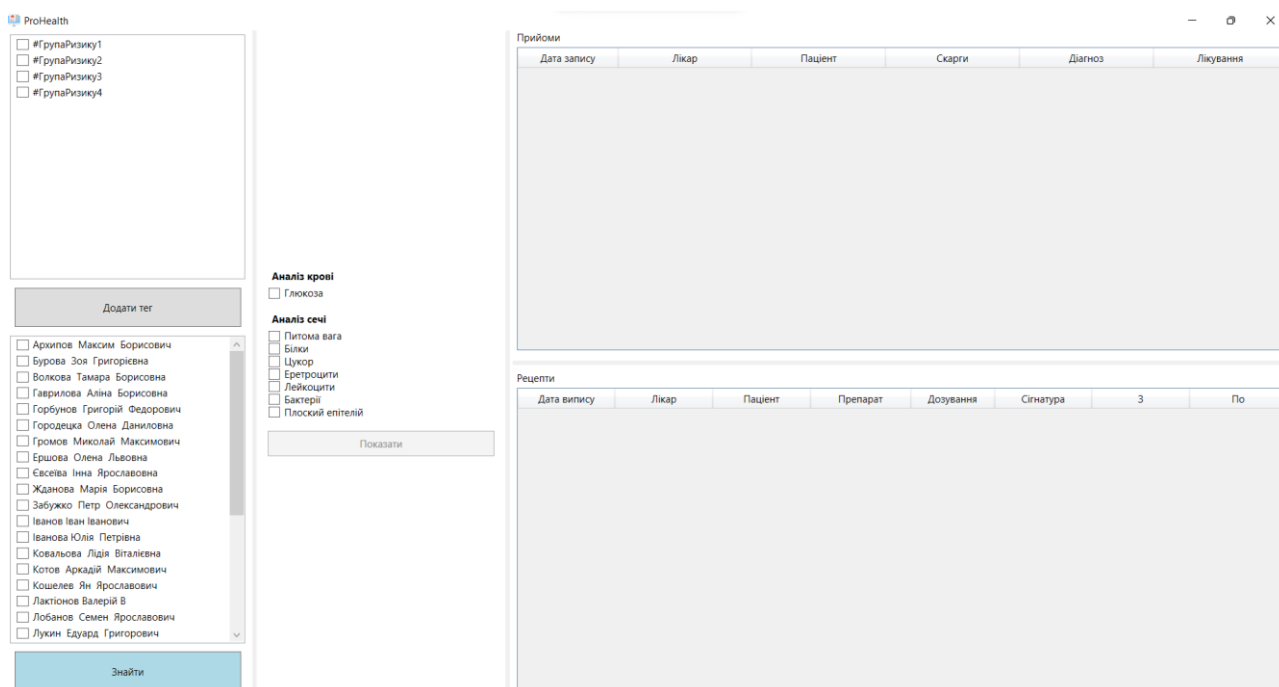


Рисунок 4.29 – Сторінка аналітики

Для пошуку даних лікар повинен обрати цікаві йому теги та натиснути кнопку «Знайти». Відбудеться пошук даних серед всіх пацієнтів у системі за обраними тегамі. Якщо жодного тегу не обрано, то користувачу надійде повідомлення, зображене на рисунку 4.30. Якщо лікарю потрібно отримати дані тільки для певного пацієнта чи пацієнтів, він має можливість обрати їх у відповідному списку пацієнтів. Результат пошуку зведених даних за тегом «#ГрупаРизику1» наведений на рисунку 4.31. Лікар бачить зведені дані по прийомам та рецептам, які знайшлися серед всіх пацієнтів з обраним тегом.

Для зручності роботи з вікном аналітики, розроблено гнучкий інтерфейс, у якому можливо зменшувати або збільшувати фільтри для пошуку та таблиці з даними.

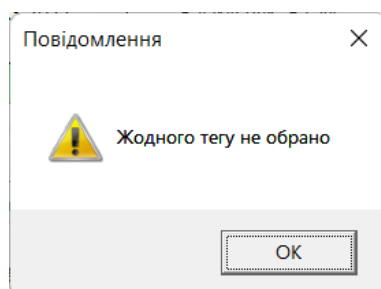


Рисунок 4.30 – Вікно повідомлення

ProHealth

☒ #ГрупаРизику1

☐ #ГрупаРизику2

☐ #ГрупаРизику3

☐ #ГрупаРизику4

Додати тег

☐ Архипов Максим Борисович

☐ Бурова Зоя Григорівна

☐ Волкова Тамара Борисовна

☐ Гаврилова Аліна Борисовна

☐ Горбунов Григорій Федорович

☐ Городецька Олена Данилівна

☐ Громов Миколай Максимович

☐ Ершова Олена Львівна

☐ Євсєєва Інна Ярославівна

☐ Жданова Марія Борисівна

☐ Забужко Петро Олександрович

☐ Іванов Іван Іванович

☐ Іванова Юлія Петрівна

☐ Ковальова Лідія Віталіївна

☐ Котов Аркадій Максимович

☐ Кошелев Ян Ярославович

☐ Лактіонов Валерій В

☐ Лобанов Семен Ярославович

☐ Лукин Едуард Григорович

Знайти

Аналіз крові

☐ Глюкоза

Аналіз сечі

☐ Питомі вага

☐ Біліки

☐ Цукор

☐ Еритроцити

☐ Лейкоцити

☐ Бактерії

☐ Плоский епітелій

Показати

Прийоми

Дата запису	Лікар	Пацієнт	Скарги	Діагноз	Лікування
1.3.2022	Васильчук Ю. М.	Архипов М. Б.	Нежить, підвищена температура, кашель #ГрупаРизику1	Гострий трахеїт	Приймати Снупред по 1 табл. 3 р. в день, сироп Лазолван по 5 мл 3 р. в день
2.3.2022	Васильчук Ю. М.	Бурова З. Г.	Нежить, підвищена температура, кашель #ГрупаРизику1	Гострий трахеїт	Приймати Снупред по 1 табл. 3 р. в день, сироп Лазолван по 5 мл 3 р. в день
5.3.2022	Васильчук Ю. М.	Бурова З. Г.	Підвищений артеріальний тиск, прискорене серцебиття	Гіпертонія	Приймати Лозартан 50 мг по 1 табл. в день #ГрупаРизику1
5.3.2022	Васильчук Ю. М.	Архипов М. Б.	Підвищений артеріальний тиск, прискорене серцебиття #ГрупаРизику1	Гіпертонія	Приймати Лозартан 50 мг по 1 табл. в день
10.3.2022	Васильчук Ю. М.	Архипов М. Б.	Підвищена температура, кашель	Трахеїт #ГрупаРизику1	Сироп Лазолван по 5 мл 3 р. в
			Кашель, підвищена		Сироп Лазолван по 5 мл 3 р. в

Рецепти

Дата випису	Лікар	Пацієнт	Препарат	Дозування	Сігнатура	З	По
1.3.2022	Миколаєва Ю. М.	Архипов М. Б.	Симвастатин	40	По 1 табл. 1 р. в день	1.3.2022	1.5.2022
1.3.2022	Миколаєва Ю. М.	Бурова З. Г.	Гліклязид	40	По 1 табл. 1 р. в день	1.3.2022	1.5.2022
4.3.2022	Миколаєва Ю. М.	Бурова З. Г.	Кетанов	40	По 2 табл. 3 р. в день	4.3.2022	1.5.2022
5.3.2022	Миколаєва Ю. М.	Архипов М. Б.	Гліклязид	60	По 1 табл. 2 р. в день	5.3.2022	1.5.2022
5.3.2022	Миколаєва Ю. М.	Бурова З. Г.	Бісопролол	60	По 2 табл. 4 р. в день	5.3.2022	1.5.2022
7.3.2022	Миколаєва Ю. М.	Архипов М. Б.	Беродуал	20	По 1 т. на день	7.3.2022	1.5.2022

Рисунок 4.31 – Сторінка аналітики зі знайденими даними

Після пошуку даних лікар має можливість використовувати другу колонку фільтрів. За допомогою цих фільтрів лікар обирає потрібні значення з аналізів для формування графіку. Якщо користувач не вибере жодного значення, буде виведено повідомлення, яке зображене на рисунку 4.32. Сформований графік за значенням глюкози наведений на рисунку 4.33.

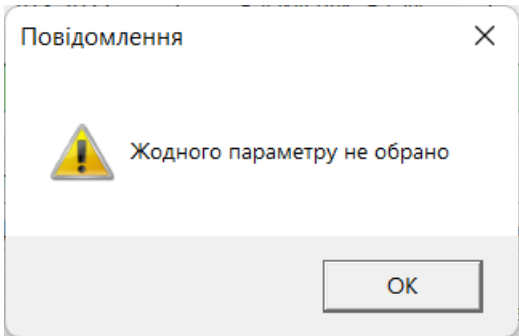


Рисунок 4.32 – Вікно повідомлення

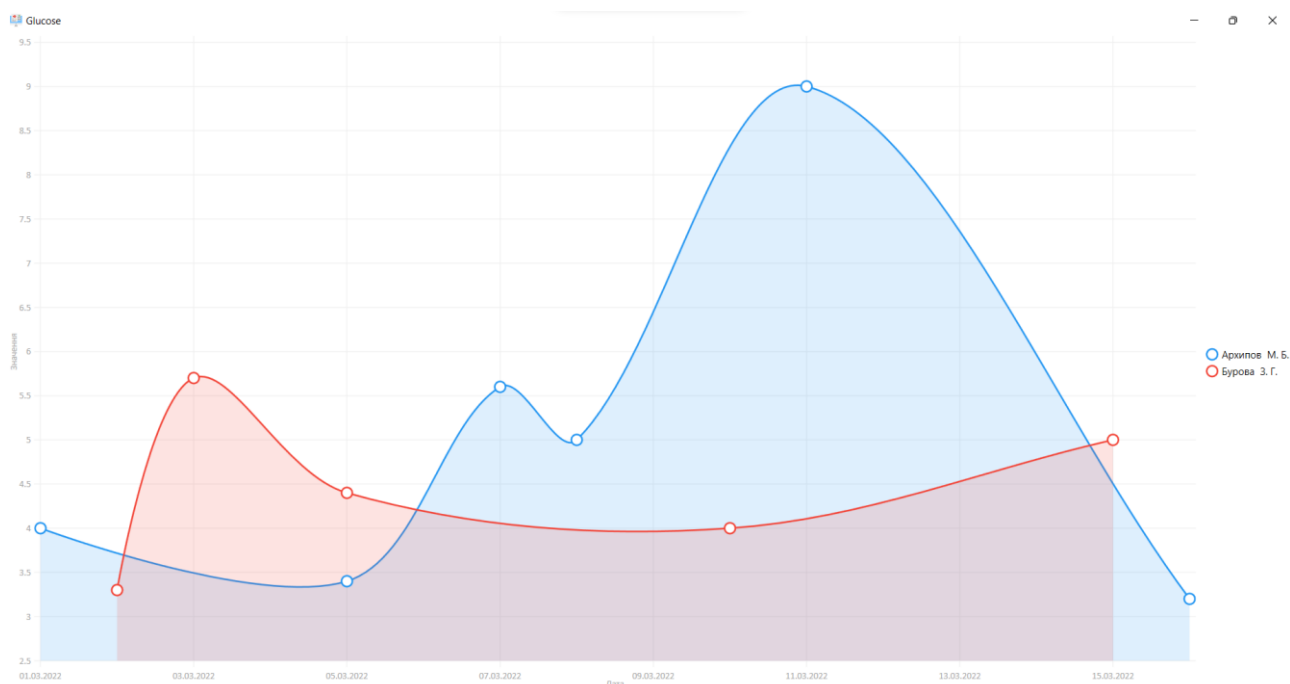


Рисунок 4.33 – Вікно графіку

На графіку видно дві криві пацієнтів за значеннями глюкози. На вісі абсцис відтворюється дата аналізу, а на вісі ординат – значення параметру аналізу. Для зручності користувач може навести курсор на точку даних та отримати інформацію за цією точкою (рисунок 4.34).

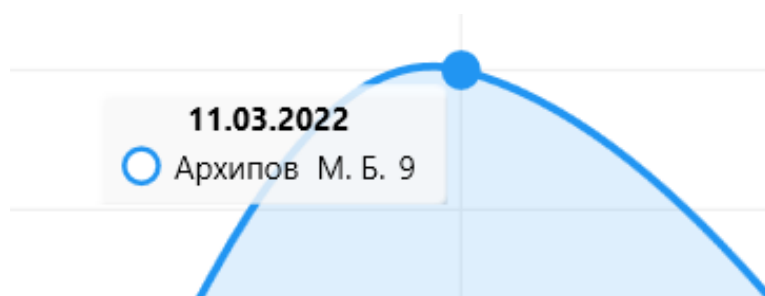


Рисунок 4.34 – Інформація за точкою

4.4 Повідомлення користувачу

В процесі роботи програми можуть виникнути помилки. В такому разі користувач отримає інформаційні повідомлення щодо деталей помилки.

У програмі реалізовано обробку виключних ситуацій за допомогою блоків try та catch, та екземпляру об'єкта Exception. Тому усі непередбачені ситуації оброблюються та програма адекватно реагує на них. Одне з можливих повідомлень, що можуть бути отримані при роботі, – помилка зв'язку з сервером (рисунок 4.35). Така помилка може виникнути у випадку, якщо клієнт не під'єднаний до мережі Інтернет або сервер вимкнений.

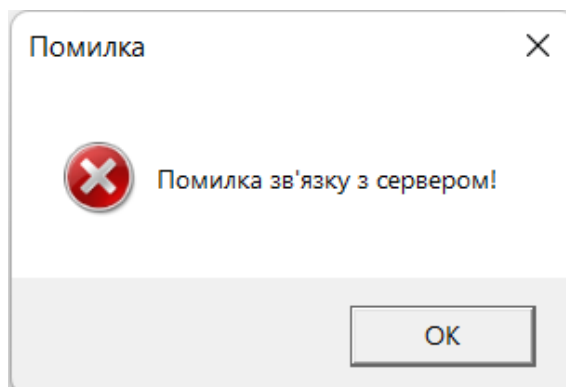


Рисунок 4.35 – Помилка зв'язку з сервером

4.5 Тестування програми

Під час розробки програмної системи велося тестування кожної доданої функції, шляхом її випробовування у різних сценаріях. У випадку виявлення помилок, коригувався та писався код для усунення помилок.

Для тестування був використаний метод тестування UX. Багато разів, за різними сценаріями використовувалися всі функції. Немало уваги було приділено валідації введених користувачем даних та збереженню їх до бази даних. Після імітації сценаріїв було виявлено незначні помилки та виправлено.

Завдяки тестуванню розробленої програмної системи, вдалося досягти стабільності роботи відповідних модулів та з'явилась впевненість в коректній роботі цих модулів програмного забезпечення.

У таблиці 4.1 наведено чек-лист тестування серверу, а у таблиці 4.2 чек-лист тестування клієнтської частини системи.

Таблиця 4.1 – Чек-лист тестування серверу

Перевірка	Результат
Робота HTTP серверу	Пройдено
Зв'язок з базою даних	Пройдено
Отримання запитів	Пройдено
Парсинг даних	Пройдено
Обробка даних	Пройдено
Збереження даних	Пройдено
Зворотній парсинг даних	Пройдено
Відповідь на запити	Пройдено
Логування	Пройдено
Безперервна робота	Пройдено

Таблиця 4.2 – Чек-лист тестування клієнту

Перевірка	Результат
Авторизація	Пройдено
Пошук пацієнтів	Пройдено
Внесення даних щодо прийому	Пройдено
Створення та перегляд декларацій	Пройдено
Виписування рецептів	Пройдено
Виписування лікарняних	Пройдено
Генерація документу «Декларація пацієнта»	Пройдено
Перегляд інформації щодо пацієнта	Пройдено
Перегляд аналітики	Пройдено
Внесення результатів аналізів	Пройдено
Перегляд медичної картки, рецептів, аналізів, лікарняних та декларацій	Пройдено

Продовження таблиці 4.2

Перевірка	Результат
Логування	Пройдено
Безперервна робота	Пройдено

Як видно з таблиць, усі тестування в результаті пройдено успішно. Усі виявлені помилки усунені та система працює стабільно.

Тестування продемонструвало, що розроблене програмне забезпечення для ведення та аналізу медичних карток пацієнтів відмінно пройшло тестування та відповідає вимогам до програмного забезпечення.

ВИСНОВКИ

Під час виконання дипломної роботи було вирішено актуальне практичне завдання створення програмного забезпечення для ведення і аналізу медичних карток пацієнтів у медичних закладах.

Для продуктивної роботи лікарень різних спеціалізацій та спрямувань доцільною є цифровізація та автоматизація роботи зі всією інформацією, що використовується в роботі. Найкращим варіантом для цього є створення електронної медичної системи.

Було розроблено програмне забезпечення, яке має функціональність для перегляду та ведення медичних карток пацієнтів, перегляду та створення рецептів лікарських засобів, перегляду та створення лікарняних для пацієнтів, перегляду та створення аналізів пацієнтів, а також широку функціональність для забезпечення інформаційної підтримки прийняття рішень лікарем.

Програмна система складається з двох застосунків – серверного та клієнтського. Обидва застосунки написані на мові програмування C# у IDE Microsoft Visual Studio 2022. Сервер являє собою консольний застосунок, який оброблює, зберігає та надає дані клієнту у відповідь на його запити. У ролі клієнта виступає графічний застосунок, створений за допомогою технології WPF. Дана програма має зручний та інтуїтивно зрозумілий інтерфейс та надає можливості користування лікарям первинної та вторинної ланки, лабораторіям та пацієнтам. Клієнт та сервер комунікують за допомогою технології REST API.

Для зв'язку сервера з базою даних було використано сучасний фреймворк для роботи з базами даних – Entity Framework, який дозволив легко встановлювати підключення та виконувати різноманітні запити до бази даних.

Головною особливістю розробленого програмного забезпечення стала функціональність аналітики, яка дозволяє лікарям переглядати зведені дані щодо пацієнта чи групи пацієнтів, а також продивлятися графіки за обраними параметрами зроблених аналізів.

Програмна система запрограмована з акцентом на легкість додавання нової функціональності, тому розроблене програмне забезпечення може в подальшому бути використане для розробки аналогічних систем чи для доопрацювання до більш масштабної системи.

Проведене тестування підтвердило працездатність розробленої програмної системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sinha, P. K. Electronic Health Record [Text] : Standards, Coding Systems, Frameworks, and Infrastructures / P. K. Sinha. – Hoboken, N.J. : John Wiley & Sons, 2013. – 376 p.
2. Thompson, D. Developing a Unified Patient-Record [Text] / D. Thompson, K. Wright. – Boca Raton : Chapman and Hall/CRC, 2018. – 144 p.
3. Електронна система Helsi.me [Електрон. ресурс]. – Режим доступу : <https://reform.helsi.me/>.
4. Електронна система Helsi.Pro [Електрон. ресурс]. – Режим доступу : <https://apt.helsi.pro/>.
5. C++ overview [Electronic resource]. – Режим доступу : https://www.tutorialspoint.com/cplusplus/cpp_overview.htm.
6. C# overview [Electronic resource]. – Режим доступу : https://www.tutorialspoint.com/csharp/csharp_overview.htm.
7. Документація по мові C# [Electronic resource]. – Режим доступу : <https://docs.microsoft.com/en-us/dotnet/csharp/>.
8. Client/Server Architecture [Electronic resource]. – Режим доступу : <https://www.techopedia.com/definition/438/clientserver-architecture>.
9. Client/Server Architecture [Electronic resource]. – Режим доступу : <https://www.britannica.com/technology/client-server-architecture>.
10. REST API overview [Electronic resource]. – Режим доступу : <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
11. Design pattern Model-View-ViewModel [Electronic resource]. – Режим доступу : <https://whatis.techtarget.com/definition/Model-View-ViewModel>.
12. Entity Framework in C# [Electronic resource]. – Режим доступу : <https://www.c-sharpcorner.com/article/entity-framework-introduction-using-c-sharp-part-one/>.
13. Завадський, І.О. Основи баз даних [Текст] : навч. посіб. / І.О. Завадський. – К. : Видавець І.О. Завадський, 2011. – 192 с.

ДОДАТОК А
Технічне завдання

Вступ

У роботі розглядається галузь розробки програмного забезпечення, пов'язана з електронними системами ведення та аналізу карток пацієнтів у медичних закладах.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломний проєкт на тему «Програмне забезпечення ведення і аналізу медичних карток пацієнтів», затверджене наказом по національному університету «Запорізька політехніка», протокол № 102 від 27 квітня 2022 р.

A.2 Призначення розробки

Програмне забезпечення ProHealth призначене для користування сімейними лікарями, лікарями вузького профілю, лабораторіями, а також пацієнтами. Лікарі отримують функціональність для ведення прийомів пацієнтів, виписування рецептів лікарських засобів, виписування лікарняних, обліку декларацій, а також аналізу даних. Лабораторії мають можливість заповнювати результати аналізів, а пацієнти – переглядати свою медичну картку, результати аналізів, рецепти, лікарняні, а також декларацію.

A.3 Основні вимоги до програмного забезпечення, що розробляється

A.3.1 Вимоги до функціональних характеристик

Функції, які повинна виконувати програма:

а) для лікаря:

1) внесення даних щодо прийому: робота з електронною карткою;

- 2) перегляд інформації щодо пацієнта: виписані рецепти, лікарняні, результати аналізів;
- 3) створення та перегляд декларацій;
- 4) пошук пацієнтів серед своїх та чужих (декларація з іншим лікарем або без декларації);
- 5) виписування рецептів та лікарняних;
- 6) перегляд аналітики;
- 7) генерація документу «Декларація пацієнта»;
- б) для лаборанта: внесення результатів аналізів;
- в) для пацієнта: перегляд медичної картки, рецептів, аналізів, лікарняних та декларацій.

А.3.2 Вимоги до надійності

Програма повинна розпізнавати ситуації нерозуміння програмою користувача (користувачем програми), а також виконувати обробку збійних ситуацій.

А.3.3 Умови експлуатації

Експлуатація програмного продукту здійснюється відповідно до експлуатаційної документації на розроблену програму, що відповідає стандартам і містить інформацію, необхідну для освоєння та експлуатації програмного продукту.

А.3.4 Вимоги до складу і параметрів технічних засобів

Для нормальної роботи розробленої програми потрібен комп'ютер під управлінням операційної системи Microsoft Windows 10 та вище з такими характеристиками: процесор з тактовою частотою не нижче 1 ГГц; оперативний

запам'ятовуючий пристрій – 4 ГБ; 10000 МБ вільного місця на жорсткому диску. Для серверного комп'ютеру обов'язково є встановлена система управління базами даних MS SQL Server Management Studio та 30000 МБ вільного місця на жорсткому диску (не враховуючи розмір СКБД MS SQL).

A.3.5 Вимоги до програмної документації

Система повинна поставлятися з технічним завданням та інструкцією з експлуатації, представленою відповідним розділом пояснювальної записки.

A.4 Порядок контролю та приймання

Дипломна робота бакалавра допускається до захисту керівником, консультантом та нормоконтролером, затверджується завідувачем кафедри та захищається у присутності членів екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Текст файла AnalyticsViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class AnalyticsViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public AnalyticsViewModel()
        {
            _ = LoadTagsListAsync();
            _ = LoadPatientListAsync();
        }

        private List<CheckBoxTagListHelper> tagList = new List<CheckBoxTagListHelper>();
        public List<CheckBoxTagListHelper> TagList
        {
            get { return tagList; }
            set
            {
                tagList = value;
                OnPropertyChanged("TagList");
            }
        }

        private List<CheckBoxPatientListHelper> patientList = new
List<CheckBoxPatientListHelper>();
        public List<CheckBoxPatientListHelper> PatientList
        {
            get { return patientList; }
            set
            {
                patientList = value;
                OnPropertyChanged("PatientList");
            }
        }

        private ObservableCollection<AnalyticsRecordsResponseModel> records = new
ObservableCollection<AnalyticsRecordsResponseModel>();
        public ObservableCollection<AnalyticsRecordsResponseModel> Records
        {
            get { return records; }
            set
            {
                records = value;
                OnPropertyChanged("Records");
            }
        }

        public ICollectionView RecordsCollection
        {
            get { return CollectionViewSource.GetDefaultView(Records); }
            set
            {
                RecordsCollection = value;
                OnPropertyChanged("RecordsCollection");
            }
        }

        private ObservableCollection<AnalyticsRecipesResponseModel> recipes = new
ObservableCollection<AnalyticsRecipesResponseModel>();
        public ObservableCollection<AnalyticsRecipesResponseModel> Recipes
        {
            get { return recipes; }
            set

```



```

        _ = LoadTagsListAsync();
    }
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
}
}
});
}

private RelayCommand findDataClick;
public RelayCommand FindDataClick
{
    get
    {
        return findDataClick ??
            (findDataClick = new RelayCommand(async obj =>
            {
                try
                {
                    // For records
                    var analyticsSearchModel = new AnalyticsSearchModel();
                    analyticsSearchModel.TagList = TagList.Where(t => t.IsChecked ==
true).Select(t => t.TagName).ToList();
                    if (analyticsSearchModel.TagList.Count < 1)
                    {
                        MessageBox.Show("Жодного тегу не обрано", "Повідомлення",
MessageBoxButton.OK, MessageBoxImage.Warning);
                        return;
                    }

                    analyticsSearchModel.PatientIdList = PatientList.Where(p =>
p.IsChecked == true).Select(p => p.PatientId).ToList();
                    if (analyticsSearchModel.PatientIdList.Count < 1)
                    {
                        analyticsSearchModel.PatientIdList = PatientList.Select(p =>
p.PatientId).ToList();
                    }

                    string sendItem =
JsonConvert.SerializeObject(analyticsSearchModel);
                    var analyticsRecordsResponseModel =
JsonConvert.DeserializeObject<List<AnalyticsRecordsResponseModel>>(await
Sender.PostToCoreAsync(sendItem, "getAnalyticsRecords"));

                    if (analyticsRecordsResponseModel != null)
                    {
                        Records = new
ObservableCollection<AnalyticsRecordsResponseModel>(analyticsRecordsResponseModel);
                        OnPropertyChanged("RecordsCollection");
                    }

                    // For recipes
                    var analyticsRecipesResponseModel =
JsonConvert.DeserializeObject<List<AnalyticsRecipesResponseModel>>(await
Sender.PostToCoreAsync(sendItem, "getAnalyticsRecipes"));

                    if (analyticsRecipesResponseModel != null)
                    {
                        Recipes = new
ObservableCollection<AnalyticsRecipesResponseModel>(analyticsRecipesResponseModel);
                        OnPropertyChanged("RecipesCollection");
                    }
                }
            }
            ));
    }
}

```

```

        ShowChartBtnEnabled = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
    });
}

private RelayCommand showChartClick;
public RelayCommand ShowChartClick
{
    get
    {
        return showChartClick ??
            (showChartClick = new RelayCommand(async obj =>
            {
                try
                {
                    if (!IsSelectedTestDictionary.ContainsValue(true))
                    {
                        MessageBox.Show("Жодного параметру не обрано",
"Повідомлення", MessageBoxButton.OK, MessageBoxImage.Warning);
                        return;
                    }
                    if (Records.Count < 1 && Recipes.Count < 1)
                    {
                        MessageBox.Show("Немає записів за обраними тегами",
"Повідомлення", MessageBoxButton.OK, MessageBoxImage.Warning);
                        return;
                    }

                    var analyticsSearchModel = new AnalyticsSearchModel();

                    analyticsSearchModel.DateFrom = Records.FirstOrDefault()?.RDate;
                    analyticsSearchModel.DateTo = Records.LastOrDefault()?.RDate;
                    analyticsSearchModel.DateFrom = analyticsSearchModel.DateFrom >
Recipes.FirstOrDefault()?.RDate ? Recipes.FirstOrDefault()?.RDate :
analyticsSearchModel.DateFrom;
                    analyticsSearchModel.DateTo = analyticsSearchModel.DateTo <
Recipes.LastOrDefault()?.RDate ? Recipes.LastOrDefault()?.RDate :
analyticsSearchModel.DateTo;

                    analyticsSearchModel.PatientIdList = Records.Select(rec =>
rec.PatientId ?? 0).ToList();
                    foreach (var item in Recipes)
                    {
                        if
(!analyticsSearchModel.PatientIdList.Contains(item.PatientId ?? 0))
                        {
                            analyticsSearchModel.PatientIdList.Add(item.PatientId ??
0);
                        }
                    }
                    analyticsSearchModel.PatientIdList.RemoveAll(s => s.Equals(0));
                    analyticsSearchModel.PatientIdList =
analyticsSearchModel.PatientIdList.Distinct().ToList();

                    foreach (var item in IsSelectedTestDictionary)
                    {
                        if (item.Value)
                        {
                            analyticsSearchModel.TestFieldName = item.Key;

```

```

        string sendItem =
JsonConvert.SerializeObject(analyticsSearchModel);
        var analyticsChartValuesResponseModel =
JsonConvert.DeserializeObject<List<AnalyticsChartValuesResponseModel>>(await
Sender.PostToCoreAsync(sendItem, "getAnalyticsChartValues"));

        if (analyticsChartValuesResponseModel != null)
        {
            ChartWindow chartWindow = new ChartWindow(item.Key,
analyticsChartValuesResponseModel);
            chartWindow.Show();
        }
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
}
}
));
}

private async Task LoadTagsListAsync()
{
    try
    {
        var newTagList = new List<CheckBoxTagListHelper>();

        string request = "getTags";
        List<string> tagListResult =
JsonConvert.DeserializeObject<List<string>>(await Sender.GetToCoreAsync(request));

        foreach (var item in tagListResult)
        {
            newTagList.Add(new CheckBoxTagListHelper(item, false));
        }

        TagList = newTagList;
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

private async Task LoadPatientListAsync()
{
    var newPatientList = new List<CheckBoxPatientListHelper>();

    string request = "getPatientList";
    List<PatientResponseModel> patientListResult =
JsonConvert.DeserializeObject<List<PatientResponseModel>>(await
Sender.GetToCoreAsync(request));

    foreach (var item in patientListResult)
    {
        newPatientList.Add(new CheckBoxPatientListHelper(item.PatientId,
item.FullName, false));
    }

    PatientList = newPatientList;
}

```

```
    }
}
```

Б.2 Текст файла ChartViewModel.cs

```
namespace ProHealthClient.ViewModels
{
    public class ChartViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

        public SeriesCollection SeriesCollection { get; set; }
        public Func<double, string> Formatter { get; set; }

        private string windowTitle;
        public string WindowTitle
        {
            get { return windowTitle; }
            set
            {
                windowTitle = value;
                OnPropertyChanged("WindowTitle");
            }
        }

        public ChartViewModel(string windowTitle,
            List<AnalyticsChartValuesResponseModel> chartValues)
        {
            WindowTitle = windowTitle;

            MakeChart(chartValues);
        }

        public void MakeChart(List<AnalyticsChartValuesResponseModel> chartValues)
        {
            try
            {
                var dayConfig = Mappers.Xy<DateTimePoint>()
                    .X(dateModel =>
dateModel.DateTime.Ticks / TimeSpan.FromDays(1).Ticks)
                    .Y(dateModel => dateModel.Value);

                Charting.For<DateTimePoint>(dayConfig);
                Formatter = value => new DateTime((long)(value *
TimeSpan.FromDays(1).Ticks)).ToString("d");

                SeriesCollection = new SeriesCollection();

                foreach (var item in chartValues)
                {
                    var values = new ChartValues<DateTimePoint>();
                    foreach (var point in item.ChartPointList)
                    {
                        values.Add(new DateTimePoint(point.XdateTime, point.YValue));
                    }

                    var lineSeries = new LineSeries
                    {
                        Title = item.Title,
                        Values = values,
                        PointGeometrySize = 15
                    };
                }
            }
        }
    }
}
```

```

        SeriesCollection.Add(lineSeries);
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
    MessageBoxImage.Error);
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
}
}
}
}
}

```

Б.3 Текст файлу DoctorViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class DoctorViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public DoctorViewModel()
        {
            DeclBtnIsEnabled = true;
            _ = ShowDataAsync();
        }

        private string doctorInfo;
        public string DoctorInfo
        {
            get { return doctorInfo; }
            set
            {
                doctorInfo = value;
                OnPropertyChanged("DoctorInfo");
            }
        }

        private Page dFrameContent;
        public Page DFrameContent
        {
            get { return dFrameContent; }
            set
            {
                if (dFrameContent == value)
                    return;

                dFrameContent = value;
                OnPropertyChanged("DFrameContent");
            }
        }

        private bool declBtnIsEnabled;
        public bool DeclBtnIsEnabled
        {
            get { return declBtnIsEnabled; }
            set
            {
                declBtnIsEnabled = value;
            }
        }
    }
}

```

```

        OnPropertyChanged("DeclBtnIsEnabled");
    }
}

private RelayCommand recordsClick;
public RelayCommand RecordsClick
{
    get
    {
        return recordsClick ??
            (recordsClick = new RelayCommand(obj =>
            {
                try
                {
                    DFrameContent = new RecordsFrame();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                        MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

private RelayCommand declarationClick;
public RelayCommand DeclarationClick
{
    get
    {
        return declarationClick ??
            (declarationClick = new RelayCommand(obj =>
            {
                try
                {
                    DFrameContent = new DeclarationFrame();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                        MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

private RelayCommand recipesClick;
public RelayCommand RecipesClick
{
    get
    {
        return recipesClick ??
            (recipesClick = new RelayCommand(obj =>
            {
                try
                {
                    DFrameContent = new RecipesFrame();
                }
                catch (Exception ex)
                {

```

```

        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
        MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
});
}

private RelayCommand sickLeavesClick;
public RelayCommand SickLeavesClick
{
    get
    {
        return sickLeavesClick ??
            (sickLeavesClick = new RelayCommand(obj =>
            {
                try
                {
                    DFrameContent = new SickLeavesFrame();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
        MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

private RelayCommand analyticsClick;
public RelayCommand AnalyticsClick
{
    get
    {
        return analyticsClick ??
            (analyticsClick = new RelayCommand(obj =>
            {
                try
                {
                    AnalyticsWindow analyticsWindow = new AnalyticsWindow();
                    analyticsWindow.Show();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
        MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

public async Task ShowDataAsync()
{
    try
    {
        string request = "getDoctorData/" + DataBank.DoctorId;
        var doctorResponseModel =
        JsonConvert.DeserializeObject<DoctorDataResponseModel>(await
        Sender.GetToCoreAsync(request));
    }
}

```

```

        if (doctorResponseModel != null)
        {
            DataBank.PositionId = doctorResponseModel.PositionId;
            if (DataBank.PositionId != 1)
            {
                DeclBtnIsEnabled = false;
            }

            DoctorInfo = doctorResponseModel.DInfo;
        }
        else
        {
            MessageBox.Show("Немає даних", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
}
}

```

Б.4 Текст файлу LaboratoryViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class LaboratoryViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

        private Page lFrameContent;
        public Page LFrameContent
        {
            get { return lFrameContent; }
            set
            {
                if (lFrameContent == value)
                    return;

                lFrameContent = value;
                OnPropertyChanged("LFrameContent");
            }
        }

        private RelayCommand glucoseTestClick;
        public RelayCommand GlucoseTestClick
        {
            get
            {
                return glucoseTestClick ??
                    (glucoseTestClick = new RelayCommand(obj =>
                    {
                        try
                        {
                            LFrameContent = new GlucoseTestFrame();

```

```

        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        }
    }
    ));
}

private RelayCommand urineTestClick;
public RelayCommand UrineTestClick
{
    get
    {
        return urineTestClick ??
            (urineTestClick = new RelayCommand(obj =>
            {
                try
                {
                    LFrameContent = new UrineTestFrame();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}
}
}
}

```

Б.5 Текст файлу LoginViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class LoginViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        private string login;
        public string Login
        {
            get { return login; }
            set
            {
                login = value;
                OnPropertyChanged("Login");
            }
        }

        public string Password { private get; set; }

        private RelayCommand authorize;
    }
}

```

```

public RelayCommand Authorize
{
    get
    {
        return authorize ??
            (authorize = new RelayCommand(async obj =>
            {
                try
                {
                    if (string.IsNullOrEmpty(Login) ||
string.IsNullOrEmpty>Password))
                    {
                        MessageBox.Show("Будьласка, заповніть всі поля!",
"Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                        return;
                    }

                    LoginSendModel loginSendModel = new LoginSendModel(Login,
Password);
                    string sendItem =
JsonConvert.SerializeObject(loginSendModel);

                    var loginResponseModel =
JsonConvert.DeserializeObject<LoginResponseModel>(await
Sender.PostToCoreAsync(sendItem, "authorize"));

                    if (loginResponseModel != null)
                    {
                        DataBank.AuthId = loginResponseModel.AuthId;
                        DataBank.DoctorId = loginResponseModel.DoctorId;
                        DataBank.PatientId = loginResponseModel.PatientId;
                    }

                    if (DataBank.AuthId == -1)
                    {
                        MessageBox.Show("Будьласка, введіть коректний логін и
пароль!", "Помилка", MessageBoxButton.OK, MessageBoxImage.Error);
                    }
                    else if (DataBank.DoctorId != -1)
                    {
                        HideDialog();
                        DoctorWindow d = new DoctorWindow();
                        d.Show();
                        CloseDialog();
                    }
                    else if (DataBank.PatientId != -1)
                    {
                        HideDialog();
                        PatientWindow p = new PatientWindow(false);
                        p.Show();
                        CloseDialog();
                    }
                    else if (DataBank.AuthId == 2)
                    {
                        HideDialog();
                        LaboratoryWindow l = new LaboratoryWindow();
                        l.Show();
                        CloseDialog();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            }
        }
    }
}

```

```

        CloseDialog();
    }
}
));
}
}

public Action HideDialog { get; set; }
public Action CloseDialog { get; set; }
}

```

Б.6 Текст файла PatientViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class PatientViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public PatientViewModel()
        {
            _ = LoadData();
        }

        private string pName;
        public string PName
        {
            get { return pName; }
            set
            {
                pName = value;
                OnPropertyChanged("PName");
            }
        }

        private DataTable recordsTable;
        public DataTable RecordsTable
        {
            get { return recordsTable; }
            set
            {
                recordsTable = value;
                OnPropertyChanged("RecordsTable");
            }
        }

        private DataTable recipesTable;
        public DataTable RecipesTable
        {
            get { return recipesTable; }
            set
            {
                recipesTable = value;
                OnPropertyChanged("RecipesTable");
            }
        }

        private DataTable glucoseTestsTable;
        public DataTable GlucoseTestsTable
        {

```

```

        get { return glucoseTestsTable; }
        set
        {
            glucoseTestsTable = value;
            OnPropertyChanged("GlucoseTestsTable");
        }
    }

    private DataTable urineTestsTable;
    public DataTable UrineTestsTable
    {
        get { return urineTestsTable; }
        set
        {
            urineTestsTable = value;
            OnPropertyChanged("UrineTestsTable");
        }
    }

    private DataTable sickLeavesTable;
    public DataTable SickLeavesTable
    {
        get { return sickLeavesTable; }
        set
        {
            sickLeavesTable = value;
            OnPropertyChanged("SickLeavesTable");
        }
    }

    private RelayCommand allInfoClick;
    public RelayCommand AllInfoClick
    {
        get
        {
            return allInfoClick ??
                (allInfoClick = new RelayCommand(async obj =>
                {
                    try
                    {
                        string request = "getDeclaration/" + DataBank.PatientId;
                        DeclarationResponseModel declarationResponseModel =
                            JsonConvert.DeserializeObject<DeclarationResponseModel>(await
                                Sender.GetToCoreAsync(request));

                        if (declarationResponseModel != null)
                        {
                            DeclarationReport(declarationResponseModel);
                        }
                        else
                        {
                            MessageBox.Show("Помилка завантаження декларації!",
                                "Повідомлення", MessageBoxButton.OK, MessageBoxImage.Information);
                        }
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                            MessageBoxImage.Error);
                        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                    }
                }
                ));
        }
    }
}

```

```

        async Task LoadData()
        {
            try
            {
                string request = "getPatientAllData/" + DataBank.PatientId;
                PatientAllDataResponseModel patientAllData =
                JsonConvert.DeserializeObject<PatientAllDataResponseModel>(await
                Sender.GetToCoreAsync(request));

                if (patientAllData != null)
                {
                    PName = patientAllData.PName;
                    RecordsTable = patientAllData.RecordsTable;
                    RecipesTable = patientAllData.RecipesTable;
                    GlucoseTestsTable = patientAllData.GlucoseTestsTable;
                    UrineTestsTable = patientAllData.UrineTestsTable;
                    SickLeavesTable = patientAllData.SickLeavesTable;
                }
                else
                {
                    MessageBox.Show("Немає даних", "Помилка", MessageBoxButton.OK,
                    MessageBoxImage.Error);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                MessageBoxImage.Error);
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            }
        }

        public void DeclarationReport(DeclarationResponseModel
        declarationResponseModel)
        {
            var wordApp = new Word.Application();
            wordApp.Visible = false;

            try
            {
                var wordDocument = wordApp.Documents.Open(DataBank.TemplateFilePath);
                ReplaceWordStrub("<Declaration_id>",
                declarationResponseModel.Declaration_id, wordDocument);
                ReplaceWordStrub("<Surname>", declarationResponseModel.PSurname,
                wordDocument);
                ReplaceWordStrub("<Name>", declarationResponseModel.PName,
                wordDocument);
                ReplaceWordStrub("<Patronymic>", declarationResponseModel.PPatronymic,
                wordDocument);
                ReplaceWordStrub("<Date_of_Birth>",
                declarationResponseModel.DateOfBirth, wordDocument);
                ReplaceWordStrub("<C_and_P_of_birth>", declarationResponseModel.Place,
                wordDocument);
                ReplaceWordStrub("<Sex>", declarationResponseModel.Sex, wordDocument);
                ReplaceWordStrub("<Phone_number>",
                declarationResponseModel.PhoneNumber, wordDocument);
                ReplaceWordStrub("<Email>", declarationResponseModel.Email,
                wordDocument);
                ReplaceWordStrub("<ITN>", declarationResponseModel.ITN, wordDocument);
                ReplaceWordStrub("<DSurname>", declarationResponseModel.DSurname,
                wordDocument);
                ReplaceWordStrub("<DName>", declarationResponseModel.DName,
                wordDocument);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                MessageBoxImage.Error);
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            }
        }
    }
}

```

```

        ReplaceWordStrub("<DPatronymic>",
declarationResponseModel.DPatronymic, wordDocument);
        ReplaceWordStrub("<DPhone_number>",
declarationResponseModel.DPhoneNumber, wordDocument);
        ReplaceWordStrub("<DEmail>", declarationResponseModel.DEmail,
wordDocument);
        ReplaceWordStrub("<Position>", declarationResponseModel.Position,
wordDocument);
        ReplaceWordStrub("<Unit>", declarationResponseModel.Unit,
wordDocument);
        ReplaceWordStrub("<Date>", declarationResponseModel.ConclusionDate,
wordDocument);
        ReplaceWordStrub("<Passport>", declarationResponseModel.Passport,
wordDocument);
        ReplaceWordStrub("<Place>", declarationResponseModel.Place,
wordDocument);

        wordDocument.SaveAs(Directory.GetCurrentDirectory() + @"\\" +
declarationResponseModel.PSurname + declarationResponseModel.ITN + ".docx");
        wordApp.Visible = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

private void ReplaceWordStrub(string strubToReplace, string text,
Word.Document wordDocument)
{
    try
    {
        var range = wordDocument.Content;
        range.Find.ClearFormatting();
        range.Find.Execute(FindText: strubToReplace, ReplaceWith: text);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
}
}

```

Б.7 Текст файлу RecipesViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class RecipesViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public RecipesViewModel()
        {
            _ = ShowDataAsync();
            _ = LoadTags();
        }
    }
}

```

```

        StartDate = DateTime.Today;
        EndDate = DateTime.Today;
    }

    private string medicament;
    public string Medicament
    {
        get { return medicament; }
        set
        {
            medicament = value;
            OnPropertyChanged("Medicament");
        }
    }

    private string quantityPerReception;
    public string QuantityPerReception
    {
        get { return quantityPerReception; }
        set
        {
            quantityPerReception = value;
            OnPropertyChanged("QuantityPerReception");
        }
    }

    private string quantityPerDay;
    public string QuantityPerDay
    {
        get { return quantityPerDay; }
        set
        {
            quantityPerDay = value;
            OnPropertyChanged("QuantityPerDay");
        }
    }

    private string durationOfTreatment;
    public string DurationOfTreatment
    {
        get { return durationOfTreatment; }
        set
        {
            durationOfTreatment = value;
            OnPropertyChanged("DurationOfTreatment");
        }
    }

    private string dosageAmount;
    public string DosageAmount
    {
        get { return dosageAmount; }
        set
        {
            dosageAmount = value;
            OnPropertyChanged("DosageAmount");
        }
    }

    private string signature;
    public string Signature
    {
        get { return signature; }
        set
        {
            signature = value;
            OnPropertyChanged("Signature");
        }
    }

```

```

    }

    private DateTime startDate;
    public DateTime StartDate
    {
        get { return startDate; }
        set
        {
            startDate = value;
            OnPropertyChanged("StartDate");
        }
    }

    private DateTime endDate;
    public DateTime EndDate
    {
        get { return endDate; }
        set
        {
            endDate = value;
            OnPropertyChanged("EndDate");
        }
    }

    private string tagNames;
    public string TagNames
    {
        get { return tagNames; }
        set
        {
            tagNames = value;
            OnPropertyChanged("TagNames");
        }
    }

    private string tags;
    public string Tags
    {
        get { return tags; }
        set
        {
            tags = value;
            OnPropertyChanged("Tags");
        }
    }

    private int selectedPatientId;
    public int SelectedPatientId
    {
        get { return selectedPatientId; }
        set
        {
            selectedPatientId = value;
            OnPropertyChanged("SelectedPatientId");
        }
    }

    private ObservableCollection<PatientResponseModel> patients;
    public ObservableCollection<PatientResponseModel> Patients
    {
        get { return patients; }
        set
        {
            patients = value;
            OnPropertyChanged("Patients");
        }
    }

    private Page rFrameContent;

```

```

public Page RFrameContent
{
    get { return rFrameContent; }
    set
    {
        if (rFrameContent == value)
            return;

        rFrameContent = value;
        OnPropertyChanged("RFrameContent");
    }
}

private RelayCommand createClick;
public RelayCommand CreateClick
{
    get
    {
        return createClick ??
            (createClick = new RelayCommand(obj =>
            {
                try
                {
                    if (SelectedPatientId != 0)
                    {
                        if (String.IsNullOrEmpty(Medicament) ||
String.IsNullOrEmpty(QuantityPerReception) ||
String.IsNullOrEmpty(QuantityPerDay) ||
                        String.IsNullOrEmpty(DurationOfTreatment) ||
String.IsNullOrEmpty(DosageAmount) || String.IsNullOrEmpty(Signature))
                        {
                            MessageBox.Show("Заповнені не всі обов'язкові поля!",
"Повідомлення", MessageBoxButton.OK, MessageBoxImage.Information);
                        }
                        else
                        {
                            SaveRecipe();
                        }
                    }
                    else
                    {
                        MessageBox.Show("Пацієнт не обраний!", "Повідомлення",
MessageBoxButton.OK, MessageBoxImage.Information);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

private RelayCommand backClick;
public RelayCommand BackClick
{
    get
    {
        return backClick ??
            (backClick = new RelayCommand(obj =>
            {
                try
                {
                    MessageBoxResult d = MessageBox.Show("Ви дійсно хочете скасувати
дії?", "Повідомлення", MessageBoxButton.YesNo, MessageBoxImage.Information);

```

```

        if (d == DialogResult.Yes)
        {
            RFrameContent = new DoctorFrame();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK,
        MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
    });
}

private RelayCommand addTagClick;
public RelayCommand AddTagClick
{
    get
    {
        return addTagClick ??
            (addTagClick = new RelayCommand(async obj =>
            {
                try
                {
                    TagSendModel tagSendModel = new TagSendModel();
                    tagSendModel.TagName = Interaction.InputBox("Введіть один тег",
                    "Додавання тегу");

                    if (!string.IsNullOrEmpty(tagSendModel.TagName))
                    {
                        tagSendModel.TagName = tagSendModel.TagName.Trim();
                        if (tagSendModel.TagName[0] != '#')
                        {
                            tagSendModel.TagName = "#" + tagSendModel.TagName;
                        }

                        string sendItem = JsonConvert.SerializeObject(tagSendModel);
                        var isSuccess = JsonConvert.DeserializeObject<bool>(await
                        Sender.PostToCoreAsync(sendItem, "addTag"));

                        if (!isSuccess)
                        {
                            MessageBox.Show("Тег не був доданий", "Помилка",
                            MessageBoxButtons.OK, MessageBoxImage.Error);
                        }
                        else
                        {
                            LoadTags();
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK,
                    MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

public async Task ShowDataAsync()
{
    try
    {
        string request = "getPatientList";
    }
}

```

```

        List<PatientResponseModel> patientList =
JsonConvert.DeserializeObject<List<PatientResponseModel>>(await
Sender.GetToCoreAsync(request));

        if (patientList != null)
        {
            Patients = new ObservableCollection<PatientResponseModel>(patientList);
        }
        else
        {
            MessageBox.Show("Немає даних пацієнтів", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

public async Task LoadTags()
{
    try
    {
        string request = "getTags";
        List<string> tagList = JsonConvert.DeserializeObject<List<string>>(await
Sender.GetToCoreAsync(request));

        Tags = String.Join(", ", tagList);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

private async void SaveRecipe()
{
    try
    {
        RecipeSendModel recipeSendModel = new RecipeSendModel(SelectedPatientId,
DataBank.DoctorId, Medicament, QuantityPerReception, QuantityPerDay,
DurationOfTreatment,
DosageAmount, Signature, DateTime.Now, StartDate, EndDate, TagNames);
        string sendItem = JsonConvert.SerializeObject(recipeSendModel);

        var result = JsonConvert.DeserializeObject<bool>(await
Sender.PostToCoreAsync(sendItem, "saveRecipe"));

        if (result)
        {
            MessageBox.Show("Рецепт успішно створений!", "Повідомлення",
MessageBoxButton.OK, MessageBoxImage.Information);
            RFrameContent = new DoctorFrame();
        }
        else
        {
            MessageBox.Show("Помилка збереження даних", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    catch (Exception ex)
    {

```

```

        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
        MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
}
}

```

Б.8 Текст файлу RecordsViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class RecordsViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public RecordsViewModel()
        {
            _ = LoadPatientList();
            _ = LoadTags();
        }

        private string declarationInfo;
        public string DeclarationInfo
        {
            get { return declarationInfo; }
            set
            {
                declarationInfo = value;
                OnPropertyChanged("DeclarationInfo");
            }
        }

        private string dateOfBirth;
        public string DateOfBirth
        {
            get { return dateOfBirth; }
            set
            {
                dateOfBirth = value;
                OnPropertyChanged("DateOfBirth");
            }
        }

        private string phoneNumber;
        public string PhoneNumber
        {
            get { return phoneNumber; }
            set
            {
                phoneNumber = value;
                OnPropertyChanged("PhoneNumber");
            }
        }

        private string placeOfResidence;
        public string PlaceOfResidence
        {
            get { return placeOfResidence; }
            set
            {
                placeOfResidence = value;
                OnPropertyChanged("PlaceOfResidence");
            }
        }
    }
}

```

```

}

private string tags;
public string Tags
{
    get { return tags; }
    set
    {
        tags = value;
        OnPropertyChanged("Tags");
    }
}

private Brush declarationInfoColor = Brushes.Black;
public Brush DeclarationInfoColor
{
    get { return declarationInfoColor; }
    set
    {
        declarationInfoColor = value;
        OnPropertyChanged("DeclarationInfoColor");
    }
}

private DataTable recordsTable;
public DataTable RecordsTable
{
    get { return recordsTable; }
    set
    {
        recordsTable = value;
        OnPropertyChanged("RecordsTable");
    }
}

private int selectedRecord;
public int SelectedRecord
{
    get { return selectedRecord; }
    set
    {
        selectedRecord = value;
        OnPropertyChanged("SelectedRecord");
    }
}

private int selectedPatientId;
public int SelectedPatientId
{
    get { return selectedPatientId; }
    set
    {
        selectedPatientId = value;
        OnPropertyChanged("SelectedPatientId");

        _ = ShowDataAsync();
        DataBank.PatientId = selectedPatientId;
    }
}

private ObservableCollection<PatientResponseModel> patients;
public ObservableCollection<PatientResponseModel> Patients
{
    get { return patients; }
    set
    {
        patients = value;
        OnPropertyChanged("Patients");
    }
}

```

```

    }

    private Page rFrameContent;
    public Page RFrameContent
    {
        get { return rFrameContent; }
        set
        {
            if (rFrameContent == value)
                return;

            rFrameContent = value;
            OnPropertyChanged("RFrameContent");
        }
    }

    private RelayCommand saveClick;
    public RelayCommand SaveClick
    {
        get
        {
            return saveClick ??
                (saveClick = new RelayCommand(obj =>
                {
                    try
                    {
                        if (SelectedPatientId != 0)
                        {
                            if (RecordsTable.Rows.Count != 1)
                            {
                                if (RecordsTable.Rows[RecordsTable.Rows.Count -
1][1].ToString() == "")
                                    {
                                        RecordsTable.Rows[RecordsTable.Rows.Count - 1][1] =
SelectedPatientId;
                                    }

                                if (RecordsTable.Rows[RecordsTable.Rows.Count -
1][2].ToString() == "")
                                    {
                                        RecordsTable.Rows[RecordsTable.Rows.Count - 1][2] =
DataBank.DoctorId;
                                    }
                                }

                                SaveRecords();
                            }
                            else
                            {
                                MessageBox.Show("Пацієнт не обраний!", "Повідомлення",
MessageBoxButton.OK, MessageBoxImage.Information);
                            }
                        }
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                    }
                }
                ));
        }
    }

    private RelayCommand backClick;
    public RelayCommand BackClick
    {
        get
        {

```

```

        return backClick ??
        (backClick = new RelayCommand(obj =>
        {
            try
            {
                MessageBoxResult d = MessageBox.Show("Ви дійсно хочете скасувати дії?", "Повідомлення", MessageBoxButton.YesNo, MessageBoxImage.Information);
                if (d == MessageBoxResult.Yes)
                {
                    RFrameContent = new DoctorFrame();
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                MessageBoxImage.Error);
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            }
        }
        ));

    private RelayCommand infoClick;
    public RelayCommand InfoClick
    {
        get
        {
            return infoClick ??
            (infoClick = new RelayCommand(obj =>
            {
                try
                {
                    if (SelectedPatientId != 0)
                    {
                        PatientWindow p = new PatientWindow(true);
                        p.Show();
                    }
                    else
                    {
                        MessageBox.Show("Пацієнт не обраний!", "Повідомлення",
                        MessageBoxButton.OK, MessageBoxImage.Information);
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                    MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
        }
    }

    private RelayCommand addTagClick;
    public RelayCommand AddTagClick
    {
        get
        {
            return addTagClick ??
            (addTagClick = new RelayCommand(async obj =>
            {
                try
                {
                    TagSendModel tagSendModel = new TagSendModel();
                    tagSendModel.TagName = Interaction.InputBox("Введіть один тег",
                    "Додавання тегу");
                }
            }
            ));
        }
    }

```

```

        if (!string.IsNullOrEmpty(tagSendModel.TagName))
        {
            tagSendModel.TagName = tagSendModel.TagName.Trim();
            if (tagSendModel.TagName[0] != '#')
            {
                tagSendModel.TagName = "#" + tagSendModel.TagName;
            }

            string sendItem = JsonConvert.SerializeObject(tagSendModel);
            var isSuccess = JsonConvert.DeserializeObject<bool>(await
Sender.PostToCoreAsync(sendItem, "addTag"));

            if (!isSuccess)
            {
                MessageBox.Show("Тег не був доданий", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
            }
            else
            {
                LoadTags();
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
});
}

public async Task LoadPatientList()
{
    try
    {
        string request = "getPatientList";
        List<PatientResponseModel> patientList =
JsonConvert.DeserializeObject<List<PatientResponseModel>>(await
Sender.GetToCoreAsync(request));

        if (patientList != null)
        {
            Patients = new ObservableCollection<PatientResponseModel>(patientList);
        }
        else
        {
            MessageBox.Show("Немає даних пацієнтів", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

public async Task LoadTags()
{
    try
    {
        string request = "getTags";

```

```

        List<string> tagList = JsonConvert.DeserializeObject<List<string>>(await
Sender.GetToCoreAsync(request));

        Tags = String.Join(", ", tagList);
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

async Task ShowDataAsync()
{
    try
    {
        string request = "getRecordsData/" + DataBank.DoctorId + "/" +
SelectedPatientId;
        RecordsDataResponseModel recordsData =
JsonConvert.DeserializeObject<RecordsDataResponseModel>(await
Sender.GetToCoreAsync(request));

        if (recordsData != null)
        {
            DeclarationInfo = recordsData.DeclarationInfo;
            DateOfBirth = recordsData.DateOfBirth;
            PhoneNumber = recordsData.PhoneNumber;
            PlaceOfResidence = recordsData.PlaceOfResidence;
            RecordsTable = recordsData.RecordsDataTable;

            DeclarationInfoColor = DeclarationInfo == "Декларація укладена" ?
Brushes.LightGreen : Brushes.Red;
        }
        else
        {
            MessageBox.Show("Немає даних", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

private async void SaveRecords()
{
    try
    {
        List<RecordsSendModel> recordsSendModelList = new List<RecordsSendModel>();

        for (int i = RecordsTable.Rows.Count - 1; i >= 0; i--)
        {
            if (String.IsNullOrEmpty(RecordsTable.Rows[i][0].ToString()))
            {
                recordsSendModelList.Add(new RecordsSendModel()
                {
                    PatientId = Int32.Parse(RecordsTable.Rows[i][1].ToString()),
                    DoctorId = Int32.Parse(RecordsTable.Rows[i][2].ToString()),
                    RDate = DateTime.Now,
                    Complaints = RecordsTable.Rows[i][4].ToString(),
                    Diagnosis = RecordsTable.Rows[i][5].ToString(),
                    Treatment = RecordsTable.Rows[i][6].ToString()
                });
            }
        }
    }
}

```

```

        else
        {
            break;
        }
    }

    string sendItem = JsonConvert.SerializeObject(recordsSendModelList);
    var result = JsonConvert.DeserializeObject<bool>(await
Sender.PostToCoreAsync(sendItem, "saveRecords"));

    if (result)
    {
        MessageBox.Show("Прийом успішно завершений", "", MessageBoxButton.OK,
MessageBoxImage.Information);
        RFrameContent = new DoctorFrame();
    }
    else
    {
        MessageBox.Show("Помилка збереження даних", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
}
}
}
}

```

Б.9 Текст файлу SickLeavesViewModel.cs

```

namespace ProHealthClient.ViewModels
{
    public class SickLeavesViewModel : INotifyPropertyChanged
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Sender Sender = new Sender();

        public SickLeavesViewModel()
        {
            _ = LoadPatientList();
        }

        private string declarationInfo;
        public string DeclarationInfo
        {
            get { return declarationInfo; }
            set
            {
                declarationInfo = value;
                OnPropertyChanged("DeclarationInfo");
            }
        }

        private string dateOfBirth;
        public string DateOfBirth
        {
            get { return dateOfBirth; }
            set
            {
                dateOfBirth = value;
                OnPropertyChanged("DateOfBirth");
            }
        }
    }
}

```

```

    }
}

private string phoneNumber;
public string PhoneNumber
{
    get { return phoneNumber; }
    set
    {
        phoneNumber = value;
        OnPropertyChanged("PhoneNumber");
    }
}

private string placeOfResidence;
public string PlaceOfResidence
{
    get { return placeOfResidence; }
    set
    {
        placeOfResidence = value;
        OnPropertyChanged("PlaceOfResidence");
    }
}

private Brush declarationInfoColor = Brushes.Black;
public Brush DeclarationInfoColor
{
    get { return declarationInfoColor; }
    set
    {
        declarationInfoColor = value;
        OnPropertyChanged("DeclarationInfoColor");
    }
}

private DataTable sickLeavesTable;
public DataTable SickLeavesTable
{
    get { return sickLeavesTable; }
    set
    {
        sickLeavesTable = value;
        OnPropertyChanged("SickLeavesTable");
    }
}

private int selectedSickLeave;
public int SelectedSickLeave
{
    get { return selectedSickLeave; }
    set
    {
        selectedSickLeave = value;
        OnPropertyChanged("SelectedSickLeave");
    }
}

private int selectedPatientId;
public int SelectedPatientId
{
    get { return selectedPatientId; }
    set
    {
        selectedPatientId = value;
        OnPropertyChanged("SelectedPatientId");

        _ = ShowDataAsync();
    }
}

```

```

    }

    private ObservableCollection<PatientResponseModel> patients;
    public ObservableCollection<PatientResponseModel> Patients
    {
        get { return patients; }
        set
        {
            patients = value;
            OnPropertyChanged("Patients");
        }
    }

    private Page sFrameContent;
    public Page SFrameContent
    {
        get { return sFrameContent; }
        set
        {
            if (sFrameContent == value)
                return;

            sFrameContent = value;
            OnPropertyChanged("SFrameContent");
        }
    }

    private RelayCommand saveClick;
    public RelayCommand SaveClick
    {
        get
        {
            return saveClick ??
                (saveClick = new RelayCommand(obj =>
                    {
                        try
                        {
                            if (SelectedPatientId != 0)
                            {
                                if (SickLeavesTable.Rows.Count != 1)
                                    if (SickLeavesTable.Rows[SickLeavesTable.Rows.Count -
1][1].ToString() == "")
                                        SickLeavesTable.Rows[SickLeavesTable.Rows.Count -
1][1] = SelectedPatientId;

                                if (SickLeavesTable.Rows[SickLeavesTable.Rows.Count -
1][2].ToString() == "")
                                    SickLeavesTable.Rows[SickLeavesTable.Rows.Count -
1][2] = DataBank.DoctorId;
                            }

                            SaveSickLeaves();
                        }
                        else
                        {
                            MessageBox.Show("Пацієнт не обраний!", "Повідомлення",
                                MessageBoxButton.OK, MessageBoxImage.Information);
                        }
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                                MessageBoxImage.Error);
                        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                    }
                )
            );
        }
    }

```

```

        }
    }
    });
}

private RelayCommand backClick;
public RelayCommand BackClick
{
    get
    {
        return backClick ??
            (backClick = new RelayCommand(obj =>
            {
                try
                {
                    MessageBoxResult d = MessageBox.Show("Ви дійсно хочете скасувати дії?", "Повідомлення", MessageBoxButton.YesNo, MessageBoxImage.Information);
                    if (d == MessageBoxResult.Yes)
                    {
                        SFrameContent = new DoctorFrame();
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
                        MessageBoxImage.Error);
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                }
            }
            ));
    }
}

public async Task LoadPatientList()
{
    try
    {
        string request = "getPatientList";
        List<PatientResponseModel> patientList =
            JsonConvert.DeserializeObject<List<PatientResponseModel>>(await
            Sender.GetToCoreAsync(request));

        if (patientList != null)
        {
            Patients = new ObservableCollection<PatientResponseModel>(patientList);
        }
        else
        {
            MessageBox.Show("Немає даних пацієнтів", "Помилка",
                MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
            MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

async Task ShowDataAsync()
{
    try
    {
        string request = "getSickLeavesData/" + DataBank.DoctorId + "/" +
            SelectedPatientId;
    }
}

```

```

        SickLeavesDataResponseModel sickLeavesData =
JsonConvert.DeserializeObject<SickLeavesDataResponseModel>(await
Sender.GetToCoreAsync(request));

        if (sickLeavesData != null)
        {
            DeclarationInfo = sickLeavesData.DeclarationInfo;
            DateOfBirth = sickLeavesData.DateOfBirth;
            PhoneNumber = sickLeavesData.PhoneNumber;
            PlaceOfResidence = sickLeavesData.PlaceOfResidence;
            SickLeavesTable = sickLeavesData.SickLeavesDataTable;

            DeclarationInfoColor = DeclarationInfo == "Декларація укладена" ?
Brushes.LightGreen : Brushes.Red;
        }
        else
        {
            MessageBox.Show("Немає даних", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}

private async void SaveSickLeaves()
{
    try
    {
        List<SickLeavesSendModel> sickLeavesSendModel = new
List<SickLeavesSendModel>();

        for (int i = SickLeavesTable.Rows.Count - 1; i >= 0; i--)
        {
            if (String.IsNullOrEmpty(SickLeavesTable.Rows[i][0].ToString()))
            {
                sickLeavesSendModel.Add(new SickLeavesSendModel()
                {
                    PatientId = Int32.Parse(SickLeavesTable.Rows[i][1].ToString()),
                    DoctorId = Int32.Parse(SickLeavesTable.Rows[i][2].ToString()),
                    SDate = DateTime.Now,
                    Regimen = SickLeavesTable.Rows[i][4].ToString(),
                    SDiagnosis = SickLeavesTable.Rows[i][5].ToString(),
                    SFrom = DateTime.Parse(SickLeavesTable.Rows[i][6].ToString()),
                    STo = DateTime.Parse(SickLeavesTable.Rows[i][7].ToString()),
                    GetToWork =
DateTime.Parse(SickLeavesTable.Rows[i][8].ToString())
                });
            }
            else
            {
                break;
            }
        }

        string sendItem = JsonConvert.SerializeObject(sickLeavesSendModel);

        var result = JsonConvert.DeserializeObject<bool>(await
Sender.PostToCoreAsync(sendItem, "saveSickLeaves"));

        if (result)
        {
            MessageBox.Show("Лікарняний успішно виписаний!", "",
MessageBoxButton.OK, MessageBoxImage.Information);
            SFrameContent = new DoctorFrame();
        }
    }
}

```

```

        }
        else
        {
            MessageBox.Show("Помилка збереження даних", "Помилка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
}
}
}

```

Б.10 Текст файлу Sender.cs

```

namespace ProHealthClient
{
    public class Sender
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

        public async Task<string> PostToCoreAsync(string postData, string command)
        {
            try
            {
                var data = new StringContent(postData, Encoding.UTF8,
                    "application/json");

                using (HttpClient client = new HttpClient())
                {
                    var response = await
                        client.PostAsync(ConfigurationManager.AppSettings["CoreUri"] + command, data);

                    var result = await response.Content.ReadAsStringAsync();

                    return result;
                }
            }
            catch (WebException webex)
            {
                Logger.Error(webex.Response);
                throw new Exception(webex.ToString());
            }
        }

        public async Task<string> GetToCoreAsync(string request)
        {
            try
            {
                using (var client = new HttpClient())
                {
                    var message = await
                        client.GetAsync(ConfigurationManager.AppSettings["CoreUri"] + request);

                    if (!message.IsSuccessStatusCode)
                        throw new Exception("Помилка отримання даних!");

                    return await message.Content.ReadAsStringAsync();
                }
            }
        }
    }
}

```

```

    }
}
catch (WebException webex)
{
    Logger.Error(webex.Response);
    throw new Exception(webex.ToString());
}
}
}
}
}

```

Б.11 Текст файла Core.cs

```

namespace ProHealthCore
{
    public class Core
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

        public LoginResponseModel Authorize(LoginSendModel loginSendModel)
        {
            try
            {
                using (DatabaseContext dB = new DatabaseContext())
                {
                    var authData = dB.Authorizations.Where(a => a.Login ==
loginSendModel.Login && a.Password == loginSendModel.Password).FirstOrDefault();

                    if (authData != null)
                    {
                        return new LoginResponseModel(authData.Auth_id, authData.Doctor_id
?? -1, authData.Patient_id ?? -1);
                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                return null;
            }
        }

        public bool SaveGlTest(GlucoseTestSendModel glucoseTestSendModel)
        {
            try
            {
                using (DatabaseContext dB = new DatabaseContext())
                {
                    GlucoseTest newGlucoseTest = new GlucoseTest()
                    {
                        Patient_id = glucoseTestSendModel.PatientId,
                        GlDate = DateTime.Now,
                        Result = glucoseTestSendModel.GlucoseValue
                    };

                    dB.GlucoseTests.Add(newGlucoseTest);
                    dB.SaveChanges();

                    return true;
                }
            }
            catch (Exception ex)

```

```

    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return false;
    }
}

public bool SaveUrTest(UrineTestSendModel urineTestSendModel)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            UrineTest newUrineTest = new UrineTest()
            {
                Patient_id = urineTestSendModel.PatientId,
                UrDate = DateTime.Now,
                Color = urineTestSendModel.Color,
                Reaction = urineTestSendModel.Reaction,
                SpecificGravity = urineTestSendModel.SpecificGravity,
                Protein = urineTestSendModel.Protein,
                Sugar = urineTestSendModel.Sugar,
                Erythrocytes = urineTestSendModel.Erythrocytes,
                Leukocytes = urineTestSendModel.Leukocytes,
                Bacteria = urineTestSendModel.Bacteria,
                FlatEpithelium = urineTestSendModel.FlatEpithelium
            };

            dB.UrineTests.Add(newUrineTest);
            dB.SaveChanges();

            return true;
        }
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return false;
    }
}

public bool SaveRecipe(RecipeSendModel recipeSendModel)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            Recipe newRecipe = new Recipe()
            {
                Patient_id = recipeSendModel.PatientId,
                Doctor_id = recipeSendModel.DoctorId,
                Medicament = recipeSendModel.Medicament,
                Quantity_per_reception = recipeSendModel.QuantityPerReception,
                Quantity_per_day = recipeSendModel.QuantityPerDay,
                Duration_of_treatment = recipeSendModel.DurationOfTreatment,
                Dosage_amount = recipeSendModel.DosageAmount,
                Signature = recipeSendModel.Signature,
                Date_of_prescription = recipeSendModel.DateOfPrescription,
                Start_date = recipeSendModel.StartDate,
                End_date = recipeSendModel.EndDate,
                TagNames = recipeSendModel.TagNames
            };

            dB.Recipes.Add(newRecipe);
            dB.SaveChanges();

            return true;
        }
    }
    catch (Exception ex)

```

```

    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return false;
    }
}

public bool SaveDeclaration(DeclarationSendModel declarationSendModel)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            Passport newPassport = new Passport()
            {
                Series_and_number = declarationSendModel.SeriesAndNumber,
                Issuing_authority = declarationSendModel.IssuingAuthority,
                Date_of_issue = declarationSendModel.DateOfIssue
            };

            dB.Passports.Add(newPassport);
            dB.SaveChanges();

            PlaceOfResidence newPlaceOfResidence = new PlaceOfResidence()
            {
                Locality_id = declarationSendModel.LocalityId,
                Street = declarationSendModel.Street,
                House = declarationSendModel.House,
                Flat = declarationSendModel.Flat,
                PIndex = declarationSendModel.PIndex
            };

            dB.PlaceOfResidences.Add(newPlaceOfResidence);
            dB.SaveChanges();

            Patient newPatient = new Patient()
            {
                Surname = declarationSendModel.Surname,
                Name = declarationSendModel.Name,
                Patronymic = declarationSendModel.Patronymic,
                Sex = declarationSendModel.Sex,
                Date_of_Birth = declarationSendModel.DateOfBirth,
                Country_of_birth = declarationSendModel.CountryOfBirth,
                Place_of_birth = declarationSendModel.PlaceOfbirth,
                ITN = declarationSendModel.ITN,
                Phone_number = declarationSendModel.PhoneNumber,
                Email = declarationSendModel.Email,
                Passport_id = newPassport.Passport_id,
                Place_id = newPlaceOfResidence.Place_id,
            };

            dB.Patients.Add(newPatient);
            dB.SaveChanges();

            Declaration newDeclaration = new Declaration()
            {
                Patient_id = newPatient.Patient_id,
                Doctor_id = declarationSendModel.DoctorId,
                Conclusion_date = DateTime.Today
            };

            dB.Declarations.Add(newDeclaration);
            dB.SaveChanges();

            Authorization newAuthorization = new Authorization()
            {
                Login = newPatient.Phone_number,
                Password = newPatient.ITN,
                Patient_id = newPatient.Patient_id
            };

```

```

        dB.Authorizations.Add(newAuthorization);
        dB.SaveChanges();

        return true;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return false;
}
}

public bool SaveRecords(List<RecordsSendModel> recordsSendModel)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            foreach (var item in recordsSendModel)
            {
                Record newRecord = new Record()
                {
                    Patient_id = item.PatientId,
                    Doctor_id = item.DoctorId,
                    RDate = item.RDate,
                    Complaints = item.Complaints,
                    Diagnosis = item.Diagnosis,
                    Treatment = item.Treatment
                };

                dB.Records.Add(newRecord);
            }

            dB.SaveChanges();

            return true;
        }
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return false;
    }
}

public bool SaveSickLeaves(List<SickLeavesSendModel> sickLeavesSendModel)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            foreach (var item in sickLeavesSendModel)
            {
                SickLeave newSickLeave = new SickLeave()
                {
                    Patient_id = item.PatientId,
                    Doctor_id = item.DoctorId,
                    SDate = item.SDate,
                    Regimen = item.Regimen,
                    SDiagnosis = item.SDiagnosis,
                    SFrom = item.SFrom,
                    STo = item.STo,
                    Get_to_work = item.GetToWork,
                };

                dB.SickLeaves.Add(newSickLeave);
            }
        }
    }
}

```

```

        }

        dB.SaveChanges();

        return true;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return false;
}
}

public bool AddTag(string tagName)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            Tag newTag = new Tag()
            {
                TagName = tagName
            };

            dB.Tags.Add(newTag);
            dB.SaveChanges();
        }

        return true;
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return false;
    }
}

public List<AnalyticsRecordsResponseModel> GetAnalyticsRecords(AnalyticsSearchModel
analyticsSearchModel)
{
    try
    {
        var analyticsRecordsResponseModel = new
List<AnalyticsRecordsResponseModel>();

        using (DatabaseContext dB = new DatabaseContext())
        {
            var records = dB.Records.Where(r =>
analyticsSearchModel.PatientIdList.Contains(r.Patient_id ?? 0) &&
(analyticsSearchModel.TagList.Any(x
=> r.Complaints.Contains(x)) ||
analyticsSearchModel.TagList.Any(x
=> r.Diagnosis.Contains(x)) ||
analyticsSearchModel.TagList.Any(x
=> r.Treatment.Contains(x))))).OrderBy(r => r.RDate).ToList();

            if (records != null)
            {
                foreach (var rec in records)
                {
                    analyticsRecordsResponseModel.Add(new
AnalyticsRecordsResponseModel(rec.Record_id, rec.Patient_id, rec.RDate, rec.Doctor.DSurname
+ " " + rec.Doctor.DName?[0] + ". " + rec.Doctor.DPatronymic?[0] + ".",
rec.Patient.Surname + " " + rec.Patient.Name?[0] + ". " + rec.Patient.Patronymic?[0] + ".",
rec.Complaints, rec.Diagnosis, rec.Treatment));
                }
            }
        }
    }
}

```

```

        }
    }

    return analyticsRecordsResponseModel;
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public List<AnalyticsRecipesResponseModel> GetAnalyticsRecipes(AnalyticsSearchModel
analyticsSearchModel)
{
    try
    {
        var analyticsRecipesResponseModel = new
List<AnalyticsRecipesResponseModel>();

        using (DatabaseContext dB = new DatabaseContext())
        {
            var recipes = dB.Recipes.Where(r =>
analyticsSearchModel.PatientIdList.Contains(r.Patient_id ?? 0) &&
analyticsSearchModel.TagList.Any(x
=> r.TagNames.Contains(x))).OrderBy(r => r.Date_of_prescription).ToList();

            if (recipes != null)
            {
                foreach (var rec in recipes)
                {
                    analyticsRecipesResponseModel.Add(new
AnalyticsRecipesResponseModel(rec.Recipe_id, rec.Patient_id, rec.Date_of_prescription,
rec.Doctor.DPatronymic + " " + rec.Doctor.DName?[0] + ". " + rec.Doctor.DPatronymic?[0] +
".",
rec.Patient.Surname + " " + rec.Patient.Name?[0] + ". " + rec.Patient.Patronymic?[0] + ". ",
rec.Medicament, rec.Dosage_amount, rec.Signature, rec.Start_date, rec.End_date));
                }
            }

            return analyticsRecipesResponseModel;
        }
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return null;
    }
}

public List<AnalyticsChartValuesResponseModel>
GetAnalyticsChartValues(AnalyticsSearchModel analyticsSearchModel)
{
    try
    {
        var analyticsChartValuesResponseModelList = new
List<AnalyticsChartValuesResponseModel>();

        if (analyticsSearchModel.TestFieldName == "Glucose")
        {
            using (DatabaseContext dB = new DatabaseContext())
            {
                foreach (var item in analyticsSearchModel.PatientIdList)
                {

```

```

        var glucoseTests = dB.GlucoseTests.Where(t => t.Patient_id ==
item && t.GlDate >= analyticsSearchModel.DateFrom && t.GlDate <=
analyticsSearchModel.DateTo).ToList();

        if (glucoseTests.Count > 0)
        {
            var newAnalyticsChartValuesResponseModel = new
AnalyticsChartValuesResponseModel();
            newAnalyticsChartValuesResponseModel.Title =
glucoseTests.FirstOrDefault().Patient.Surname + " " +
glucoseTests.FirstOrDefault().Patient.Name?[0] + ". " +
glucoseTests.FirstOrDefault().Patient.Patronymic?[0] + ".";

            var chartPointsList = new List<ChartPoint>();

            foreach (var test in glucoseTests)
            {
                try
                {
                    chartPointsList.Add(new ChartPoint(test.GlDate ??
DateTime.Now, double.Parse(test.Result)));
                }
                catch { }
            }

            newAnalyticsChartValuesResponseModel.ChartPointList =
chartPointsList;
analyticsChartValuesResponseModelList.Add(newAnalyticsChartValuesResponseModel);
        }
    }
}
else
{
    using (DatabaseContext dB = new DatabaseContext())
    {
        foreach (var item in analyticsSearchModel.PatientIdList)
        {
            var urineTests = dB.UrineTests.Where(t => t.Patient_id == item
&& t.UrDate >= analyticsSearchModel.DateFrom && t.UrDate <=
analyticsSearchModel.DateTo).ToList();

            if (urineTests.Count > 0)
            {
                var newAnalyticsChartValuesResponseModel = new
AnalyticsChartValuesResponseModel();
                newAnalyticsChartValuesResponseModel.Title =
urineTests.FirstOrDefault().Patient.Surname + " " +
urineTests.FirstOrDefault().Patient.Name?[0] + ". " +
urineTests.FirstOrDefault().Patient.Patronymic?[0];

                var chartPointsList = new List<ChartPoint>();

                foreach (var test in urineTests)
                {
                    try
                    {
                        if (analyticsSearchModel.TestFieldName ==
"SpecificGravity")
                            chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.SpecificGravity)));
                        if (analyticsSearchModel.TestFieldName ==
"Protein")
                            chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.Protein)));
                        if (analyticsSearchModel.TestFieldName == "Sugar")
                            chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.Sugar)));

```

```

                                if (analyticsSearchModel.TestFieldName ==
"Erythrocytes")
                                    chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.Erythrocytes)));
                                if (analyticsSearchModel.TestFieldName ==
"Leukocytes")
                                    chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.Leukocytes)));
                                if (analyticsSearchModel.TestFieldName ==
"Bacteria")
                                    chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.Bacteria)));
                                if (analyticsSearchModel.TestFieldName ==
"FlatEpithelium")
                                    chartPointsList.Add(new ChartPoint(test.UrDate
?? DateTime.Now, double.Parse(test.FlatEpithelium)));
                                }
                                catch { }
                            }

                                newAnalyticsChartValuesResponseModel.ChartPointList =
chartPointsList;
analyticsChartValuesResponseModelList.Add(newAnalyticsChartValuesResponseModel);
                        }
                    }
                }

                return analyticsChartValuesResponseModelList;
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                return null;
            }
        }

        public DoctorDataResponseModel GetDoctorData(int doctorId)
        {
            try
            {
                using (DatabaseContext dB = new DatabaseContext())
                {
                    var doctor = dB.Doctors.Where(d => d.Doctor_id ==
doctorId).FirstOrDefault();

                    if (doctor != null)
                    {
                        return new DoctorDataResponseModel(doctor.Position_id ?? -1,
doctor.Position.Name + ": " + doctor.DName + " " + doctor.DPatronymic + " " +
doctor.DSurname);
                    }
                }

                return null;
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                return null;
            }
        }

        public List<PatientResponseModel> GetPatientList()
        {
            try
            {
                using (DatabaseContext dB = new DatabaseContext())

```

```

        {
            var patients = dB.Patients.OrderBy(p => p.Surname).ToList();

            if (patients != null)
            {
                var patientResponseModel = new List<PatientResponseModel>();

                foreach (var pat in patients)
                {
                    patientResponseModel.Add(new
PatientResponseModel(pat.Patient_id, pat.Surname + " " + pat.Name + " " + pat.Patronymic));
                }

                return patientResponseModel;
            }

            return null;
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return null;
        }
    }

    public List<RegionResponseModel> GetRegionList()
    {
        try
        {
            using (DatabaseContext dB = new DatabaseContext())
            {
                var regions = dB.Regions.OrderBy(r => r.Name).ToList();

                if (regions != null)
                {
                    var regionResponseModel = new List<RegionResponseModel>();

                    foreach (var reg in regions)
                    {
                        regionResponseModel.Add(new RegionResponseModel(reg.Region_id,
reg.Name));
                    }

                    return regionResponseModel;
                }

                return null;
            }
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return null;
        }
    }

    public List<DistrictResponseModel> GetDistrictList(int regionId)
    {
        try
        {
            using (DatabaseContext dB = new DatabaseContext())
            {
                var districts = dB.Districts.Where(d => d.Region_id ==
regionId).OrderBy(d => d.Name).ToList();

                if (districts != null)
                {
                    var districtResponseModel = new List<DistrictResponseModel>();

```

```

        foreach (var dis in districts)
        {
            districtResponseModel.Add(new
DistrictResponseModel(dis.District_id, dis.Name));
        }

        return districtResponseModel;
    }

    return null;
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public List<LocalityResponseModel> GetLocalityList(int districtId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var localities = dB.Localities.Where(l => l.District_id ==
districtId).OrderBy(l => l.Name).ToList();

            if (localities != null)
            {
                var localityResponseModel = new List<LocalityResponseModel>();

                foreach (var loc in localities)
                {
                    localityResponseModel.Add(new
LocalityResponseModel(loc.Locality_id, loc.Name));
                }

                return localityResponseModel;
            }
        }

        return null;
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return null;
    }
}

public DataTable GetPatientDataTable(int doctorId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            var sda = new SqlDataAdapter("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
"FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
"FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +
"dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
"FROM dbo.Declarations INNER JOIN dbo.Patients ON
dbo.Declarations.Patient_id = dbo.Patients.Patient_id " +

```

```

        "INNER JOIN dbo.Doctors ON dbo.Declarations.Doctor_id =
dbo.Doctors.Doctor_id " +
        "WHERE(dbo.Declarations.Doctor_id = '" + doctorId + "') ORDER BY
dbo.Patients.Surname", con);
        var dt = new DataTable();
        sda.Fill(dt);

        return dt;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public DataTable GetPatientDataTableSearch(int doctorId, string surname, string
name)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var con = new SqlConnection(dB.Database.Connection.ConnectionString);

            if (!String.IsNullOrEmpty(surname) &&
String.IsNullOrEmpty(name))
            {
                var cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
                "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
                "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +
                "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
                "FROM dbo.Declarations INNER JOIN dbo.Patients ON
dbo.Declarations.Patient_id = dbo.Patients.Patient_id " +
                "INNER JOIN dbo.Doctors ON dbo.Declarations.Doctor_id =
dbo.Doctors.Doctor_id " +
                "WHERE (dbo.Declarations.Doctor_id = '" + doctorId + "') " +
                "AND (dbo.Patients.Surname LIKE '" + @Surname + "%')", con);

                cmd.Parameters.AddWithValue("@Surname", surname);
                var sda = new SqlDataAdapter(cmd);

                var dt = new DataTable();
                sda.Fill(dt);

                if (dt.Rows.Count > 0)
                {
                    return dt;
                }
                else
                {
                    return null;
                }
            }

            if (String.IsNullOrEmpty(surname) &&
!String.IsNullOrEmpty(name))
            {
                SqlCommand cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
                "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
                "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +

```

```

        "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
        "FROM dbo.Declarations INNER JOIN dbo.Patients ON
dbo.Declarations.Patient_id = dbo.Patients.Patient_id " +
        "INNER JOIN dbo.Doctors ON dbo.Declarations.Doctor_id =
dbo.Doctors.Doctor_id " +
        "WHERE (dbo.Declarations.Doctor_id = '" + doctorId + "') AND
(dbo.Patients.Name LIKE '" + @Name + "%')", con);

        cmd.Parameters.AddWithValue("@Name", name);
        var sda = new SqlDataAdapter(cmd);

        var dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            return dt;
        }
        else
        {
            return null;
        }
    }

    if (!String.IsNullOrEmpty(surname) &&
!String.IsNullOrEmpty(name))
    {
        SqlCommand cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
        "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
        "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +
        "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
        "FROM dbo.Declarations INNER JOIN dbo.Patients ON
dbo.Declarations.Patient_id = dbo.Patients.Patient_id " +
        "INNER JOIN dbo.Doctors ON dbo.Declarations.Doctor_id =
dbo.Doctors.Doctor_id " +
        "WHERE (dbo.Declarations.Doctor_id = '" + doctorId + "')
AND (dbo.Patients.Surname LIKE '" + @Surname + "%') " +
        "AND (dbo.Patients.Name LIKE '" + @Name + "%')", con);

        cmd.Parameters.AddWithValue("@Surname", surname);
        cmd.Parameters.AddWithValue("@Name", name);
        var sda = new SqlDataAdapter(cmd);

        var dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            return dt;
        }
        else
        {
            return null;
        }
    }

    if (String.IsNullOrEmpty(surname) &&
String.IsNullOrEmpty(name))
    {
        return GetPatientDataTable(doctorId);
    }

    return null;
}

```

```

    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return null;
    }
}

public DataTable GetPatientDataTableSearchAll(int doctorId, string surname, string
name)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var con = new SqlConnection(dB.Database.Connection.ConnectionString);

            if (!String.IsNullOrEmpty(surname) &&
String.IsNullOrEmpty(name))
            {
                SqlCommand cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
                "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
                "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +
                "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
                "FROM dbo.Declarations INNER JOIN dbo.Doctors ON
dbo.Declarations.Doctor_id = dbo.Doctors.Doctor_id " +
                "INNER JOIN dbo.Patients ON dbo.Declarations.Patient_id =
dbo.Patients.Patient_id " +
                "WHERE (dbo.Patients.Surname LIKE ' ' + @Surname + '%')", con);

                cmd.Parameters.AddWithValue("@Surname", surname);
                var sda = new SqlDataAdapter(cmd);

                var dt = new DataTable();
                sda.Fill(dt);

                if (dt.Rows.Count > 0)
                {
                    return dt;
                }
                else
                {
                    return null;
                }
            }

            if (String.IsNullOrEmpty(surname) &&
!String.IsNullOrEmpty(name))
            {
                SqlCommand cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
                "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
Date_of_Birth, dbo.Patients.Phone_number, " +
                "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
Conclusion_date, " +
                "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
dbo.Doctors.DPatronymic AS Doctor " +
                "FROM dbo.Declarations INNER JOIN dbo.Doctors ON
dbo.Declarations.Doctor_id = dbo.Doctors.Doctor_id " +
                "INNER JOIN dbo.Patients ON dbo.Declarations.Patient_id =
dbo.Patients.Patient_id " +
                "WHERE (dbo.Patients.Name LIKE ' ' + @Name + '%')", con);

                cmd.Parameters.AddWithValue("@Name", name);
                var sda = new SqlDataAdapter(cmd);

```

```

        var dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            return dt;
        }
        else
        {
            return null;
        }
    }

    if (!String.IsNullOrEmpty(surname) &&
        !String.IsNullOrEmpty(name))
    {
        SqlCommand cmd = new SqlCommand("SELECT dbo.Patients.Patient_id,
        dbo.Patients.Surname, dbo.Patients.Name, dbo.Patients.Patronymic, " +
            "FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-yyyy') as
        Date_of_Birth, dbo.Patients.Phone_number, " +
            "FORMAT (dbo.Declarations.Conclusion_date, 'dd-MM-yyyy') as
        Conclusion_date, " +
            "dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
        dbo.Doctors.DPatronymic AS Doctor " +
            "FROM dbo.Declarations INNER JOIN dbo.Doctors ON
        dbo.Declarations.Doctor_id = dbo.Doctors.Doctor_id " +
            "INNER JOIN dbo.Patients ON dbo.Declarations.Patient_id =
        dbo.Patients.Patient_id " +
            "WHERE (dbo.Patients.Surname LIKE ' ' + @Surname + '%') " +
            "AND (dbo.Patients.Name LIKE ' ' + @Name + '%')", con);

        cmd.Parameters.AddWithValue("@Surname", surname);
        cmd.Parameters.AddWithValue("@Name", name);
        var sda = new SqlDataAdapter(cmd);

        var dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            return dt;
        }
        else
        {
            return null;
        }
    }

    return null;
}
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public bool GetResaveDeclaration(int doctorId, int patientId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            string query = "update Declarations set Doctor_id = @Doctor_id where
        Patient_id = @id";

```

```

        SqlCommand cmd = new SqlCommand(query, con);
        cmd.Parameters.AddWithValue("@Doctor_id", doctorId);
        cmd.Parameters.AddWithValue("@id", patientId);

        con.Open();
        cmd.ExecuteNonQuery();
        con.Close();

        return true;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return false;
}
}

public DeclarationResponseModel GetDeclaration(int patientId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            var sda = new SqlDataAdapter("SELECT dbo.Patients.Surname,
dbo.Patients.Name, dbo.Patients.Patronymic, " +
            "dbo.Patients.Sex, FORMAT (dbo.Patients.Date_of_Birth, 'dd-MM-
yyyy') as Date_of_Birth, " +
            "dbo.Patients.Place_of_birth + ', ' + dbo.Patients.Country_of_birth
AS C_and_P_of_birth, " +
            "dbo.Patients.ITN, dbo.Patients.Phone_number, dbo.Patients.Email,
dbo.Doctors.DSurname, dbo.Doctors.DName, " +
            "dbo.Doctors.DPatronymic, dbo.Positions.Name, dbo.Doctors.Unit,
dbo.Doctors.DPhone_number, " +
            "dbo.Doctors.DEmail, FORMAT (dbo.Declarations.Conclusion_date, 'dd-
MM-yyyy') as Conclusion_date, dbo.Declarations.Declaration_id " +
            "FROM dbo.Declarations INNER JOIN dbo.Doctors ON
dbo.Declarations.Doctor_id = dbo.Doctors.Doctor_id " +
            "INNER JOIN dbo.Patients ON dbo.Declarations.Patient_id =
dbo.Patients.Patient_id " +
            "INNER JOIN dbo.Positions ON dbo.Doctors.Position_id =
dbo.Positions.Position_id " +
            "WHERE(dbo.Patients.Patient_id = '" + patientId + "')", con);

            DataTable dt3 = new DataTable();
            sda.Fill(dt3);

            sda = new SqlDataAdapter("SELECT dbo.Passports.Series_and_number + ', ' +
+ dbo.Passports.Issuing_authority + N', вид ' + FORMAT(dbo.Passports.Date_of_issue, 'dd-MM-
yyyy') AS Expr1" +
            " FROM dbo.Passports " +
            "INNER JOIN dbo.Patients ON dbo.Passports.Passport_id =
dbo.Patients.Passport_id " +
            "WHERE(dbo.Patients.Patient_id = '" + patientId + "')", con);

            DataTable dt1 = new DataTable();
            sda.Fill(dt1);

            sda = new SqlDataAdapter("SELECT dbo.Localities.Name + ', ' +
dbo.Districts.Name + N' район, ' + " +
            "dbo.Regions.Name + N' обл., в. ' + dbo.PlaceOfResidences.Street +
N', буд. ' + dbo.PlaceOfResidences.House AS Place, " +
            "dbo.PlaceOfResidences.Flat FROM dbo.Districts INNER JOIN
dbo.Localities ON dbo.Districts.District_id = dbo.Localities.District_id " +
            "INNER JOIN dbo.PlaceOfResidences ON dbo.Localities.Locality_id =
dbo.PlaceOfResidences.Locality_id INNER JOIN dbo.Patients " +
            "ON dbo.PlaceOfResidences.Place_id = dbo.Patients.Place_id INNER
JOIN dbo.Regions ON dbo.Districts.Region_id = dbo.Regions.Region_id " +

```

```

        "WHERE(dbo.Patients.Patient_id = '" + patientId + "')", con);

        DataTable dt2 = new DataTable();
        sda.Fill(dt2);

        var declarationResponseModel = new
DeclarationResponseModel(dt3.Rows[0][0].ToString(), dt3.Rows[0][1].ToString(),
dt3.Rows[0][2].ToString(),
        (dt3.Rows[0][3].ToString() == "0" ? "чоловіча" : "жіноча"),
dt3.Rows[0][4].ToString(), dt3.Rows[0][5].ToString(), dt3.Rows[0][6].ToString(),
        dt3.Rows[0][7].ToString(), dt3.Rows[0][8].ToString(),
dt3.Rows[0][9].ToString(), dt3.Rows[0][10].ToString(), dt3.Rows[0][11].ToString(),
        dt3.Rows[0][14].ToString(), dt3.Rows[0][15].ToString(),
dt3.Rows[0][12].ToString(), dt3.Rows[0][13].ToString(), dt3.Rows[0][16].ToString(),
        dt3.Rows[0][17].ToString(), dt1.Rows[0][0].ToString(),
(dt2.Rows[0][1].ToString() != "" ? dt2.Rows[0][0].ToString() + ", кв. " +
dt2.Rows[0][1].ToString() : dt2.Rows[0][0].ToString()));

        return declarationResponseModel;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public RecordsDataResponseModel GetRecordsData(int doctorId, int patientId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var recordsDataResponseModel = new RecordsDataResponseModel();

            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            var sda = new SqlDataAdapter("select FORMAT (Date_of_Birth, 'dd-MM-
yyyy') as Date_of_Birth, Phone_number " +
            "from Patients where Patient_id = '" + patientId + "'", con);
            var dt = new DataTable();
            sda.Fill(dt);

            recordsDataResponseModel.DateOfBirth = "Дата народження: " +
dt.Rows[0][0].ToString();
            recordsDataResponseModel.PhoneNumber = "Номер телефону: " +
dt.Rows[0][1].ToString();

            sda = new SqlDataAdapter("SELECT dbo.Localities.Name + ', ' +
dbo.Districts.Name + N' район, ' + " +
            "dbo.Regions.Name + N' обл., в. ' + dbo.PlaceOfResidences.Street +
N', буд. ' + dbo.PlaceOfResidences.House AS Place, " +
            "dbo.PlaceOfResidences.Flat FROM dbo.Districts INNER JOIN
dbo.Localities ON dbo.Districts.District_id = dbo.Localities.District_id " +
            "INNER JOIN dbo.PlaceOfResidences ON dbo.Localities.Locality_id =
dbo.PlaceOfResidences.Locality_id INNER JOIN dbo.Patients " +
            "ON dbo.PlaceOfResidences.Place_id = dbo.Patients.Place_id INNER
JOIN dbo.Regions ON dbo.Districts.Region_id = dbo.Regions.Region_id " +
            "WHERE(dbo.Patients.Patient_id = '" + patientId + "')", con);

            dt = new DataTable();
            sda.Fill(dt);

            if (dt.Rows[0][1].ToString() != "")
                recordsDataResponseModel.PlaceOfResidence =
dt.Rows[0][0].ToString() + ", кв. " + dt.Rows[0][1].ToString();
            else
                recordsDataResponseModel.PlaceOfResidence =
dt.Rows[0][0].ToString();
        }
    }
}

```

```

        sda = new SqlDataAdapter("select Declaration_id from Declarations where
Patient_id = '" + patientId + "'" +
        "and Doctor_id = '" + doctorId + "'", con);
        dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            recordsDataResponseModel.DeclarationInfo = "Декларація укладена";
        }
        else
        {
            recordsDataResponseModel.DeclarationInfo = "Декларація не
укладена";
        }

        sda = new SqlDataAdapter("select Record_id, Patient_id, Doctor_id,
FORMAT (RDate, 'dd-MM-yyyy') as RDate, " +
        "Complaints, Diagnosis, Treatment from Records where Patient_id =
'" + patientId + "'", con);
        dt = new DataTable();
        sda.Fill(dt);

        recordsDataResponseModel.RecordsDataTable = dt;

        return recordsDataResponseModel;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public SickLeavesDataResponseModel GetSickLeavesData(int doctorId, int patientId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var sickLeavesDataResponseModel = new SickLeavesDataResponseModel();

            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            var sda = new SqlDataAdapter("select FORMAT (Date_of_Birth, 'dd-MM-
yyyy') as Date_of_Birth, Phone_number " +
            "from Patients where Patient_id = '" + patientId + "'", con);
            var dt = new DataTable();
            sda.Fill(dt);

            sickLeavesDataResponseModel.DateOfBirth = "Дата народження: " +
            dt.Rows[0][0].ToString();
            sickLeavesDataResponseModel.PhoneNumber = "Номер телефону: " +
            dt.Rows[0][1].ToString();

            sda = new SqlDataAdapter("SELECT dbo.Localities.Name + ', ' +
            dbo.Districts.Name + N' район, ' + " +
            "dbo.Regions.Name + N' обл., в. ' + dbo.PlaceOfResidences.Street +
            N', буд. ' + dbo.PlaceOfResidences.House AS Place, " +
            "dbo.PlaceOfResidences.Flat FROM dbo.Districts INNER JOIN
            dbo.Localities ON dbo.Districts.District_id = dbo.Localities.District_id " +
            "INNER JOIN dbo.PlaceOfResidences ON dbo.Localities.Locality_id =
            dbo.PlaceOfResidences.Locality_id INNER JOIN dbo.Patients " +
            "ON dbo.PlaceOfResidences.Place_id = dbo.Patients.Place_id INNER
            JOIN dbo.Regions ON dbo.Districts.Region_id = dbo.Regions.Region_id " +
            "WHERE(dbo.Patients.Patient_id = '" + patientId + "'", con);

            dt = new DataTable();

```

```

        sda.Fill(dt);

        if (dt.Rows[0][1].ToString() != "")
            sickLeavesDataResponseModel.PlaceOfResidence =
dt.Rows[0][0].ToString() + ", кв. " + dt.Rows[0][1].ToString();
        else
            sickLeavesDataResponseModel.PlaceOfResidence =
dt.Rows[0][0].ToString();

        sda = new SqlDataAdapter("select Declaration_id from Declarations where
Patient_id = '" + patientId + "'" +
            "and Doctor_id = '" + doctorId + "'", con);
        dt = new DataTable();
        sda.Fill(dt);

        if (dt.Rows.Count > 0)
        {
            sickLeavesDataResponseModel.DeclarationInfo = "Декларація
укладена";
        }
        else
        {
            sickLeavesDataResponseModel.DeclarationInfo = "Декларація не
укладена";
        }

        sda = new SqlDataAdapter("SELECT Sick_leave_id, Patient_id, Doctor_id,
FORMAT(SDate, 'dd-MM-yyyy') AS SDate, " +
            "Regimen, SDiagnosis, FORMAT(SFrom, 'dd-MM-yyyy') AS SFrom,
FORMAT(STo, 'dd-MM-yyyy') AS STo, " +
            "FORMAT(Get_to_work, 'dd-MM-yyyy') AS Get_to_work FROM
dbo.SickLeaves " +
            "WHERE(Patient_id = '" + patientId + "'", con);
        dt = new DataTable();
        sda.Fill(dt);

        sickLeavesDataResponseModel.SickLeavesDataTable = dt;

        return sickLeavesDataResponseModel;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public PatientAllDataResponseModel GetPatientAllData(int patientId)
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var patientAllDataResponseModel = new PatientAllDataResponseModel();

            var con = new SqlConnection(dB.Database.Connection.ConnectionString);
            var sda = new SqlDataAdapter("SELECT dbo.Patients.Surname + ' ' +
dbo.Patients.Name + ' ' + dbo.Patients.Patronymic + N', " +
            "лікп: ' + dbo.Doctors.DSurname + ' ' + dbo.Doctors.DName + ' ' +
            dbo.Doctors.DPatronymic AS Expr1 " +
            "FROM dbo.Doctors INNER JOIN dbo.Declarations ON dbo.Doctors.Doctor_id
= dbo.Declarations.Doctor_id " +
            "INNER JOIN dbo.Patients ON dbo.Declarations.Patient_id =
            dbo.Patients.Patient_id " +
            "WHERE(dbo.Patients.Patient_id) = '" + patientId + "'", con);
            var dt = new DataTable();
            sda.Fill(dt);

```

```

        patientAllDataResponseModel.PName = dt.Rows[0][0].ToString();

        sda = new SqlDataAdapter("SELECT dbo.Doctors.DSurname + ' ' +
        dbo.Doctors.DName + ' ' + dbo.Doctors.DPatronymic + " +
        "' (' + dbo.Positions.Name + ')" AS Doctor, FORMAT (RDate, 'dd-MM-
        yyyy') as RDate, dbo.Records.Complaints, " +
        "dbo.Records.Diagnosis, dbo.Records.Treatment " +
        "FROM dbo.Doctors INNER JOIN dbo.Positions ON
        dbo.Doctors.Position_id = dbo.Positions.Position_id " +
        "INNER JOIN dbo.Records ON dbo.Doctors.Doctor_id =
        dbo.Records.Doctor_id " +
        "WHERE(dbo.Records.Patient_id = '" + patientId + "')" , con);
        dt = new DataTable();
        sda.Fill(dt);

        patientAllDataResponseModel.RecordsTable = dt;

        sda = new SqlDataAdapter("SELECT dbo.Doctors.DSurname + ' ' +
        dbo.Doctors.DName + ' ' + dbo.Doctors.DPatronymic + " +
        "' (' + dbo.Positions.Name + ')" AS Doctor, dbo.Recipes.Medicament,
        dbo.Recipes.Dosage_amount, dbo.Recipes.Signature, " +
        "FORMAT(dbo.Recipes.Start_date, 'dd-MM-yyyy') AS Start_date,
        FORMAT(dbo.Recipes.End_date, 'dd-MM-yyyy') AS End_date " +
        "FROM dbo.Recipes INNER JOIN dbo.Doctors ON dbo.Recipes.Doctor_id =
        dbo.Doctors.Doctor_id " +
        "INNER JOIN dbo.Positions ON dbo.Doctors.Position_id =
        dbo.Positions.Position_id " +
        "WHERE(dbo.Recipes.Patient_id = '" + patientId + "')" , con);
        dt = new DataTable();
        sda.Fill(dt);

        patientAllDataResponseModel.RecipesTable = dt;

        sda = new SqlDataAdapter("SELECT FORMAT(GlDate, 'dd-MM-yyyy') AS
        GlDate, Result FROM dbo.GlucoseTests " +
        "WHERE(Patient_id = '" + patientId + "')" , con);
        dt = new DataTable();
        sda.Fill(dt);

        patientAllDataResponseModel.GlucoseTestsTable = dt;

        sda = new SqlDataAdapter("SELECT FORMAT(UrDate, 'dd-MM-yyyy') AS
        UrDate, Color, Reaction, SpecificGravity, Protein, " +
        "Sugar, Erythrocytes, Leukocytes, Bacteria, FlatEpithelium FROM
        dbo.UrineTests " +
        "WHERE(Patient_id = '" + patientId + "')" , con);
        dt = new DataTable();
        sda.Fill(dt);

        patientAllDataResponseModel.UrineTestsTable = dt;

        sda = new SqlDataAdapter("SELECT dbo.Doctors.DSurname + ' ' +
        dbo.Doctors.DName + ' ' + dbo.Doctors.DPatronymic + " +
        "' (' + dbo.Positions.Name + ')" AS Doctor,
        FORMAT(dbo.SickLeaves.SDate, 'dd-MM-yyyy') AS SDate, " +
        "dbo.SickLeaves.Regimen, dbo.SickLeaves.SDiagnosis,
        FORMAT(dbo.SickLeaves.SFrom, 'dd-MM-yyyy') AS SFrom, " +
        "FORMAT(dbo.SickLeaves.STo, 'dd-MM-yyyy') AS STo, " +
        "FORMAT(dbo.SickLeaves.Get_to_work, 'dd-MM-yyyy') AS Get_to_work
        FROM dbo.SickLeaves " +
        "INNER JOIN dbo.Doctors ON dbo.SickLeaves.Doctor_id =
        dbo.Doctors.Doctor_id " +
        "INNER JOIN dbo.Positions ON dbo.Doctors.Position_id =
        dbo.Positions.Position_id " +
        "WHERE(dbo.SickLeaves.Patient_id = '" + patientId + "')" , con);
        dt = new DataTable();
        sda.Fill(dt);

        patientAllDataResponseModel.SickLeavesTable = dt;

```

```

        return patientAllDataResponseModel;
    }
}
catch (Exception ex)
{
    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    return null;
}
}

public List<string> GetTags()
{
    try
    {
        using (DatabaseContext dB = new DatabaseContext())
        {
            var tagList = dB.Tags.OrderBy(t => t.TagName).ToList();

            if (tagList != null)
            {
                List<string> tags = new List<string>();

                foreach (var item in tagList)
                {
                    tags.Add(item.TagName);
                }

                return tags;
            }

            return null;
        }
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return null;
    }
}
}
}
}

```

Б.12 Текст файла CoreLauncher.cs

```

namespace ProHealthCore
{
    internal class CoreLauncher
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

        static void Main(string[] args)
        {
            try
            {
                var configuration = new HostConfiguration()
                {
                    UrlReservations = new UrlReservations() { CreateAutomatically =
true }
                };

                var host = new NancyHost(configuration, new
Uri("http://localhost:8001"));
                host.Start();
            }
            catch (Exception ex)
            {
                Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            }
        }
    }
}

```

```

        Logger.Info("Host started");

        while ("exit" != Console.ReadLine())
        {
        }

        host.Stop();

        Logger.Info("Host stoped");
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
    }
}
}
}
}

```

Б.13 Текст файла MainModule.cs

```

namespace ProHealthCore
{
    public class MainModule : NancyModule
    {
        private static readonly Logger Logger = LogManager.GetCurrentClassLogger();
        private static readonly Core Core = new Core();

        public MainModule()
        {
            Post("/authorize", x =>
            {
                try
                {
                    Logger.Info("Received authorize ...");
                    var loginSendModel = this.Bind<LoginSendModel>();
                    Logger.Info("SendModel: " +
                        JsonConvert.SerializeObject(loginSendModel));

                    var loginResponseModel = Core.Authorize(loginSendModel);
                    Logger.Info("ResponseModel: " +
                        JsonConvert.SerializeObject(loginResponseModel));

                    return JsonConvert.SerializeObject(loginResponseModel);
                }
                catch (Exception ex)
                {
                    Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
                    return HttpStatusCode.BadRequest;
                }
            });

            Post("/saveGlTest", x =>
            {
                try
                {
                    Logger.Info("Received saveGlTest ...");
                    var glucoseTestSendModel = this.Bind<GlucoseTestSendModel>();
                    Logger.Info("SendModel: " +
                        JsonConvert.SerializeObject(glucoseTestSendModel));

                    var isSuccess = Core.SaveGlTest(glucoseTestSendModel);
                    Logger.Info("ResponseModel: " +
                        JsonConvert.SerializeObject(isSuccess));
                }
            });
        }
    }
}

```

```

        return JsonConvert.SerializeObject(isSuccess);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
    });

    Post("/saveUrTest", x =>
    {
        try
        {
            Logger.Info("Received saveUrTest ...");
            var urineTestSendModel = this.Bind<UrineTestSendModel>();
            Logger.Info("SendModel: " +
JsonConvert.SerializeObject(urineTestSendModel));

            var isSuccess = Core.SaveUrTest(urineTestSendModel);
            Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

    Post("/saveRecipe", x =>
    {
        try
        {
            Logger.Info("Received saveRecipe ...");
            var recipeSendModel = this.Bind<RecipeSendModel>();
            Logger.Info("SendModel: " +
JsonConvert.SerializeObject(recipeSendModel));

            var isSuccess = Core.SaveRecipe(recipeSendModel);
            Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

    Post("/saveDeclaration", x =>
    {
        try
        {
            Logger.Info("Received saveDeclaration ...");
            var declarationSendModel = this.Bind<DeclarationSendModel>();
            Logger.Info("SendModel: " +
JsonConvert.SerializeObject(declarationSendModel));

            var isSuccess = Core.SaveDeclaration(declarationSendModel);
            Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)

```

```

        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

    Post("/saveRecords", x =>
    {
        try
        {
            Logger.Info("Received saveRecords ...");
            var recordsSendModel = this.Bind<List<RecordsSendModel>>();
            Logger.Info("SendModel: " +
                JsonConvert.SerializeObject(recordsSendModel));

            var isSuccess = Core.SaveRecords(recordsSendModel);
            Logger.Info("ResponseModel: " +
                JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

    Post("/saveSickLeaves", x =>
    {
        try
        {
            Logger.Info("Received saveSickLeaves ...");
            var sickLeavesSendModel = this.Bind<List<SickLeavesSendModel>>();
            Logger.Info("SendModel: " +
                JsonConvert.SerializeObject(sickLeavesSendModel));

            var isSuccess = Core.SaveSickLeaves(sickLeavesSendModel);
            Logger.Info("ResponseModel: " +
                JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

    Post("/addTag", x =>
    {
        try
        {
            Logger.Info("Received addTag ...");
            var tagSendModel = this.Bind<TagSendModel>();
            Logger.Info("SendModel: " + JsonConvert.SerializeObject(tagSendModel));

            var isSuccess = Core.AddTag(tagSendModel.TagName);
            Logger.Info("ResponseModel: " +
                JsonConvert.SerializeObject(isSuccess));

            return JsonConvert.SerializeObject(isSuccess);
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
            return HttpStatusCode.BadRequest;
        }
    });

```

```

});

Post("/getAnalyticsRecords", x =>
{
    try
    {
        Logger.Info("Received getAnalyticsRecords ...");
        var analyticsSearchModel = this.Bind<AnalyticsSearchModel>();
        Logger.Info("SendModel: " +
JsonConvert.SerializeObject(analyticsSearchModel));

        var analyticsRecordsResponseModel =
Core.GetAnalyticsRecords(analyticsSearchModel);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(analyticsRecordsResponseModel));

        return JsonConvert.SerializeObject(analyticsRecordsResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Post("/getAnalyticsRecipes", x =>
{
    try
    {
        Logger.Info("Received getAnalyticsRecipes ...");
        var analyticsSearchModel = this.Bind<AnalyticsSearchModel>();
        Logger.Info("SendModel: " +
JsonConvert.SerializeObject(analyticsSearchModel));

        var analyticsRecipesResponseModel =
Core.GetAnalyticsRecipes(analyticsSearchModel);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(analyticsRecipesResponseModel));

        return JsonConvert.SerializeObject(analyticsRecipesResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Post("/getAnalyticsChartValues", x =>
{
    try
    {
        Logger.Info("Received getAnalyticsChartValues ...");
        var analyticsSearchModel = this.Bind<AnalyticsSearchModel>();
        Logger.Info("SendModel: " +
JsonConvert.SerializeObject(analyticsSearchModel));

        var analyticsChartValuesResponseModel =
Core.GetAnalyticsChartValues(analyticsSearchModel);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(analyticsChartValuesResponseModel));

        return JsonConvert.SerializeObject(analyticsChartValuesResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

```

```

});

Get("/getDoctorData/{doctorId}", x =>
{
    try
    {
        Logger.Info("Received getDoctorData ...");
        int doctorId = x.doctorId;
        Logger.Info("SendData: doctorId: " + doctorId);

        var doctorDataResponseModel = Core.GetDoctorData(doctorId);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(doctorDataResponseModel));

        return JsonConvert.SerializeObject(doctorDataResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getPatientList", x =>
{
    try
    {
        Logger.Info("Received getPatientList ...");

        List<PatientResponseModel> patientList = Core.GetPatientList();
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(patientList));

        return JsonConvert.SerializeObject(patientList);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getRegionList", x =>
{
    try
    {
        Logger.Info("Received getRegionList ...");

        List<RegionResponseModel> regionList = Core.GetRegionList();
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(regionList));

        return JsonConvert.SerializeObject(regionList);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getDistrictList/{regionId}", x =>
{
    try
    {
        Logger.Info("Received getDistrictList ...");
        int regionId = x.regionId;
        Logger.Info("SendData: regionId: " + regionId);

```

```

        List<DistrictResponseModel> districtList =
Core.GetDistrictList(regionId);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(districtList));

        return JsonConvert.SerializeObject(districtList);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getLocalityList/{districtId}", x =>
{
    try
    {
        Logger.Info("Received getLocalityList ...");
        int districtId = x.districtId;
        Logger.Info("SendData: districtId: " + districtId);

        List<LocalityResponseModel> localityList =
Core.GetLocalityList(districtId);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(localityList));

        return JsonConvert.SerializeObject(localityList);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getPatientDataTable/{doctorId}", x =>
{
    try
    {
        Logger.Info("Received getPatientDataTable ...");
        int doctorId = x.doctorId;
        Logger.Info("SendData: doctorId: " + doctorId);

        DataTable patientDataTable = Core.GetPatientDataTable(doctorId);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(patientDataTable));

        return JsonConvert.SerializeObject(patientDataTable);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getPatientDataTableSearch/{surname}/{name}/{doctorId}", x =>
{
    try
    {
        Logger.Info("Received getPatientDataTableSearch ...");
        int doctorId = x.doctorId;
        string surname = x.surname;
        string name = x.name;
        Logger.Info("SendData: doctorId: " + doctorId + ", name:" + name + ",
surname:" + surname);

```

```

        DataTable patientDataTableSearch =
Core.GetPatientDataTableSearch(doctorId, surname, name);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(patientDataTableSearch));

        return JsonConvert.SerializeObject(patientDataTableSearch);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getPatientDataTableSearchAll/{surname}/{name}/{doctorId}", x =>
{
    try
    {
        Logger.Info("Received getPatientDataTableSearchAll ...");
        int doctorId = x.doctorId;
        string surname = x.surname;
        string name = x.name;
        Logger.Info("SendData: doctorId: " + doctorId + ", name:" + name + ",
surname:" + surname);

        DataTable patientDataTableSearchAll =
Core.GetPatientDataTableSearchAll(doctorId, surname, name);
        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(patientDataTableSearchAll));

        return JsonConvert.SerializeObject(patientDataTableSearchAll);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getResaveDeclaration/{doctorId}/{patientId}", x =>
{
    try
    {
        Logger.Info("Received getResaveDeclaration ...");
        int doctorId = x.doctorId;
        int patientId = x.patientId;
        Logger.Info("SendData: doctorId: " + doctorId + ", patientId:" +
patientId);

        bool isSuccess = Core.GetResaveDeclaration(doctorId, patientId);
        Logger.Info("ResponseModel: " + isSuccess);

        return JsonConvert.SerializeObject(isSuccess);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getDeclaration/{patientId}", x =>
{
    try
    {
        Logger.Info("Received getDeclaration ...");
        int patientId = x.patientId;
        Logger.Info("SendData: patientId:" + patientId);

```

```

        DeclarationResponseModel declarationResponseModel =
Core.GetDeclaration(patientId);

        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(declarationResponseModel));

        return JsonConvert.SerializeObject(declarationResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getRecordsData/{doctorId}/{patientId}", x =>
{
    try
    {
        Logger.Info("Received getRecordsData ...");
        int doctorId = x.doctorId;
        int patientId = x.patientId;
        Logger.Info("SendData: doctorId: " + doctorId + ", patientId:" +
patientId);

        RecordsDataResponseModel recordsDataResponseModel =
Core.GetRecordsData(doctorId, patientId);

        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(recordsDataResponseModel));

        return JsonConvert.SerializeObject(recordsDataResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getSickLeavesData/{doctorId}/{patientId}", x =>
{
    try
    {
        Logger.Info("Received getSickLeavesData ...");
        int doctorId = x.doctorId;
        int patientId = x.patientId;
        Logger.Info("SendData: doctorId: " + doctorId + ", patientId:" +
patientId);

        SickLeavesDataResponseModel sickLeavesDataResponseModel =
Core.GetSickLeavesData(doctorId, patientId);

        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(sickLeavesDataResponseModel));

        return JsonConvert.SerializeObject(sickLeavesDataResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getPatientAllData/{patientId}", x =>
{
    try
    {

```

```

        Logger.Info("Received getPatientAllData ...");
        int patientId = x.patientId;
        Logger.Info("SendData: patientId:" + patientId);

        PatientAllDataResponseModel patientAllDataResponseModel =
Core.GetPatientAllData(patientId);

        Logger.Info("ResponseModel: " +
JsonConvert.SerializeObject(patientAllDataResponseModel));

        return JsonConvert.SerializeObject(patientAllDataResponseModel);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});

Get("/getTags", x =>
{
    try
    {
        Logger.Info("Received getTags ...");

        List<string> tags = Core.GetTags();
        Logger.Info("ResponseModel: " + JsonConvert.SerializeObject(tags));

        return JsonConvert.SerializeObject(tags);
    }
    catch (Exception ex)
    {
        Logger.Error(ex.Message + "StackTrace: " + ex.StackTrace);
        return HttpStatusCode.BadRequest;
    }
});
}
}
}
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

кафедра програмних засобів

Дипломна робота бакалавра
на тему

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВЕДЕННЯ І АНАЛІЗУ МЕДИЧНИХ
КАРТОК ПАЦІЄНТІВ**

Виконав:
студент групи кнт-228

Лактіонов В. В.

Керівник:
к.т.н., доцент

Льовкін В. М.

2022 р.

Рисунок В.1 – Слайд №1**Мета, об'єкт та предмет дослідження**

- **Мета роботи** – розробка програмного забезпечення ведення і аналізу медичних карток пацієнтів для забезпечення інформаційної підтримки прийняття рішень лікарем.
- **Об'єкт дослідження** – процес ведення і аналізу медичних карток пацієнтів.
- **Предмет дослідження** – програмне забезпечення ведення і аналізу медичних карток пацієнтів для медичних закладів.

Рисунок В.2 – Слайд №2

Завдання до роботи

- Визначення вимог до програмного забезпечення
- Проектування програмного забезпечення
- Реалізація і налагодження програмного забезпечення
- Тестування програмного забезпечення

3

Рисунок В.3 – Слайд №3



4

Рисунок В.4 – Слайд №4

Вибір мови програмування

Критерії порівняння	Мови програмування		
	C#	C++	Python
Підтримка об'єктно-орієнтованої парадигми програмування	+	+	+
Зручність використання для створення Windows-застосунків	++	+	-
Зручність використання засобів розробки	++	+	+
Сприйняття коду	++	+	+
Стандартні бібліотеки шаблонів	++	+	+
Кросплатформність	+	+	+
Продуктивність роботи	++	+	+

Рисунок В.5 – Слайд №5

Структура бази даних

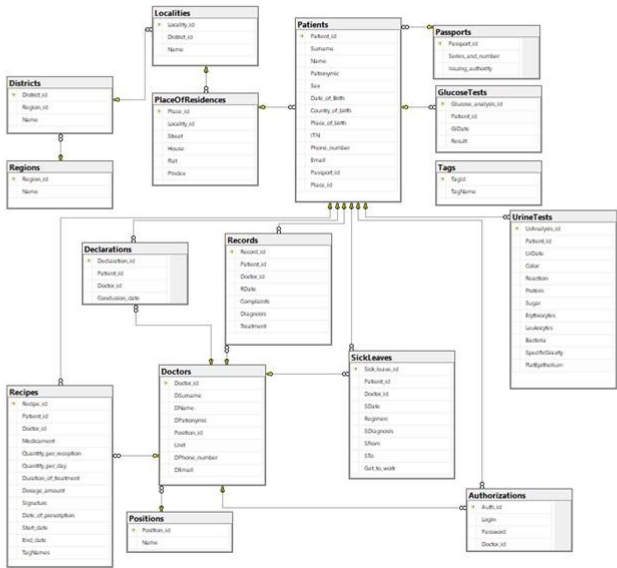
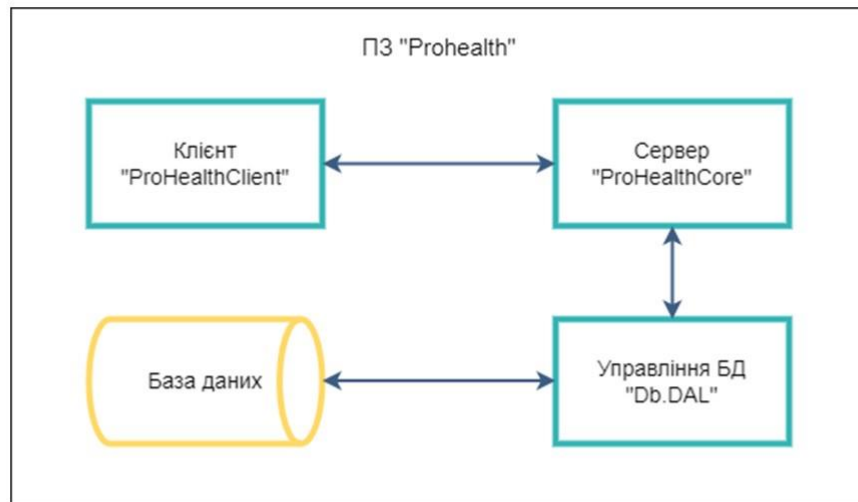


Рисунок В.6 – Слайд №6

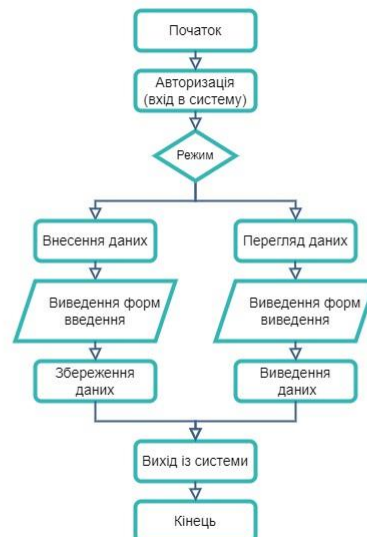
Структурна схема



7

Рисунок В.7 – Слайд №7

Функціональна схема



8

Рисунок В.8 – Слайд №8

Діаграми послідовності

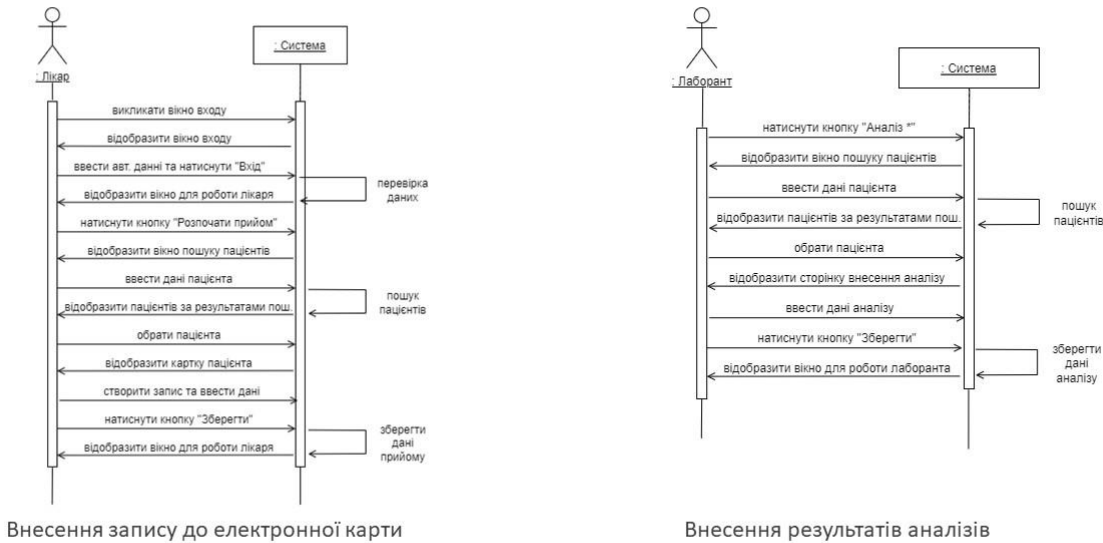


Рисунок В.9 – Слайд №9

Структура проекту

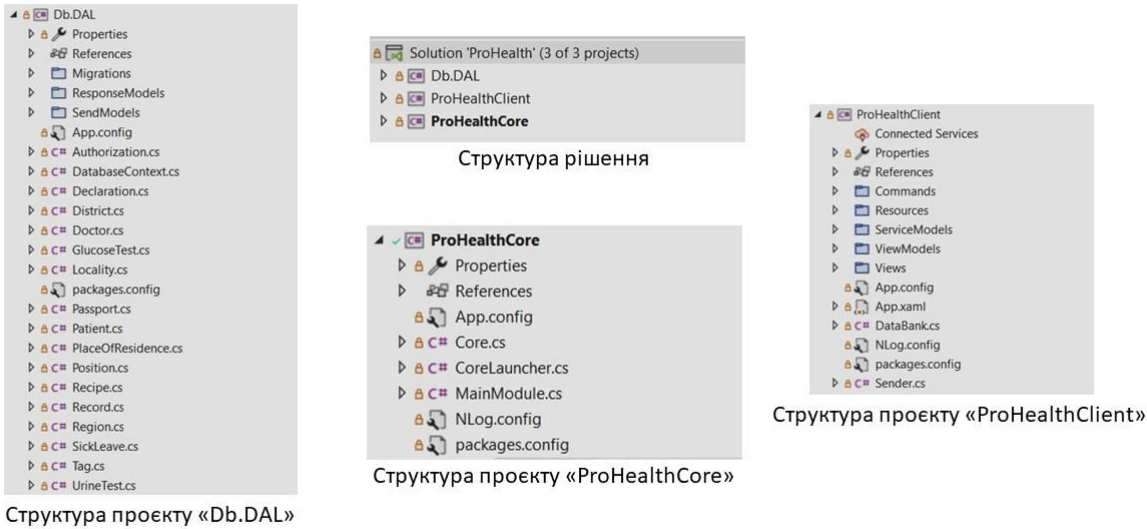
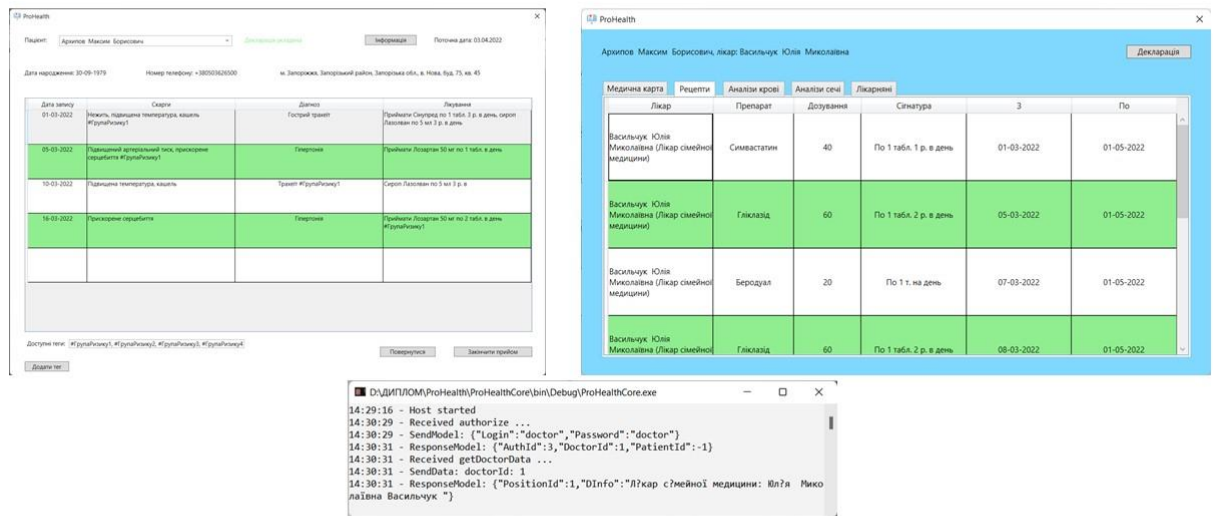


Рисунок В.10 – Слайд №10

Розроблене програмне забезпечення

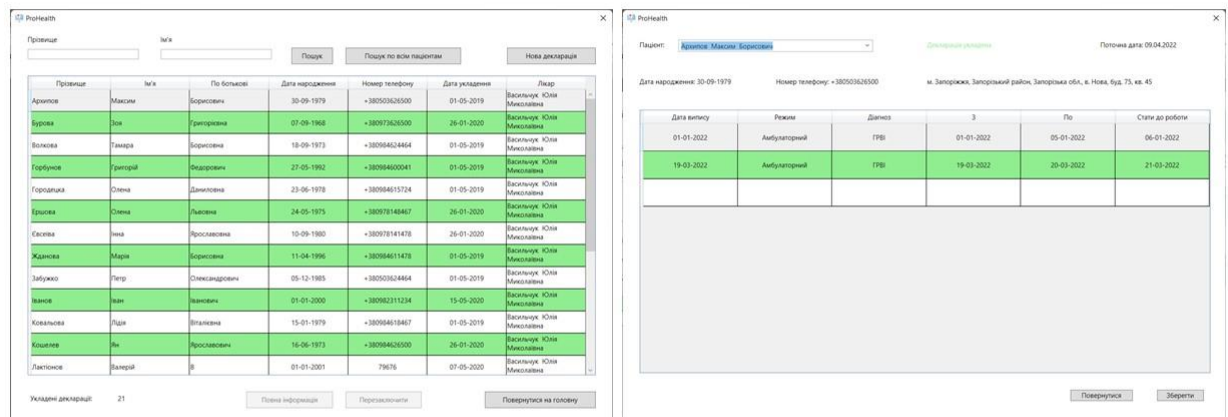


Сторінка прийому пацієнта, вікно інформації пацієнта та сервер ПЗ

11

Рисунок В.11 – Слайд №11

Розроблене програмне забезпечення

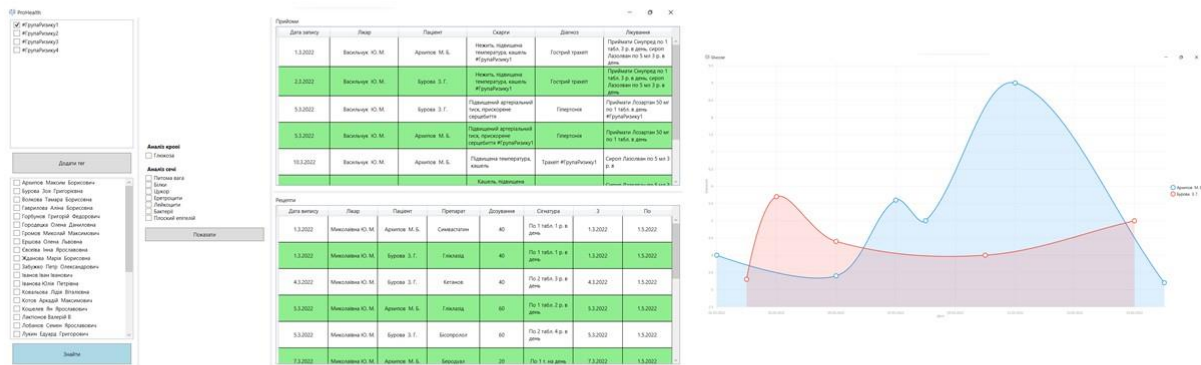


Сторінка декларацій та сторінка лікарняних

12

Рисунок В.12 – Слайд №12

Розроблене програмне забезпечення



Сторінка аналітики та вікно графіку

Рисунок В.13 – Слайд №13

Результати тестування

Перевірка	Результат
Робота HTTP серверу	Пройдено
Зв'язок з базою даних	Пройдено
Отримання запитів	Пройдено
Парсинг даних	Пройдено
Обробка даних	Пройдено
Збереження даних	Пройдено
Зворотній парсинг даних	Пройдено
Відповідь на запити	Пройдено
Логування	Пройдено
Безперервна робота	Пройдено

Чек-лист тестування серверу

Перевірка	Результат
Авторизація	Пройдено
Пошук пацієнтів	Пройдено
Внесення даних щодо прийому	Пройдено
Створення та перегляд декларацій	Пройдено
Виписування рецептів	Пройдено
Виписування лікарняних	Пройдено
Генерація документу «Декларація пацієнта»	Пройдено
Перегляд інформації щодо пацієнта	Пройдено
Перегляд аналітики	Пройдено
Внесення результатів аналізів	Пройдено
Перегляд медичної картки, рецептів, аналізів, лікарняних та декларацій	Пройдено
Логування	Пройдено
Безперервна робота	Пройдено

Чек-лист тестування клієнту

Рисунок В.14 – Слайд №14

Висновки

Було вирішено актуальне практичне завдання створення програмного забезпечення для ведення і аналізу медичних карток пацієнтів у медичних закладах.

Розроблено програмне забезпечення, яке має функціональність для перегляду та ведення медичних карток пацієнтів, перегляду та створення рецептів лікарських засобів, перегляду та створення лікарняних для пацієнтів, перегляду та створення аналізів пацієнтів, а також широку функціональність для забезпечення інформаційної підтримки прийняття рішень лікарем.

Головною особливістю розробленого програмного забезпечення стала функціональність аналітики, яка дозволяє лікарям переглядати зведені дані щодо пацієнта, чи групи пацієнтів, а також продивлятися графіки за обраними параметрами зроблених аналізів.

Програмна система запрограмована з акцентом на легкість додавання нової функціональності, тому розроблене програмне забезпечення може в подальшому бути використане для розробки аналогічних систем чи для доопрацювання до більш масштабної системи.