Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

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

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КЕРУВАННЯ ІНДИВІДУАЛЬНИМИ МЕДИЧНИМИ ЗАПИСАМИ ПАЦІЄНТІВ
SOFTWARE DEVELOPMENT FOR MEDICAL RECORDS MANAGEMENT

Виконав: студент(ка) 4 курсу, групи КНТ-117
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Чигрін Н.С.
(прізвище та ініціали)
Керівник Зайко Т.А.
(прізвище та ініціали)
Рецензент Голуб Т.В.
(прізвище та ініціали)

2021

Форма № 25

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

Інститут, факультет ІІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
“____” _____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Чигрін Наталії Сергіївни
(прізвище, ім’я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення керування індивідуальними медичними записами пацієнтів
Software Development for Medical Records Management

керівник проєкту (роботи) Зайко Тетяна Анатоліївна, к.т.н., доцент ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “ 30 ” березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) травень 2021 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. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Зайко Т.А., доцент		
Нормоконтролер	Липовець М.В., асистент		

7. Дата видачі завдання “ 15 ” березня 2021 року.

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	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка) _____ Чигрін Н.С.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Зайко Т.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 86 с., 28 рис., 1 табл., 3 дод., 19 джерел.

МЕДИЧНІ ЗАПИСИ, МЕДИЧНА КАРТКА, РЕЦЕПТ, ЛАБОРАТОРНІ ДОСЛІДЖЕННЯ, МОВА C#, MODEL-VIEW-VIEWMODEL.

Об’єкт дослідження – процес керування індивідуальними медичними записами пацієнтів. Предмет дослідження – програмне забезпечення керування індивідуальними медичними записами пацієнтів.

Мета роботи – розроблення програмного забезпечення керування індивідуальними медичними записами пацієнтів для забезпечення автоматизації даного процесу.

Матеріали, методи та технічні засоби: мова програмування C#, архітектурний шаблон Model-View-ViewModel, система керування базами даних Microsoft SQL Server.

Результати. Розроблено програмне забезпечення, яке дозволяє виконувати внесення, перегляд і редагування медичних записів пацієнта, їх пошук за заданий період, укладання декларацій пацієнта з лікарем з відповідним підтвердженням з двох сторін, пошук пацієнтів, внесення, редагування, пошук і перегляд результатів проведених лабораторних досліджень, внесення даних про сплату послуг, виписування, редагування, перегляд і пошук рецептів, продаж ліків за рецептом.

Висновки. Створено програмне забезпечення, яке придатне до подальшого розширення в медичній сфері, і може використовуватися для роботи пацієнтів, лікарів, адміністраторів аптек та представників лабораторій.

Галузь використання. Медична сфера в розрізі роботи з електронними медичними картками пацієнтів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 86 pages, 28 figures, 1 table, 3 appendixes, 19 sources.

MEDICAL RECORDS, MEDICAL CARD, RECIPE, LABORATORY RESEARCH, C# LANGUAGE, MODEL-VIEW-VIEWMODEL.

The object of research is the process of managing individual medical records of patients. The subject of research is software for managing individual medical records of patients.

The purpose of the work is to develop software for managing individual medical records of patients to ensure the automation of this process.

Materials, methods and technical means: C# programming language, architectural template Model-View-ViewModel, Microsoft SQL Server database management system.

Results. The developed software allows users to enter, view and edit patient medical records, search for a given period, subscrive patient declarations with a doctor with appropriate confirmation from both sides, search for patients, to enter, edit, search and view the results of laboratory tests, to enter data on payment for services, to prescribe, edit, view and search for prescriptions, to sale drugs according to the prescriptions.

Conclusions. The software has been developed in a way which is suitable for further expansion in the medical sphere and should be used for patients, doctors, pharmacy administrators and laboratory representatives.

Field of usage. Medical sphere in the context of working with electronic medical records of patients.

ЗМІСТ

С.

Перелік скорочень та умовних познак 7
Вступ 8
1 Аналіз предметної області 10
1.1 Основні сучасні програмні засоби, що використовуються в медичній сфері 10
1.2 Огляд програмних аналогів 15
1.3 Висновки за розділом 1 19
2 Розробка архітектури програми 20
2.1 Вимоги до програмного забезпечення 20
2.2 Вибір мови програмування 27
2.3 Проєктування бази даних 28
2.4 Вибір архітектури програмного забезпечення 32
2.5 Висновки за розділом 2 33
3 Розробка програми 34
3.1 Структура розробленого програмного забезпечення 34
3.2 Опис прийнятих під час реалізації рішень 37
3.3 Висновки за розділом 3 41
4 Експлуатація, тестування та експериментальне дослідження програми 42
4.1 Призначення програми 42
4.2 Умови виконання програми 42
4.3 Виконання програми 42
Висновки 51
Перелік джерел посилання 53
Додаток А Технічне завдання 55
Додаток Б Текст програми 60
Додаток В Слайди презентації 80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVVM – Movel-View-VIewModel;

БД – база даних.

ВСТУП

Вплив медичної галузі на життя людини величезний, що можна було оцінити протягом останнього періоду пандемії.

Інформація про лікування пацієнтів раніше була представлена в паперовому вигляді у картках пацієнтів. Це було традиційним підходом і це призводило з одного боку до того, що ці дані могли бути втрачені, з іншого боку – до їх фрагментації, адже вони накопичувались в паперовому вигляді в окремій установі, відповідно були недоступними в інших установах. До того ж ще однією проблемою була незручність пошуку, адже чим більше накопичено індивідуальних засобів, тим довше шукати потрібну інформацію і відповідно аналізувати її.

Створення програмного забезпечення, яке дозволить керувати індивідуальними медичними записами пацієнтів, дозволяє значно полегшити даний процес і виконувати в результаті аналіз швидше. Якщо декілька організацій будуть працювати з системою, то одночасно виписки різних лікарів, лабораторні дослідження, рецепти будуть доступні для всіх лікарів, і це дозволить приймати якісні рішення в результати, беручи до уваги всю наявну інформацію про пацієнта.

Розвиток медичної сфери в Україні нарешті досяг того стану, коли для підтримки роботи з пацієнтами стало використовуватися програмне забезпечення. Проте такі програмні системи на даний момент активно підтримують, наприклад, запис до лікарів на прийом, забезпечення інформаційної підтримки деяких медичних послуг, а робота над підтримкою електронних медичних карток пацієнтів тільки розпочинається. Тому проблему розроблення програмного забезпечення керування індивідуальними медичними записами пацієнтів, що розглядається в даній роботі, можна вважати актуальною в Україні.

Об’єкт дослідження – процес керування індивідуальними медичними записами пацієнтів.

Предмет дослідження – програмне забезпечення керування індивідуальними медичними записами пацієнтів.

Мета роботи полягає в розробленні програмного забезпечення керування індивідуальними медичними записами пацієнтів для забезпечення автоматизації даного процесу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні сучасні програмні засоби, що використовуються в медичній сфері

У даному підрозділі необхідно розглянути основні принципи організації програмного забезпечення керування індивідуальними медичними записами пацієнтів і основні типи програмних засобів, які загалом використовуються в медичній сфері.

У медичній сфері розрізняють програмні засоби, які направлені на роботу з електронними медичними картками пацієнтів, та програмні засоби, які направлені на роботу з медико-санітарною документацією. При цьому другі часто включають функціональність, пов’язану з медичними записами пацієнтів.

Щоб забезпечити найкращий можливий догляд, медичні працівники повинні мати можливість швидко розуміти історію пацієнтів та підтримувати ефективне постійне спілкування з пацієнтами щодо їхніх планів забезпечення здоров’я та лікування, саме програмне забезпечення керування індивідуальними медичними записами пацієнтів дозволяє використовувати шаблони, які фіксують відповідну інформацію під час візитів пацієнтів, і надає можливість швидко згадувати цю інформацію за потреби, а оскільки взаємодія між медичними інформаційними системами стає більш звичним явищем, записи пацієнтів стають більш доступними всій медичній сфері (наприклад, легкий доступ до історії хвороби пацієнта може суттєво покращити якість медичної допомоги, яку вони отримують під час відвідування лікарні швидкої допомоги, коли час є найважливішим), окрім того, щоб зробити записи більш доступними, сучасні системи можуть допомогти значно зменшити помилки, особливо при призначенні ліків [1].

Медичні системи також покращують канал зв'язку між постачальниками та пацієнтами: наприклад, така система може інтегруватися із системами планування, щоб автоматично нагадувати пацієнтам про

майбутні зустрічі, тоді провайдери можуть використовувати портали для пацієнтів, щоб надійно повідомляти пацієнтам плани лікування, інструкції щодо призначення рецептів та іншу корисну інформацію, поряд з порталами для пацієнтів, поява телемедицини ще більше полегшує охорону здоров'я: сеанси телемедицини дозволяють провайдерам проводити діагностику та лікування пацієнтів по телефону або за допомогою відеочату, пацієнти можуть скористатися медичними порадами, не виходячи з власного будинку, та заощадити час та гроші, які витрачали б на поїздки [1].

Медичні програмні засоби можуть надавати засоби виконання наступних функцій:

– електронне призначення ліків: можна роздрукувати та передати рецепти в аптеки на вибір пацієнта, окрім того отримувати автоматичні сповіщення, пов'язані з різними взаємодіями з аптеками щодо ліків, такими як кількість дозувань, алергія та рецепти;

– портал для пацієнтів: надається можливість пацієнтам входити в систему та отримувати доступ до різної інформації, наприклад, історії їх прийомів, кількості відвідувань лікаря та результатів лабораторних досліджень, що у свою чергу має загалом призвести до поліпшеного залучення пацієнтів, сприяючи активній участі пацієнтів у процесах, пов'язаних з їх власним здоров'ям;

– введення замовлення: вводити, зберігати та передавати замовлення на лабораторні тести, замовлення на ліки та інші послуги;

– підтримка прийняття рішень: отримувати автоматичні сповіщення про необхідне лікування, нагадування або рекомендації, які призначені допомогти пацієнтам на основі їхніх конкретних станів та демографічних показників [2].

Переваги використання такого програмного забезпечення:

– підвищення ефективності практик за рахунок більш зручного охоплення великого набору даних;

– впорядковані адміністративні завдання;

– покращена прибутковість;

– поліпшення залучення пацієнтів;

– зниження стаціонарних витрат;

– автоматизовані попередження і нагадування, що допомагають пацієнтам з’явитися вчасно;

– покращена безпека [3].

Таке програмне забезпечення може використовуватися установами різного рівня:

– медичні практики невеликих або індивідуальних установ: маленькі або самостійні практики з невеликими офісними приміщеннями та обмеженою кількістю співробітників, як правило, не мають великого бюджету порівняно з більшими практиками, через ці обмеження багато таких установ просто віддають перевагу хмарному програмному забезпеченню та використовують безкоштовне програмне забезпечення або вибирають доступного постачальника програмного забезпечення з низькою фіксованою щомісячною платою;

– установи середнього розміру: часто такі установи надають практики, що належать до однієї і тієї ж медичної спеціальності, середні установи від програмного забезпечення вимагають унікальних функцій, таких як сумісність, спеціальна підтримка клієнтів, підтримка декількох пристроїв, а також широка функціональність звітування, щоб стежити за загальною ефективністю практик, оскільки їхній бюджет також порівняно більший;

– установи великого розміру: більші медичні установи зазвичай пропонують медичні послуги, які підпадають під різні медичні спеціальності (наприклад, ортопедія, хірургія, сімейна медицина тощо), при цьому медичні послуги можуть надаватися від п’ятнадцяти до двадцяти постачальниками медичних послуг, ці практики часто вимагають підтримки декількох користувачів для одночасного доступу до систем, програмне забезпечення має підтримувати розміщення в декількох різних місцях, доступ різних постачальників до різних ресурсів;

– лікарні підприємств: у лікарнях підприємств є найбільш складний перелік вимог, які, як правило, задокументовані у відповідній формі для оцінки та внесення до списку постачальників, які потенційно можуть їм відповідати [3].

Деякі типи програмного забезпечення, які тісно пов’язані з системами, що працюють з індивідуальними медичними записами пацієнтів, зокрема включають:

– програмне забезпечення для проблем психічного здоров'я, що охоплює системи електронних медичних карток для закладів, що надають послуги підтримки психічного та поведінкового здоров'я, вони мають унікальні функції для консультантів, клінік психічного здоров'я та групових практик;

– програмне забезпечення виставлення рахунків в медичній галузі: медичні системи виставлення рахунків допомагають формувати виписки пацієнтів, таке програмне забезпечення може використовуватись тоді, коли клієнти хочуть самостійно обробляти рахунки;

– програмне забезпечення для порталів пацієнтів: системи порталів пацієнтів дозволяють пацієнтам отримувати доступ до власної інформації про охорону здоров’я, оплачувати рахунки, планувати консультації та спілкуватися безпосередньо з постачальниками послуг;

– програмне забезпечення для планування розкладу роботи з пацієнтами: системи планування спрощують створення графіків за рахунок автоматизації процесу, можна дозволити пацієнтам планувати консультації без необхідності дзвонити у кабінет, а також можна надсилати автоматичне підтвердження зустрічі та сповіщення для нагадування, щоб зменшити рівень неявок;

– програмне забезпечення для підтримки телемедицини: системи телемедицини – це інструменти, які дозволяють медичним працівникам надавати допомогу пацієнтам віддалено, використовуючи такі функції, як безпечна відеоконференція, чат та обмін повідомленнями [2].

Для більшості медичних кабінетів процес збору інформації традиційно полягав у заповненні або надсиланні пакету з кількома паперовими формами для нового пацієнта, який необхідно заповнити перед будь-яким контактом із медичним працівником, потім лікар переглядав ці документи під час контакту з пацієнтом, щоб знайти дані, що стосуються візиту, найбільш очевидний спосіб перевести цей процес в електронну систему – це використовувати комп'ютер або планшетний пристрій певного типу, що використовує, наприклад, веббраузер як інтерфейс користувача, в ідеалі медичні установи дозволять пацієнтам отримувати доступ до медичної картки та вхідних даних у зручний для них спосіб, окрім того альтернативою могло б бути продовжувати використовувати папір, але потім сканувати документи в систему після завершення, після того, як основна інформація про пацієнта буде зібрана та введена в систему, розпочинається процес надання допомоги: зазвичай це починається зі списку питань для визначення симптомів, це критична частина процесу, оскільки в кінцевому підсумку це допоможе сформулювати діагноз і перелік дій, як варіант, якщо пацієнт потребує подальшого спостереження або специфічного лікування, такого як мануальна терапія або фізіотерапія, тоді перелік дій зазвичай визначається заздалегідь, однак прогрес пацієнта у виконанні плану повинен бути оцінений та задокументований, а отримана в результаті інформація повинна бути позначена та надійно збережена, у будь-якому випадку, фактично наданий догляд повинен бути задокументований та закодований відповідно до вимог, що регулюються страхуванням, щоб забезпечити належне відшкодування [4].

Для прийняття рішень, які стосуються програмного забезпечення, що розробляється, тепер потрібно провести аналіз реально існуючих на даний момент програмних аналогів.

1.2 Огляд програмних аналогів

Kareo Clinical [5] (рис. 1.1) є одним з програмних засобів, які можуть використовуватися для керування індивідуальними медичними записами пацієнтів. Цей програмний засіб спеціально створений для невеликих незалежних практик, а не для розширених медичних центрів та лікарень, відповідно для кінцевого користувача це виражається в тому, що в програмному засобів немає додаткових функцій, які фактично на практиці не будуть використовуватися, але при цьому вартість програмного забезпечення буде враховувати такі можливості, окрім того щодо даного програмного засобу можна відзначити наступне:

– розгортання системи є достатньо простим, а хмарний інтерфейс простий у використанні, відповідно це призводить до зменшення потреби в навчанні, хоча варто відзначити, що таке навчання потребує тільки додаткових витрат часу, але не вимагає додаткової плати;

– інформаційна панель забезпечує легкий доступ для ліцензованих користувачів до призначень для пацієнта та записів, а також виставлення рахунків та вбудованої аналітики, існує також портал для пацієнтів, який оптимізований для мобільних пристроїв, тому пацієнти можуть легко перевірити своє призначення або деталі рецепта зі свого мобільного телефону, і є навіть вбудована опція відеоконференцій;

– Kareo Clinical також пропонує один з найнижчих тарифів серед постачальників такого типу програмного забезпечення [6]-[7].

Ще одним програмним аналогом є WRS Health [8]-[9], в якому надається велика кількість засобів для зручності роботи лікаря з індивідуальними медичними записами пацієнтів: диктофон та інструменти малювання, вбудовані в інструмент створення діаграм, є корисними для лікарів для документування інформації, включаючи не лише те, що вони чують від пацієнтів, але те, що вони розповідають пацієнтам, при цьому реалізуючи даний процес у режимі реального часу, практикуючий спеціаліст

може також змінювати шаблони, додаючи рядки до випадаючих меню та додаючи до списків симптоми, щоб зробити оцінки доречними та індивідуальними для пацієнта в режимі реального часу [10].

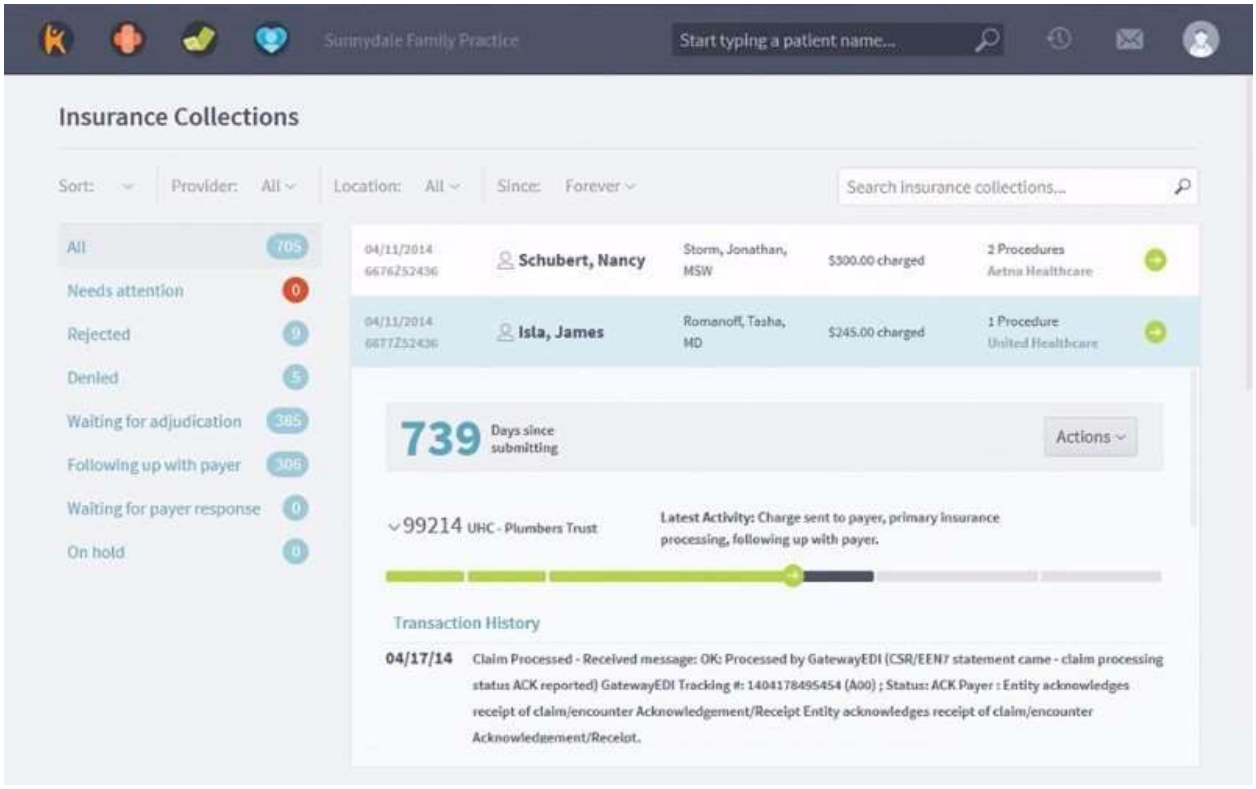

Рисунок 1.1 – Kareo Clinical [8]

WRS Health (рис. 1.2) добре функціонує для інтегративних клінік, які потребують функцій, що забезпечують роботу багатьох типів спеціалістів, та великих груп, які мають декілька фізичних локацій та персонал, який виконує функціональні адміністративні ролі, дана платформа добре підходить для підтримки потреб великої групової практики або для зайнятого індивідуального спеціаліста із допоміжним персоналом, який особливо потребує функцій електронного призначення, у такому випадку можливості налаштування можуть зробити цей інструмент достатньо придатним для змішаних практик, які надають як медичні послуги загалом, так і послуги, пов’язані з психічним здоров’ям [11].

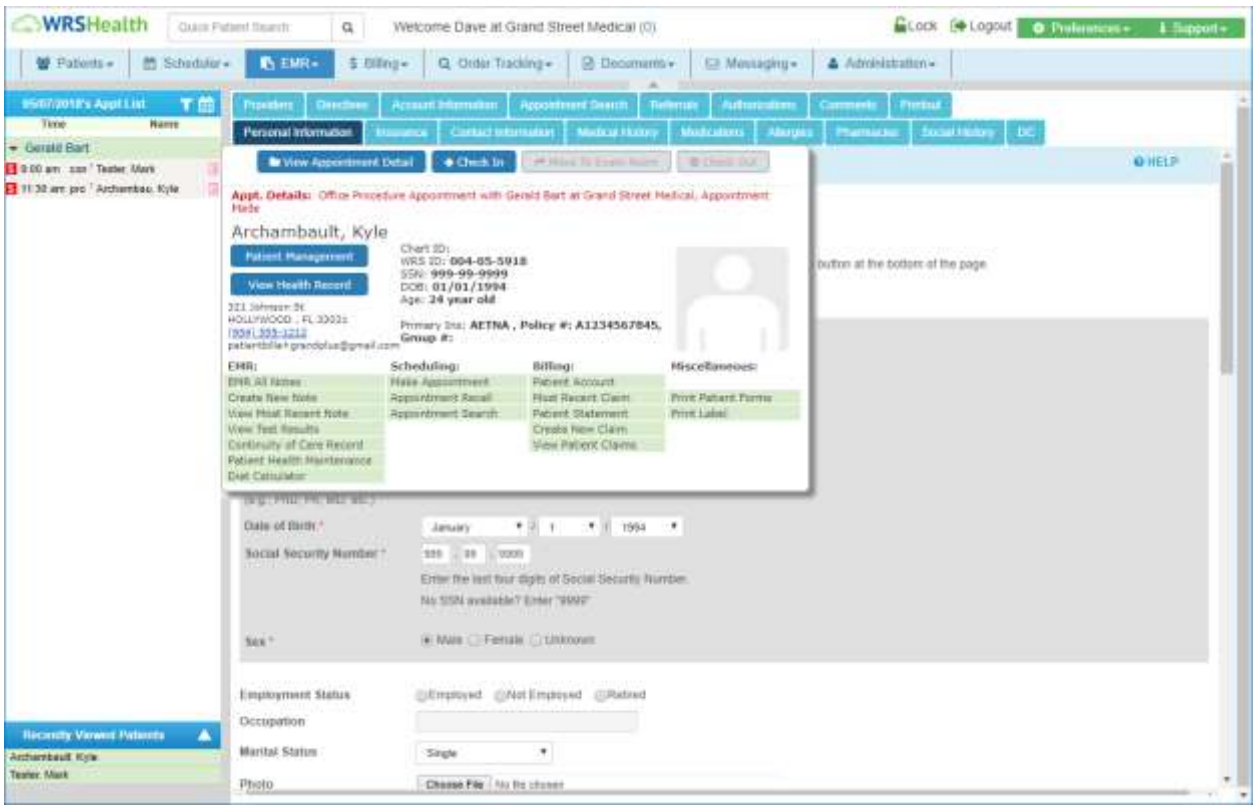

Рисунок 1.2 – WRS Health [10]

iPatientCare [12] (рис. 1.3) надає орієнтоване на клієнта програмне забезпечення, яке постачається з низкою налаштувань робочого циклу, щоб полегшити роботу для практики, що надається, незалежно від розміру та місцезнаходження установи, доступні дві основні версії, із доступом до хмарної платформи, яка забезпечує надійне зберігання записів пацієнтів з дотриманням вимог Health Insurance Portability and Accountability Act, але за необхідності використання власних серверів, доступна локальна версія, при цьому доступні сотні адаптерів, що дозволяють програмному забезпеченню підключатись до лабораторій, аптек та різних реєстрів, а також до варіантів інтеграції, таких як системи управління практиками, відповідно дана функціональність також працює з порталом iPatientCare та іншими програмними системами, загалом, iPatientCare EHR прагне зробити своє програмне забезпечення відносно простим і зокрема простим у використанні, забезпечуючи при цьому всю інформацію, необхідну для ведення медичної документації пацієнтів [7].

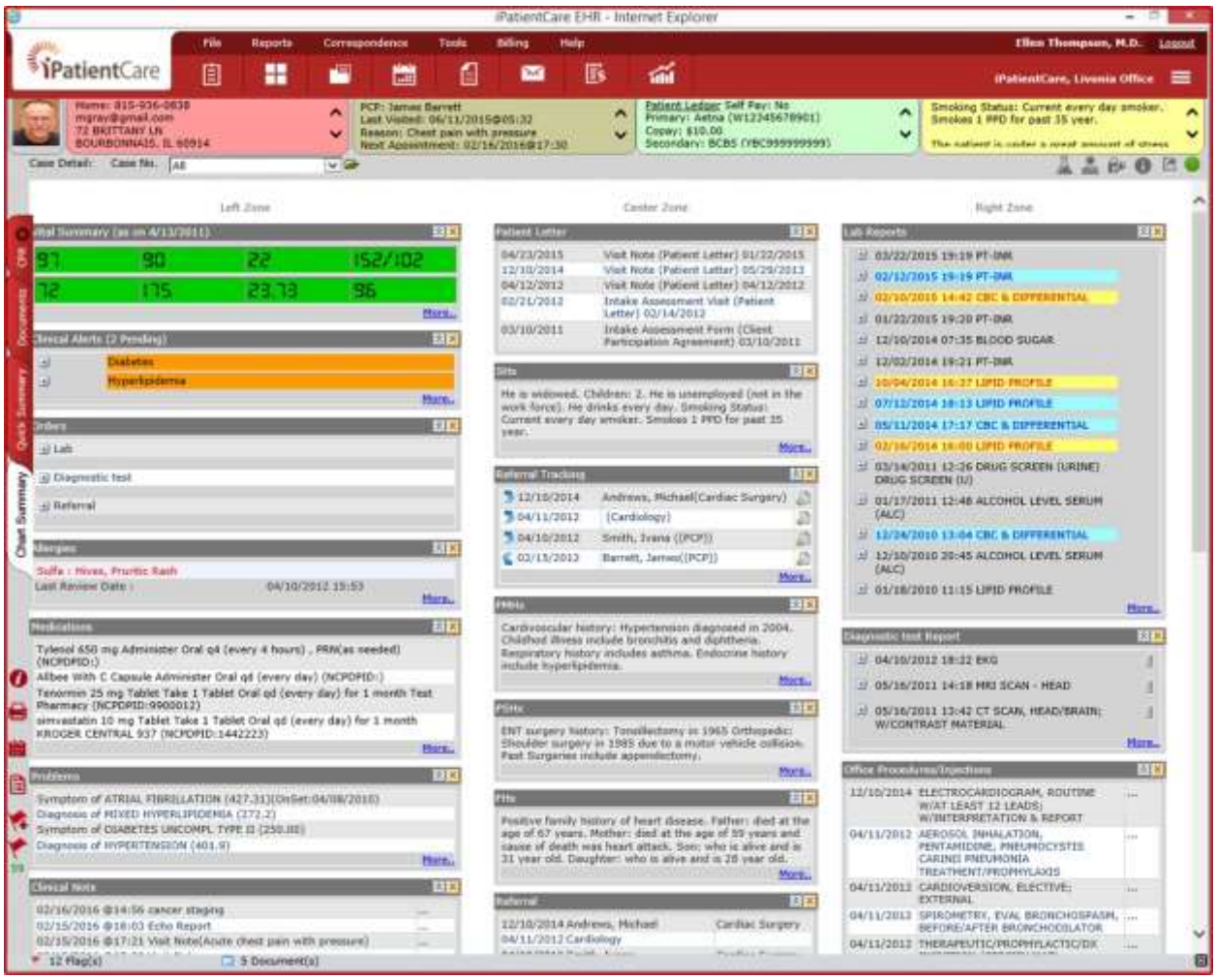

Рисунок 1.3 – iPatientCare [13]

Незважаючи на велику кількість програмних засобів у цій сфері, лише невелика частина з яких була детально представлена в даному підрозділі, в них існує певна перевантаженість функціями, яка з часом призводить до того, що безкоштовні засоби стають платними, відповідно можливості не використовувати додаткові функції, які не потрібні конкретному спеціалісту або клініці, немає. Тому важливим завданням є розроблення програмного забезпечення керування індивідуальними медичними записами пацієнтів, яке буде концентруватися саме на забезпеченні процесу роботи з індивідуальними медичними записами пацієнтів і буде достатньо простим у використанні.

1.3 Висновки за розділом 1

Розглянуто основні типи медичних систем та особливості їх застосування для керування індивідуальними медичними записами пацієнтів. Проаналізовано аналоги програмного засобу, що розробляється, включаючи Kareo Clinical, WRS Health, iPatientCare.

Завдання, які поставлені на виконання в дипломній кваліфікаційній роботі:

– визначення вимог до програмного забезпечення, що розробляється;

– проєктування програмного забезпечення;

– реалізація програмного забезпечення;

– тестування і налагодження програмного забезпечення.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вимоги до програмного забезпечення

Оскільки основним недоліком існуючих програмних засобів у цій сфері є їхня функціональна перевантаженість, то в даному випадку центровим елементом було вирішено зробити індивідуальні медичні записи пацієнтів, а також інформацію безпосередньо пов’язану з ними. Безпосередньо пов’язаними з ними є дані профілю пацієнта, які зазвичай заповнюються адміністратором, та дані рецептів, адже це часто є результатом обслідування, який зокрема потрібний не тільки лікарю, який є основним користувачем системи.

Окрім лікаря базовим користувачем програмної системи є також пацієнт, оскільки він окрім надання власних даних також повинен мати доступ до таких даних, виписані йому медикаменти, сам курс лікування – це все те, що потрібно пацієнту.

Але окрім лікаря та пацієнта є ще групи користувачів, які будуть взаємодіяти з системою, адже вони також використовують описану вище інформацію. Це, як було зазначено вище, адміністратори, а також представники аптек та лабораторій. Кожен з них потребує певного рівня доступу, тому спочатку потрібно визначити загальні вимоги до програмного забезпечення, а далі виділити сценарії використання системи кожним типом користувача.

Пацієнт може отримувати доступ тільки до власної інформації. Інформація про будь-якого з інших пацієнтів йому має бути не доступна. Ця інформація включає його декларацію, укладену з лікарем, результати лабораторних досліджень стану його здоров’я, результати спостережень лікаря, а також висновки лікаря, що формують безпосередньо медичну картку пацієнта, отримані рецепти, а також придбані за цими рецептами фактично ліки.

Адміністратор може отримувати доступ до даних профілю пацієнта під час первинного внесення цих даних, окрім того до даних профілю пацієнтів лікарні, в якій він працює адміністратором.

Лікар може працювати з будь-якими медичними даними, які стосуються пацієнта, з яким ним було укладено декларацію. При цьому лікарем може бути не обов’язково сімейний лікар. А будь-який з лікарів пацієнта може отримати доступ до всіх даних, які було внесено іншими лікарями. Це дозволяє уникати необхідності подвійного внесення даних. Якщо один з лікарів оцифрував якісь дані, то вся повністю історія пацієнта буде доступна всім для повноцінного виконання аналізу цих даних та діагностування. Лікар може працювати тільки з даними власних пацієнтів, до даних всіх інших пацієнтів лікар доступ отримати не може, таким чином реалізується захист даних.

Представник аптеки може працювати з даними будь-якого пацієнта, але тільки тими, що стосуються виписаних йому, при чому активних рецептів. Усі інші дані закриті від аптекаря, тобто він не може переглянути історію хвороби пацієнта.

Представники лабораторій можуть тільки вносити результати лабораторних досліджень стану здоров’я пацієнта, а також переглядати дані профілю пацієнта (тобто фактично його ім’я, прізвище тощо). Ні до якої іншої інформації такі учасники доступу не мають.

Програмне забезпечення керування індивідуальними медичними записами пацієнтів повинно надавати можливість виконання наступних функцій:

– внесення і редагування медичних записів пацієнта;

– перегляд всіх медичних записів пацієнта;

– пошук медичних записів пацієнта;

– укладання декларації пацієнта з лікарем;

– пошук пацієнтів;

– внесення і редагування результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– пошук результатів проведених лабораторних досліджень;

– внесення даних про сплату послуг;

– виписування рецептів;

– редагування рецептів;

– перегляд рецептів;

– пошук рецептів;

– продаж ліків за рецептом;

– авторизація облікового запису;

– реєстрація облікового запису;

– внесення даних профілю пацієнта.

Адміністратор лікарні може використовувати наступну функціональність системи (рис. 2.1):

– пошук пацієнтів;

– авторизація облікового запису;

– реєстрація облікових записів лікарів, пацієнтів;

– внесення даних про сплату послуг;

– внесення даних профілю пацієнта.

Лікар може використовувати наступну функціональність системи:

– внесення і редагування медичних записів пацієнта;

– перегляд всіх медичних записів пацієнта;

– пошук медичних записів пацієнта;

– укладання декларації з пацієнтом;

– пошук пацієнтів;

– перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– пошук результатів проведених лабораторних досліджень;

– виписування рецептів;

– редагування рецептів;

– перегляд рецептів;

– пошук рецептів;

– авторизація облікового запису.

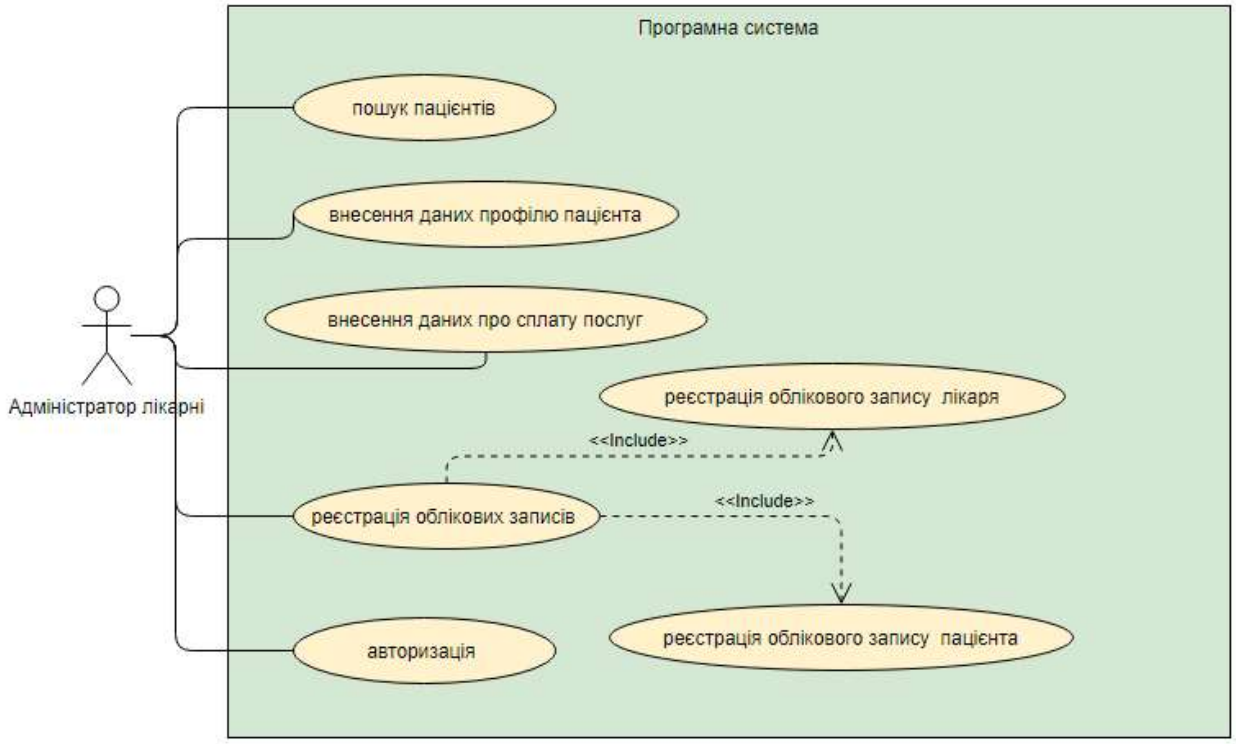

Рисунок 2.1 – Діаграма прецедентів адміністратора лікарні

Діаграму прецедентів для роботи лікаря з програмною системою приведено на рис. 2.2.

Пацієнт може використовувати наступну функціональність системи:

– перегляд всіх медичних записів пацієнта;

– пошук медичних записів пацієнта;

– укладання декларації з лікарем;

– перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– пошук результатів проведених лабораторних досліджень;

– перегляд рецептів;

– пошук рецептів;

– авторизація облікового запису.

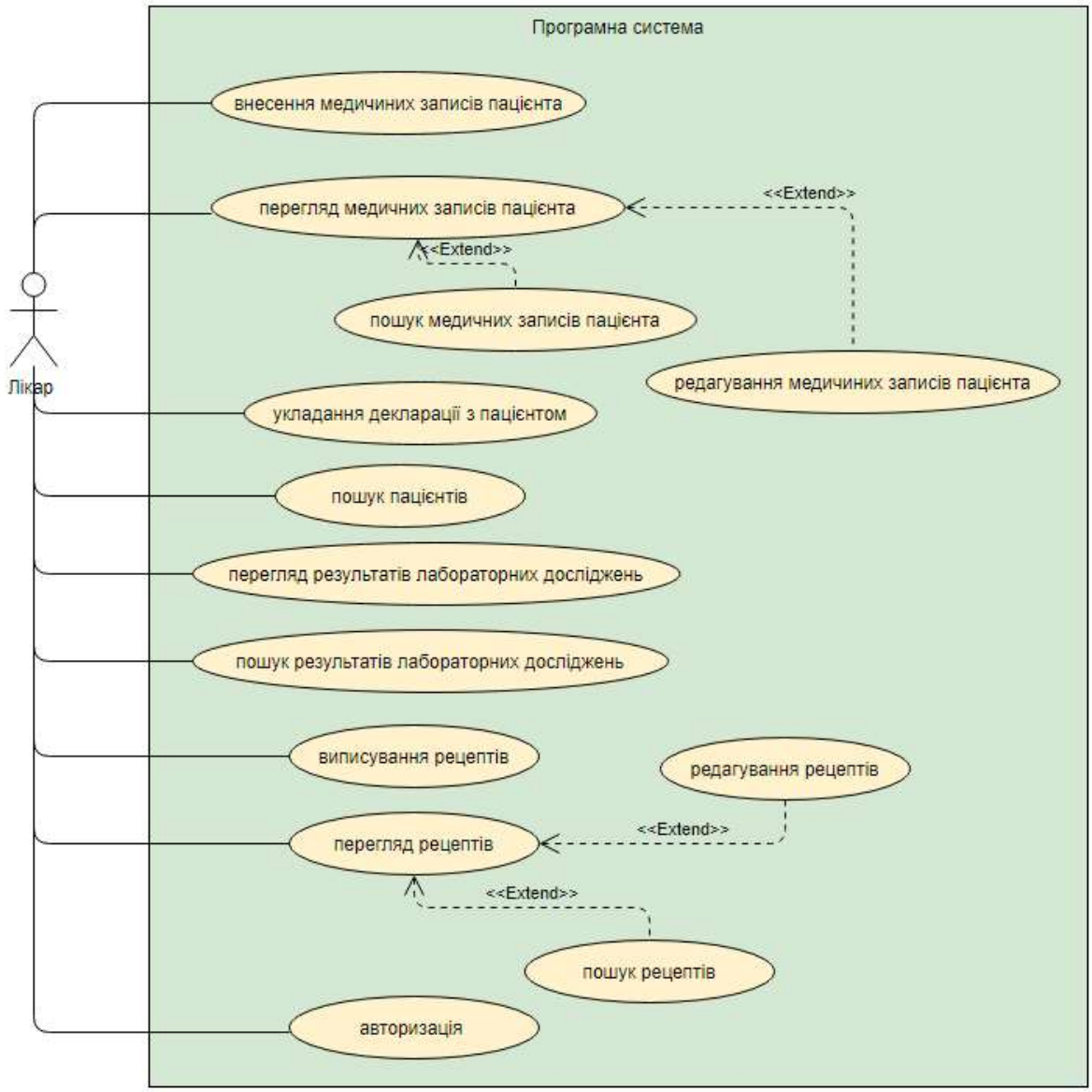

Рисунок 2.2 – Діаграма прецедентів лікаря

Діаграму прецедентів для роботи пацієнта з програмною системою приведено на рис. 2.3.

Представник аптеки може використовувати наступну функціональність системи:

– пошук пацієнтів;

– перегляд і пошук рецептів;

– продаж ліків за рецептом;

– авторизація облікового запису.

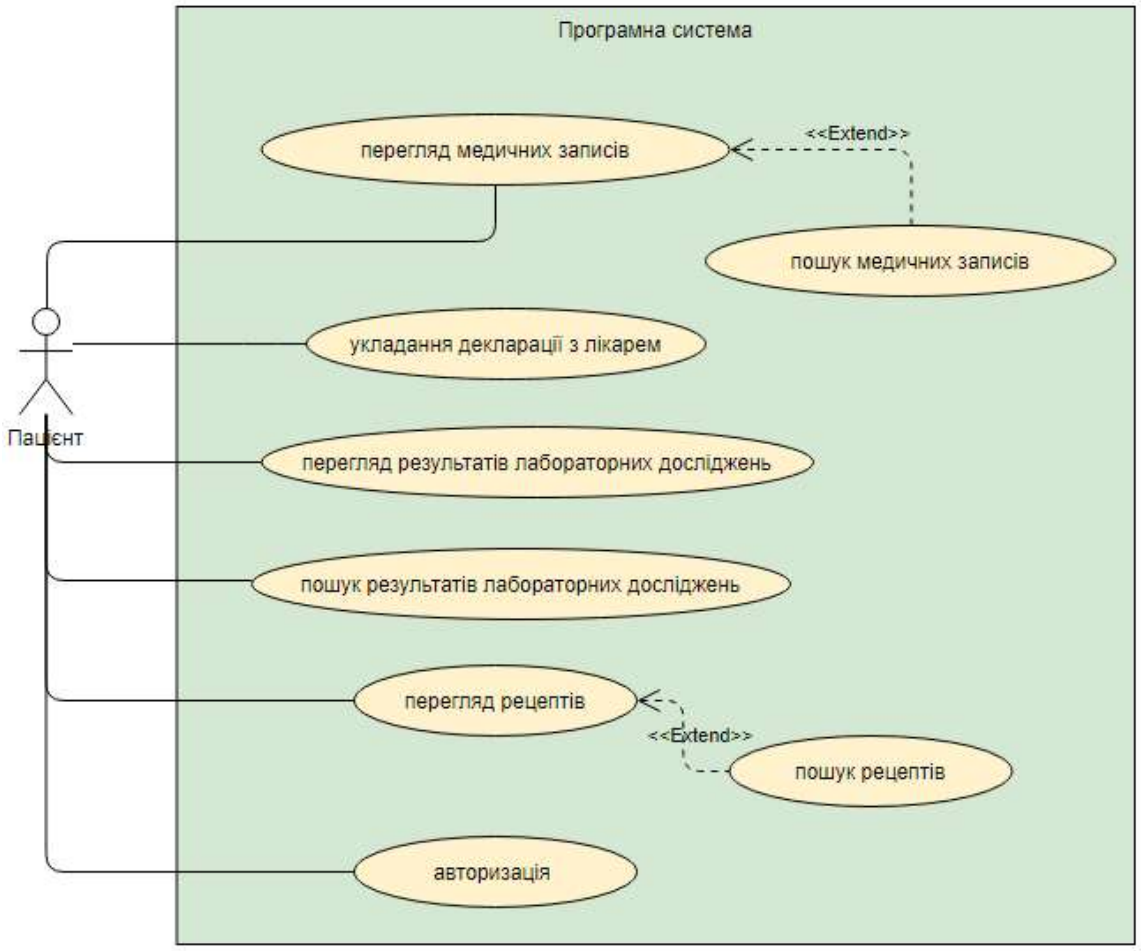

Рисунок 2.3 – Діаграма прецедентів пацієнта

Діаграму прецедентів представника аптеки приведено на рис. 2.4.

Представник лабораторії може використовувати наступну функціональність системи (рис. 2.5):

– пошук пацієнтів;

– внесення і редагування результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– пошук результатів проведених лабораторних досліджень;

– внесення даних про сплату послуг;

– авторизація облікового запису.

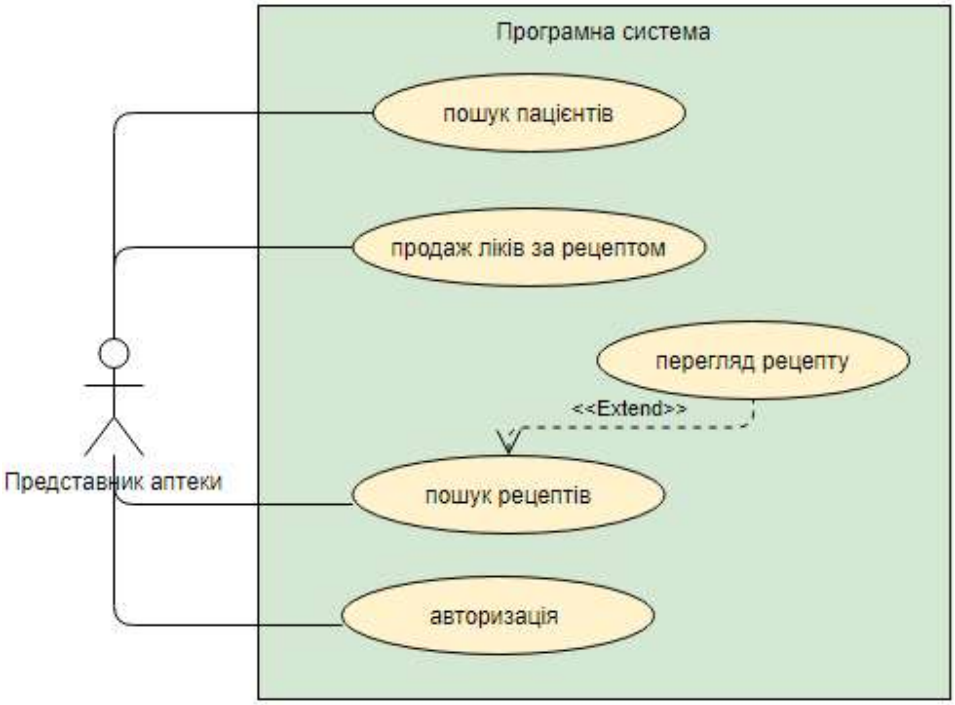

Рисунок 2.4 – Діаграма прецедентів представника аптеки

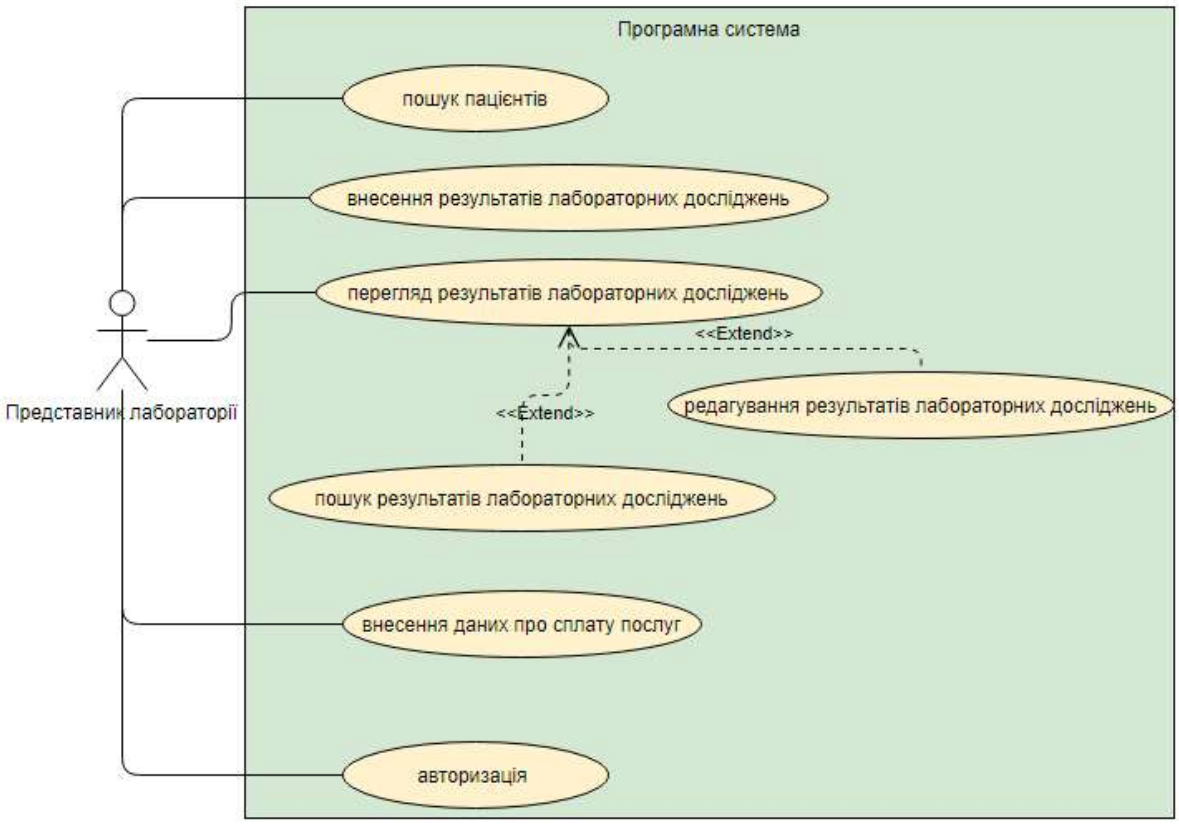

Рисунок 2.5 – Діаграма прецедентів представника лабораторії

2.2 Вибір мови програмування

Програмне забезпечення керування індивідуальними медичними записами пацієнтів має бути розроблено у вигляді десктопного застосунку. Підтримка такого рішення буде здійснюватися за рахунок віддаленого розташування бази даних. Саме віддалене розташування на сервері даних призведе до того, що клієнтські застосунки будуть звертатися до сервера і отримувати необхідні дані.

Мови програмування Java [14] і C# [15]-[17], як одні з найпопулярніших в цій галузі, були порівняні відносно можливості реалізації цього програмного забезпечення (табл. 2.1).

Таблиця 2.1 – Вибір мови програмування

Показник	C#	Java
Парадигма програмування	Об’єктно-орієнтована, процедурна, узагальнена, функціональна	Об’єктно-орієнтована, структурна
Платформи	Windows, кросплатформна, за рахунок .NET Core	Кросплатформна
Збирач сміття	Наявний	Наявний
Створення графічного інтерфейсу	Легке за допомогою візуальних засобів середовища	Ускладнене, передбачає роботу на основі додаткових бібліотек

Для подальшої розробки було обрано мову програмування C#, оскільки вона дозволяє значно простіше створювати графічний інтерфейс для Windows-застосунків зокрема за допомогою середовища Visual Studio, до

того ж дозволяє дещо гнучкіше створювати код за рахунок підтримки більшої кількості сучасних парадигм програмування.

2.3 Проєктування бази даних

Проєктування і розробку бази даних виконано на основі використання системи керування базами даних (БД) MS SQL Server [18].

Сутностями, що було покладено в основу створення бази даних, були визначені такі:

– Account – обліковий запис;

– Patient – пацієнт;

– Record – медичний запис (основа побудови медичної картки);

– Diagnosis – діагноз;

– Recipe – рецепт;

– Drug – ліки;

– Sale – продаж ліків;

– Laboratory – лабораторне дослідження;

– Declaration – декларація між пацієнтом та лікарем.

Кожну сутність було представлено в результаті відповідною таблицею БД.

Таблиця Account передбачає такі поля:

– AccountId – номер облікового запису;

– Name – ім’я;

– FrName – по батькові;

– Surname – прізвище;

– Login – логін;

– Password – пароль;

– Typeid – тип облікового запису (адміністратор лікарні, лікар, пацієнт, аптекар, представник лабораторії);

– Phone – номер телефону;

– Job – місце роботи;

– Position – посада (в тому числі передбачає конкретну спеціалізацію лікаря).

Таблиця Patient передбачає такі поля:

– PatientId – номер пацієнта;

– AccountId – номер облікового запису (зовнішній ключ з таблиці Account);

– Passport – інформація про паспорт пацієнта;

– Idcode – ідентифікаційний код пацієнта;

– BthDay – дата народження пацієнта;

– Sex – стать пацієнта;

– Home – адреса проживання пацієнта.

Таблиця Record передбачає такі поля:

– RecordId – номер медичного запису;

– DeclarationId – номер декларації;

– DiagnosisId – номер діагнозу;

– DiagnosisState – стан визначення діагнозу (попередній, остаточний або повторний);

– AddDiagnosis – додаткові діагнози;

– RecDate – дата запису;

– Problems – скарги пацієнта та визначені спостереження під час огляду;

– Therapy – призначене лікування;

– Research – призначене подальше дослідження;

– Description – опис будь-якої додаткової інформації про виконане спостереження.

Таблиця Diagnosis передбачає такі поля:

– DiagnosisId – номер діагнозу;

– Title – назва.

Таблиця Recipe передбачає такі поля:

– RecipeId – номер рецепту;

– DeclarationId – номер декларації;

– RecDate – дата виписування рецепту;

– StartDate – дата початку дії рецепту;

– EndDate – дата завершення дії рецепту;

– Description – опис лікування за рецептом;

– RecStatus – стан рецепту (активний чи неактивний).

Таблиця Drug передбачає такі поля:

– DrugId – номер ліків;

– Title – назва ліків;

– Dose – дозування.

Таблиця Sale передбачає такі поля:

– SaleId – номер операції;

– AccountId – номер облікового запису працівника аптеки, який виконав продаж;

– RecipeId – номер рецепту;

– SaleDate – дата операції;

– Price – вартість одиниці ліків;

– Quantity – кількість придбаних ліків.

Таблиця Laboratory передбачає такі поля:

– LaboratoryId – номер лабораторного дослідження;

– AccountId – номер облікового запису працівника лабораторії, який вніс дані;

– PatientId – номер пацієнта;

– LabDate1 – дата і час виконання лабораторного дослідження;

– LabDate2 – дата і час отримання результату лабораторного дослідження;

– LabFile – файл лабораторного дослідження;

– Description – опис дослідження.

В результаті була створена структура БД (рис. 2.6).

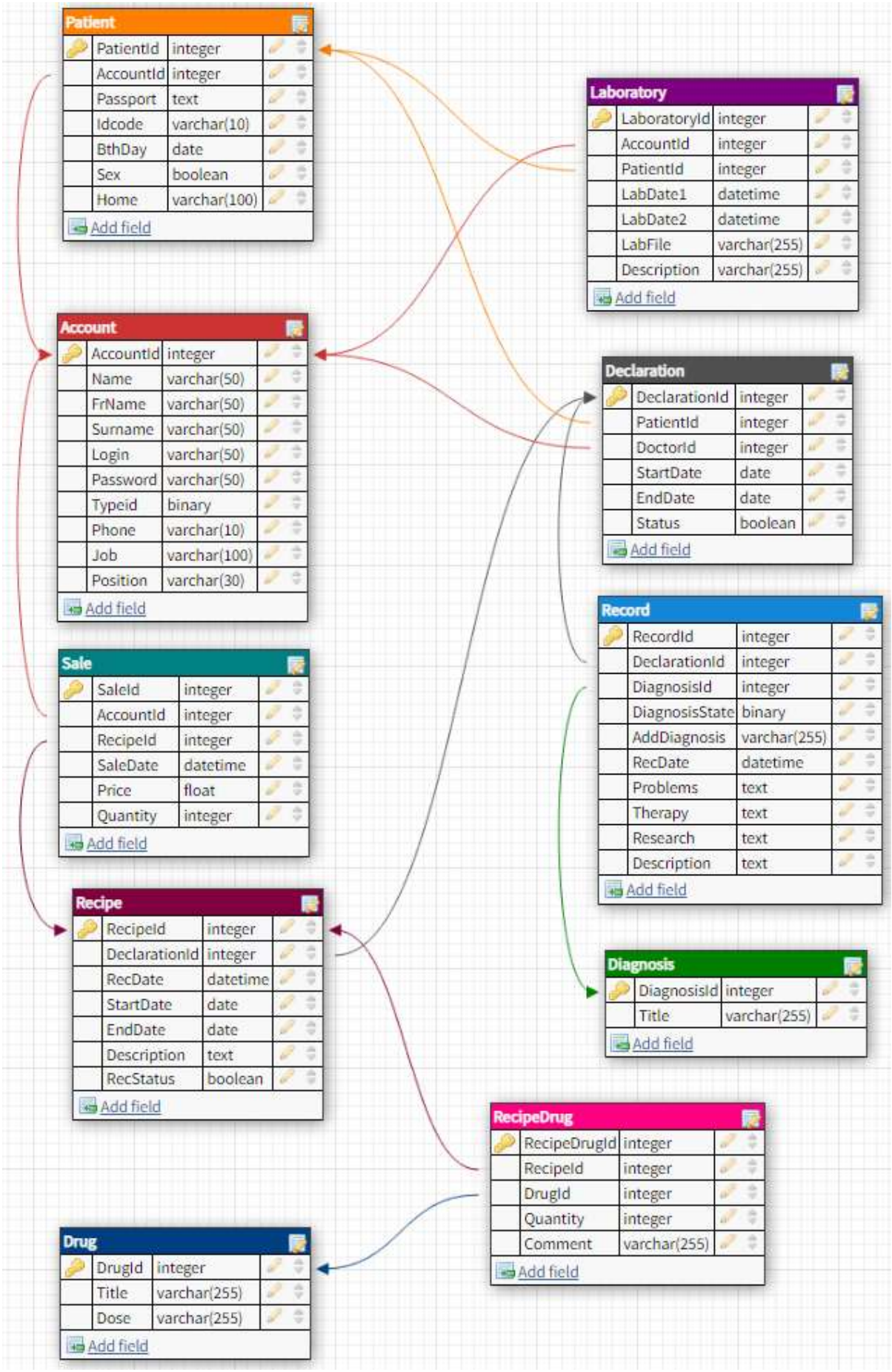

Рисунок 2.6 – Структура БД

Таблиця Declaration передбачає такі поля:

– DeclarationId – номер декларації пацієнта з лікарем;

– PatientId – номер пацієнта;

– DoctorId – номер лікаря (облікового запису фактично);

– StartDate – дата початку дії декларації;

– EndDate – дата завершення дії декларації;

– Status – поточний стан дії декларації (активна або неактивна).

Для реалізації зв’язку багато до багатьох між таблицями Recipe і Drug було створено таблицю RecipeDrug, що передбачає такі поля:

– RecipeDrugId – номер запису;

– RecipeId – номер рецепту;

– DrugId – номер ліків;

– Quantity – кількість ліків;

– Comment – коментар до лікування.

2.4 Вибір архітектури програмного забезпечення

Для побудови архітектури програмного забезпечення було використано архітектурний шаблон Model-View-ViewModel (MVVM) [19].

Використання архітектурного шаблону дозволить у подальшому використовувати дану розробку як модуль в більшій системі підтримки медичних організацій за необхідності.

Відповідно під час визначення структури застосунку необхідно буде виділити відповідний набір представлень, моделей та моделей представлень у відповідності зі структурою БД та визначеними функціональними особливостями. Це і визначає використання архітектурного шаблону в програмному забезпечення.

2.5 Висновки за розділом 2

Представлено результати проєктування функціональних вимог до програмного забезпечення з відповідними сценаріями роботи адміністратора лікарні, лікаря, пацієнта, представників аптек та лабораторій.

Обрано мову програмування C# і систему керування БД Microsoft SQL Server в якості засобів розробки програмного забезпечення.

Виконано проєктування БД і обрано архітектурний шаблон MVVM для побудови програмного забезпечення.

3 РОЗРОБКА ПРОГРАМИ

3.1 Структура розробленого програмного забезпечення

Використовуючи архітектурний шаблон MVVM, було розроблено структуру програмного забезпечення (рис. 3.1).

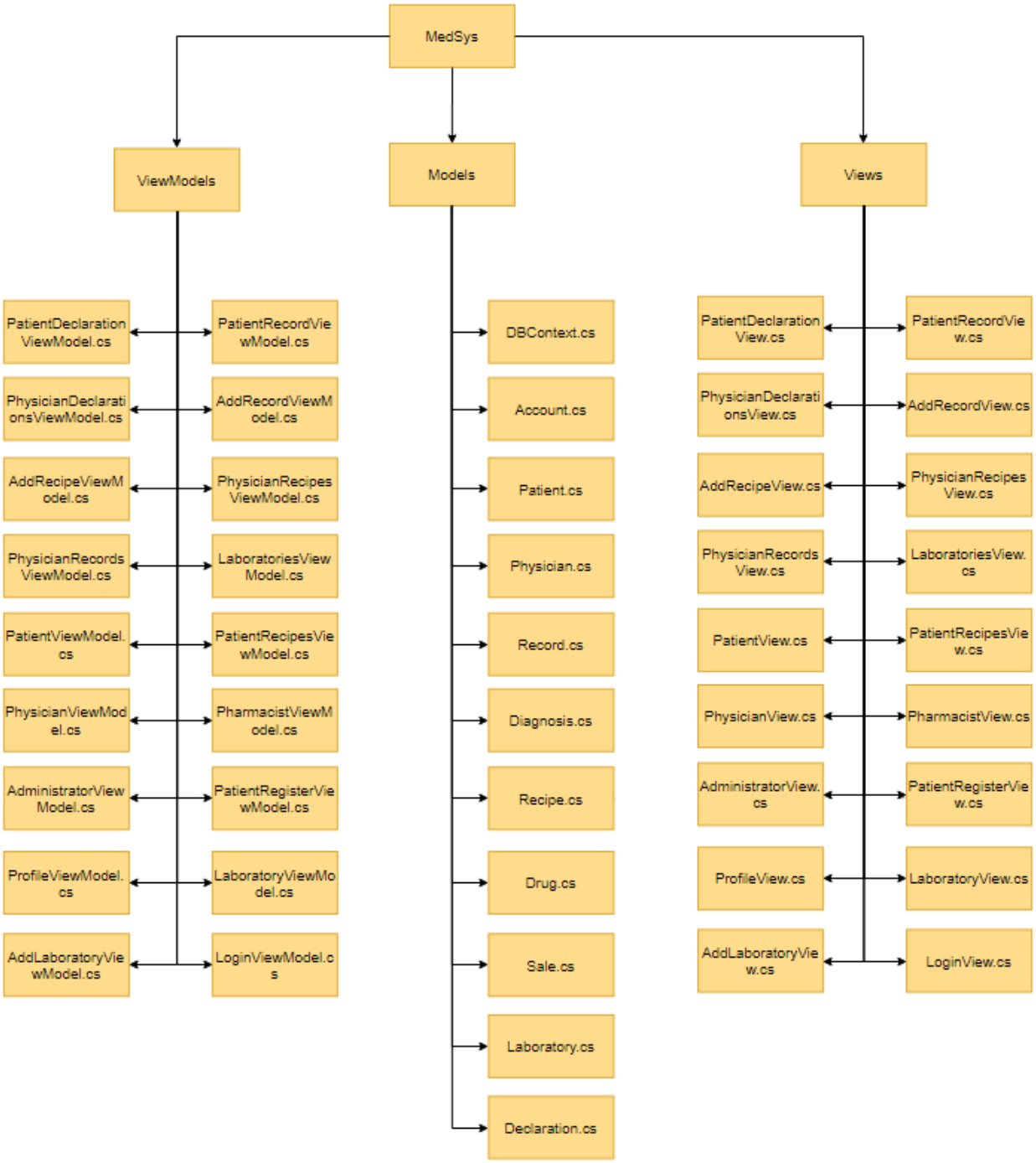

Рисунок 3.1 – Структура програмного забезпечення

Серед моделей представлень було розроблено наступні класи:

– PatientDeclarationViewModel – модель представлення початкової ініціалізації декларації з лікарем, виконаної пацієнтом;

– PatientRecordViewModel – модель представлення перегляду власних медичних записів пацієнтом;

– PhysicianDeclarationsViewModel – модель представлення роботи лікаря з укладеними з ним деклараціями та над підтвердженням ним декларацій;

– AddRecordViewModel – модель представлення створення нового індивідуального медичного запису пацієнта;

– AddRecipeViewModel – модель представлення виписування рецепту лікарем;

– PhysicianRecordsViewModel – модель представлення роботи лікаря зі всіма медичними записами пацієнта;

– PhysicianRecipesViewModel – модель представлення роботи лікаря зі всіма рецептами пацієнта;

– LaboratoriesViewModel – модель представлення роботи лікаря зі всіма лабораторними дослідженнями пацієнта;

– PatientViewModel – модель основного представлення роботи пацієнта з програмою;

– PatientRecipesViewModel – модель представлення перегляду власних рецептів пацієнтом;

– PhysicianViewModel – модель основного представлення роботи лікаря з програмою;

– PharmacistViewModel – модель основного представлення роботи аптекаря з програмою;

– AdministratorViewModel – модель основного представлення роботи адміністратора лікарні з системою;

– PatientRegisterViewModel – модель представлення реєстрації облікового запису пацієнта;

– ProfileViewModel – модель представлення для роботи з даними профілю облікового запису;

– LaboratoryViewModel – модель основного представлення для роботи представника лабораторії;

– AddLaboratoryViewModel – модель представлення внесення результатів лабораторного дослідження;

– LoginViewModel – модель представлення авторизації облікового запису.

Розроблені класи моделей:

– Account – модель облікового запису;

– Patient – модель пацієнта;

– Physician – модель лікаря, яку було створено на основі моделі Account, оскільки лікар потребує окремих засобів у своїй роботі, тому зручно було реалізувати роботу з його даними окремим класом;

– Record – модель медичного запису;

– Diagnosis – модель діагнозу;

– Recipe – модель рецепту;

– Drug – модель ліків;

– Sale – модель продажу ліків;

– Laboratory – модель лабораторного дослідження;

– Declaration – модель декларації.

Розроблені представлення застосунку:

– PatientDeclarationView – представлення початкової ініціалізації декларації з лікарем, виконаної пацієнтом;

– AddRecipeView – представлення виписування рецепту лікарем;

– PhysicianView – основне представлення роботи лікаря з програмою;

– ProfileView – представлення для роботи з даними профілю облікового запису;

– AddLaboratoryView – представлення внесення результатів лабораторного дослідження;

– LaboratoriesView – представлення роботи лікаря зі всіма лабораторними дослідженнями пацієнта;

– AddRecordView – представлення створення нового індивідуального медичного запису пацієнта;

– PharmacistView – основне представлення роботи аптекаря з програмою;

– AdministratorView – основне представлення роботи адміністратора лікарні з системою;

– PatientRecordView – представлення перегляду власних медичних записів пацієнтом;

– PhysicianDeclarationsView – представлення роботи лікаря з укладеними з ним деклараціями та над підтвердженням ним декларацій;

– PhysicianRecipesView – представлення роботи лікаря зі всіма медичними записами пацієнта;

– PhysicianRecordsView – представлення роботи лікаря зі всіма рецептами пацієнта;

– PatientView – основне представлення роботи пацієнта з програмою;

– PatientRecipesView – представлення перегляду власних рецептів пацієнтом;

– PatientRegisterView – представлення реєстрації облікового запису пацієнта;

– LaboratoryView – основне представлення для роботи представника лабораторії;

– LoginView – представлення авторизації облікового запису.

3.2 Опис прийнятих під час реалізації рішень

Клас DBContext реалізує доступ до засобів моделей у вигляді відповідних колекцій класу DbSet на основі атрибутів Accounts, Patients,

Physicians (DbSet<Physician>), Records, Diagnoses, Recipes, Drugs, Laboratories, Declarations.

Розроблені моделі представлень відповідають за взаємодію в межах інтерфейсу зв’язування даних з представленнями, підготовку даних в потрібній формі, реалізацію команд, що обробляють дії користувача.

Кожен клас моделей представлень реалізовано за допомогою наслідування інтерфейсу зв’язування властивостей INotifyPropertyChanged.

Реалізацію моделей представлень виконано на основі створення відповідного набору методів get і set для відповідних властивостей (полів, переліків тощо представлень), а також методів, що визначають команди, тобто дії, які виконуються за натискання на відповідні кнопки. Окрім того для кожного класу було створено базовий конструктор, який має параметри, що визначають вхідні властивості (наприклад, в моделях представлень роботи пацієнта це ідентифікаційний номер авторизованого користувача-пацієнта).

Кожен клас моделей представлень має посилання на контекст бази даних DBContext, що дозволяє завантажувати та змінювати дані за потреби.

Безпосередньо завантажування даних з БД реалізується через контекст роботи з базою даних DBContext:

```
db = new DBContext();
db.Physicians.Load();
ObservableCollection<Physician> phlist = db.Physicians.Local.
ToBindingList ();
```

Для підтримки роботи пошуку без необхідності натискання на окрему кнопку, тобто коли результати пошуку відображаються користувачу одразу під час введення запиту в поле пошуку, що є простішим і зручнішим для користувача підходом, коли він одразу в процесі роботи бачить, як введені ним дані змінюють результати, було використано реалізацію зміни переліку

результатів (у даному випадку переліку лікарів для укладання декларації) під час зміни (встановлення нового значення в полі пошуку) за допомогою set-методу:

```
SearchedPhysicianList.Clear();
foreach (Physician ps in FullPhysicianList)
{
   string fullname;
   fullname = ps.Surname;
   fullname = fullname + " " + ps.Name;
   fullname = fullname + " " + ps.FrName;
   if (fullname.Contains(_chosenPhys) || ps.Position.Contains
(_chosenPhys.ToUpper()))
      SearchedPhysicianList.Add(ps);
}
```

У даному випадку перевірка відбувалась одразу і за прізвищем, іменем та по батькові лікаря, і за його посадою, тобто користувач міг вводити і ті, і ті дані в рядок пошуку, і відбувався пошук даних як за посадою, так і за особистими даними.

Такий підхід є достатньо простим і зручним, роблячи процес пошуку для звичайного користувача, який може часом мати мінімальний досвід роботи з комп’ютерними засобами (що характерно для користувачів даних систем), достатньо простим.

Під час створення нової декларації з лікарем було використано підхід, який дозволяє створювати декларації не тільки з сімейними лікарями, але і з будь-якими іншими спеціалістами, але при цьому з кожним спеціалістом відповідного типу може бути укладено тільки одну декларацію. Тобто пацієнт не може мати двох сімейних лікарів або двох фізіотерапевтів, але він може мати декларацію з одним сімейним лікарем і з одним фізіотерапевтом.

Для цього було перед створенням нової декларації спочатку виконано перевірку наявності декларації зі спеціалістом даного типу в авторизованого пацієнта:

```
Declaration dec = db.Declarations.Find(Pat, ChosenPhys.Position);
```

У випадку відсутності такої декларації або у випадку підтвердження створення нової декларації відбувалось безпосередньо внесення даних в БД про нову декларацію та деактивація старої:

```
Declaration dec = db.Declarations.Add(Pat, ChosenPhys.AccountId, DateTime.Now, EndDate, true);
db.Declarations.Get(dec.DeclarationId).Status = false;
db.SaveChanges();
```

У даному випадку після успішного створення декларації потрібно повернутися до переліку індивідуальних медичних записів пацієнтів, оскільки саме це представлення PatientRecordView вважається базовим під час взаємодії, виходячи з обраної концепції програмного забезпечення:

```
PatientRecordView rv = new PatientRecordView (Pat);
rv.ShowDialog();
```

Таким чином, моделі представлень окрім передачі даних між моделями і представленням виконують функції керування відображенням представлень за необхідності та завантаження потрібних даних з моделей також за відповідної необхідності. Основою цього є механізм прив’язування даних.

3.3 Висновки за розділом 3

За результатами реалізації програмного забезпечення керування індивідуальними медичними записами пацієнтів представлено структуру розробленого рішення. Описано основний процес реалізації відповідних класів моделей, моделей представлень та представлень.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення програми

Розроблене програмне забезпечення призначене для створення централізованого доступу до медичних даних пацієнтів, які вносяться в систему лікарями, представниками лабораторій. Всі ці дані потрібні пацієнтам, лікарям для прийняття їх рішень і є основою для створення нових таких записів, рецептів.

4.2 Умови виконання програми

Технічні засоби клієнтської сторони мають відповідати наступним:

– стаціонарний або мобільний (ноутбук) комп'ютер під керуванням операційної системи Windows, починаючи з версії 7;

– процесор не нижче Intel Celeron з тактовою частотою не менше 300 МГц;

– не менше 1 Гб оперативної пам’яті;

– 100 Мб вільного простору на жорсткому диску;

– монітор і відеокарта, які мають забезпечувати роботу з роздільною здатністю не менше 800x600.

4.3 Виконання програми

Для роботи програми спочатку має бути забезпечено запуск сервера бази даних.

Після реалізації авторизації виконується вибір потрібного сценарію роботи в залежності від ролі, визначеної типом облікового запису.

На рис. 4.1 представлено основну форму роботи лікаря.

Лікар може перейти до перегляду укладених з ним декларацій та підтвердження ініційованих пацієнтами декларацій (тільки після цього декларація буде вважатися укладеною). Для цього необхідно натиснути на кнопку «Декларації».

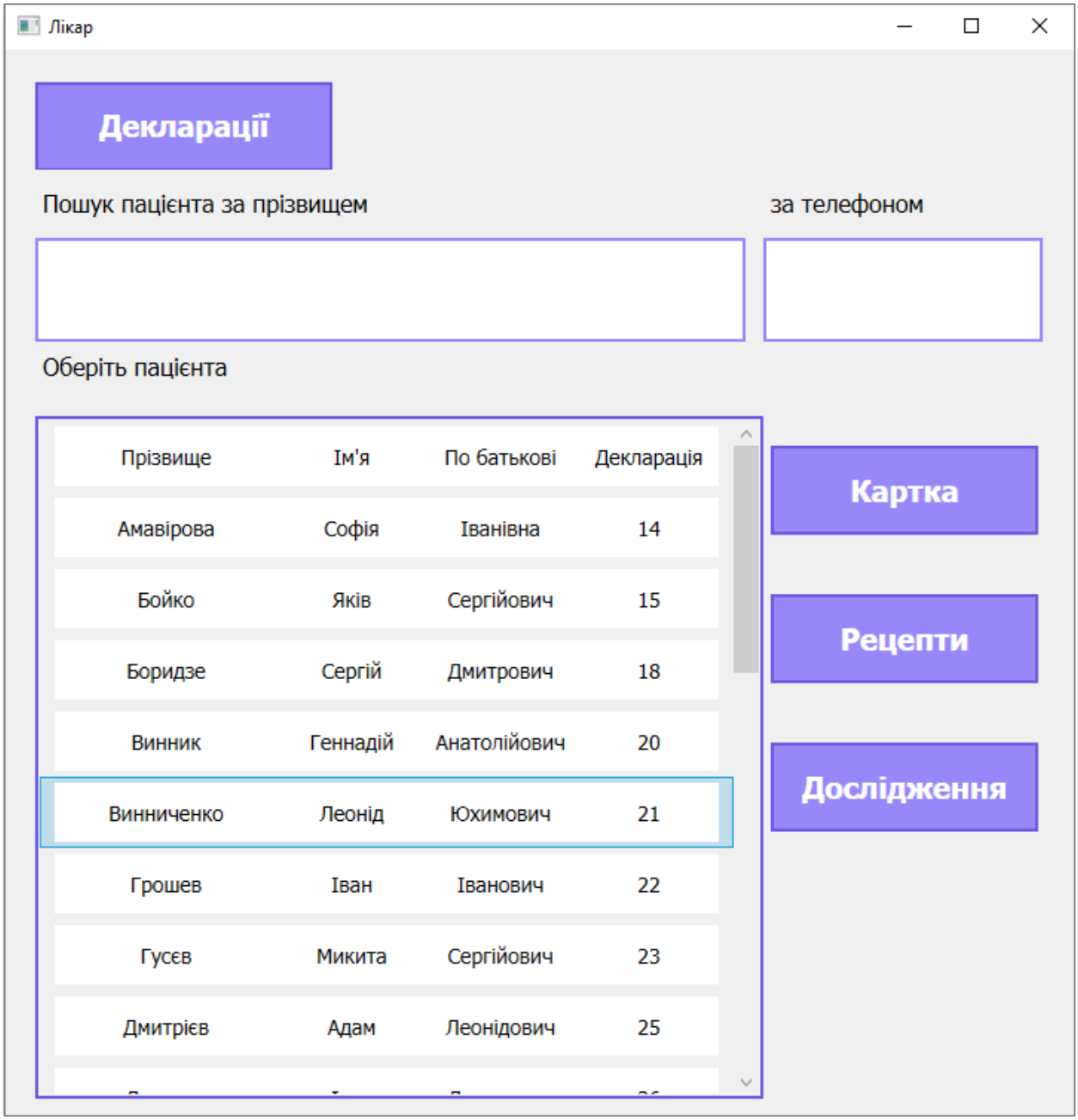

Рисунок 4.1 – Основна форма роботи лікаря

Основний сценарій роботи лікаря з програмою включає вибір його пацієнта. Для цього передбачено таблицю зі всіма пацієнтами, які уклали з авторизованим лікарем декларацію. Можна обрати конкретного пацієнта з

таблиці або ввести прізвище пацієнта в поле «Пошук пацієнта за прізвищем» або номер телефону в поле «за телефоном», тоді зміст таблиці одразу буде скорочено і він буде відповідати заданим критеріям пошуку (рис. 4.2).

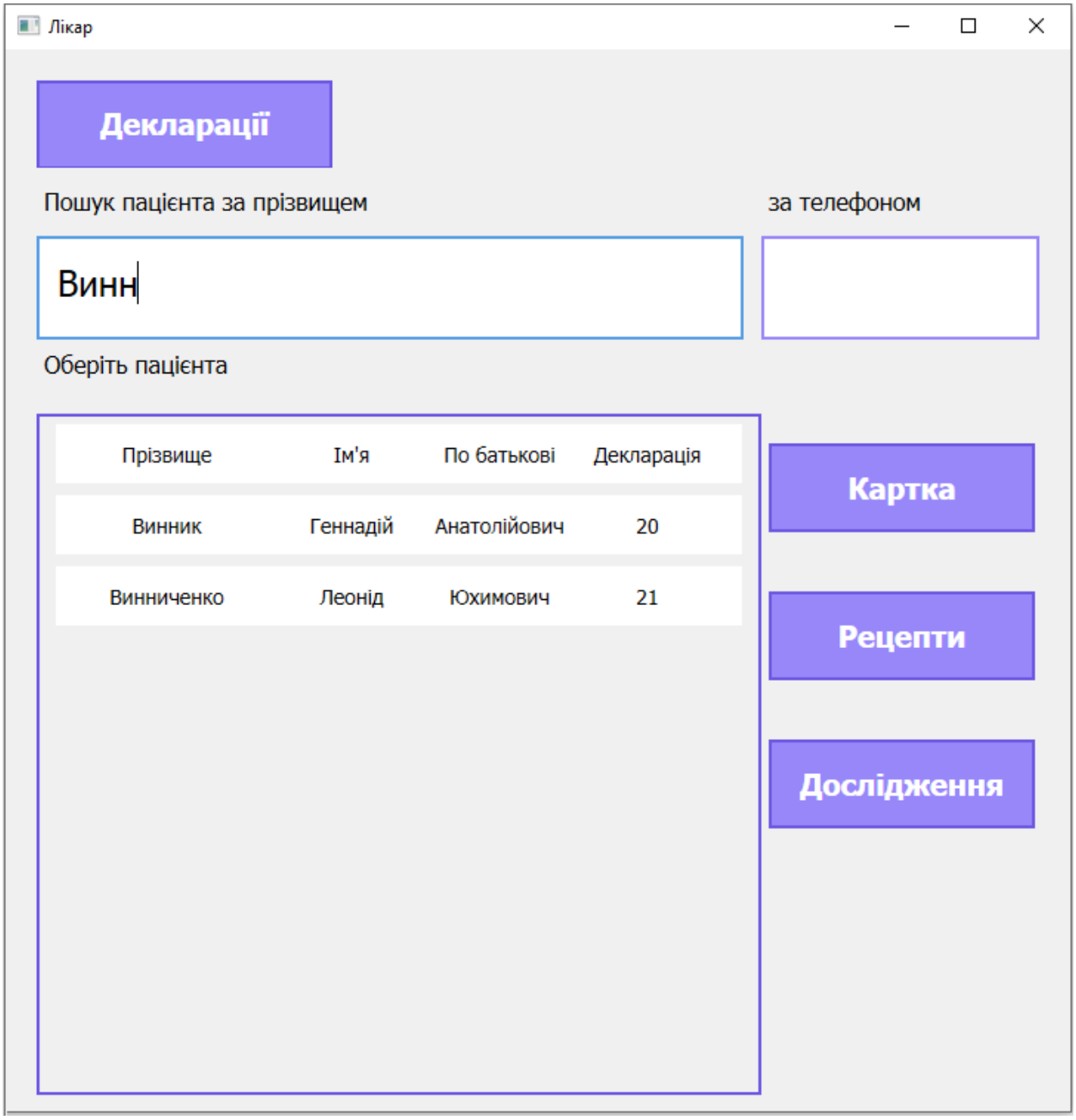

Рисунок 4.2 – Пошук лікарем пацієнта

Далі потрібно обрати конкретного пацієнта з таблиці і натиснути одну з кнопок. Кнопка «Картка» призначена для переходу до форми перегляду і редагування медичних записів пацієнта, звідти можна перейти до створення нового запису (рис. 4.3).

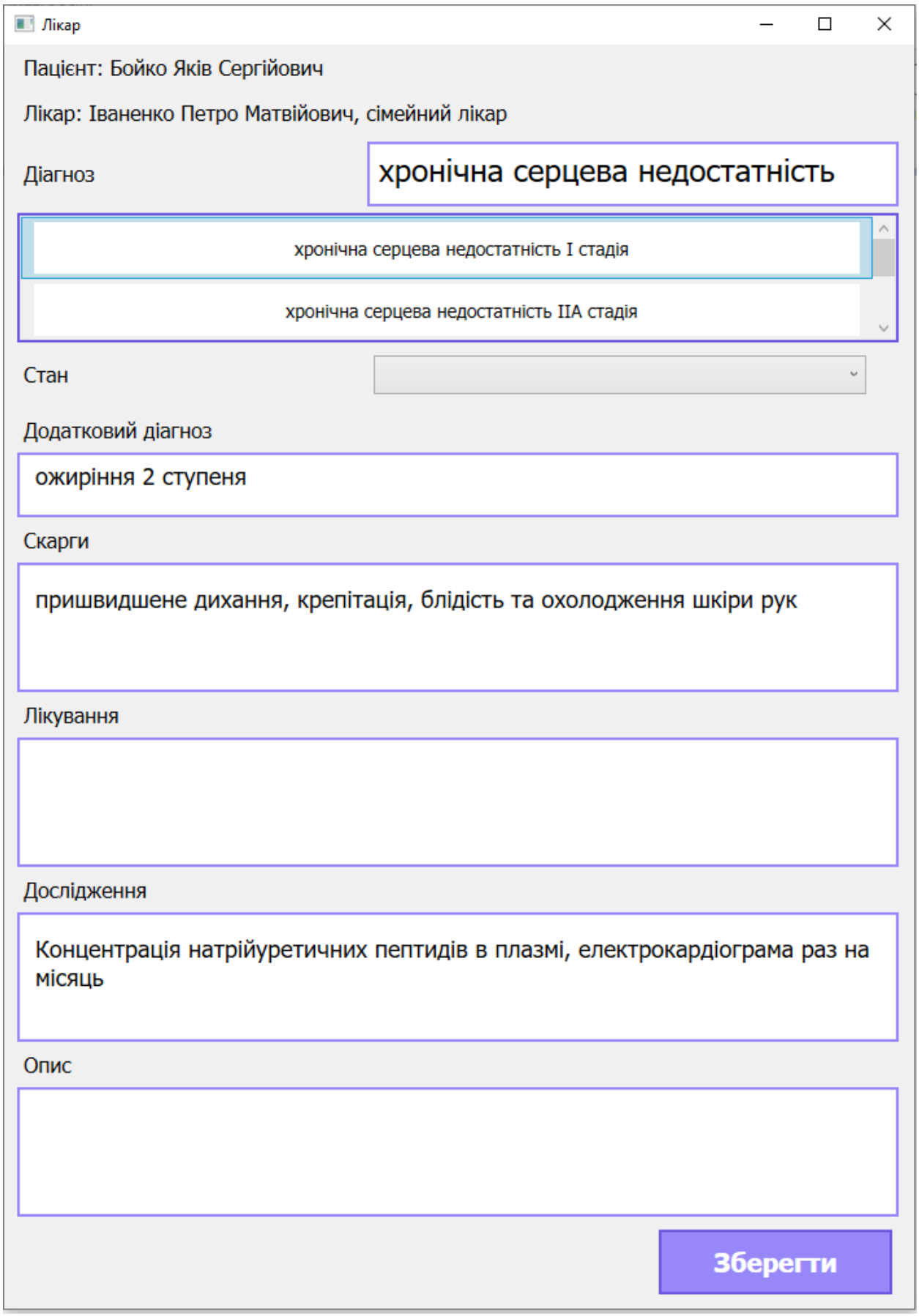

Рисунок 4.3 – Внесення медичного запису лікарем

Кнопка «Рецепти» основної форми роботи лікаря призначена для переходу до форми перегляду і редагування рецептів пацієнта, звідти можна перейти до створення нового рецепта;

Кнопка «Дослідження» основної форми роботи лікаря призначена для переходу до форми перегляду лабораторних досліджень пацієнта.

На формі створення нового медичного запису потрібно ввести діагноз в поле пошуку, тоді нижче в таблиці буде відображено відомі діагнози, які відповідають заданому. З таблиці потрібно обрати той, що відповідає необхідному.

Далі потрібно обрати стан діагнозу з випадаючого переліку (первинний, повторний), задати за наявності додатковий діагноз, вказати скарги, описати процес лікування, необхідні дослідження, а також задати будь-який додатковий опис. Для підтвердження внесених даних потрібно натиснути на кнопку «Зберегти».

Основна форма роботи пацієнта (рис. 4.4) складається з наступних груп:

– групи декларацій, звідки можна переглянути всі укладені декларації пацієнта та перейти до укладання нової шляхом натискання на кнопку «Укласти»;

– групи медичних записів, де пацієнт може переглянути всі власні медичні записи, а також задати потрібний проміжок дат, в який має входити медичний запис, пошук якого відбувається, а обравши один з записів з таблиці, можна натиснути на кнопку «Детальніше» і перейти до окремої форми перегляду всіх даних запису;

– групи рецептів, де пацієнт може переглянути всі власні рецепти, а також задати потрібний проміжок дат, в який має входити рецепт, пошук якого відбувається, або вказати номер рецепта, а обравши один з рецептів з таблиці, можна натиснути на кнопку «Детальніше» і перейти до окремої форми перегляду всіх даних рецепту;

– групи лабораторних досліджень, де пацієнт може переглянути всі власні дослідження, а також, натиснувши на кнопку «Детальніше», перейти до окремої форми перегляду всіх даних дослідження.

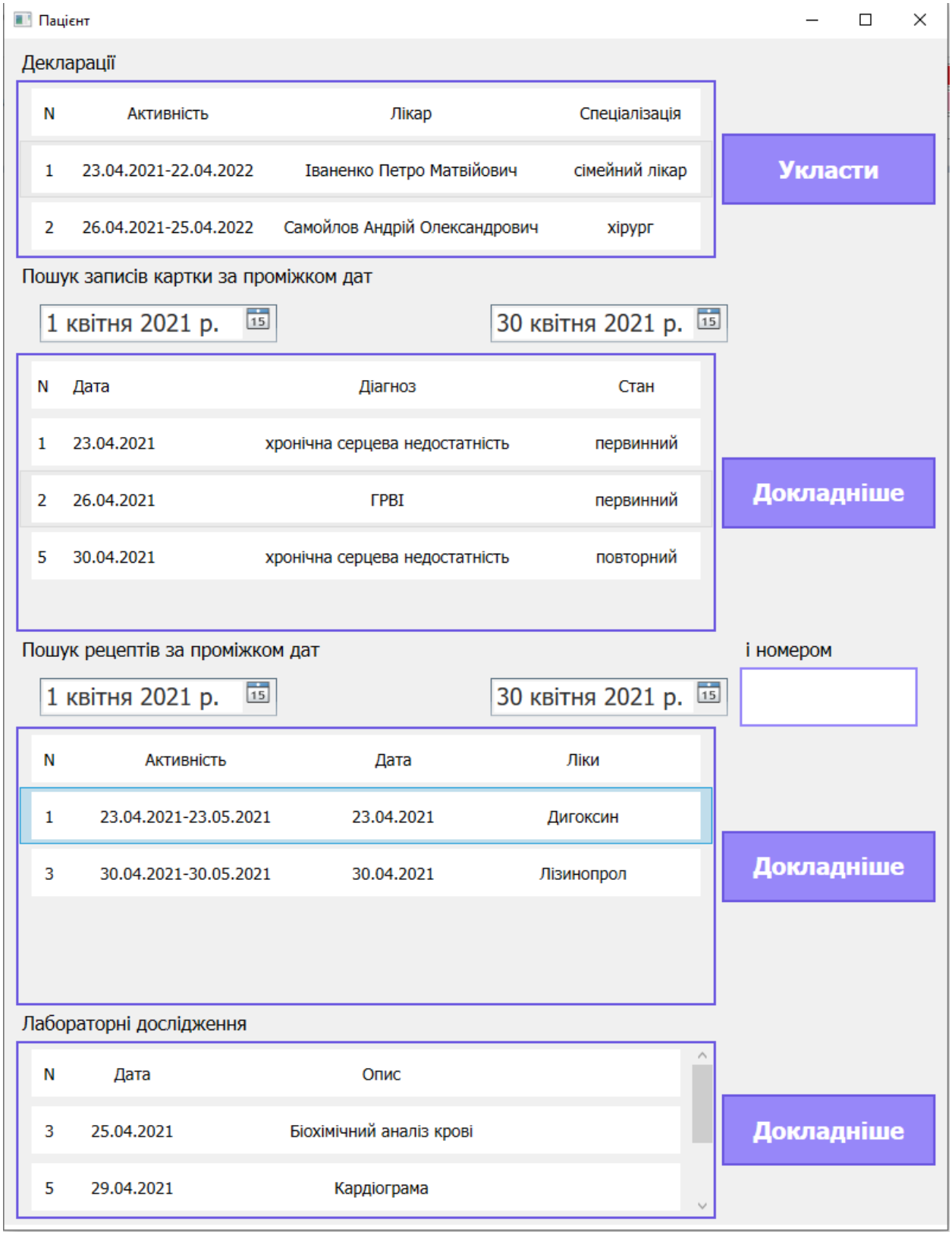

Рисунок 4.4 – Основна форма роботи пацієнта

У випадку натискання на кнопку «Укласти» відкривається окрема форма (рис. 4.5), де можна виконати пошук лікаря, обрати його з відповідної таблиці, зафіксувати дату початку дії декларації та підтвердити вибір кнопкою «Укласти».

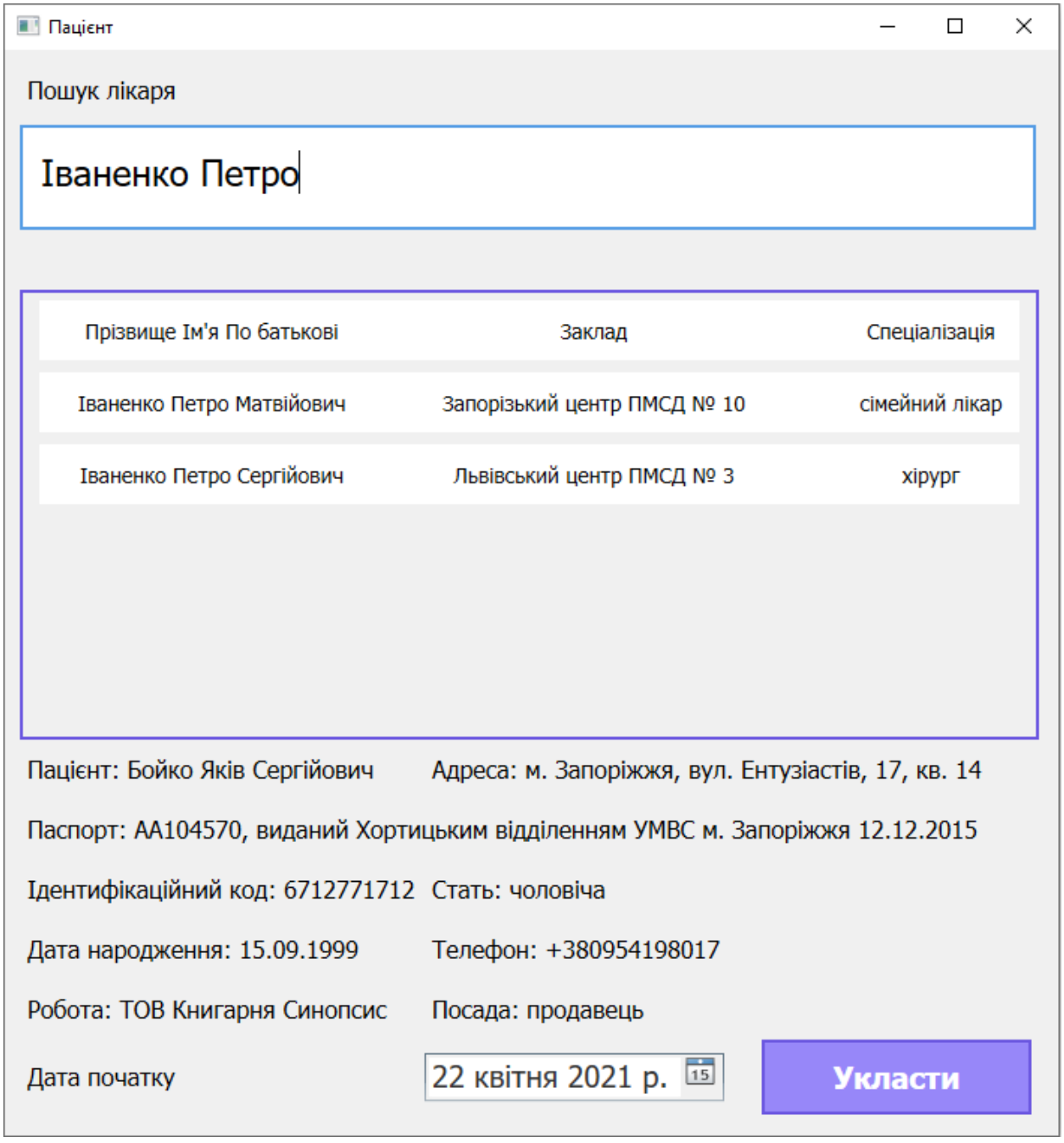

Рисунок 4.5 – Укладання декларації пацієнтом

Аптекар під час роботи з програмою на основній формі (рис. 4.6) виконує пошук рецепта, для цього він має задати вірний номер рецепта та

дані пацієнта, що включають або прізвище, або номер телефону. Тільки при співпадінні цих даних буде виведено переліків ліків рецепту.

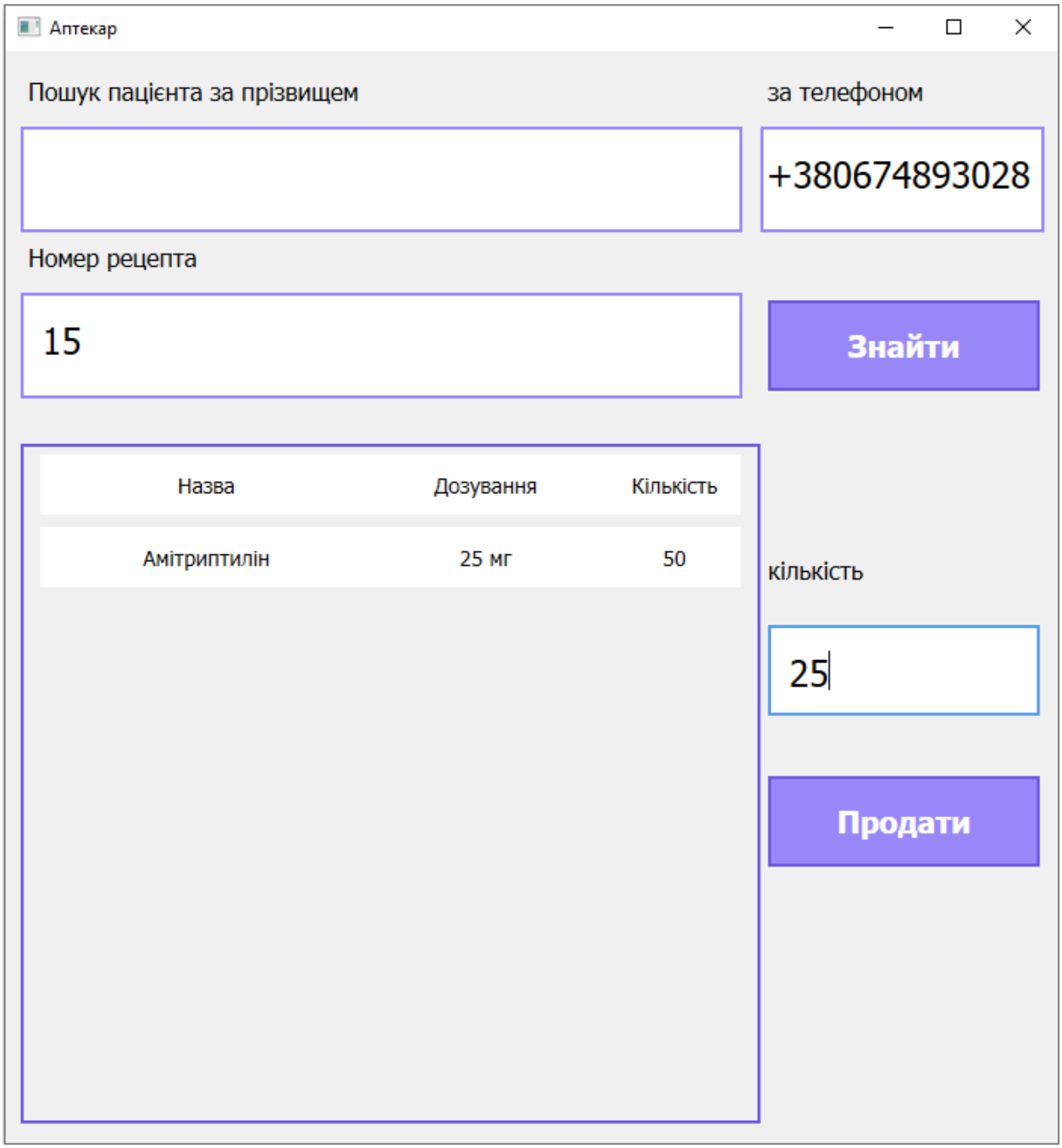

Рисунок 4.6 – Основна форма роботи аптекаря

Аптекар може задати кількість ліків, які він продає, та підтвердити продаж натисканням на кнопку «Продати».

Адміністратор лікарні під час своєї роботи на основній формі виконує пошук пацієнтів, для яких він може редагувати профіль та вносити дані про сплату послуг, має доступ до форм реєстрації пацієнта та лікаря.

Представник лабораторії виконує на основній формі пошук пацієнтів, а для знайденого пацієнта може переглядати його попередні дослідження, редагувати їх дані, виконувати пошук досліджень, а також вносити нові результати дослідження і вносити дані про їх сплату.

ВИСНОВКИ

Роботу присвячено проблемі розроблення програмного забезпечення керування індивідуальними медичними записами пацієнтів.

Розглянуто основні типи медичних систем та особливості їх застосування для керування індивідуальними медичними записами пацієнтів. Проаналізовано програмні аналоги, включаючи Kareo Clinical, WRS Health, iPatientCare.

Сформульовано функціональні вимоги до програмного забезпечення з відповідними сценаріями роботи адміністратора лікарні, лікаря, пацієнта, представників аптек та лабораторій, для яких приведено відповідні діаграми прецедентів.

За результатами порівняння обрано мову програмування C# і систему керування БД Microsoft SQL Server в якості засобів розробки програмного забезпечення.

Виконано проєктування БД з визначенням структури БД і описом відповідних таблиць. Обрано архітектурний шаблон MVVM для побудови програмного забезпечення, що дозволяє в подальшому зручним чином розширювати за необхідності систему.

Розроблено моделі, моделі представлень та представлення для реалізації функціональних вимог. Описано процес роботи користувачів з програмою для виконання відповідних функцій.

Розроблене програмне забезпечення дозволяє виконувати внесення і редагування медичних записів пацієнта, перегляд всіх медичних записів пацієнта, пошук медичних записів пацієнта за заданий період, укладання декларації пацієнта з лікарем з відповідним підтвердженням з двох сторін, внесення даних профілю пацієнта під час реєстрації, пошук пацієнтів, внесення, редагування, пошук і перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта, внесення даних про

сплату послуг, виписування, редагування, перегляд і пошук рецептів, продаж ліків за рецептом.

Завдання дипломної кваліфікаційної роботи в результаті повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. EMR & EHR Software [Electronic resource] : Electronic Medical Record Service. – Access mode : https://www.advancedmd.com/emr-ehr-software/

2. Top Electronic Medical Records Software – 2021 Reviews [Electronic resource]. – Access mode : https://www.softwareadvice.com/medical/electronic-medical-record-software-comparison/#buyers-guide

3. Best Electronic Medical Record (EMR) Software List 2021 [Electronic resource]. – Access mode : https://www.emrfinder.com/all-emr-software/

4. Medical Software for Your Independent Practice | Kareo [Electronic resource]. – Access mode : https://www.kareo.com/

5. Ferrill, P. The Best Electronic Medical Record (EMR) Managers [Electronic resource] / P. Ferrill, T. Liddle. – Access mode : https://www.pcmag.com/picks/the-best-electronic-medical-record-emr-managers

6. De Muro, J. P. Kareo review [Electronic resource] / J. P. De Muro. – Access mode : https://www.techradar.com/reviews/kareo

7. Turner, B. Best Electronic Health Record (EHR) software of 2021 [Electronic resource] / Brian Turner. – Access mode : https://www.techradar.com/best/best-electronic-health-record-ehr-software

8. Kareo Clinical HER Software – EHR pricing, demo & Comparison [Electronic resource]. – Access mode : https://www.ehrinpractice.com/kareo-clinical-ehr-software-profile.html

9. Cloud-Based EHR Software And Practice Management Software | WRS Health [Electronic resource]. – Access mode : https://www.wrshealth.com/

10. Ferrill, P. WRS Health Review [Electronic resource] / P. Ferrill, T. Liddle. – Access mode : https://www.pcmag.com/reviews/wrs-health

11. WRS Health – Person Centered Tech [Electronic resource]. – Access mode : https://personcenteredtech.com/vendorreview/wrs-health/

12. Certified Ambulatory EHR, Practice Management, Medical Billing

Software, Revenue Cycle Management [Electronic resource]. – Access mode : https://ipatientcare.com/

13. iPatientCare EHR Software – 2021 Reviews, Pricing & Demo [Electronic resource]. – Access mode : https://www.softwareadvice.com/ie/medical/ipatientcare-profile/

14. Horstmann, C. S. Core Java Volume I – Fundamentals [Text] / C. S. Horstmann. – Boston : Prentice Hall, 2018. – 889 p.

15. Price, M. J. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development [Text] : Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code / M. J. Price. – Birmingham : Packt Publishing, 2019. – 818 p.

16. Skeet, J. C# in Depth [Text] : Fourth Edition / J. Skeet. – New York : Manning Publications, 2019. – 528 p.

17. Griffiths, I. Programming C# 8.0 [Text] : Build Windows, Web, and Desktop Applications / I. Griffiths. – Farnham : O'Reilly UK Ltd., 2020. – 779 p.

18. Elk, K. SQL Server with C# [Text] / K. Elk. – Scotts Valley : CreateSpace Independent Publishing Platform, 2019. – 146 p.

19. Шаблон Model-View-ViewModel [Электрон. ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

ДОДАТОК А

Технічне завдання

Вступ

Програмне забезпечення, яке розробляється в роботі, повинно надати мультикористувацький доступ до роботи з індивідуальними медичними записами пацієнтів, адже саме ці дані є основою подальшої роботи різних спеціалістів.

А.1 Підстави для розробки

Тема дипломної кваліфікаційної роботи була визначена як «Розробка програмного забезпечення керування індивідуальними медичними записами пацієнтів» на основі наказу № 103 від 30 березня 2021 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Призначенням розробки є створення централізованого доступу до медичних даних пацієнтів, які вносяться в систему лікарями, представниками лабораторій. Всі ці дані потрібні пацієнтам, лікарям для прийняття їх рішень і є основою для створення нових таких записів, рецептів.

Таким чином, програма має убезпечити різних спеціалістів медичної сфери від необхідності повторного внесення даних в систему.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програма повинна надавати можливість виконання за її допомогою наступних функцій:

– внесення і редагування медичних записів пацієнта;

– перегляд всіх медичних записів пацієнта;

– пошук медичних записів пацієнта;

– укладання декларації пацієнта з лікарем;

– пошук пацієнтів;

– внесення і редагування результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– перегляд результатів проведених лабораторних досліджень стану здоров’я пацієнта;

– пошук результатів проведених лабораторних досліджень;

– внесення даних про сплату послуг;

– виписування рецептів;

– редагування рецептів;

– перегляд рецептів;

– пошук рецептів;

– продаж ліків за рецептом;

– авторизація облікового запису;

– реєстрація облікового запису;

– внесення даних профілю пацієнта.

А.3.2 Вимоги до надійності

Під час роботи програми має виконуватися перевірка внесення даних в БД. У випадку якщо такі запити не досягли результатів, має виводитися відповідне повідомлення.

При внесенні даних має виконуватися також перевірка заповнення всіх обов’язкових полів.

А.3.3 Умови експлуатації

Для забезпечення віддаленої роботи з програмою потрібно забезпечити на сервері доступ до БД.

Всі інші екземпляри програми запускаються на клієнтських комп’ютерах.

А.3.4 Вимоги до складу та параметрів технічних засобів

Технічні засоби клієнтської сторони мають відповідати наступним:

– стаціонарний або мобільний (ноутбук) комп'ютер під керуванням операційної системи Windows, починаючи з версії 7;

– процесор не нижче Intel Celeron з тактовою частотою не менше 300 МГц;

– не менше 1 Гб оперативної пам’яті;

– 100 Мб вільного простору на жорсткому диску;

– монітор і відеокарта, які мають забезпечувати роботу з роздільною здатністю не менше 800x600.

Серверні вимоги мають відповідати вимогам до серверів Microsoft SQL Server.

А.3.5 Вимоги до транспортування та збереження

Можливе транспортування і зберігання програмного забезпечення на фізичних носіях, хоча достатньо збереження в репозиторії на сервері.

А.3.6 Вимоги до програмної документації

Окремі програмні документи окрім технічного завдання не передбачаються. Технічне завдання має обов’язково бути доданим до пояснювальної записки.

Опис процедури взаємодії користувачів з програмою забезпечується в четвертому розділі пояснювальної записки.

А.4 Порядок контролю та приймання

Контроль виконання дипломної кваліфікаційної роботи забезпечується керівником. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б

Текст програми

Б.1 Текст файлу PatientDeclarationViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

using MedSys.Models;
using MedSys.Views;

namespace MedSys.ViewModels
{
    public class DeclarationViewModel : INotifyPropertyChanged
    {

        DBContext db;
        int Pat;
        DateTime Nowtime;
        private RelayCommand _declare;

        public DeclarationViewModel (int patientId)
        {

            db = new DBContext();

            Pat = patientId;

            db.Physicians.Load();
            ObservableCollection<Physician> phlist =
db.Physicians.Local.ToBindingList();
            FullPhysicianList = new
ObservableCollection<Physician>();

            foreach (Physician ps in phlist)
            {
                FullPhysicianList.Add(new Physician (ps));
            }

            PhysSearchParam = "";
```

```
                ChosenPhys = new Physician ();
                Nowtime = DateTime.Now;
                EndDate = Nowtime.AddYears(1);
            }

            private ObservableCollection<Physician>
_fullPhysicianList;
            public ObservableCollection<Physician> FullPhysicianList
            {
                get { return _fullPhysicianList; }
                set
                {
                    _fullPhysicianList = value;
                    OnPropertyChanged("Physicians");
                }
            }

            private ObservableCollection<Physician>
_searchedPhysicianList;
            public ObservableCollection<Physician>
SearchedPhysicianList
            {
                get { return _searchedPhysicianList; }
                set
                {
                    _searchedPhysicianList = value;
                    OnPropertyChanged("SearchPhysicians");
                }
            }

            private Physician _chosenPhys;
            public Physician ChosenPhys
            {
                get { return _chosenPhys; }
                set
                {
                    _chosenPhys = value;
                    OnPropertyChanged("Physician");
                }
            }

            private DateTime _nowtime;
            public DateTime Nowtime
```

```
{
    get { return _nowtime; }
    set
    {
        _nowtime = value;
        OnPropertyChanged("Nowtime");
    }
}

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

private string _physSearchParam;
public string PhysSearchParam
{
    get { return _physSearchParam; }
    set
    {
        _physSearchParam = value;
        if (_physSearchParam == "")
        {
            SearchedPhysicianList.Clear();

            foreach (Physician ps in FullPhysicianList)
            {
                   SearchedPhysicianList.Add(ps);
            }
        }
        else
        {
            SearchedPhysicianList.Clear();

            foreach (Physician ps in FullPhysicianList)
            {
                string fullname;
```

```
                                fullname = ps.Surname;
                                fullname = fullname + " " + ps.Name;
                                fullname = fullname + " " + ps.FrName;
                                if (fullname.Contains(_chosenPhys) ||
ps.Position.Contains(_chosenPhys.ToUpper()))
                                    SearchedPhysicianList.Add(ps);
                            }
                        }
                        OnPropertyChanged("SearchPhysicians");
                    }
                }

                public RelayCommand Declare
                {
                    get
                    {
                        return _declare?? (new RelayCommand(obj =>
                        {
                            if (!ChosenPhys.IsNull())
                            {
                                Declaration dec =
db.Declarations.Find(Pat, ChosenPhys.Position);
                                if (!db.IsNull())
                                    if (MessageBox.Show("Ви вже маєте
поточну декларацію, укладену з лікарем з заданою спеціалізацією. У
випадку підтвердження дії поточну декларацію буде розірвано і укладено
нову", "Підтвердження", MessageBoxButtons.YesNo,
MessageBoxIcon.Question) == DialogResult.Yes)
                                    {
                                        if (!EndDate.IsNull())
                                        {
                                            Declaration dec =
db.Declarations.Add(Pat, ChosenPhys.AccountId, DateTime.Now, EndDate,
true);

db.Declarations.Get(dec.DeclarationId).Status = false;
                                            db.SaveChanges();
                                            if (!dec.IsNull())
                                            {
                                                PatientRecordView rv =
new PatientRecordView (Pat);
                                                rv.ShowDialog();
                                            }
```

```
                                        else
                                        {
                                            MessageBox.Show("Не
вдалося створити декларацію. Спробуйте пізніше", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
                                        }
                                    }
                                    else
                                    {
                                        MessageBox.Show("Для
укладання декларації потрібно задати дату завершення", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
                                    }
                                }
                                else
                                {
                                    if (!EndDate.IsNull())
                                    {
                                        Declaration dec =
db.Declarations.Add(Pat, ChosenPhys.AccountId, DateTime.Now, EndDate,
true);
                                        db.SaveChanges();
                                        if (!dec.IsNull())
                                        {
                                            PatientRecordView rv =
new PatientRecordView (Pat);
                                            rv.ShowDialog();
                                        }
                                        else
                                        {
                                            MessageBox.Show("Не
вдалося створити декларацію. Спробуйте пізніше", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
                                        }
                                    }
                                    else
                                    {
                                        MessageBox.Show("Для
укладання декларації потрібно задати дату завершення", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
                                    }
                                }
                            else
```

```
                        {
                            MessageBox.Show("Для укладання догору
спочатку потрібно обрати лікаря.", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                        }
                    }));
                }
            }    }
        }
```

Б.2 Текст файлу PatientRecordViewModel.cs

```
        using System;
        using System.Collections.Generic;
        using System.Collections.ObjectModel;
        using System.Linq;
        using System.Text;
        using System.Threading.Tasks;
        using System.Windows;

        using MedSys.Models;
        using MedSys.Views;

        namespace MedSys.ViewModels
        {
            class PatientRecordViewModel : INotifyPropertyChanged
            {

                DBContext db;
                int Pat;

                public PatientRecordViewModel () {

                    db = new DBContext();

                    Pat = patientId;
                    FullRecordList = new ObservableCollection<Record>();
                    PhysicianList = new
ObservableCollection<Physician>();

                    db.Declarations.Load();
```

```
                ObservableCollection<Declaration> declist =
db.Declarations.Local.ToBindingList();

                foreach (Declaration d1 in declist)
                {
                    db.Records.Where(x => x.DeclarationId ==
d1).Load();
                    reclist = db.Records.Local.ToBindingList();

                    foreach (Record rl in reclist)
                    {
                        FullRecordList.Add(new Record (rl));
                    }

                    Declaration dc =
db.Declarations.Find(d1.DeclarationId);
                    int pnum = dc.DoctorId;
                    Account ac = db.Accounts.Find(pnum);
                    Physician dr = new Physician (ac);
                    if (!PhysicianList.Contains(dr))
                        PhysicianList.Add(dr);
                }

                Physname = "";
                ChosenPhys = new Physician ();
                RecordId = -1;
                Diagnosisname = "";
                Adddiagnosis = "";
                RecDate = new DateTime();
                Problems = "";
                Therapy = "";
                Research = "";
                Description = "";
                Diagnosisstate = -1;
                BeginDate = new DateTime();
                EndDate = new DateTime();
                EndDate = Nowtime.AddYears(1);
            }

            private Physician _chosenRecord;
            public Physician ChosenRecord
            {
                get { return _chosenRecord; }
```

```
                set
                {
                    _chosenRecord = value;
                    RecordId = ChosenRecord.RecordId;
                    int dnum = ChosenRecord.DeclarationId;
                    Declaration dc = db.Declarations.Find(dnum);
                    int pnum = dc.DoctorId;
                    Account ac = db.Accounts.Find(pnum);
                    Physname = ac.Surname + " " + ac.Name + " " +
ac.FrName;
                    int dnum = ChosenRecord. DiagnosisId;
                    Diagnosis dgn = db.Diagnoses.Find(dnum);
                    Diagnosisname = dgn.Title;
                    Diagnosisstate = ChosenRecord.DiagnosisState;
                    Adddiagnosis = ChosenRecord.AddDiagnosis
                    RecDate = ChosenRecord.RecDate;
                    Problems = ChosenRecord.Problems;
                    Therapy = ChosenRecord.Therapy;
                    Research = ChosenRecord.Research;
                    Description = ChosenRecord.Description;
                    OnPropertyChanged("ChosenRecord");
                }
            }

            private Physician _chosenPhys;
            public Physician ChosenPhys
            {
                get { return _chosenPhys; }
                set
                {
                    _chosenPhys = value;
                    if (_chosenPhys.IsNull())
                    {
                        FullRecordList.Clear();
                        db.Records.Load();
                        ObservableCollection<Record> reclist =
db.Records.Local.ToBindingList();
                        foreach (Record rl in reclist)
                        {
                            reclist.Add(new Record (rl));
                        }
                    }
                    else
```

```
                        {
                            FullRecordList.Clear();
                            int pnum = _chosenPhys.AccountId;
                            Account ac = db.Accounts.Find(pnum);
                            Physname = ac.Surname + " " + ac.Name + " "
+ ac.FrName;
                            Declaration dc = db.Declarations.Get(Pat,
pnum);
                            db.Records.Where(x => x.DeclarationId ==
dc.DeclarationId).Load();
                            reclist = db.Records.Local.ToBindingList();
                        }

                        foreach (Record rl in reclist)
                        {
                            bool flag1 = true;
                            if (!BeginDate.IsNull())
                                flag1 = rl.RecDate >= BeginDate;
                            bool flag2 = true;
                            if (!EndDate.IsNull())
                                flag2 = rl.RecDate <= EndDate;
                            if (flag1 && flag2)
                                FullRecordList.Add(new Record (rl));
                        }
                        OnPropertyChanged("Physician");
                    }
                }

                private Physician _recordid;
                public Physician RecordId
                {
                    get { return _chosenRecord; }
                    set
                    {
                        _chosenRecord = value;
                        OnPropertyChanged("Record");
                    }
                }

                private string _physname;
                public string Physname
                {
                    get { return _physname; }
```

```
    set
    {
        _physname = value;
        OnPropertyChanged("PhysicianName");
    }
}

private string _diagnosisname;
public string Diagnosisname
{
    get { return _diagnosisname; }
    set
    {
        _diagnosisname = value;
        OnPropertyChanged("Diagnosisname");
    }
}

private string _adddiagnosis;
public string Adddiagnosis
{
    get { return _adddiagnosis; }
    set
    {
        _adddiagnosis = value;
        OnPropertyChanged("AdditionalDiagnosis");
    }
}

private DateTime _recdate;
public DateTime RecDate
{
    get { return _recdate; }
    set
    {
        _recdate = value;
        OnPropertyChanged("RecordDate");
    }
}

private int _Diagnosisstate;
public int Diagnosisstate
{
```

```
    get { return _Diagnosisstate; }
    set
    {
        _Diagnosisstate = value;
        OnPropertyChanged("Diagnosisstate");
    }
}

private string _problems;
public string Problems
{
    get { return _problems; }
    set
    {
        _problems = value;
        OnPropertyChanged("RecordProblems");
    }
}

private string _therapy;
public string Therapy
{
    get { return _therapy; }
    set
    {
        _therapy = value;
        OnPropertyChanged("RecordTherapy");
    }
}

private string _research;
public string Research
{
    get { return _research; }
    set
    {
        _research = value;
        OnPropertyChanged("RecordResearch");
    }
}
private string _description;
public string Description
{
```

```
                get { return _description; }
                set
                {
                    _description = value;
                    OnPropertyChanged("RecordDescription");
                }
            }

            private ObservableCollection<Record> _fullRecordList;
            public ObservableCollection<Record> FullRecordList
            {
                get { return _fullRecordList; }
                set
                {
                    _fullRecordList = value;
                    OnPropertyChanged("Records");
                }
            }

            private ObservableCollection<Physician> _physicianList;
            public ObservableCollection<Physician> PhysicianList
            {
                get { return _physicianList; }
                set
                {
                    _physicianList = value;
                    OnPropertyChanged("DecPhysicians");
                }
            }

            private DateTime _begindate;
            public DateTime BeginDate
            {
                get { return _begindate; }
                set
                {
                    _begindate = value;
                    if (_chosenPhys.IsNull())
                    {
                        FullRecordList.Clear();
                        db.Records.Load();
                        ObservableCollection<Record> reclist =
db.Records.Local.ToBindingList();
```

```
                        foreach (Record rl in reclist)
                        {
                            reclist.Add(new Record (rl));
                        }
                    }
                    else
                    {
                        FullRecordList.Clear();
                        int pnum = _chosenPhys.AccountId;
                        Account ac = db.Accounts.Find(pnum);
                        Physname = ac.Surname + " " + ac.Name + " "
+ ac.FrName;
                        Declaration dc = db.Declarations.Get(Pat,
pnum);
                        db.Records.Where(x => x.DeclarationId ==
dc.DeclarationId).Load();
                        reclist = db.Records.Local.ToBindingList();
                    }

                    foreach (Record rl in reclist)
                    {
                        bool flag1 = true;
                        if (!BeginDate.IsNull())
                            flag1 = rl.RecDate >= BeginDate;
                        bool flag2 = true;
                        if (!EndDate.IsNull())
                            flag2 = rl.RecDate <= EndDate;
                        if (flag1 && flag2)
                            FullRecordList.Add(new Record (rl));
                    }
                    OnPropertyChanged("BeginDate");
                }
            }

            private DateTime _enddate;
            public DateTime EndDate
            {
                get { return _enddate; }
                set
                {
                    _enddate = value;
                    if (_chosenPhys.IsNull())
                    {
```

```
                            FullRecordList.Clear();
                            db.Records.Load();
                            ObservableCollection<Record> reclist =
db.Records.Local.ToBindingList();
                            foreach (Record rl in reclist)
                            {
                                reclist.Add(new Record (rl));
                            }
                        }
                        else
                        {
                            FullRecordList.Clear();
                            int pnum = _chosenPhys.AccountId;
                            Account ac = db.Accounts.Find(pnum);
                            Physname = ac.Surname + " " + ac.Name + " "
+ ac.FrName;
                            Declaration dc = db.Declarations.Get(Pat,
pnum);
                            db.Records.Where(x => x.DeclarationId ==
dc.DeclarationId).Load();
                            reclist = db.Records.Local.ToBindingList();
                        }

                        foreach (Record rl in reclist)
                        {
                            bool flag1 = true;
                            if (!BeginDate.IsNull())
                                flag1 = rl.RecDate >= BeginDate;
                            bool flag2 = true;
                            if (!EndDate.IsNull())
                                flag2 = rl.RecDate <= EndDate;
                            if (flag1 && flag2)
                                FullRecordList.Add(new Record (rl));
                        }
                        OnPropertyChanged("EndDate");
                    }
                }

            }
        }
```

Б.3 Текст файлу AddRecordViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

using MedSys.Models;
using MedSys.Views;

namespace MedSys.ViewModels
{
    public class AddRecordViewModel: INotifyPropertyChanged
    {

        DBContext db;
        int Dec;

        DateTime Nowtime;
        private RelayCommand _create;

        public AddRecordViewModel (int declId)
        {

            Dec = declId;

            db = new DBContext();
            db.Diagnoses.Load();

            ObservableCollection<Diagnosis> dlist =
db.Diagnoses.Local.ToBindingList();
            FullDiagnosisList = new
ObservableCollection<Diagnosis>();

            foreach (Diagnosis d in dlist)
            {
                FullDiagnosisList.Add(new Diagnosis (d));
            }
```

```
                ChosenDiag = new Diagnosis ();
                Nowtime = DateTime.Now;
            }

            private ObservableCollection<Diagnosis>
_fullDiagnosisList;
            public ObservableCollection<Diagnosis> FullDiagnosisList
            {
                get { return _fullDiagnosisList; }
                set
                {
                    _fullDiagnosisList = value;
                    OnPropertyChanged("Diagnoses");
                }
            }

            private Diagnosis _chosenDiag;
            public Diagnosis ChosenDiag
            {
                get { return _chosenPhys; }
                set
                {
                    _chosenDiag = value;
                    OnPropertyChanged("Diagnos");
                }
            }

            private DateTime _nowtime;
            public DateTime Nowtime
            {
                get { return _nowtime; }
                set
                {
                    _nowtime = value;
                    OnPropertyChanged("Nowtime");
                }
            }

            private string _adddiagnosis;
            public string Adddiagnosis
            {
                get { return _adddiagnosis; }
                set
```

```
    {
        _adddiagnosis = value;
        OnPropertyChanged("AdditionalDiagnosis");
    }
}

private int _Diagnosisstate;
public int Diagnosisstate
{
    get { return _Diagnosisstate; }
    set
    {
        _Diagnosisstate = value;
        OnPropertyChanged("Diagnosisstate");
    }
}

private string _problems;
public string Problems
{
    get { return _problems; }
    set
    {
        _problems = value;
        OnPropertyChanged("RecordProblems");
    }
}

private string _therapy;
public string Therapy
{
    get { return _therapy; }
    set
    {
        _therapy = value;
        OnPropertyChanged("RecordTherapy");
    }
}

private string _research;
public string Research
{
    get { return _research; }
```

```
                set
                {
                    _research = value;
                    OnPropertyChanged("RecordResearch");
                }
            }

            private string _description;
            public string Description
            {
                get { return _description; }
                set
                {
                    _description = value;
                    OnPropertyChanged("RecordDescription");
                }
            }

            public RelayCommand Create
            {
                get
                {
                    return _create?? (new RelayCommand(obj =>
                    {
                        int num = -1;
                        if (!ChosenDiag.IsNull())
                            num = ChosenDiag.DiagnosisId;

                        Record rcrd = db.Records.Add(Dec, num,
Diagnosisstate, Adddiagnosis, Nowtime, Problems, Therapy, Research,
Description);
                        db.SaveChanges();

                        if (!rcrd.IsNull())
                        {
                            PhysicianRecordView rv = new
PhysicianRecordView (Dec);
                            rv.ShowDialog();
                        }
                        else
                        {
```

```
                                MessageBox.Show("Не вдалося створити
медичний запис. Спробуйте пізніше", "Помилка", MessageBoxButton.OK,
MessageBoxImage.Error);
                            }
                        }));
                    }
                }    }
        }
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд № 1

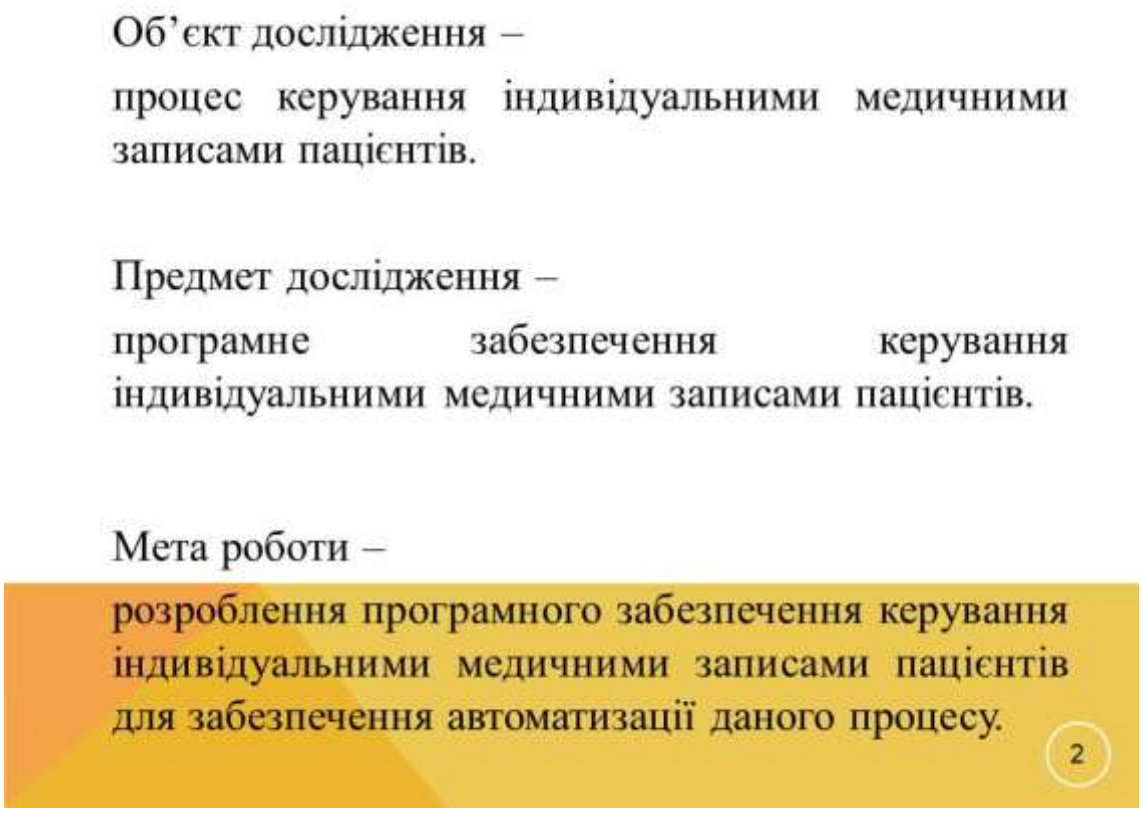

Рисунок В.2 – Слайд № 2

Завдання

- ✓ визначення вимог до програмного забезпечення, що розробляється;
- ✓ проєктування програмного забезпечення;
- ✓ реалізація програмного забезпечення;
- ✓ тестування і налагодження програмного забезпечення.

3

Рисунок В.3 – Слайд № 3

Вибір мови програмування

Показник	C#	Java
Парадигма програмування	Об'єктно-орієнтована, процедурна, узагальнена, функціональна	Об'єктно-орієнтована, структурна
Платформи	Windows, кросплатформна, за рахунок .NET Core	Кросплатформна
Збирач сміття	Наявний	Наявний
Створення графічного інтерфейсу	Легке за допомогою візуальних засобів середовища	Ускладнене, передбачає роботу на основі додаткових бібліотек

4

Рисунок В.4 – Слайд № 4

Рисунок В.5 – Слайд № 5

Рисунок В.6 – Слайд № 6

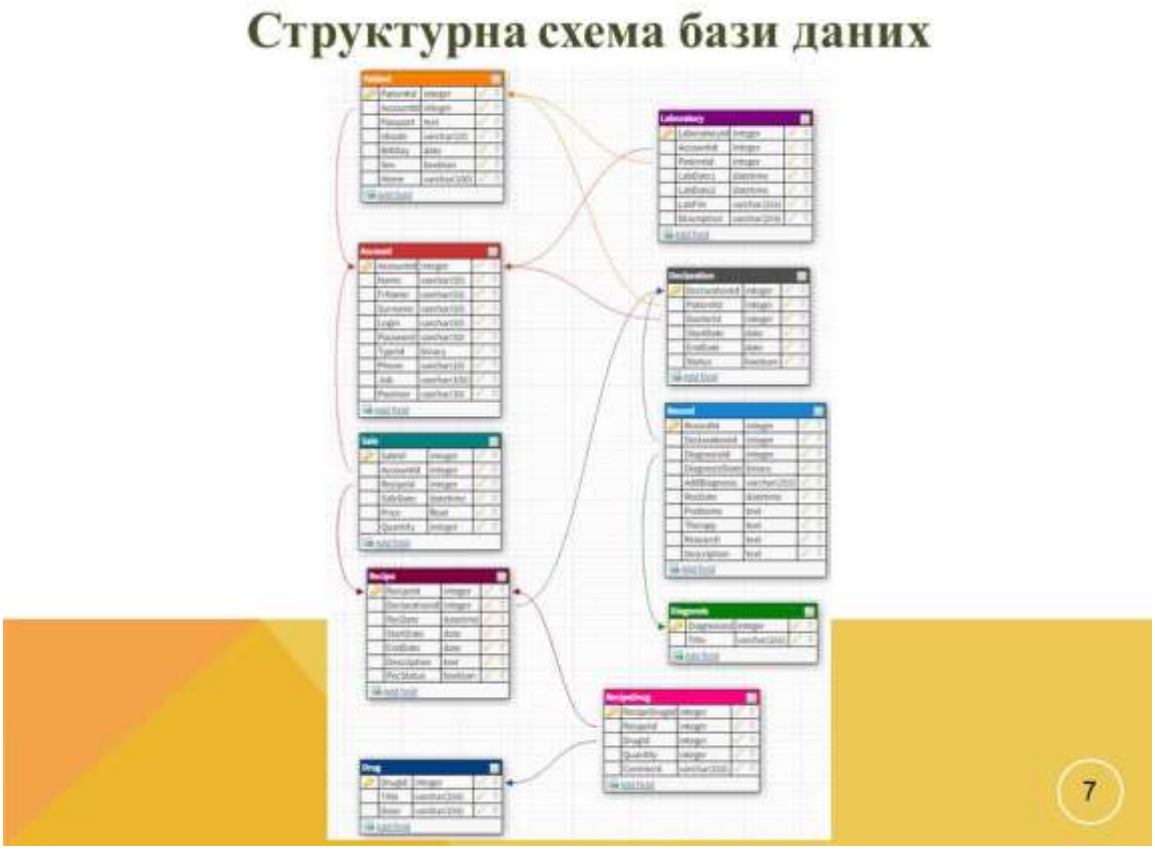

Рисунок В.7 – Слайд № 7

Рисунок В.8 – Слайд № 8

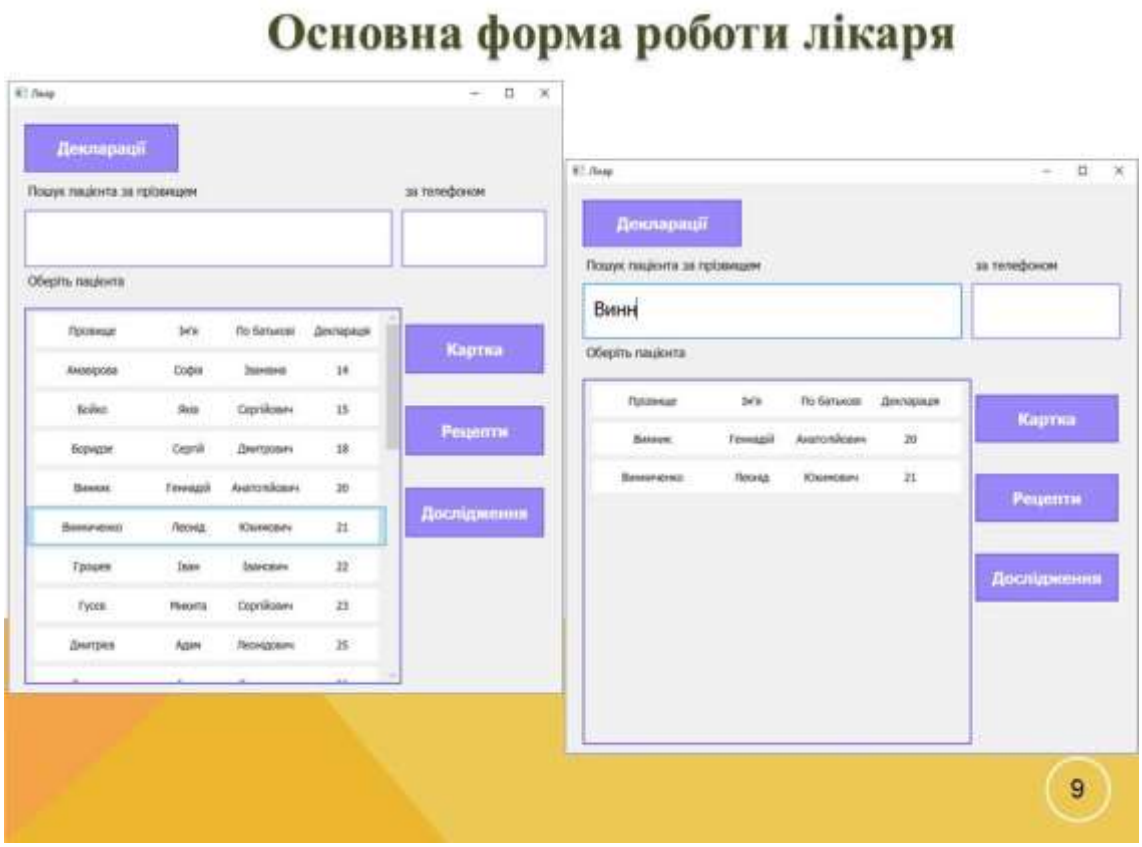

Рисунок В.9 – Слайд № 9

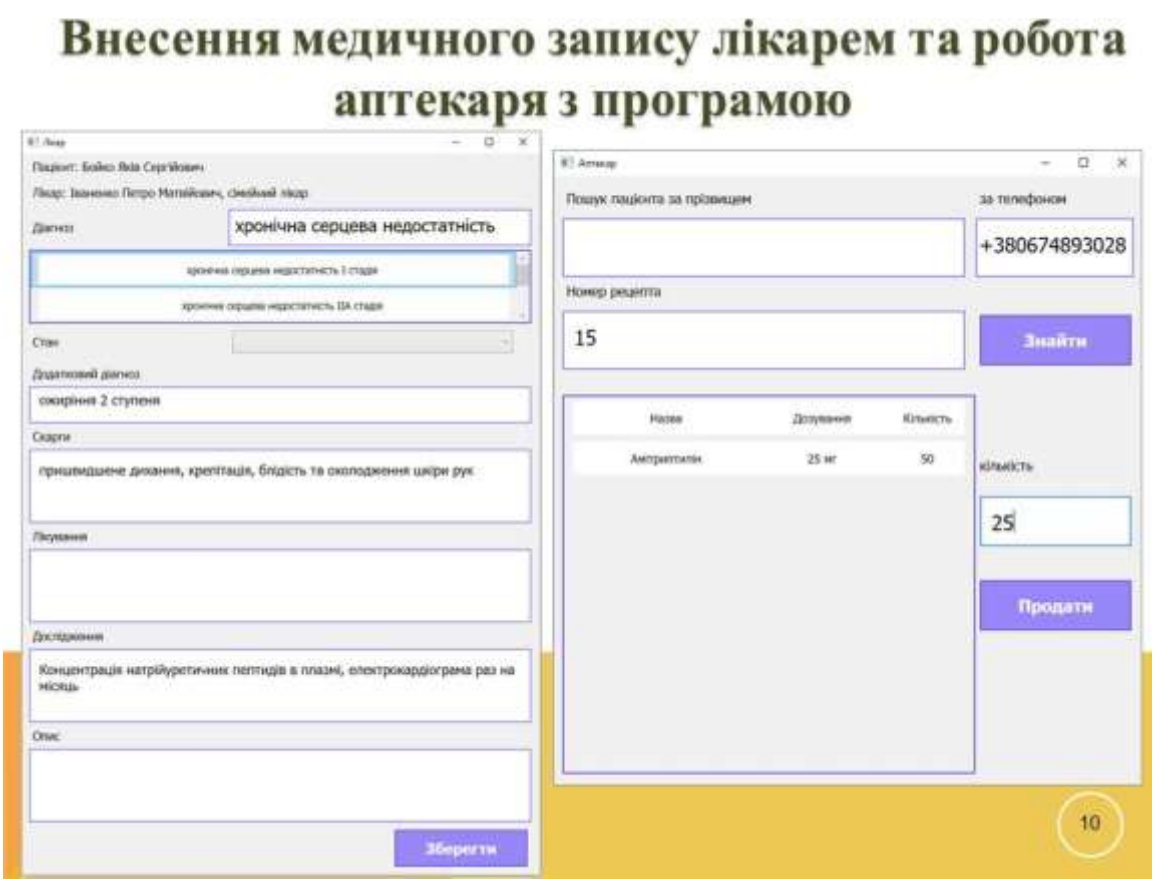

Рисунок В.10 – Слайд № 10

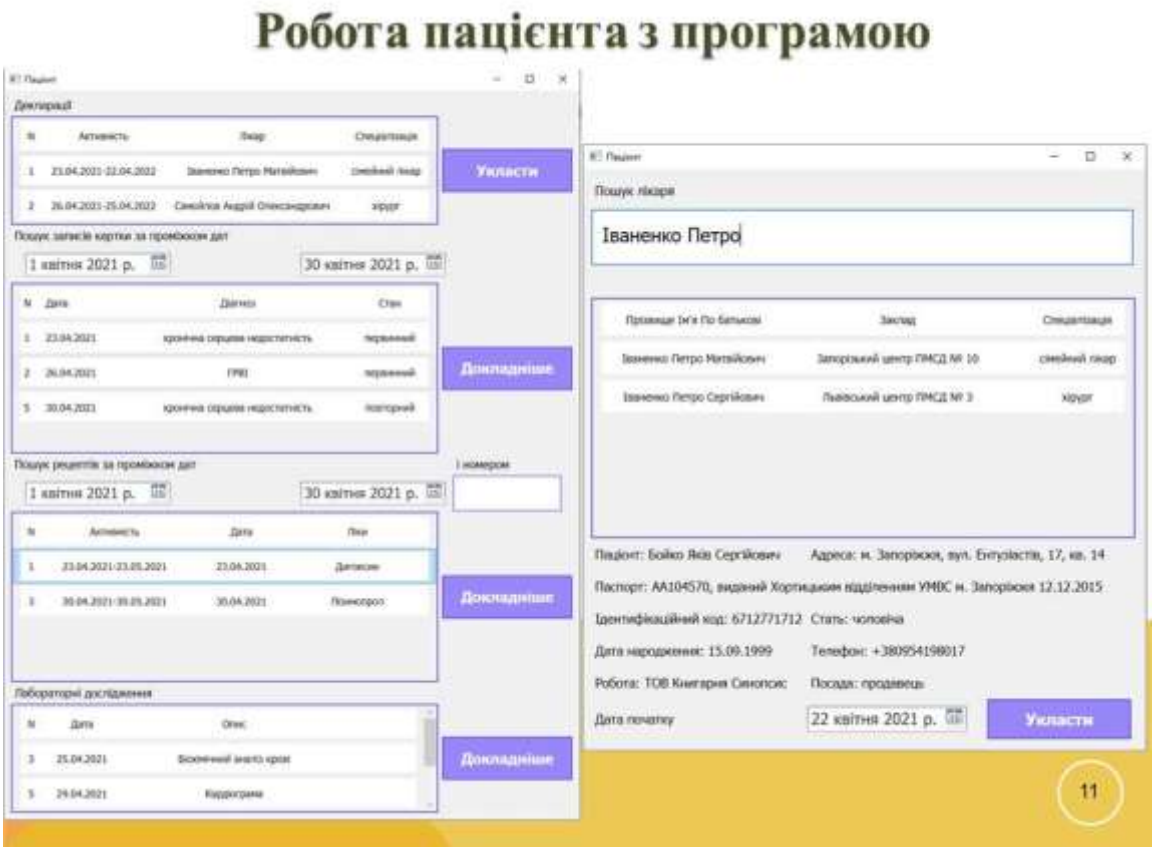

Рисунок В.11 – Слайд № 11

Рисунок В.12 – Слайд № 12