НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ Голуб Б.Л.
(підпис) (ПІБ)

"__" _____ 20__ р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Програмне забезпечення системи обліку пацієнтів в районній поліклініці»

Спеціальність 121 – «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

Гарант освітньої програми

К.т.н., доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Голуб Б.Л.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Стахнюк Т.П.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)
_____ Голуб Б.Л.
(підпис) (ПІБ)

"16" грудня 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту

Стахнюку Тарасу Павловичу
(прізвище, ім'я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення системи обліку пацієнтів в районній поліклініці
затверджена наказом ректора НУБіП України від "16" грудня 2024 р.
№ 2248"С"
Термін подання завершеної роботи на кафедру 2025.05.25
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи: опис організації та змісту роботи амбулаторно-поліклінічних закладів

Перелік питань, які потрібно розробити:

1. Районна поліклініка як об'єкт дослідження
2. Проєктування інформаційного забезпечення
3. Розробка програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання "16" грудня 2024 р.

Керівник бакалаврської кваліфікаційної роботи

К.т.н., доцент _____ Голуб Б.Л.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Стахнюк Т.П.
_

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- **CRUD** – Create, Read, Update, Delete
- **ER** – Entity-Relationship
- **LLM** – Large Language Model
- **LLaMA** – Large Language Model Meta AI
- **RDLC** – Report Definition Language Client-side
- **SQL** – Structured Query Language
- **UI** – User Interface
- **UML** – Unified Modeling Language
- **WPF** – Windows Presentation Foundation
- **НФ** – нормальна форма
- **ПЗ** – програмне забезпечення
- **СУБД** – система управління базами даних

ВСТУП

Актуальність. У наші дні в багатьох районних поліклініках облік пацієнтів ще досі ведеться вручну або за допомогою застарілого програмного забезпечення, які в переважній більшості є незручними та неефективними у використанні. І у висновку це призводить до плутанини у документації, втрати важливої інформації або даремної витрати часу. Крім того, в деяких районах, наприклад невеликих містах чи селищах, немає стабільного доступу до інтернету, що ускладнює використання сучасних електронних систем.

Через це виникає багато проблем у щоденній роботі медичного персоналу. Наприклад, пошук інформації про пацієнта займає багато часу, а лікарі змушені витрачати години на заповнення документів натомість того, щоб приділити більше уваги лікуванню. Це впливає негативно на якість надання послуг, створює ризик допущення помилок і знижує продуктивність роботи медзакладу загалом.

Для того щоб вирішити ці проблеми, було вирішено створити програмне забезпечення, яке дозволить автоматизувати процес обліку пацієнтів в районній поліклініці і зможе працювати навіть без постійного підключення до інтернету. Ця система надасть не лише зручне введення, обробку та зберігання медичної інформації, а й сприятиме підвищенню якості медичних послуг завдяки інтеграцій сучасних технологій, зокрема елементів штучного інтелекту, які можуть допомагати лікарям у постановці діагнозів на основі аналізу наявних даних.

Розроблений програмний продукт зможе зробити щоденну роботу медичного персоналу більш зручною та продуктивною, завдяки скорочення часу на рутинні операції та значно полегшить процес прийняття рішень.

Мета розробки. Мета розробки полягає в автоматизації процесів обліку пацієнтів та покращенні управління медичними даними на основі створення програмного забезпечення у районній поліклініці. Завдяки цьому можна буде спростити пошук та збереження інформації, зменшити кількість виникнення

помилок під час заповнення документації, в загальному зробити роботу медичного персоналу більш організованою.

Основна ідея полягає в тому, щоб вирішити існуючі проблеми з паперовим обліком та застарілими системами. Розроблене програмне забезпечення дозволить централізовано зберігати дані про пацієнтів та працівників поліклініки, інтегрувати всі процеси обліку, запису на прийому, введення даних в єдину систему, що забезпечить швидкий доступ до актуальної інформації та використання новітніх технологій для допомоги в постановці діагнозів.

Методи та технології. Під час створення цієї програми використовувалася мова програмування C# у середовищі Visual Studio. Для створення графічного інтерфейсу обрано технологію WPF на платформі .NET, яка дозволяє створити сучасний настільний застосунок, який буде не тільки зручним для користувачів, а й досить привабливим зовні.

Щоб зберігати всю інформацію про пацієнтів, працівників поліклініки, прийоми, медичну інформацію та інше, було використано MS SQL Server – це надійна система управління базами даних, яка добре справляється з великими обсягами інформації. Для зручного адміністрування використовувалось SQL Server Management Studio. Для формування різних звітів в системі застосовано компонент Microsoft Report Viewer, що дає можливість зручно переглядати й роздруковувати потрібну інформацію.

Крім цього, для реалізації функції допомоги в постановці діагнозу було використано модель штучного інтелекту. За основу взято модель LLaMA3.1 (локальна версія через платформу Ollama), яку було адаптовано та донавчено під потреби проєкту. Це дало змогу реалізувати базову можливість аналізу введених симптомів і пропонування можливих варіантів діагнозів. Це може стати у пригоді лікарям – як додатковий інструмент підтримки під час прийняття рішень.

Апробація програмного продукту. Проєкт приймав участь у І турі Всеукраїнського конкурсу студентських наукових робіт за спеціальністю «Інженерія програмного забезпечення», де посів перше місце. Відповідний диплом представлено у Додатку Л.

Результати розробки продукту були представлені на VII Всеукраїнській науково-практичній конференції студентів і аспірантів “Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025” у вигляді тез доповіді з назвою «Програмне забезпечення системи обліку пацієнтів в районній поліклініці», які наведено в Додатку М.

Структура записки. Записка кваліфікаційної роботи складається зі вступу, чотирьох основних розділів та висновків. Також до роботи додається список використаних джерел та додатки. Загальний обсяг записки – 77 сторінок основного тексту, кількість використаних джерел - 24, кількість додатків - 9.

У першому розділі «Районна поліклініка як об’єкт дослідження» розглянуто особливості функціонування поліклініки як предметної області, змодельовано основні процеси за допомогою діаграм, проаналізовано існуючі рішення, а також чітко сформульовано завдання, яке покликана вирішити розроблена система.

Другий розділ «Проєктування інформаційного забезпечення» присвячено побудові логічної моделі даних у вигляді ER-діаграми, вибору СУБД, розробці структури інформаційної бази та визначенню фізичної організації бази даних.

Третій розділ «Розробка програмного забезпечення» описує процес створення програмної частини системи: від вибору інструментів і середовищ розробки до реалізації програмних модулів, алгоритмів, а також структури взаємодії компонентів та архітектури застосунку.

У четвертому розділі «Рекомендації щодо впровадження та експлуатації системи» подано результати тестування розробленого програмного забезпечення, визначено вимоги до технічного середовища для його функціонування, описано склад інсталяційного пакета, а також рекомендації щодо впровадження системи.

1 РАЙОННА ПОЛІКЛІНІКА ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Аналіз роботи поліклініки як предметної області

Районна поліклініка – це заклад охорони здоров'я, який надає первинну медичну допомогу населенню певної території. Основними функціями поліклініки є прийом пацієнтів, діагностика захворювань, профілактичних оглядів та амбулаторне лікування. У такому закладі пацієнти отримують консультації терапевтів, вузькопрофільних спеціалістів, проходять обстеження, здають аналізи та, за потреби, скеровуються на лікування до стаціонарів або інших медичних установ [1].

Одним з ключових етапів взаємодії пацієнта з поліклінікою є реєстрація. Під час першого звернення пацієнта до медичного закладу проводиться первинна реєстрація, де створюється особова медична картка. У неї вносяться основні дані про пацієнта: ПІБ, дата народження, адреса проживання, номер телефону, а також відомості про вже перенесені захворювання або хронічні стани [2]. Надалі, при кожному новому зверненні до поліклініки, у картку вносяться нові записи – дані про діагнози, лікування, результати прийомів тощо.

Обслуговування пацієнта передбачає проходження ним певного шляху: від запису на прийом – до консультації з лікарем, можливої діагностики та лікування. Пацієнт може записатися як особисто через реєстратуру, так і через телефонний дзвінок або через електронні сервіси, якщо такі існують. Під час прийому, якщо це необхідно, можуть бути застосовані лікарські засоби лікарем, а після прийому лікар вносить записи до медичної картки, призначає лікування або направлення до інших спеціалістів чи додаткові дослідження.

Визначення діагнозу – один із найважливіших процесів. На цьому етапі лікар проводить опитування пацієнта, проводить фізичний огляд, призначає лабораторні або інструментальні обстеження. Залежно від результатів

діагностики лікар встановлює діагноз і формує відповідні призначення. Цей процес потребує точності та повного обліку, оскільки на його основі формуються подальші дії щодо лікування пацієнта.

Інформація про пацієнта має бути доступною, структурованою і такою, що постійно оновлюється. Зокрема, важливо мати змогу швидко переглянути історію звернень, раніше поставлені діагнози, призначене лікування, алергії, результати обстежень тощо. Це дозволяє лікарям приймати більш обґрунтовані рішення та знижує ризик медичних помилок. В умовах паперового обліку це займає багато часу і часто пов’язане з людським фактором.

Окрім медичного обслуговування населення, у поліклініці здійснюється детальний облік роботи персоналу. Ведеться фіксація даних працівників, кількості прийомів лікарів, обсяг часу витраченого на обслуговування пацієнтів. Це дозволяє контролювати рівень навантаження на медичних працівників та раціонально розподіляти робочі ресурси. Також проводиться облік використаних лікарських засобів, що необхідно для подальшого планування закупівель та уникнення дефіциту. За результатами роботи формуються статистичні звіти, які дозволяють аналізувати ефективність діяльності поліклініки та приймати управлінські рішення на рівні адміністрації [1].

Таким чином, районна поліклініка – це складна багатофункціональна система, в якій взаємодіють пацієнти, медичний персонал, адміністрація та інші структурні елементи. Автоматизація основних процесів є важливим кроком до підвищення якості медичних послуг і зменшення навантаження на персонал.

1.2 Моделювання предметної області

Для побудови ефективного програмного забезпечення важливо не лише розуміти логіку функціонування предметної області, але й вміти структурувати інформацію про неї у вигляді чітких моделей. Моделювання предметної області дозволяє описати її основні об’єкти, процеси, взаємозв’язки та сценарії використання. Це сприяє більш глибокому аналізу потреб користувачів,

визначенню вимог до системи, а також уникненню непорозумінь між замовником і розробником [3].

У випадку поліклініки як об’єкта дослідження моделювання дає змогу виявити ключові елементи взаємодії: пацієнтів, працівників реєстратури, лікарів, адміністрацію тощо. Також важливо відобразити послідовність дій, що відбуваються в процесі надання медичних послуг – від реєстрації пацієнта до завершення прийому, включаючи збереження даних, ведення електронної документації і формування звітів.

Для опису предметної області зручно використовувати нотацію UML (Unified Modeling Language) – уніфіковану мову моделювання, яка дає можливість графічно відображати функціональні й логічні аспекти системи у вигляді об’єктно-орієнтованих моделей. У межах даного розділу наведено основні типи UML-діаграм, зокрема: діаграма прецедентів, діаграма послідовності та діаграма активності. Надалі буде детально розглянуто кожну з цих діаграм у контексті предметної області.

1.2.1 Діаграма прецедентів. Діаграма прецедентів (use case diagram) є одним із ключових інструментів моделювання предметної області. Вона дає змогу візуалізувати основних учасників процесів у межах реального середовища (акторів) і типові дії (прецеденти), які вони виконують. Такий тип діаграм дозволяє сформувати цілісне уявлення про поточну організацію роботи в установі до впровадження інформаційної системи.

Ця діаграма є ефективним засобом для ідентифікації ключових процесів, що відбуваються в межах предметної області, а також для встановлення меж відповідальності між учасниками [4]. Вона дозволяє ще на початковому етапі аналізу визначити, які саме функції виконує кожен із суб’єктів взаємодії та які події ініціюються акторами. Таким чином, діаграма прецедентів виступає як основа для подальшої деталізації сценаріїв роботи та розробки логіки програмної системи.

Основні елементи діаграми прецедентів включають акторів і прецеденти. Актори – це зовнішні суб’єкти, які взаємодіють із процесами в межах предметної

області. Вони можуть бути як людьми, так і іншими системами або пристроями, що беруть участь у виконанні конкретних функцій. Прецеденти – це дії або послідовності дій, які виконуються в межах певного процесу. Вони відображаються основні функціональні сценарії, що здійснюються актором для досягнення конкретних результатів.

На рис. 1 представлено діаграму прецедентів, що моделює предметну область функціонування районної поліклініки. Вона охоплює основних користувачів системи, типові сценарії використання, а також зв’язки між акторами та прецедентами.

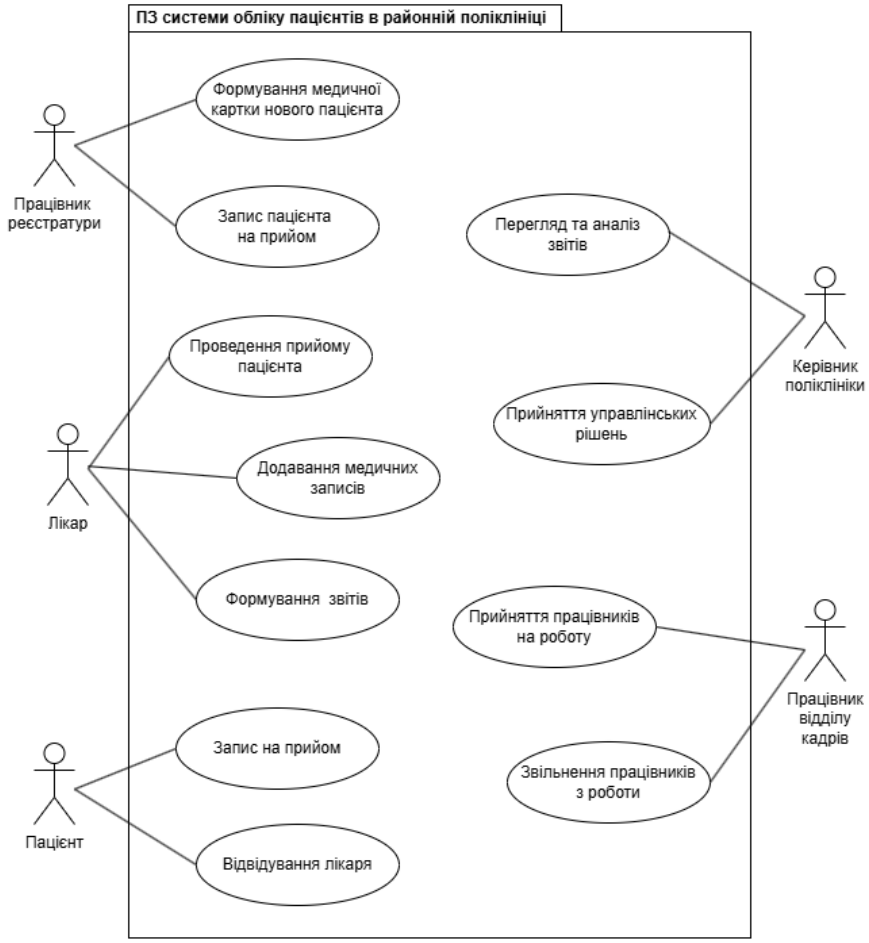


Рис. 1 Діаграма прецедентів ПЗ системи обліку пацієнтів в районній поліклініці

Опис акторів

Було виокремлено п’ять ключових акторів, які взаємодіють між собою та беруть участь у процесах.

- *Працівник реєстратури* – займається первинним обліком пацієнтів та організацією прийомів до лікаря.

- *Лікар* – здійснює прийом пацієнтів, веде медичні записи та формує звітну інформацію.
- *Пацієнт* – користується послугами медичного закладу.
- *Керівник поліклініки* – здійснює контроль за діяльністю поліклініки, аналізує статистичні звіти та приймає управлінські рішення.
- *Працівник відділу кадрів* – забезпечує кадрове адміністрування: прийом на роботу та звільнення персоналу.

Опис прецедентів

Працівник реєстратури:

- формування медичної картки нового пацієнта – створення медичної картки при першому зверненні;
- запис пацієнта на прийом – внесення інформації до розкладу лікаря відповідно до звернення пацієнта.

Лікар:

- проведення прийому пацієнта – обстеження, консультація, призначення лікування тощо;
- додавання медичних записів – ведення медичної документації щодо пацієнта;
- формування звітів – складання підсумкових звітів, наприклад про кількість прийомів за місяць.

Пацієнт:

- запис на прийом – ініціація звернення до лікаря, запис через реєстратуру;
- відвідування лікаря – фізичне відвідування поліклініки з метою отримання медичних послуг.

Керівник поліклініки:

- перегляд та аналіз звітів – ознайомлення зі статистикою щодо діяльності лікарів, кількості прийомів тощо;

- прийняття управлінських рішень – ухвалення дій, пов’язаних з оптимізацією роботи медичного закладу.

Працівник відділу кадрів:

- прийняття працівників на роботу – оформлення кадрових документів та введення нових працівників у штат;
- звільнення працівників з роботи – оформлення звільнення працівників, зняття з обліку.

Діаграма прецедентів демонструє загальну структуру організаційної взаємодії у предметній області районної поліклініки, що дає змогу краще зрозуміти логіку процесів та ролі учасників перед переходом до проєктування інформаційної системи.

1.2.2 Діаграма послідовності. Діаграма послідовності (sequence diagram) в UML використовується для моделювання динаміки предметної області, тобто порядку та взаємозв’язку дій між об’єктами або учасниками процесу з часом. Такий тип діаграм дозволяє зрозуміти логіку виконання типового сценарію, визначити учасників та їхню взаємодію в межах реального бізнес-процесу.

Діаграма послідовності включає кілька ключових елементів, таких як актори, об’єкти, повідомлення, лінії життєвого циклу та смуги активації [5]. Актори представляють зовнішніх учасників процесу, які взаємодіють із елементами предметної області, тоді як об’єкти – це внутрішні учасники або компоненти, що беруть участь у реалізації процесу. Лінії життєвого циклу позначають часову шкалу кожного учасника, по якій розташовуються повідомлення – дії чи запити, що передаються між об’єктами або акторами. Смуги активації відображають періоди, коли конкретний об’єкт або актор є активним і виконує певні дії у відповідь на отримані повідомлення.

В Додатку А представлено діаграму послідовності, яка ілюструє типовий сценарій надання медичних послуг в районній поліклініці – від прийняття працівників на роботу, запису пацієнта на прийом до аналізу статистичних даних

керівництвом поліклініки. Усі кроки відображають реальні дії, які виконуються персоналом поліклініки, пацієнтами та допоміжними службами.

Опис акторів та об’єктів діаграми

- *Керівник поліклініки* – ініціює призначення нового працівника та аналізує статистичні звіти.
- *Працівник відділу кадрів* – оформлює прийом нового працівника.
- *Пацієнт* – ініціює запис на прийом до лікаря та отримує медичну допомогу.
- *Працівник реєстратури* – приймає запит пацієнта, формує медичну картку, перевіряє доступність лікарів та фіксує запис.
- *Медична картка* – використовується як накопичувач усіх медичних даних пацієнта.
- *Графік прийомів* – представлення робочих розкладів лікарів, які оновлюються залежно від доступності.
- *Лікар* – проводить прийом пацієнта, призначає лікування, запитує наявність ліків, формує медичні звіти.
- *Аптека* – здійснює перевірку доступності ліків.

Опис сценарію

Адміністративна діяльність:

- керівник поліклініки надає вказівку щодо прийому нового працівника;
- працівник відділу кадрів оформлює прийняття на роботу нового лікаря або працівника реєстратури.

Запис пацієнта до лікаря:

- пацієнт звертається до працівника реєстратури з проханням запису на прийом;
- якщо пацієнт новий – створюється медична картка;
- працівник перевіряє розклад лікарів;

- отримується список доступних дат, часу і лікарів, після чого здійснюється вибір з боку пацієнта;
- запис фіксується і графік оновлюється.

Прийом у лікаря:

- пацієнт прибуває на прийом;
- лікар проводить обстеження, призначає лікування;
- за необхідності під час прийому можуть бути застосовані ліки, які є в наявності;
- результати прийому фіксуються в медичній картці.

Формування звітності:

- лікар формує медичні звіти на основі своєї діяльності;
- керівник аналізує отримані звіти для ухвалення управлінських рішень.

Цей сценарій демонструє послідовність взаємодій між учасниками у типових процесах районної поліклініки. Діаграма дозволяє зрозуміти часову логіку процесів та їх взаємозв’язки, що є важливою складовою для подальшого моделювання автоматизованої системи.

1.2.3 Діаграма активності. Під час аналізу предметної області було створено графічне представлення процесів, у якому відображено послідовність дій та логіку їх виконання. Це представлення, що є діаграмою активності (activity diagram), дозволяє візуалізувати взаємодію між основними учасниками процесу [5]. Такий підхід дає змогу краще розуміти внутрішню організацію роботи в умовах районної поліклініки та визначити основні етапи взаємодії.

У Додатку Б наведено діаграму активності, яка відображає узагальнену модель процесів, що відбуваються в межах діяльності районної поліклініки. Вона демонструє взаємозв’язки між основними учасниками та етапи виконання дій у межах типових процедур, що дає змогу візуально оцінити структуру та логіку організації роботи.

Діаграма включає основні елементи, зокрема доріжки, які поділяють простір відповідно до ролей, що виконують певні дії. Початкова подія позначається чорною круглою точкою і означає початок процесу, а кінцева – чорною точкою з білим кільцем усередині, що вказує на його завершення. Дії подаються у вигляді прямокутник блоків, потік керування відображено стрілками, які з'єднують ці дії між собою та вказують напрям виконання процесу, а ромбовидні елементи означають логічні розгалуження залежно від умов. Також діаграма може містити паралельні потоки, що відображають одночасне виконання кількох дій та їх подальше об'єднання в єдиний процес.

Опис діаграми

Процес прийому нового працівника:

- ініціюється керівником поліклініки;
- працівник відділу кадрового відділу оформлює документацію, додає працівника у графік роботи, фіксує в облікових записах.

Процес звернення пацієнта до лікаря:

- пацієнт ініціює запит на прийом;
- працівник реєстратури перевіряє наявність медичної картки:
  - якщо картки немає – створює нову;
  - якщо є, то пацієнта записують на прийом.
- відбувається запис на прийом: вибір дати, часу і лікаря;
- пацієнт з'являється на прийом;
- лікар проводить огляд, за потреби – визначає, чи потрібно застосувати медикаменти:
  - якщо медикаменти потрібні – перевіряється їх наявність;
  - якщо ліки в наявності – застосовуються;
  - якщо відсутні – видається рецепт.
- після завершення призначається лікування;
- формуються звіти;
- звіт переглядається та аналізується відповідальними особами.

Ця діаграма дозволяє побачити типову логіку взаємодії в поліклініці, виявити характерні сценарії роботи та оцінити організацію внутрішніх процесів. Завдяки розмежуванню ролей і поетапному опису дій, така форма представлення є зручною для подальшого аналізу та формалізації процесів.

1.3 Огляд існуючих аналогічних систем

В сучасних умовах медичні заклади активно використовують програмне забезпечення для автоматизації обліку пацієнтів, управління медичними процесами та формування звітності. Серед найбільш популярних систем, що використовуються в Україні, виділяються Helsi, МедІнфоСервіс та BAS Медицина. Розглянемо кожну з них, а також проведемо порівняльний аналіз їх функціональності та недоліків.

Helsi

Helsi – одна з найбільш популярних систем для автоматизації обліку пацієнтів в Україні. Вона пропонує широкий набір функцій, серед яких є реєстрація пацієнтів, запис на прийом до лікаря, введення електронних медичних карток, а також формування звітів для державних органів [6]. Однією з ключових переваг системи є можливість інтеграції з електронною системою охорони здоров’я та її широке використання серед медичних установ по всій країні.

На рис. 2 представлено скріншот інтерфейсу системи Helsi.

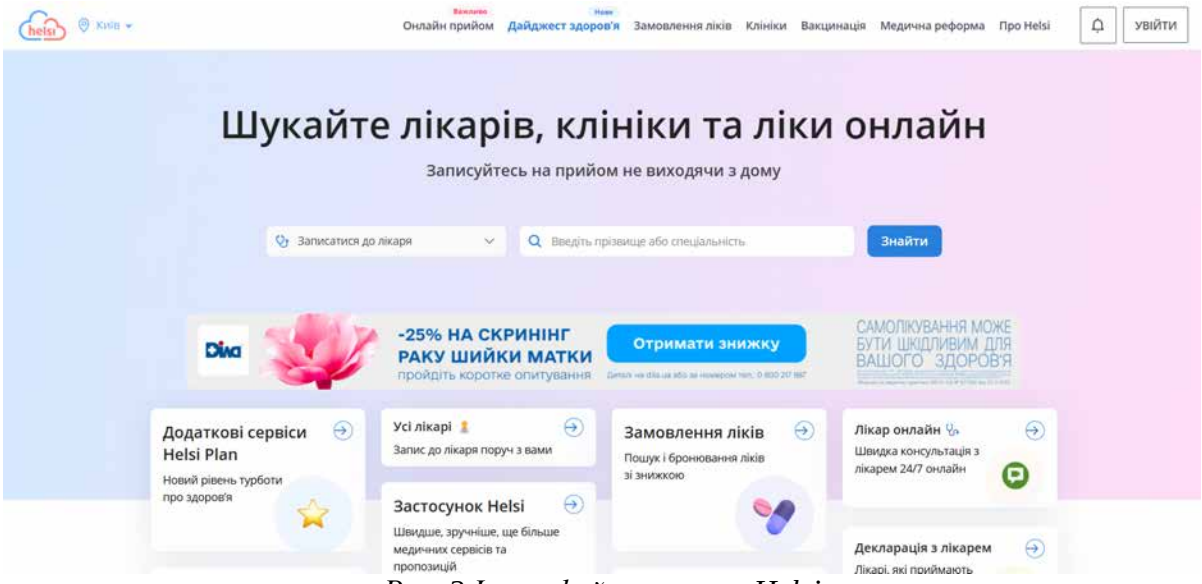


Рис. 2 Інтерфейс системи Helsi

Переваги:

- широка популярність і підтримка на державному рівні;
- можливість інтеграції з електронною системою охорони здоров'я;
- наявність мобільного додатку для пацієнтів.

Недоліки:

- високі вимоги до інтернет-з'єднання та інфраструктури;
- обмеження у використанні в районних поліклініках, де інтернет-з'єднання не завжди стабільне;
- не всі функції доступні в офлайн-режимі.

МедІнфоСервіс

МедІнфоСервіс є однією з найбільш відомих систем автоматизації обліку пацієнтів та лікарських прийомів. Ця система зручна для медичних працівників завдяки простому інтерфейсу, а також можливості легко формувати звіти та аналізувати дані [7].

На рис. 3 показано інтерфейс МедІнфоСервіс.

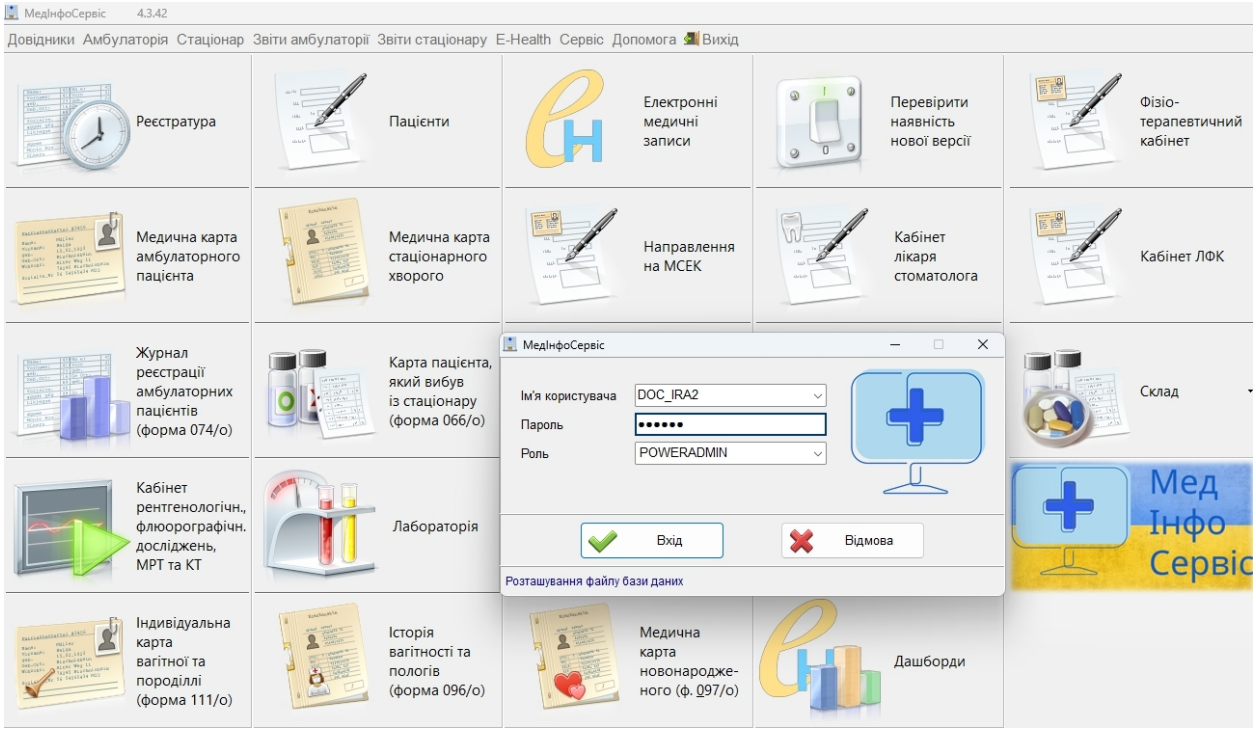


Рис. 3 Інтерфейс системи МедІнфоСервіс

Переваги:

- простота у використані;

- можливість налаштування різних функцій для конкретних медичних установ;
- доступність для невеликих медичних установ.

Недоліки:

- відсутність інтеграції з іншими системами охорони здоров’я;
- обмежена підтримка офлайн-режиму;
- немає можливості автоматичного планування прийомів.

BAS Медицина

BAS Медицина – це система автоматизації медичних установ, яка забезпечує облік пацієнтів, планування прийомів, а також автоматичне формування звітів і аналізу даних [8]. Вона пропонує зручний інтерфейс і хорошу підтримку для медичних працівників, що дозволяє їм швидко знаходити необхідну інформацію.

Інтерфейс системи BAS Медицина наведено на рис. 4.

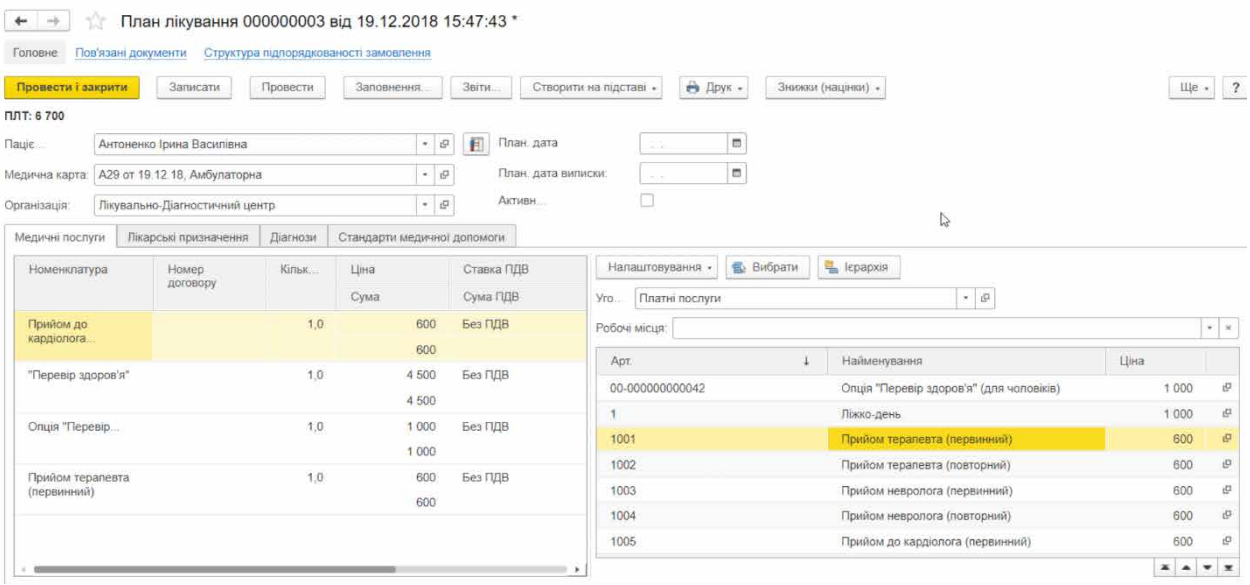


Рис. 4 Інтерфейс системи BAS Медицина

Переваги:

- широка функціональність для різних медичних процесів;
- інтеграція з іншими бухгалтерськими та фінансовими системами;
- можливість формування різноманітних звітів та аналітики.

Недоліки:

- вузька спрямованість на управління закладом, а не на зручність користувачам;
- відсутність функції скасування прийому;
- високі вимоги до апаратного забезпечення.

Незважаючи на високу популярність та ефективність систем, таких як Helsi, МедІнфоСервіс та BAS Медицина, їх використання не завжди забезпечує повноцінної підтримки для медичних закладів із обмеженими технічними ресурсами, особливо мова йде про наявність нестабільного інтернет-з'єднання. Окрім того, відсутність інтеграції з інтерактивними час-асистентами обмежує їх здатність підтримувати лікарів під час діагностики, що є критично важливим для забезпечення високої якості медичних послуг.

1.4 Постановка завдання

Метою даного проєкту є розробка програмного забезпечення для автоматизації діяльності медичного закладу, яке дозволить ефективно керувати даними пацієнтів, персоналу, записами на прийом та надавати інтерактивну допомогу лікарям під час консультування [9]. Програма має враховувати обмежені технічні ресурси окремих медичних установ, працювати автономно та забезпечувати високу зручність і безпеку.

Вхідні дані системи

Система повинна обробляти такі вхідні дані:

- *дані пацієнтів:* ПІБ, стать, дата народження, контактна інформація;
- *дані працівників:* ПІБ, спеціалізація (для лікарів), контактна інформація;
- *прийоми:* дата й час, пацієнт, лікар, заключення;
- *медичні записи:* аналізи, діагнози, призначення, лікування тощо;
- *кабінети:* номер, назва, поверх, закріплений лікар;

- *лікарські препарати:* АТС-код, назва, форма випуску, доза, одиниця вимірювання.

Вихідні дані системи

Система повинна генерувати такі результати:

- графік прийомів лікарів;
- медичні картки пацієнтів із основною інформацією, записами про прийоми, діагнози та призначення;
- статистичні звіти (завантаження лікарів, використані лікарські засоби тощо);
- інтерфейс з виведенням інформації на екран (швидкий доступ).

Умовно-постійна інформація

Умовно-постійна інформація передбачає:

- список лікарських спеціалізацій;
- дані про наявні кабінети;
- список лікарських засобів.

Функціональні вимоги до системи

Авторизація та розмежування доступу

Система повинна підтримувати багаторівневу авторизацію для трьох основних груп користувачів:

- працівники відділу кадрів (адміністратор);
- працівники реєстратури;
- лікарі.

Усі користувачі повинні мати логін/пароль, при цьому права доступу розмежовані. Кожному користувачу доступні лише ті функції, які відповідають його ролі.

Функції для працівників відділу кадрів

Облік персоналу:

- створення нових профілів працівників;
- редагування даних працівників за потреби;

- деактивація профілю у разі звільнення;
- перегляд повної інформації про персонал медичного закладу.

Функції для працівників реєстратури

Управління пацієнтами:

- додавання інформації про пацієнтів в систему;
- формування та редагування електронної медичної картки;
- перегляд медичних записів.

Запис на прийом:

- запис пацієнтів до лікаря з урахуванням графіка;
- доступ до графіка кожного із лікарів;
- можливість скасування або редагування запису.

Функції для лікарів

Введення медичних карток:

- внесення діагнозів, результатів прийомів, призначень;
- доступ до медичних карток пацієнтів;
- перегляд графіка прийомів.

Генерація звітів:

- формування звітів про прийоми за вказаний період;
- облік використаних ліків;
- вивантаження звітів в Word/PDF/Excel.

Інтерактивний чат-асистент:

- надання підказок щодо діагностичних дій;
- пошук інформації за симптомами та наданими даними;
- генерація рекомендацій щодо лікування на основі вхідних даних.

Нефункціональні вимоги до системи

Безпека:

- захист персональних даних відповідно до законодавства України (зокрема, ЗУ «Про захист персональних даних»);

- захист від несанкціонованого доступу.

Зручність використання:

- стандартний та зрозумілий інтерфейс для всіх ролей;
- швидкий доступ до основних функцій з головного меню;
- автоматичне збереження внесених змін.

Масштабованість:

- можливість додавання нових модулів без необхідності змінювати основну архітектуру;
- гнучка архітектура на основі модулів, що не потребує повної перебудови системи при оновленнях.

Сумісність:

- робота на операційній системі Windows 10 та новіших версіях;
- необов’язкова наявність стабільного інтернет-з’єднання.

Запропонована система орієнтована на зручне використання в умовах районних медичних установ з обмеженим ресурсним забезпеченням. Завдяки впровадженню інтелектуального асистента, автономності роботи без необхідності постійного підключення до інтернету та модульному підходу, програмне забезпечення стане ефективним інструментом для підвищення якості медичних.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних відіграє ключову роль у процесі проєктування інформаційних систем, оскільки дозволяє формалізувати структуру зберігання, обробки та взаємозв'язків між ними, забезпечуючи таким чином основу для створення фізичної структури бази даних. За допомогою ER-діаграми (діаграми «сутність-зв'язок») можна візуально відобразити цю логіку, що значно спрощує розуміння структури інформаційної системи на етапі проєктування [10].

У розробленій системі обліку пацієнтів в районній поліклініці були визначені такі ключові сутності: пацієнти, працівники (у тому числі лікарі та працівники реєстратури), медичні препарати, кабінети, спеціалізації, медичні картки, записи на прийом, звіт про прийом тощо. Кожна з цих сутностей має набір атрибутів, які ідентифікують її або описують відповідні характеристики. Для зв'язку між сутностями використовуються зовнішні ключі, що дозволяє забезпечити цілісність і зв'язність даних у базі.

ER-діаграма розробленої логічної моделі розміщена в Додатку В.

Опис сутностей та атрибутів

Patient (Пацієнт) – батьківська сутність:

- ID_patient – ідентифікаційний код пацієнта
- Last_name – прізвище пацієнта
- First_name – ім'я пацієнта
- Patronymic – по батькові пацієнта
- Gender – стать
- Date_birth – дата народження
- Home_address – домашня адреса
- Phone_number – номер телефону

- Email – електронна пошта
- Status – статус (наприклад, активний або неактивний)

Employee (Працівник) – батьківська сутність:

- ID_employee – ідентифікаційний код працівника
- Last_name, First_name, Patronymic – ПІБ працівника
- Phone_number – контактний номер
- Email – електронна пошта
- Status – статус працівника (активний або неактивний)

Medical_product (Медичні препарати) – батьківська сутність:

- ID_drug – унікальний ідентифікатор препарату
- ATC_code – міжнародний код препарату
- Drug_name – назва препарату
- Release_form – форма випуску (таблетки, ампули тощо)
- Dosage_unit – одиниця вимірювання дози
- Quantity_unit – одиниця вимірювання кількості

Office (Кабінет) – батьківська сутність:

- Office_number – номер кабінету
- Office_name – назва кабінету
- Floor – поверх

Specialization (Спеціалізація) – батьківська сутність:

- ID_specialization – ідентифікатор спеціалізації
- Name_specialization – назва спеціалізації

Medical_card (Медична картка) – дочірня сутність:

- ID_medical_card – унікальний ідентифікатор картки
- Blood_type – група крові
- Chronic_diseases – хронічні захворювання
- Allergies – алергії
- Start_date – дата відкриття картки

- End_date – дата закриття картки
- ID_patient (FK) – посилання на пацієнта

Registry_employee (Працівник реєстратури) – дочірня сутність:

- ID_employee (FK) – посилання на працівника

Doctor (Лікар) – дочірня сутність:

- ID_employee (FK) – посилання на працівника
- ID_specialization (FK) – посилання на спеціалізацію

Workplace (Робоче місце) – дочірня сутність:

- ID_employee (FK) – посилання на лікаря
- Office_number (FK) – посилання на кабінет

Appointment_record (Запис на прийом) – дочірня сутність:

- ID_patient (FK) – посилання па пацієнта
- ID_employee (FK) – посилання на лікаря
- Date_time – дата і час прийому

Consultation_report (Звіт про прийом) – дочірня сутність:

- ID_consultation – унікальний ідентифікатор прийому
- Start_date_time – дата та час початку прийому
- End_date_time – дата та час завершення
- Complaints – скарги пацієнта
- Diagnosis – діагноз
- Conlusion – заключення
- ID_medical_card (FK) – посилання на медичну картку
- ID_employee (FK) – посилання на лікаря

Administered_drugs (Застосовані ліки) – дочірня сутність:

- ID_consultation (FK) – посилання на прийом
- ID_drug (FK) – посилання на препарат
- Dosage – доза
- Quantity_used – використана кількість

Опис зв’язків

- Один пацієнт може мати лише одну медичну картку (неідентифікуючий зв’язок). Зв’язок реалізований через зовнішний ключ ID_patient у таблиці Medical_card, що посилається на первинний ключ ID_patient у таблиці Patient.
- Одна медична картка може мати багато звітів про прийом (неідентифікуючий зв’язок). Звя’зок реалізовано через зовнішній ключ ID_medical_card у таблиці Consultation_report.
- Один звіт про прийом може включати декілька записів про використані ліки, і один препарат може бути використаний у багатьох звітах (ідентифікуючі зв’язки). Ці зв’язки реалізовано через проміжну таблицю Administered_drugs, що має зовнішні ключі ID_consultation і ID_drug.
- Кожен звіт при прийом пов’язаний з одним лікарем (неідентифікуючий зв’язок). Зв’язок реалізується через зовнішній ключ ID_employee у таблиці Consultation_report.
- Працівники класифікуються як лікарі або реєстратори відповідно до запису у таблицях Registry_employee та Doctor. Зв’язок здійснюється через спільний ключ ID_employee.
- Один лікар має одну спеціалізацію, але одна спеціалізація може охоплювати багатьох лікарів (неідентифікуючий зв’язок). Зв’язок реалізовано через зовнішній ключ ID_specialization у таблиці Doctor.
- Один лікар може мати багато прийомів з пацієнтами, і один пацієнт може бути записаний до кількох лікарів (ідентифікуючі зв’язки). Для реалізації зв’язків використовується таблиця Appointment_record, яка містить зовнішні ключі ID_patient і ID_employee.

- Один лікар має одне робоче місце, що визначається через таблицю Workplace. Вона зв’язує ID_employee з Office_number. Один кабінет може бути закріплений за одним або кількома лікарями (ідентифікуючий зв’язок).

Перевірка відповідності нормальним формам

Щоб забезпечити ефективне зберігання, обробку та уникнення надлишковості даних у базі даних, важливо привести модель до форми, що відповідає принципам нормалізації. Нормалізація – це процес організації атрибутів та відношень у базі даних таким чином, щоб зменшити дублювання даних і забезпечити цілісність [11]. Основними нормальними формами є перша (1НФ), друга (2НФ) і третя (3НФ). У процесі розробки структури бази даних було дотримано основних нормальних форм до третьої включно. Нижче наведено обґрунтування відповідності моделі цим формам.

Перша нормальна форма (1НФ)

Перша нормально форма передбачає, що всі атрибути у таблицях повинні бути атомарними, тобто не ділитися на підчастини або множинні значення [11]. У розробленій моделі кожен стовпець таблиці містить лише одне значення, наприклад, у таблицях Patient, Employee та Medical_card усі поля є неподільними. Таблиця Administered_drugs не містить масивів чи списків препаратів – кожен запис відображає призначення одного конкретного препарату в межах одного прийму. Також у кожній таблиці чітко визначено первинний ключ, який забезпечує унікальність записів. Відсутність вкладених списків чи повторювальних груп підтверджує дотримання першої нормальної форми.

Друга нормальна форма (2НФ)

Друга нормальна форма вимагає, щоб усі неключові атрибути були повністю функціонально залежними від усього первинного ключа [11]. Це означає, що в таблицях з композитними ключами жоден атрибут не залежить лише від частини ключа. Наприклад, у таблиці Administered_drugs, де використовується складений ключ з полів ID_consultation та ID_drug, атрибути

Dosage та Quantity залежать одночасно від обох частин ключа. Таблиці з простими первинними ключами, як-от Doctor чи Specialization, також не містять атрибутів, що порушували б це правило, оскільки всі залежності визначені чітко й однозначно. Отже, жоден з неключових атрибутів не залежить частково від ключа, що підтверджує відповідність 2НФ.

Третя нормальна форма (3НФ)

Третя нормальна форма виключає транзитивні залежності, тобто ситуації, коли неключовий атрибут залежить не напряму від первинного ключа, а через інший неключовий атрибут [12]. У створеній логічній моделі це правило також дотримано. Наприклад, у таблиці Doctor ID_specialization напряму залежить від ID_employee, що є первинним ключем цієї таблиці. Самі дані про спеціалізацію винесено в окрему таблицю Specialization, а не дублюються у Doctor. Завдяки цьому відсутнє дублювання даних і забезпечується логічна цілісність структури. Інші таблиці, зокрема Appoitment_record, Medical_card, Employee та Patient, не містять транзитивних залежностей – усі атрибути функціонально залежать лише від первинного ключа відповідної таблиці.

Таким чином, логічна модель даних є такою, що повністю відповідає вимогам трьох нормальних форм. Це дозволяє зменшити обсяг зайвої інформації, уникнути логічних аномалій під час оновлення чи видалення даних, а також забезпечує узгодженість і зручність при подальшій роботі з базою даних.

2.2 Вибір системи управління базами даних

Розроблена модель бази даних перебуває у третій нормальній формі, що передбачає чітку структурованість, відсутність надлишковості та логічну цілісність даних. Для зберігання й обробки таких структурованих даних найбільш доречною є реляційна система управління базами даних (СУБД). Це пояснюється тим, що реляційні СУБД забезпечують ефективну підтримку складних зв’язків між таблицями, гарантуючи логічну узгодженість та коректність операцій над даними.

У процесі вибору відповідної СУБД було розглянуто два основні типи архітектур: файл-серверні та клієнт-серверні рішення.

Файл-серверні СУБД (наприклад, Microsoft Access, SQLite):

- підходять для невеликих локальних проєктів з мінімальними вимогами до навантаження;
- не забезпечуються високої продуктивності при великій кількості запитів або користувачів;
- обмежена підтримка транзакцій, безпеки та паралельної роботи;
- відсутні ефективні механізми масштабування, що критично для розширюваної медичної системи;
- через обмежену функціональність не є придатними для реалізації повноцінної багатокористувацької системи обліку пацієнтів [13].

Клієнт-серверні СУБД (наприклад, Microsoft SQL Server, PostgreSQL, MySQL):

- підтримують ефективну паралельну обробку запитів і масштабування;
- забезпечують високий рівень безпеки, керування доступом, аудит дій користувачів;
- підтримують складні транзакції, реплікацію, резервне копіювання, збережені процедури;
- оптимізовані для роботи з великими обсягами медичних записів та аналітичних запитів [14].

З огляду на необхідність централізованого зберігання інформації, багатокористувацького доступу, інтеграції з десктоп-додатком на WPF (.NET), безпеки та масштабованості було прийнято рішення використати Microsoft SQL Server як основну СУБД проєкту.

Обґрунтування вибору Microsoft SQL Server

Microsoft SQL Server був обраний як оптимальне рішення завдяки наступним вказаним перевагам.

Інтеграція з середовищем розробки:

- SQL Server безпосередньо підтримується платформою .NET;
- працює з ADO.NET, Entity Framework, Dapper – що забезпечує зручну роботу базою через C# WPF-додаток;
- просте налаштування підключень до бази даних, висока сумісність із Visual Studio.

Продуктивність і масштабованість:

- оптимізатор запитів, індексація, планувальник виконання дозволяють швидко обробляти великі обсяги даних;
- підтримка паралельного доступу на навантаження без зниження стабільності;
- можливість масштабування на рівні апаратного забезпечення або хмарної інфраструктури.

Безпека та управління доступом:

- розширені механізми аутентифікації (Windows/SQL);
- підтримка ролей, дозволів, шифрування, аудит доступу;
- відповідає стандартам безпеки, включаючи контроль доступу до конкретних об’єктів.

Надійність і відновлення:

- повна підтримка резервного копіювання, планового збереження даних;
- журнал транзакцій дає змогу відновити стан бази до будь-якого моменту;
- інструменти для моніторингу продуктивності та попередження помилок.

Адміністрування:

- SQL Server Management Studio (SSMS) надає зручний графічний інтерфейс для адміністрування;

- можливість автоматизації через SQL Agent, моніторинг через вбудовані функції [15].

Для обґрунтування вибору Microsoft SQL Server проведено порівняльний аналіз з іншими популярними клієнт-серверними системами управління базами даних і відобразимо це у табл. 1.

Таблиця 1

Порівняльна характеристика клієнт-серверних СУБД

Параметр	**SQL Server**	**PostgreSQL**	**MySQL**
Інтеграція з .NET	Відмінна (рідка підтримка)	Посередня (через сторонні драйвери)	Базова підтримка через конектори
Адміністрування	Зручне (SSMS, Azure Studio)	Складніше (через psql, pgAdmin)	Просте, але менш гнучке
Безпека	Розширена	Потужна, але складна в налаштуванні	Обмежена
Масштабованість	Висока	Висока	Середня
Підтримка Windows	Рідна	Обмежена	Добра

Враховуючи зазначені параметри, Microsoft SQL Server є найбільш раціональним вибором для медичної системи, яка розробляється як багатокористувацький WPF-додаток. Він забезпечує оптимальний баланс між продуктивністю, інтеграцією, безпекою та масштабованістю.

2.3 Розробка структури інформаційної бази

У процесі створення інформаційної системи особливу увагу приділено розробці структури інформаційної бази, оскільки саме вона є основою збереження, обробки та взаємодії з даними. Ретельно спроектована структура бази даних забезпечує логічну цілісність, ефективність виконання запитів та можливість масштабування у майбутньому.

Для підтримки повноцінного функціонування інформаційної системи структура бази даних містить не лише основні таблиці, а й уявлення, збережені процедури, тригери та окремо налаштованих користувачів із розмежованими правами доступу. Такий підхід дозволяє автоматизувати обробку даних, підвищити рівень безпеки системи та забезпечити контроль над цілісністю інформації на рівні СУБД. У наступних підрозділах описано реалізацію кожного з цих компонентів.

2.3.1 Створення бази даних. Створення бази даних є першим етапом у процесі розробки інформаційної системи. Це важливий крок, що забезпечує фундамент для подальшого зберігання та організації даних. На рис. 5 наведено код для створення бази даних.

```
CREATE DATABASE DistrictPolyclinic
```

Рис. 5 Створення бази даних "DistrictPolyclinic"

Цей код ініціює процес створення нової бази, після чого можна приступати до формування таблиць, запитів та інших компонентів.

2.3.2 Створення таблиць. Процес створення таблиць є важливим етапом у розробці структури бази даних, оскільки кожна таблиця відображає окремий об'єкт або сутність системи і містить стовпці з відповідними атрибутами. Для забезпечення ефективного зберігання даних необхідно визначати правильну структуру таблиць, що відповідає вимогам бізнес-логіки.

Створення таблиць для зберігання інформації про пацієнтів та записів на прийом до лікаря було реалізовано за допомогою SQL-запитів, зокрема через команду CREATE TABLE. На прикладі цих таблиць розглянемо процес створення: фрагмент створення таблиці "Пацієнт" наведено на рис. 6, а таблиці "Запис на прийом" – на рис. 7. Перед їх створенням здійснювалась перевірка на наявність відповідних таблиць у базі даних, що зображена на рис. 8. Це дозволяє уникнути помилок при спробі створити дублікати таблиць.

```
CREATE TABLE Patient (
    ID_patient CHAR(10) PRIMARY KEY NOT NULL,
    Last_name NVARCHAR(50) NOT NULL,
    First_name NVARCHAR(50) NOT NULL,
    Patronymic NVARCHAR(50),
    Gender NVARCHAR(50),
    Date_birth DATE NOT NULL,
    Home_address NVARCHAR(255),
    Phone_number VARCHAR(20),
    Email VARCHAR(100),
    Status_patient NVARCHAR(50) NOT NULL
)
```

Рис. 6 Створення таблиці "Пацієнт"

```
CREATE TABLE Appointment_record (
    ID_patient CHAR(10) NOT NULL,
    ID_employee CHAR(10) NOT NULL,
    Date_time DATETIME NOT NULL,
    PRIMARY KEY (ID_patient, ID_employee, Date_time),
    FOREIGN KEY (ID_patient) REFERENCES Patient(ID_patient),
    FOREIGN KEY (ID_employee) REFERENCES Doctor(ID_employee)
)
```

Рис. 7 Створення таблиці "Запис на прийом"

```
USE DistrictPolyclinic
GO

IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Appointment_record' AND type_desc = 'USER_TABLE')
    DROP TABLE Appointment_record
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Patient' AND type_desc = 'USER_TABLE')
    DROP TABLE Patient
```

Рис. 8 Перевірка на існування таблиць

Повний код створення всіх таблиць подано в Додатку Д.

2.3.3 Створення уявлень. Уявлення (views) – це віртуальні таблиці, які формуються на основі результатів запиту до однієї або кількох таблиць бази даних [16]. Вони дозволяють спростити доступ до складної інформації, покращити безпеку даних та зручно організувати звітність.

На рис. 9 зображено приклад уявлення, яке містить інформацію про застосовані лікарські засоби, що може бути використаним для аналітики та створення відповідних звітів.

```sql
IF EXISTS (SELECT name FROM sys.objects
 WHERE name = 'vw_AdministeredDrugs  ' AND type_desc = 'VIEW')
DROP VIEW vw_AdministeredDrugs
GO

-- Застосовані медикаменти
CREATE VIEW vw_AdministeredDrugs AS
SELECT
    cr.ID_consultation,
    cr.Start_date_time,
    e.Last_name + ' ' + e.First_name + ' ' + e.Patronymic AS DoctorName,
    mp.ATC_code,
    mp.Drug_name,
    mp.Release_form,
    ad.Dosage,
    mp.Dosage_unit,
    ad.Quantity_used,
    mp.Quantity_unit
FROM Administered_drugs ad
JOIN Consultation_report cr ON ad.ID_consultation = cr.ID_consultation
JOIN Employee e ON cr.ID_employee = e.ID_employee
JOIN Medicinal_product mp ON ad.ID_drug = mp.ID_drug;
```

Рис. 9 Створення уявлення

2.3.4 Створення збережених процедур та тригерів. Одним із важливих інструментів роботи з базою даних є збережені процедури, які дозволяють зберігати часто використовувані SQL-команди у вигляді окремих об’єктів. Вони сприяють підвищенню ефективності виконання операцій, забезпечують повторне використання код та зменшують ризик помилок при ручному введені запитів [16]. На рис. 10 подано приклад створення збереженої процедури, що дозволяє додавати нового працівника у систему. Процедура приймає необхідні параметри та автоматично виконує вставку даних у відповідну таблицю.

```sql
-- Процедура додавання працівника
CREATE PROCEDURE add_employee
    @ID_employee CHAR(10),
    @Last_name NVARCHAR(50),
    @First_name NVARCHAR(50),
    @Patronymic NVARCHAR(50),
    @Phone_number VARCHAR(20),
    @Email VARCHAR(100),
    @Status_employee NVARCHAR(50),
    @Type_employee NVARCHAR(50)
AS
BEGIN
    INSERT INTO Employee (ID_employee, Last_name, First_name, Patronymic, Phone_number, Email, Status_employee, Type_employee)
    VALUES (@ID_employee, @Last_name, @First_name, @Patronymic, @Phone_number, @Email, @Status_employee, @Type_employee)
END
GO
```

Рис. 10 Створення збереженої процедури

Ще одним механізмом автоматизації контролю за даними в базі є тригери, які автоматично виконуються при настанні певної події, наприклад, додавання або зміни запису. Вони особливо корисні для реалізації логіки перевірки та

підтримки цілісності даних [16]. На рис. 11 наведено приклад тригера, який контролює додавання записів на прийом до лікаря.

```
-- Перевірка на перетин часу прийомів лікаря
CREATE TRIGGER check_consultation_overlap
ON Consultation_report
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        INNER JOIN Consultation_report c ON i.ID_employee = c.ID_employee
        WHERE (
            (i.Start_date_time >= c.Start_date_time AND i.Start_date_time < c.End_date_time) OR
            (i.End_date_time > c.Start_date_time AND i.End_date_time <= c.End_date_time) OR
            (i.Start_date_time <= c.Start_date_time AND i.End_date_time >= c.End_date_time)
        )
          AND i.ID_consultation <> c.ID_consultation
    )
    BEGIN
        RAISERROR('На цей час у лікаря вже є інший запис!', 16, 1)
        ROLLBACK
    END
END
GO
```

Рис. 11 Створення тригера

2.3.5 Створення користувачів. У рамках організації доступу до бази даних було реалізовано механізм створення користувачів із поділом на ролі відповідно до функціональних обов'язків. Було визначено три основні ролі: "Administrator", "Registry" та "Doctor". Такий підхід дозволяє ефективно керувати правами доступу та забезпечити належний рівень безпеки даних.

На прикладі створення користувача "administrator" можна прослідкувати основні етапи додавання нового користувача до системи. На рис. 12 показано створення облікового запису, а на рис. 13 – безпосереднє створення користувача в базі даних.

```
-- Створення логіна адміністратора
sp_addlogin
@loginame = 'administrator',
@passwd = 'administrator'
GO
```

Рис. 12 Створення облікового запису

```
-- Створення користувача адміністратора
sp_adduser
@loginame = 'administrator',
@name_in_db = 'administrator'
GO
```

Рис. 13 Створення користувача

Далі були створені ролі з відповідними назвами, кожній з яких надано необхідні привілеї. На рис. 14 представлено процес створення ролі “Administrator” та надання їй відповідних прав. Після цього користувача “administrator” було додано до ролі “Administrator”, що зображено на рис. 15.

```
-- Створення ролі Administrator і надання їй прав
sp_addrole 'Administrator'
GRANT SELECT ON Patient TO Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Employee TO Administrator
GRANT SELECT ON Medicinal_product TO Administrator
GRANT SELECT ON Office TO Administrator
GRANT SELECT ON Specialization TO Administrator
GRANT SELECT ON Medical_card TO Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Registry_employee TO Administrator
GRANT SELECT, INSERT, UPDATE, DELETE ON Doctor TO Administrator
GRANT SELECT ON Workplace TO Administrator
GRANT SELECT ON Appointment_record TO Administrator
GRANT SELECT ON Consultation_report TO Administrator
GRANT SELECT ON Administered_drugs TO Administrator
GRANT SELECT ON vw_MedicalCard TO Administrator
GRANT SELECT ON vw_DoctorAppointments TO Administrator
GRANT SELECT ON vw_AdministeredDrugs TO Administrator
GO
```

Рис. 14 Створення ролі та надання їй привілей

```
-- Призначення ролі адміністратору
sp_addrolemember 'Administrator', 'administrator'
GO
```

Рис. 15 Додання користувача до ролі

У подальшому, при створенні нових користувачів, процедура передбачає спочатку формування облікового запису, потім створення користувача в системі та, відповідно, його включення до потрібної ролі, яка вже має налаштований набір привілеїв. Такий підхід забезпечує гнучке та безпечне управління доступом до функціональності бази даних.

2.4 Схема та фізична структура бази даних

Фізична структура бази даних є важливим елементом проєктування, який визначає зберігання, організацію та взаємозв’язки даних. У межах даної інформаційної системи таблиці бази даних створювалися за допомогою SQL-запитів. Це дозволило детально налаштувати атрибути таблиць, встановити

обмеження цілісності даних (включаючи первинні та зовнішні ключі), а також задати типи даних відповідно до специфіки збережуваної інформації [17].

Загалом у базі даних реалізовано 12 основних таблиць, які описані нижче.

- *Patient* – зберігає персональні дані пацієнтів.
- *Employee* – містить інформацію про всіх працівників, включаючи контактні дані та тип працівника.
- *Doctor і Registry_employee* – таблиці для розмежування типів працівників відповідно до їхньої ролі.
- *Medical_card* – медична картка пацієнта.
- *Appointment_record* – записи на прийом.
- *Consultation_report* – результати медичних консультацій.
- *Administered_drugs* – перелік застосованих ліків під час прийому.
- *Medicinal_product* – довідкова таблиця з лікарськими препаратами;
- *Specialization* – спеціалізації лікарів.
- *Office* – кабінети поліклініки.
- *Workplace* – робоче місце лікаря.

Кожна таблиця має первинний ключ, який забезпечує унікальність записів. Для встановлення взаємозв’язків між таблицями використовувалися зовнішні ключі (FOREIGN KEY). Наприклад, поле ID_patient у таблицях Medical_card і Appointment_record пов’язане з первинним ключем таблиці Patient, що забезпечує зв’язок між пацієнтами та їх медичними картками або записами на прийом. Аналогічно, ID_employee пов’язує таблиці лікарів і працівників реєстратури з загальною таблицею працівників.

Для коректного представлення текстових даних використовувалися типи:

- *CHAR(n)* – для зберігання фіксованої кількості символів, зокрема для ідентифікаторів;
- *NVARCHAR(n)* – для зберігання рядків, що підтримують кирилицю, зокрема ПІБ, адреси, назви препаратів тощо;

- *VARCHAR(n)* – для даних без потреби в юнікоді, наприклад, телефонні номери чи електронних адрес;
- *NVARCHAR(MAX)* – для зберігання довгих текстових описів;
- *DATE та DATETIME* – для роботи з датами та часом;
- *INT* – для числових значень.

На схемі бази даних, яка зображена в Додатку Е, наочно зображено всі таблиці, поля, типи даних, а також взаємозв’язки між ними. Така схема дозволяє краще зрозуміти логіку структури, зручно аналізувати зв’язки між сутностями та визначити залежності, необхідні для ефективного функціонування системи.

Як зазначалося раніше, повний SQL-код створення таблиць представлений у Додатку Д, що дозволяє відтворити структуру бази даних у будь-якому середовищі з підтримкою SQL Server.

Отже, фізична структура бази даних є логічно впорядкованою, гнучкою та масштабованою. Вона враховує специфіку медичних даних, забезпечує надійне зберігання інформації та підтримує важливі механізми забезпечення цілісності. Такий підхід дозволяє легко розширювати систему у майбутньому, додаючи нові сутності або функціональність, без шкоди для стабільності або якості даних.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

На етапі проєктування програмного забезпечення важливою складовою є виділення ключових абстракцій предметної області, які відображають реальні об’єкти, учасників і процеси, характерні для діяльності поліклініки. Абстракції – це узагальнені моделі, що містять у собі найважливіші властивості та обов’язки відповідних сутностей, необхідні для реалізації логіки системи [18]. На рис. 16 зображено загальну структуру абстракцій предметної області, яка є основою для подальшого проєктування системи.

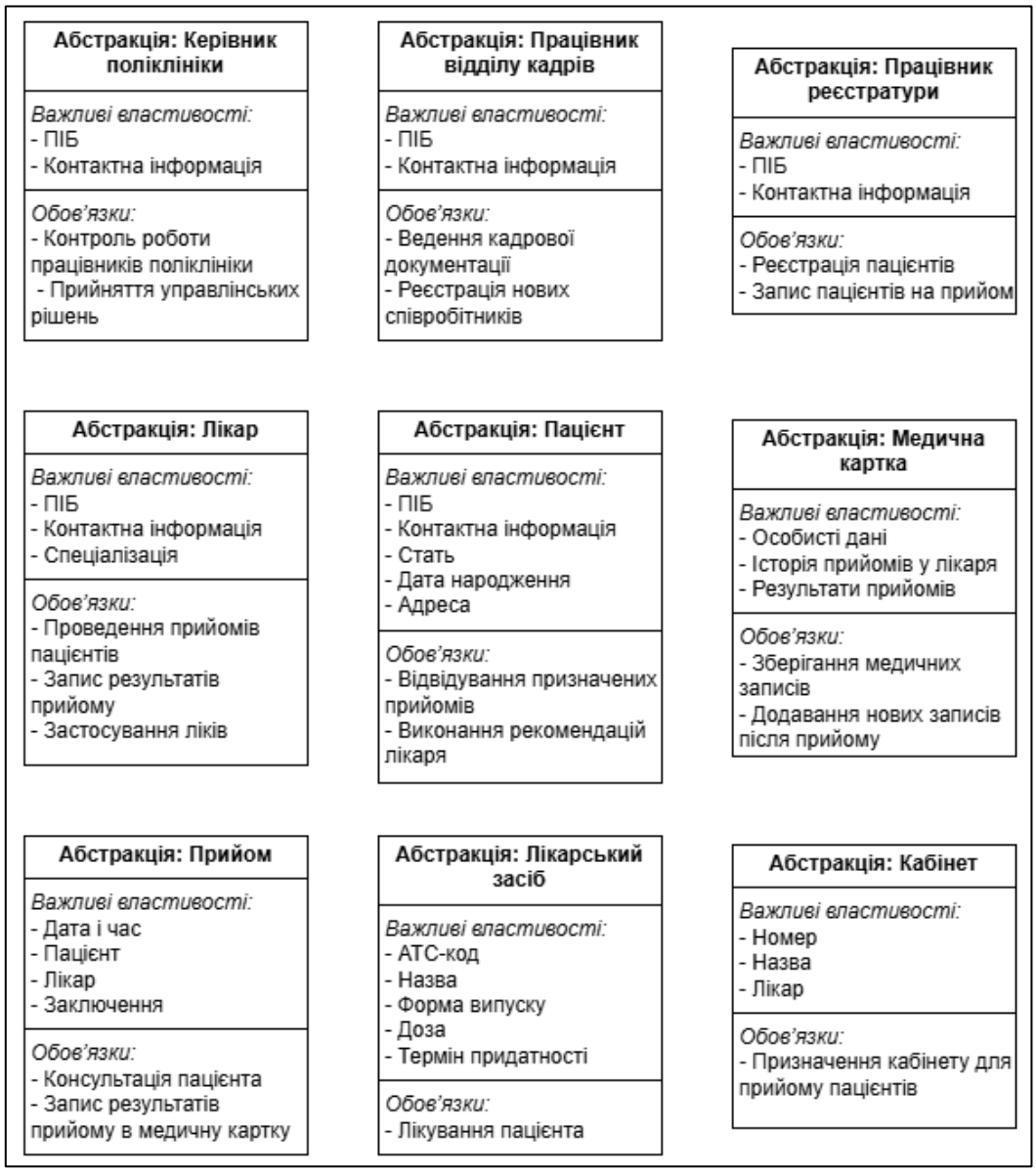


Рис. 16 Абстракції предметної області

Кожна з абстракцій охоплює типові ролі, елементи та процеси, що мають місце в роботі медичного закладу. Зокрема, до абстракцій належать такі суб’єкти, як керівник поліклініки, працівник відділу кадрів, працівник реєстратури, лікар, пацієнт, а також об’єкти, що опосередковують взаємодію між ними, такі як медична картка, прийом, лікарський засіб та кабінет.

Кожна абстракція визначає характерні властивості та обов’язки, які необхідні для побудови функціональної структури системи. Це дозволяє описати логіку взаємодії між користувачами, забезпечити належну обробку даних, а також встановити зв’язки між об’єктами системи.

Таким чином, абстракції предметної області дозволяють створити чітку й логічну модель взаємодії між усіма ключовими учасниками процесів у поліклініці. Це, у свою чергу, забезпечує ефективне структурування даних та функціоналу майбутнього програмного забезпечення, що сприятиме автоматизації основних бізнес-процесів закладу.

3.2 Моделювання структури та взаємодії об’єктів

Наступним кроком є моделювання структури та взаємодії об’єктів, що дозволяє чітко визначити складові системи та їхні взаємозв’язки. На цьому етапі важливо не лише відобразити логіку функціонування системи, а й чітко структурувати її складові, визначити, які об’єкти існують, які зв’язки між ними встановлюються, та як вони взаємодіють у процесі виконання функціоналу. Моделювання структури допомагає візуалізувати загальну архітектуру системи та слугує основою для подальшого її програмного втілення.

Для моделювання застосовується стандартна мова UML, що є загальноприйнятим інструментом для побудови об’єктно-орієнтованих моделей. Особливу роль на цьому етапі відіграють діаграми, що відображають як статичні, так і динамічні аспекти системи. Статичні аспекти представлені діаграмою класів, яка формує основу логічної структури системи, тоді як динамічні аспекти

ілюструються через діаграму кооперацій, що показує взаємодію об’єктів у процесі виконання певних сценаріїв [19].

Діаграма класів була побудована ще на етапі аналізу, оскільки саме вона дозволяє сформувати уявлення про функціональну модель системи та спроектувати об’єкти, що реалізуватимуть ключові можливості майбутнього програмного продукту. На основі цієї діаграми згодом формуються програмні класи з відповідними атрибутами та методами, а також реалізуються міжоб’єктні взаємозв’язки.

3.2.1 Діаграма класів. Діаграма класів – це один із основних типів UML-діаграм, призначений для зображення об’єктної структури системи. Вона демонструє основні класи, які було ідентифіковано на етапі аналізу, їхні атрибути, методи та зв’язки між ними. Особливу увагу діаграма приділяє асоціаціям між класами, що визначають логіку взаємодії об’єктів у реальній системі.

На рис. 17 зображено діаграму класів із асоціаціями.

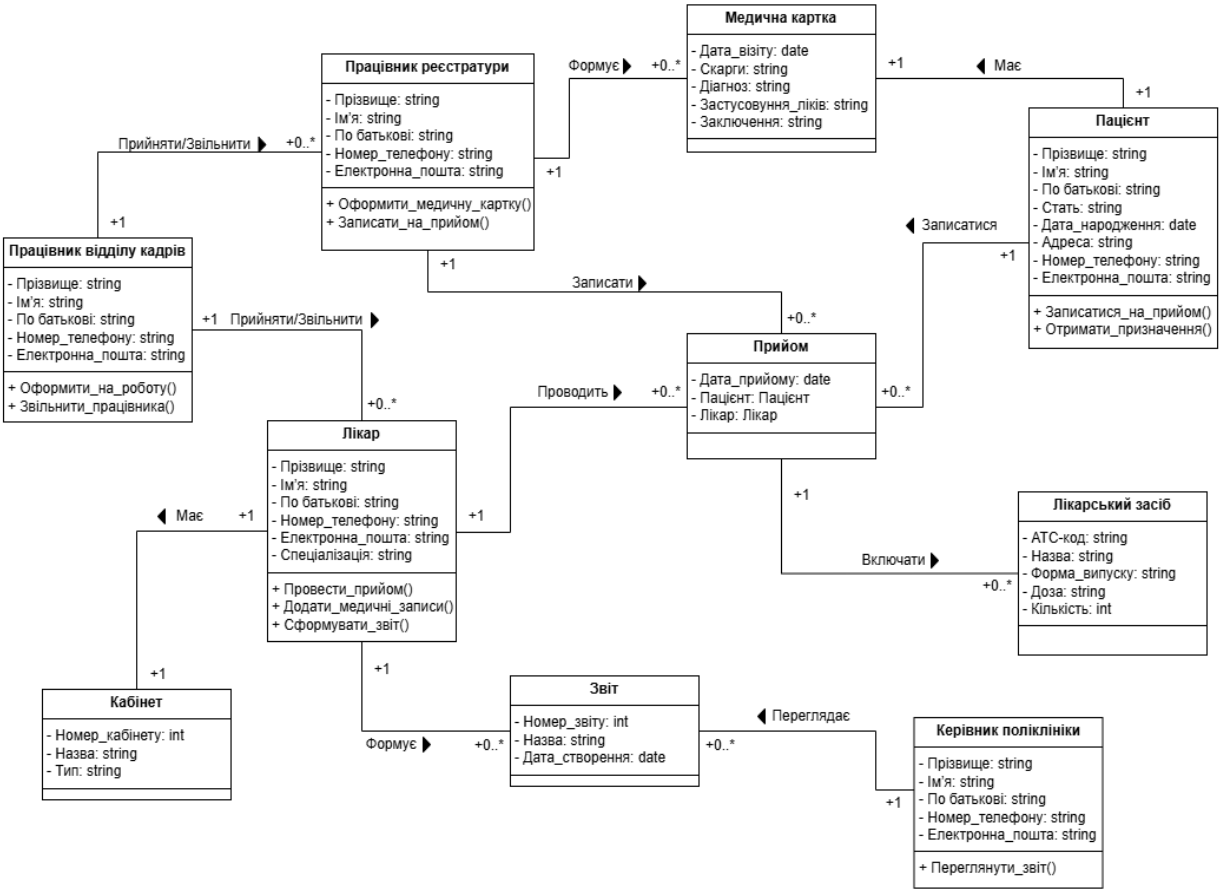


Рис. 17 Діаграма класів із асоціаціями

Ця діаграма є основою для подальшого проектування й реалізації функціональності системи. Вона показує, які об’єкти існують у системі, які дані вони зберігають (атрибути), які дії можуть виконувати (методи), а також як ці об’єкти пов’язані між собою (асоціації) [19].

Діаграма класів включає такі основні сутності: Працівник відділу кадрів, Працівник реєстратури, Пацієнт, Медична картка, Прийом, Лікар, Кабінет, Лікарський засіб, Звіт, Керівник поліклініки. Кожен клас має власний набір атрибутів, які зберігають ключову інформацію, а також методи, що відповідають за функціональну поведінку системи.

Особливу увагу приділено асоціаціям між класами, які чітко визначають ролі та взаємозв’язки. Наприклад, працівник відділу кадрів має можливість приймати або звільняти працівників реєстратури та лікарів. Це реалізовано у вигляді асоціації типу один до багатьох (1 до +0..*). Аналогічно, працівник реєстратури може оформлювати медичні картки та записувати пацієнтів на прийом, також це показано через зв’язок один до багатьох.

Клас “Пацієнт” пов’язаний з медичкою карткою у відношенні 1 до 1, що означає, що кожен пацієнт має лише одну картку. Крім того, пацієнт може бути записаний на декілька прийомів, що відображено через асоціацію 1 до +0..* з класом “Прийом”.

Лікар виконує ключові функції в системі, зокрема проведення прийомів, ведення медичних записів та формування звітів. Лікар також пов’язаний з кабінетом (1 до 1), прийомами (1 до +0..*), а також із звітами, які він формує. Звіти, у свою чергу, переглядає керівник поліклініки, що реалізовано асоціацією 1 до +0..*.

Клас “Лікарський засіб” асоційований з класом “Прийом”, оскільки лікарські засоби можуть бути застосовані під час прийому пацієнта. Це дозволяє системі фіксувати використовувані препарати.

Таким чином, діаграма класів із асоціаціями формує основу для подальшої реалізації системи, оскільки вона узагальнює ключові об’єкти та зв’язки між ними, і вже на цьому етапі відображає структуру майбутнього функціоналу. Вона

є інструментом, що поєднує аналітичний і проєктний підходи до створення системи, забезпечуючи перехід від логічної моделі до фізичної реалізації.

3.2.2 Діаграми кооперацій.

Діаграми кооперацій є важливим інструментом моделювання взаємодії між об’єктами у системі. Вони дозволяють детально відобразити, як об’єкти різних класів взаємодіють між собою для досягнення певної функціональності. У таких діаграмах використовуються різні типи зв’язків, зокрема асоціації, залежності, агрегації, композиції та узагальнення, що забезпечує повну картину логіки системи.

На рис. 17 представлено діаграм кооперації для процесу додавання працівників до системи.

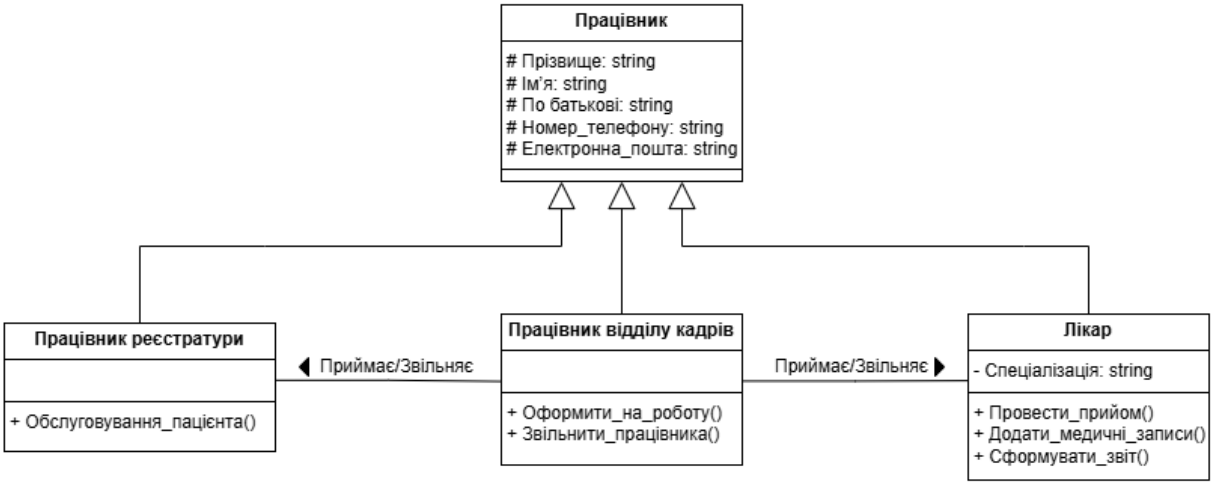


Рис. 17 Діаграма кооперації: додавання працівників до системи

У цьому сценарії центральним є клас “Працівник”, який виступає базовим для таких підкласів як “Працівник відділу кадрів”, “Працівник реєстратури” та “Лікар”. Зв’язок узагальнення забезпечує спадкування загальних характеристик, а також можливість обробки об’єктів усіх типів працівників через спільний інтерфейс. Працівник відділу кадрів виступає ініціатором створення нових об’єктів типу Лікар або Працівник реєстратури, що реалізується через асоціацію «приймає/звільняє». Таким чином, діаграма відображає логіку кадрового управління персоналом у системі.

На рис. 18 зображено діаграму кооперації для процесу запису на прийом. У цьому сценарії взаємодіють об’єкти класів “Пацієнт”, “Реєстратура”, “Прийом” та “Медична картка”. Пацієнт звертається до реєстратури, яка створює новий запис прийому (залежність) і зберігає його у відповідному об’єкті Прийом. Якщо у пацієнта ще немає медичної картки, її створюють – це реалізується через композицію, яка підкреслює, що медична картка не може існувати без пацієнта. Асоціації між класами демонструють основні дії під час запису: обробку запиту, збереження інформації та оновлення відповідних об’єктів.

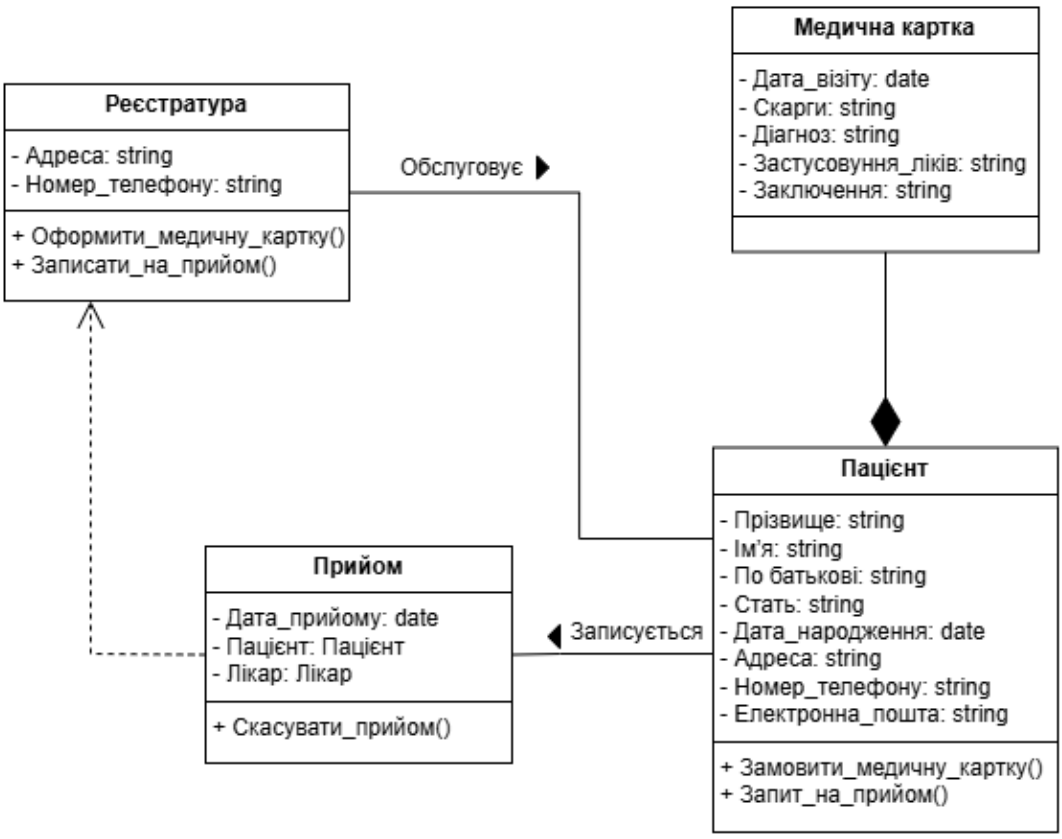


Рис. 18 Діаграма кооперації: записи на прийом

На рис.19 показано діаграму кооперації для сценарію прийому у лікаря. Учасники взаємодії – “Лікар”, “Пацієнт”, “Медична картка” та “Кабінет”. Лікар приймає пацієнта у своєму кабінеті (асоціація), проводить огляд і вносить відповідну інформацію до медичної картки пацієнта. Доступ до картки забезпечується через асоціацію, а сама картка, як і раніше, пов’язана з пацієнтом через композицію. Ця діаграма демонструє ключову функцію системи – обробку медичних даних під час прийому.

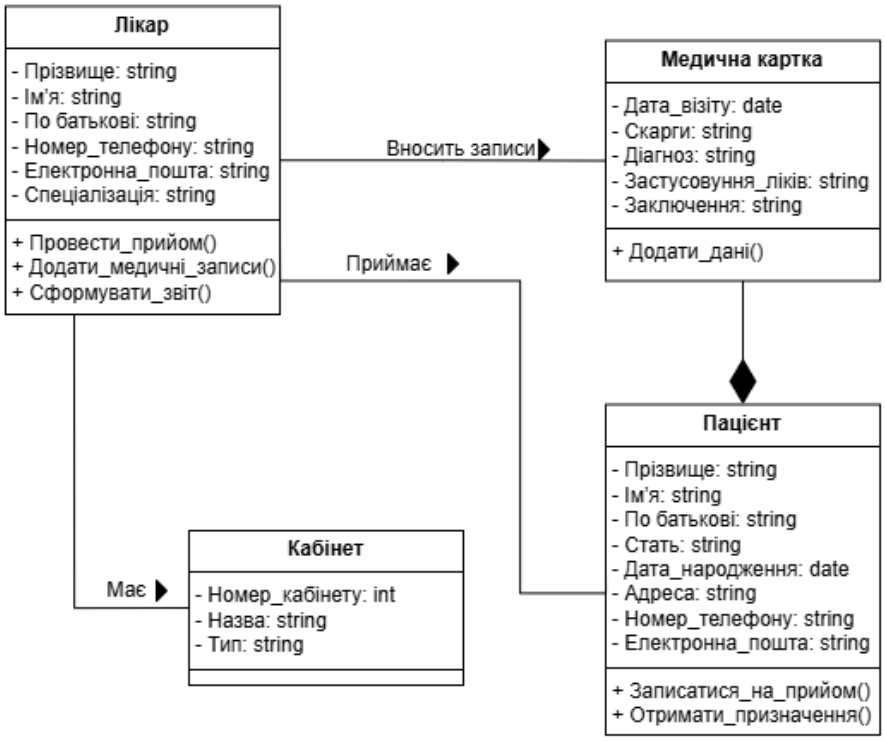


Рис. 19 Діаграма кооперації: прийом у лікаря

На рис. 20 подано діаграму кооперації для встановлення діагнозу.

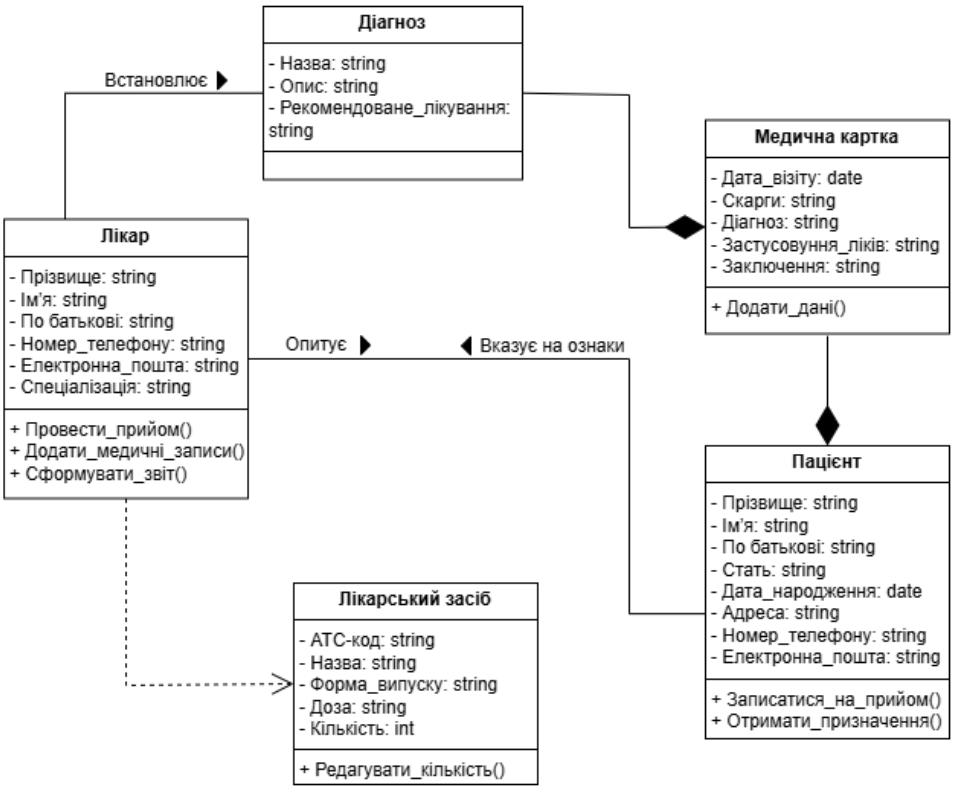


Рис. 20 Діаграма кооперації: встановлення діагнозу

У цьому процесі беруть участь “Лікар”, “Пацієнт”, “Медична картка”, “Діагноз” та “Лікарський засіб”. Лікар опитує пацієнта, визначає симптоми та

встановлює діагноз, який фіксується у медичній картці. Діагноз стає частиною картки (композиція), а зв’язки між лікарем, діагнозом і лікарськими засобами описуються через асоціації та залежності. У випадку потреби, лікар може використати відповідний лікарський засіб, що також відображено на діаграмі. Така діаграма дозволяє прослідкувати процес встановлення діагнозу та взаємодію між лікарем та пацієнтом і відповідними іншими даними.

3.3 Організаційна структура програмного забезпечення

Організаційна структура ПЗ представлена у вигляді діаграми пакетів, яка відображає логічний поділ системи обліку пацієнтів у районній поліклініці на окремі функціональні модулі (пакети) та взаємозв’язки між ними [20]. Діаграма пакетів зображена в Додатку Ж.

Уся система представлена у вигляді єдиного головного пакета, який охоплює чотири підпакети (кооперації), кожен із яких відповідає за реалізацію окремого бізнес-процесу. Така модульна структура забезпечує зручність супроводу, повторне використання компонентів та чіткий розподіл відповідальностей.

Опис пакетів

- *Додавання працівників до системи* – це базовий пакет, що відповідає за створення та облік співробітників закладу – працівників реєстратури і лікарів. Він є незалежним і не потребує функціоналу інших пакетів. У межах цього модуля реалізується створення об’єктів відповідних класів працівників, а також їхнє внесення у систему кадровим підрозділом.
- *Запис на прийом* – у цьому пакеті реалізується механізм взаємодії пацієнта з реєстратурою з метою запису до лікаря. Він залежить від пакета “Додавання працівників до системи”, оскільки для запису необхідна наявність зареєстрованих у системі лікарів та працівників

реєстратури. Цей пакет також оперує створенням медичної картки пацієнта при першому зверненні.

- *Прийом у лікаря* – цей модуль відповідає за проведення лікарського прийому: огляд пацієнта, взаємодію з медичною карткою, внесення попередніх спостережень. Він залежить від пакета “Запис на прийом”, оскільки прийом можливий тільки після відповідного запису.
- *Встановлення діагнозу* – описує процес встановлення діагнозу лікерам на основі прийому та даних у медичній картці. Також включає використання лікарських засобів. Пакет залежить від “Прийому у лікаря”, бо діагноз не може бути сформульований без попереднього огляду.

Між зазначеними пакетами реалізовано залежності. Ці залежності забезпечують логічну послідовність функціонування системи та її модульну архітектуру, що дозволяє зручно масштабувати або оновлювати окремі частини без порушення цілісності всього програмного забезпечення.

3.4 Компонентна структура системи

Компонентна структура ПЗ системи обліку пацієнтів у районній поліклініці представлена у вигляді діаграми компонентів. Вона моделює логічну архітектуру застосунку, розподілену на три основні рівні: інтерфейс користувача, бізнес-логіку та рівень доступу до даних [20]. Такий підхід дозволяє досягти високого рівня модульності, розширюваності та зручності супроводу системи. Діаграма компонентів представлена в Додатку К.

Далі розглянемо основні складові діаграми.

Інтерфейси користувачів

У системі передбачено декілька типів користувачів, кожен з яких має доступ до певних функціональних вікон:

- *IAdministrator* – доступ до: EmployeeManagement.xaml (модуль керування працівниками);
- *IRegistryEmployee* – доступ до: PatientManagement.xaml (управління пацієнтами) та AppointmentManagement.xaml (керування записами на прийом);
- *IDoctor* – доступ до: MedicalCardManagement.xaml (робота з медичними картками пацієнтів).

Всі інші вікна доступні кожному із типів користувачів.

Компоненти користувацького інтерфейсу (UI)

Інтерфейс представлений XAML-файлами, які реалізують вікна та сторінки графічного інтерфейсу: Login.xaml, MainMenu.xaml, xEmployeeManagement.xaml, PatientManagement.xaml, Assistant.xaml, AppointmentManagement.xaml, News.xaml, MedicalCardManagement.xaml.

Ці компоненти забезпечують взаємодію користувачів із системою та викликають відповідну бізнес логіку.

Сервісні компоненти (бізнес-логіка)

Реалізують основну логіку роботи програми та обробляють дані, отримані з інтерфейсу:

- AuthService.cs – обробка авторизації;
- EmployeeService.cs, PatientService.cs, AppointmentService.cs, MedicalCardService.cs – обробка основних операцій із даними;
- NewService.cs, AssistantService.cs – допоміжна логіка.

Репозиторії (доступ до бази даних)

Компоненти, які безпосередньо взаємодіють з базою даних, реалізуючи CRUD-операції: UserRepository.cs, EmployeeRepository, PatientRepository.cs, AppointmentRepository.cs, MedicalRepository.cs.

Вони служать посередником між сервісами та фізичним зберіганням інформації.

Зв’язки між компонентами поділяються на два типи: залежності та реалізації. Залежності вказують на використання одного компонента іншим, наприклад, xaml-файли залежать від відповідних сервісів, які у свою чергу взаємодіють з репозиторіями. Реалізації демонструють пряме впровадження функціональності – інтерфейси користувачів реалізуються через відповідні xaml-файли, а репозиторії безпосередньо пов’язані з базою даних.

Загалом, компонентна структура забезпечує розділення обов’язків між модулями, що значно спрощує розробку, тестування та підтримку системи на всіх етапах життєвого циклу проєкту.

3.5 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі розробки прикладного програмного забезпечення для пацієнтів у районній поліклініці було використано сучасний інструментарій, що забезпечує зручність розробки, гнучкість та ефективну інтеграцію між компонентами системи.

В якості основного середовища розробки було обрано **Visual Studio**. Це одна з найпотужніших та найзручніших інтегрованих середовищ розробки, яка надає вбудовану підтримку для роботи з .NET, мовою C#, базами даних, звітами, а також широким спектром інструментів для налагодження та тестування [21]. Visual Studio дозволяє зручно керувати великими проєктами, працювати з графічними інтерфейсами, підключати потрібні пакети, а також забезпечує ефективну інтеграцію з системами контролю версій.

Як мову програмування було обрано **C#**, що є основною мовою платформи .NET. Її основними перевагами є висока продуктивність, підтримка об’єктно-орієнтованого підходу, наявність великої кількості бібліотек та активна спільнота [21]. Завдяки синтаксичній строгості та чіткості C# дозволяє створювати надійні, масштабовані рішення, що особливо важливо для медичних інформаційних систем.

Для реалізації графічного інтерфейсу користувача було застосовано технологію **WPF (Windows Presentation Foundation)** на платформі **.NET**. Це рішення обрано через його можливості створення адаптивних, сучасних інтерфейсів з розділенням логіки та подання, а також підтримку декларативного опису UI в XAML. Крім того, WPF забезпечує широкі можливості налаштування інтерфейсу та зручну інтеграцію з іншими модулями програми.

Як вже зазначалося раніше, зберігання даних у системі здійснюється за допомогою **Microsoft SQL Server**. Для зручного адміністрування та розробки структури бази даних було використано **SQL Server Management Studio (SSMS)** – офіційний клієнт Microsoft, який дозволяє керувати таблицями, обробляти запити, створювати резервні копії та моніторити продуктивність СУБД [15].

Для реалізації функціоналу створення звітів у програмному забезпеченні були використані **RDLC-звіти** (Report Definition Language Client-side) від Microsoft. Звіти інтегруються у Visual Studio через **Microsoft Report Viewer**, який забезпечує відображення звітів прямо в інтерфейсі програми. Це дозволило формувати звіти без потреби в додаткових зовнішніх компонентах.

Однією з інноваційних складових проєкту стала реалізація функції інтелектуальної допомоги при постановці діагнозу. Для цього була використана велика мовна модель **LLaMA 3.1**, розроблена компанією Meta. Модель було обрано за її відкрите ліцензування, високу точність генерації відповідей та можливість локального розгортання. Використання локальної моделі забезпечує повну конфіденційність даних без потреби в зовнішньому API.

Для локального запуску LLaMA було використано **Ollama** – зручний інструмент, що дозволяє розгортати та керувати великими мовними моделями на локальному рівні [22]. Ollama підтримує ефективне використання апаратних ресурсів, простий у налаштуванні та сумісний з різними LLM-архітектурами. Завдяки цьому рішення не лише підвищує інтелектуальні можливості системи, а й забезпечує захист персональних медичних даних, що обробляються ШІ-модулем.

Модель LLaMA 3.1 була адаптована під специфіку медичних консультацій шляхом донавчання на власному датасеті, який включав медичні кейси та спеціалізовані словники. Для цього було використано мову програмування Python разом із бібліотекою Hugging Face та методом PEFT (Parameter-Efficient Fine-Tuning), зокрема технікою LoRA (Low-Rank Adaptation). Це дозволило додати до базової моделі невеликі адаптивні шари, які навчилися на специфічних даних, не змінюючи при цьому всю архітектуру моделі. Така медотика зробила процес донавчання більш швидким і економним щодо обчислювальних ресурсів. Наприкінці донавчені параметри було інтегровано з базовою моделлю, що дозволило отримати покращену версію, здатну точніше інтерпретувати та генерувати релевантні відповіді в межах медичної тематики. Це значно підвищило якість інтелектуальної підтримки лікарів, допомагаючи уникнути неточностей на початковому етапі діагностування.

Таким чином, обраний інструментарій забезпечив високий рівень гнучкості, розширюваності та надійності системи. Він дозволяє ефективно підтримувати всі етапи розробки, від проєктування інтерфейсу до створення інтелектуальних функцій, що в сукупності робить програмне забезпечення сучасним, функціональним і готовим до впровадження в медичних установах.

3.6 Алгоритмізація та програмування програмних модулів

Алгоритмізація функціональних процесів системи є важливою складовою програмної реалізації, що забезпечує послідовність, точність та ефективність виконання основних операцій. На цьому етапі визначаються ключові дії, що виконуються під час взаємодії користувача з інтерфейсом системи, обробки введених даних, їх перевірки та збереження у базу даних. Для ілюстрації буде наведено приклади двох алгоритмів: алгоритму запису результатів прийому у медичну картку пацієнта та алгоритму консультації лікаря із асистентом.

На рис. 21 зображена блок-схема алгоритму запису даних у медичну картку. Вона демонструє послідовність дій при внесенні результатів прийому до

медичної картки пацієнта. Після початку процесу користувач вводить інформацію про сам прийом і застосовані лікарські засоби. Далі система виконує перевірку коректності введених даних: якщо виявлено помилки, виводиться відповідне повідомлення, і користувач повертається до етапу введення даних. У разі правильного введення даних вони проходять обробку і зберігаються в базі даних. Після цього система перевіряє, чи збереження пройшло успішно, та формує користувача або про помилку і направляє знову на етап введення даних, або про успішне завершення операції.

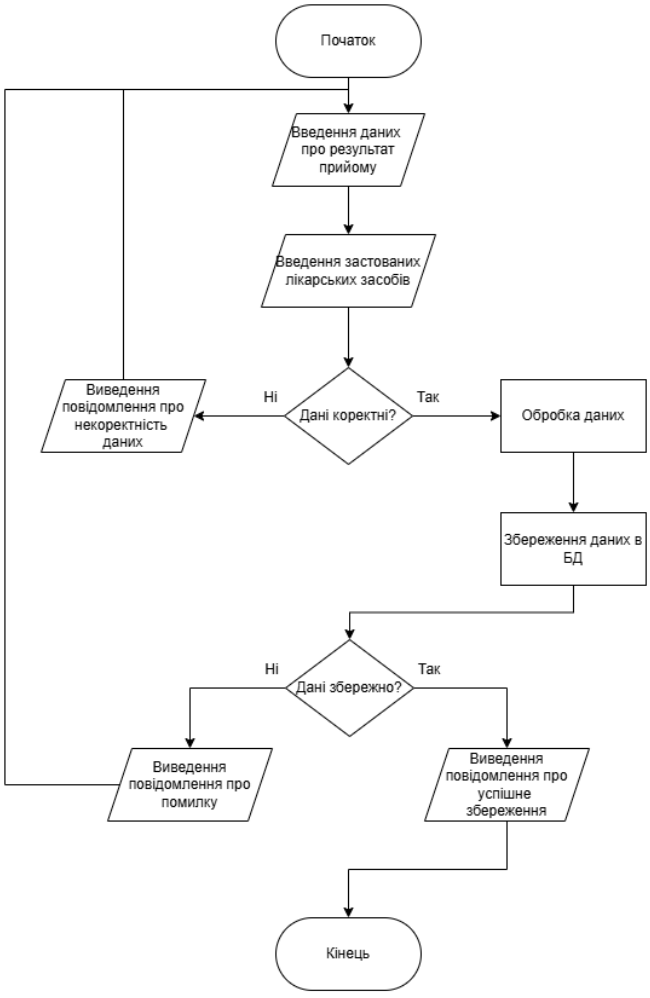


Рис. 21 Блок-схема алгоритму запису результатів прийому в медичну картку

Далі розглянемо основні частини коду, які реалізовують цей алгоритм. Фрагмент коду, наведений на рис. 22, представляє клас AdministeredDrugTemp, який використовується для тимчасового зберігання інформації про лікарські засоби, що були використані для тимчасового зберігання інформації про лікарські засоби, що були використані під час прийому.

```
public class AdministeredDrugTemp
{
    3 references
    public string ID_drug { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string DosageDisplay { get; set; }
    1 reference
    public string QuantityDisplay { get; set; }
    2 references
    public int Dosage { get; set; }
    2 references
    public int Quantity { get; set; }
}
```

Рис. 22 Клас AdministeredDrugTemp

На рис. 23 зображено метод AddMedicinal_Click, який викликається при додаванні нового лікарського засобу до списку на формі. Цей метод виконує послідовну перевірку коректності введених користувачем даних, після чого за допомогою методу GetGrugDetailsById класу DatabaseHelper (в якому реалізовано логіку доступу до бази даних), зображеного на рис. 24, підвантажується повна інформація про лікарський засіб із бази даних за його ідентифікатором. У результаті формується новий об’єкт AdministeredDrugTemp, який додається до колекції administeredDrugs, що згодом буде збережена.

```
private void AddMedicinal_Click(object sender, RoutedEventArgs e)
{
    string drugId = txtMedicinalProduct.SelectedValue.ToString();
    int dosage = int.Parse(txtDosage.Text);
    int quantity = int.Parse(txtQuantity.Text);

    if (txtMedicinalProduct.SelectedValue == null ||
        string.IsNullOrWhiteSpace(txtDosage.Text) ||
        string.IsNullOrWhiteSpace(txtQuantity.Text))
    {
        MessageBox.Show("Будь ласка, оберіть лікарський засіб та введіть дозу із кількістю!", "Інформація!");
        return;
    }

    if (administeredDrugs.Any(d => d.ID_drug == drugId))
    {
        MessageBox.Show("Такий лікарський засіб вже додано!", "Помилка!");
        return;
    }

    string drugName, releaseForm, dosageUnit, quantityUnit;
    try
    {
        (drugName, releaseForm, dosageUnit, quantityUnit) = DatabaseHelper.GetDrugDetailsById(drugId);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка при отриманні даних про лікарський засіб: {ex.Message}", "Помилка!");
        return;
    }

    var item = new AdministeredDrugTemp
    {
        ID_drug = drugId,
        Name = $"{drugName} ({releaseForm})",
        Dosage = dosage,
        Quantity = quantity,
        DosageDisplay = $"{dosage} {dosageUnit}",
        QuantityDisplay = $"{quantity} {quantityUnit}"
    };

    administeredDrugs.Add(item);
    lstMedicinal.Items.Refresh();
}
```

Рис. 23 Метод додавання лікарського засобу до списку на формі

```
public static (string Name, string ReleaseForm, string DosageUnit, string QuantityUnit) GetDrugDetailsById(string drugId)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        SqlCommand cmd = new SqlCommand(
            "SELECT Drug_name, Release_form, Dosage_unit, Quantity_unit FROM Medicinal_product WHERE ID_drug = @ID",
            connection);
        cmd.Parameters.AddWithValue("@ID", drugId);

        using (SqlDataReader reader = cmd.ExecuteReader())
        {
            if (reader.Read())
            {
                string name = reader["Drug_name"].ToString();
                string form = reader["Release_form"].ToString();
                string dosageUnit = reader["Dosage_unit"].ToString();
                string quantityUnit = reader["Quantity_unit"].ToString();
                return (name, form, dosageUnit, quantityUnit);
            }
            else
            {
                throw new Exception("Лікарський засіб не знайдено в базі даних.");
            }
        }
    }
}
```

Рис. 24 Метод отримання детальної інформації про лікарський засіб з бази даних

Наступний важливий етап показано на рис. 25 – це метод SaveButton_Click, який активується при натисканні кнопки збереження. Він перевіряє заповненість усіх обов’язкових полів форми, і у випадку їх наявності викликає відповідні методи збереження даних. Зокрема, UpdateConsultationReport зберігає інформацію про прийом, а InsertAdministeredDrugs записує список лікарських засобів до таблиці бази даних.

```
private void SaveButton_Click(object sender, RoutedEventArgs e)
{
    string consultationId = cmbNumberAppointment.SelectedValue?.ToString();
    string complaints = txtComplaints.Text.Trim();
    string diagnosis = txtDiagnosis.Text.Trim();
    string conclusion = txtConclusion.Text.Trim();

    if (string.IsNullOrEmpty(consultationId) ||
        string.IsNullOrEmpty(complaints) ||
        string.IsNullOrEmpty(diagnosis) ||
        string.IsNullOrEmpty(conclusion))
    {
        MessageBox.Show("Будь ласка, заповніть всі поля консультації!", "Помилка!");
        return;
    }

    try
    {
        // Оновлення даних консультації
        DatabaseHelper.UpdateConsultationReport(consultationId, complaints, diagnosis, conclusion);

        // Збереження лікарських засобів
        if (administeredDrugs.Any())
        {
            DatabaseHelper.InsertAdministeredDrugs(consultationId, administeredDrugs.ToList());
        }

        MessageBox.Show("Інформація успішно збережена!", "Інформація!");
    }
    catch (Exception ex)
    {
        MessageBox.Show("Сталася помилка при збереженні даних: " + ex.Message, "Помилка");
    }
}
```

Рис. 25 Метод збереження результатів прийому в базу даних

На рис. 26 зображено частину класу DatabaseHelper, зокрема методи, які викликаються у методі SaveButton_Click. Метод UpdateConsultationReport

оновлює відповідний запис у таблиці Consultation_report, встановлюючі нові значення полів скарг, діагнозу і заключення за вказаний ідентифікатором консультації. Метод InsertAdministeredDrugs проходить по списку лікарських засобів, які були додані користувачем на формі, і вставляє відповідні записи у таблицю Administered_drugs. Цей процес відповідає кроку збереження основної медичної інформації у блок-схемі.

```
public static class DatabaseHelper
{
    private static string connectionString =
        ConfigurationManager.ConnectionStrings[
            "DistrictPolyclinic.Properties.Settings.DistrictPolyclinicConnectionString"]
        .ConnectionString;

    1 reference
    public static void UpdateConsultationReport(string consultationId, string complaints, string diagnosis, string conclusion)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand updateCmd = new SqlCommand(
                @"UPDATE Consultation_report
                  SET Complaints = @Complaints, Diagnosis = @Diagnosis, Conclusion = @Conclusion
                  WHERE ID_consultation = @ID", connection);

            updateCmd.Parameters.AddWithValue("@Complaints", complaints);
            updateCmd.Parameters.AddWithValue("@Diagnosis", diagnosis);
            updateCmd.Parameters.AddWithValue("@Conclusion", conclusion);
            updateCmd.Parameters.AddWithValue("@ID", consultationId);
            updateCmd.ExecuteNonQuery();
        }
    }

    1 reference
    public static void InsertAdministeredDrugs(string consultationId, List<AdministeredDrugTemp> drugs)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            foreach (var drug in drugs)
            {
                SqlCommand insertCmd = new SqlCommand(
                    @"INSERT INTO Administered_drugs (ID_consultation, ID_drug, Dosage, Quantity_used)
                      VALUES (@ID_consultation, @ID_drug, @Dosage, @Quantity)", connection);

                insertCmd.Parameters.AddWithValue("@ID_consultation", consultationId);
                insertCmd.Parameters.AddWithValue("@ID_drug", drug.ID_drug);
                insertCmd.Parameters.AddWithValue("@Dosage", drug.Dosage);
                insertCmd.Parameters.AddWithValue("@Quantity", drug.Quantity);
                insertCmd.ExecuteNonQuery();
            }
        }
    }
```

Рис. 26 Методи класу DatabaseHelper

Таким чином, програмна реалізація алгоритму охоплює перевірку даних, роботу з тимчасовими структурами, а також збереження текстової інформації та списку препаратів у базу даних. Всі ці етапи відповідають крокам, наведеним у блок-схемі.

На рис. 27 представлено блок-схему алгоритму консультації з асистентом, що реалізує логіку взаємодії користувача із вбудованим медичним асистентом. Процес починається з того, що користувач вводить запит у відповідне текстове поле та надсилає його за допомогою клавіші Enter або кнопки “Відправити”. Далі здійснюється перевірка факту відправлення повідомлення. Якщо повідомлення

не надіслано, система очікує дій користувача. Якщо ж запит був відправлений, повідомлення користувача додається до інтерфейсу, після чого воно передається локальній моделі для обробки. У відповідь асистент генерує повідомлення, яке відображається на екрані. Після цього система знову переходить у режим очікування нового запиту. Якщо користувач вводить нове повідомлення – процес повторюється. У разі завершення сеансу взаємодія припиняється.

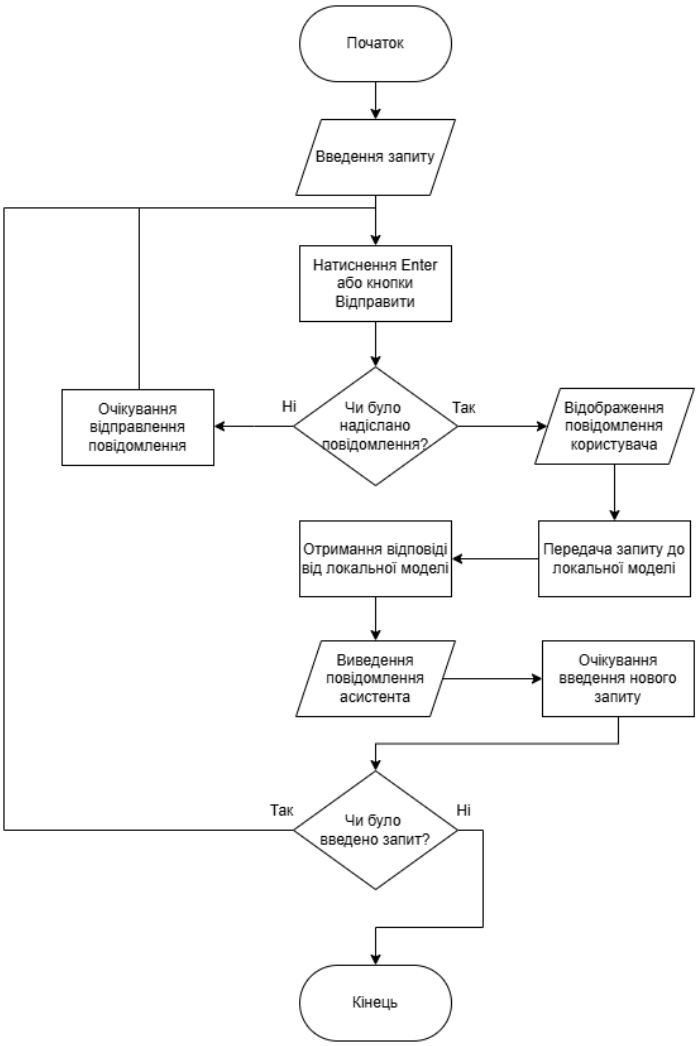


Рис. 27 Блок-схема алгоритму консультації з асистентом

Перейдемо до аналізу основних частин коду, які забезпечують виконання цього алгоритму. На рис. 28 зображено конструктор класу Assistant, який ініціалізує колекцію повідомлень messages, прив’язану до візуального елементу ItemsControl. У разі першого запуску, у список додається вітальне повідомлення асистента з пропозицією допомоги. Це дозволяє створити початкову взаємодію з користувачем ще до введення першого запиту.

```
public Assistant()
{
    InitializeComponent();
    messages = SessionManager.SessionMessages;

    if (messages.Count == 0)
    {
        messages.Add(new Message
        {
            IsRequest = false,
            Text = "Привіт! Як я можу допомогти вам сьогодні?"
        });
    }

    messagesItemsControl.ItemsSource = messages;
}
```

Рис. 28 Конструктор сторінки асистента

На рис. 29 представлено метод messageTextBox_PreviewKeyDown, який реагує на натискання клавіші Enter. Якщо введений текст не є порожнім, викликається асинхронний метод SendUserMessageAsync, який формує повідомлення користувача, додає його до списку повідомлень та ініціює запит до локальної моделі асистента.

```
private async void messageTextBox_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (e.Key == Key.Enter && !Keyboard.Modifiers.HasFlag(ModifierKeys.Shift))
    {
        e.Handled = true;

        if (!string.IsNullOrWhiteSpace(messageTextBox.Text))
            await SendUserMessageAsync(messageTextBox.Text.Trim());
    }
}
```

Рис. 29 Метод для обробки клавіші Enter

На рис. 30 зображено метод sendButton_Click, який є обробником події натискання кнопки “Відправити”. Аналогічно до попереднього методу, перевіряється наявність введеного тексту, після чого викликається SendUserMessageAsync.

```
private async void sendButton_Click(object sender, RoutedEventArgs e)
{
    if (!string.IsNullOrWhiteSpace(messageTextBox.Text))
        await SendUserMessageAsync(messageTextBox.Text.Trim());
}
```

Рис. 30 Метод для обробки кнопки “Відправити”

На рис. 31 наведено метод SendUserMessageAsync, що відповідає за додавання повідомлення користувача до списку, очищення поля введення, попередню підготовку порожнього об’єкта для відповіді асистента, а також запуск методу GetAssistantResponseStreamingAsync для надсилання запиту до локального API й поетапного формування відповіді.

```
private async Task SendUserMessageAsync(string userMessage)
{
    try
    {
        var requestMessage = new Message { IsRequest = true, Text = userMessage };
        messages.Add(requestMessage);
        messageTextBox.Text = "";

        var assistantMessage = new Message { IsRequest = false, Text = "" };
        messages.Add(assistantMessage);

        messagesScrollViewer.ScrollToEnd();

        await GetAssistantResponseStreamingAsync(userMessage, assistantMessage);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Помилка: {ex.Message}", "Помилка!");
    }
}
```

Рис. 31 Метод надсилання повідомлення користувача

Метод GetAssistantResponseStreamingAsync, який представлено на рис. 32, здійснює HTTP-запит до локального сервера (на базі моделі medical-assistant), з параметром «stream = true», що дозволяє поступово отримувати відповідь. Зчитування рядків потоку виконується в циклі, а кожна частина відповіді додається до тексту повідомлення асистента з ефектом поступового друку.

```
private async Task GetAssistantResponseStreamingAsync(string userMessage, Message assistantMessage)
{
    try
    {
        using (var client = new HttpClient { BaseAddress = new Uri(serverAddress) })
        {
            var content = new StringContent(
            $"{{\"model\": \"medical-assistant\", \"prompt\": \"{userMessage}\", \"stream\": true}}",
            Encoding.UTF8, "application/json");

            var request = new HttpRequestMessage(HttpMethod.Post, "api/generate") { Content = content };
            var response = await client.SendAsync(request, HttpCompletionOption.ResponseHeadersRead);

            if (!response.IsSuccessStatusCode)
            {
                assistantMessage.Text = "Сталась помилка при отриманні відповіді від асистента!";
                return;
            }

            using (var stream = await response.Content.ReadAsStreamAsync())
            using (var reader = new StreamReader(stream))
            {
                while (!reader.EndOfStream)
                {
                    var line = await reader.ReadLineAsync();
                    if (string.IsNullOrWhiteSpace(line)) continue;

                    try
                    {
                        var json = JObject.Parse(line);
                        var part = json["response"]?.ToString() ?? json["text"]?.ToString();
                        if (!string.IsNullOrEmpty(part))
                        {
                            assistantMessage.Text += part;
                            assistantMessage.NotifyTextChanged();
                            messagesScrollViewer.ScrollToEnd();
                            await Task.Delay(5);
                        }
                    }
                    catch
                    {
                        continue;
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        assistantMessage.Text = $"Помилка при зв'язку з сервером: {ex.Message}";
    }
}
```

Рис. 32 Метод отримання відповіді асистента

На рис. 33 подано клас Message, який реалізує інтерфейс INotifyPropertyChanged і містить властивості Text та IsRequest. Це дозволяє динамічно оновлювати інтерфейс при зміні вмісту повідомлень.

```
public class Message : INotifyPropertyChanged
{
    private string _text;
    private bool _isRequest;

    public event PropertyChangedEventHandler PropertyChanged;

    4 references
    public bool IsRequest
    {
        get => _isRequest;
        set
        {
            _isRequest = value;
            OnPropertyChanged(nameof(IsRequest));
        }
    }

    8 references
    public string Text
    {
        get => _text;
        set
        {
            _text = value;
            OnPropertyChanged(nameof(Text));
        }
    }

    1 reference
    public void NotifyTextChanged() => OnPropertyChanged(nameof(Text));

    3 references
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Рис. 33 Клас повідомлення Message

Ця реалізація забезпечує інтерактивну консультацію в режимі реального часу з використанням локальної LLM-моделі, що імітує живе спілкування користувача з цифровим асистентом.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Ретельне тестування є частиною процесу розробки будь-якої програмної системи, оскільки воно дозволяє виявити помилки, недоліки та забезпечити відповідність функціональних можливостей заявленим вимогам. У ході тестування було проаналізовано різні сценарії використання системи, охоплено взаємодію між компонентами та перевірено роботу під різними ролями користувачів. Це дозволило оцінити не лише окремі функції, а й загальну надійність та зручність роботи системи.

Для забезпечення надійної роботи розробленої системи було проведено комплексне тестування, що включало кілька основних видів.

- **Модульне тестування (Unit Testing)** – проводилось для перевірки окремих компонентів системи, таких як обробка форм введення, перевірка введених даних, окремі функції API та базові обчислення. Метою цього тестування було виявлення логічних помилок на рівні окремих програмних блоків.
- **Інтеграційне тестування (Integration Testing)** – здійснювалось з метою перевірки коректної взаємодії між окремими модулями системи. Воно дозволило виявити можливі помилки, що виникають при обміні даними між компонентами під час спільної роботи.
- **Тестування зручності використання (Usability Testing)** – здійснювалося за участі тестових користувачів, що імітували роботу всіх типів користувачів системи. Вивчалась логічність інтерфейсу, швидкість доступу до основних функцій, а також загальна зручність навігації.

- **Системне тестування (System Testing)** – охоплювало перевірку всієї системи в цілому, включаючи базу даних, серверну частину та інтерфейс користувача. Перевірялась стабільність роботи при виконанні стандартних сценаріїв користувача та відповідність вимогам.
- **Тестування “чорної скриньки” (Black Box Testing)** – було проведене з метою перевірки поведінки системи без доступу до її внутрішнього коду. Оцінювався функціонал з точки зору кінцевого користувача [23].

Нижче наведено один із прикладів проведення тестування системи за методом “чорної скриньки”, під час якого, не маючи доступу до внутрішнього коду, задаються вхідні дані, очікуються конкретні результати та виконується послідовність дій під різними ролями користувача з метою виявлення помилок у роботі основного функціоналу системи. На кожному етапі перевіряється відповідність фактичних результатів очікуваним, що дозволяє оцінити коректність виконання фактичних результатів очікуваним, що дозволяє оцінити коректність виконання операцій та стабільність роботи системи.

Приклад тестування за методом “чорної скриньки”

Наведений приклад імітує типову поведінку користувача без знання внутрішньої реалізації системи, що відповідає принципам тестування методом “чорної скриньки”.

- **Вхід у систему під обліковим записом працівника реєстратури**

Користувач відкриває форму авторизації та вводить облікові дані співробітника реєстратури. На рис. 34 показано відповідне вікно авторизації. Після успішного входу система автоматично перенаправляє користувача до головного меню.

Рис. 34 Форма авторизації для входу в систему

- **Перевірка головного меню**

На головному екрані відображаються оголошення та інформація про кількість завершених і запланованих прийомів на поточний день. Це зображено на рис. 35.

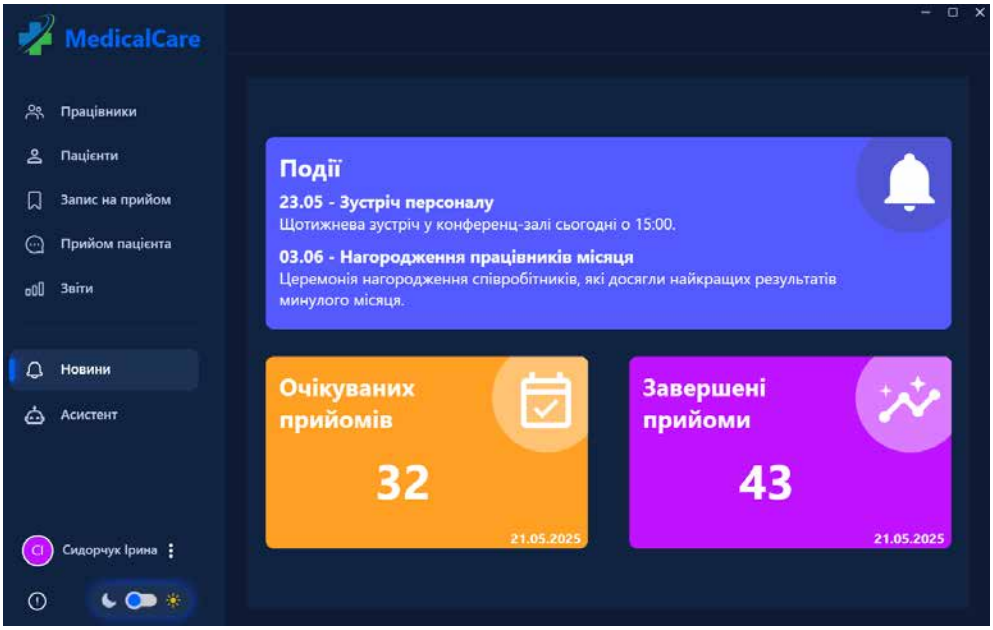


Рис. 35 Головне меню після авторизації користувача

- **Створення нового запису на прийом**

Користувач відкриває розділ “Запис на прийом”, де обирає лікаря зі списку, як показано на рис. 36. Після цього, відповідно до рис. 37, користувач у вікні з розкладом натискає на комірку відповідного дня та часу для створення запису. Після підтвердження запис з’являється на графіку, як показано на рис. 38.

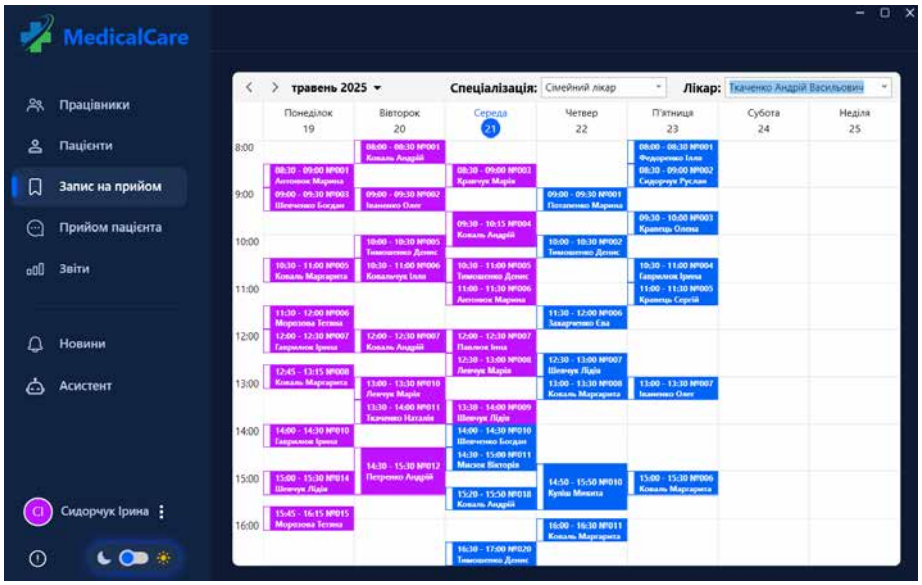


Рис. 36 Інтерфейс розділу "Запис на прийом"

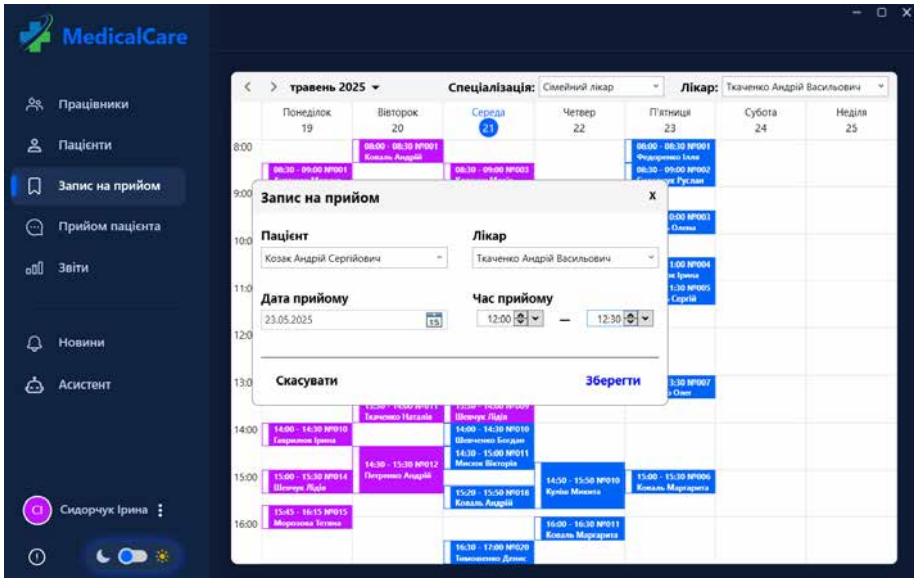


Рис. 37 Вибір пацієнта, дати та часу для створення запису

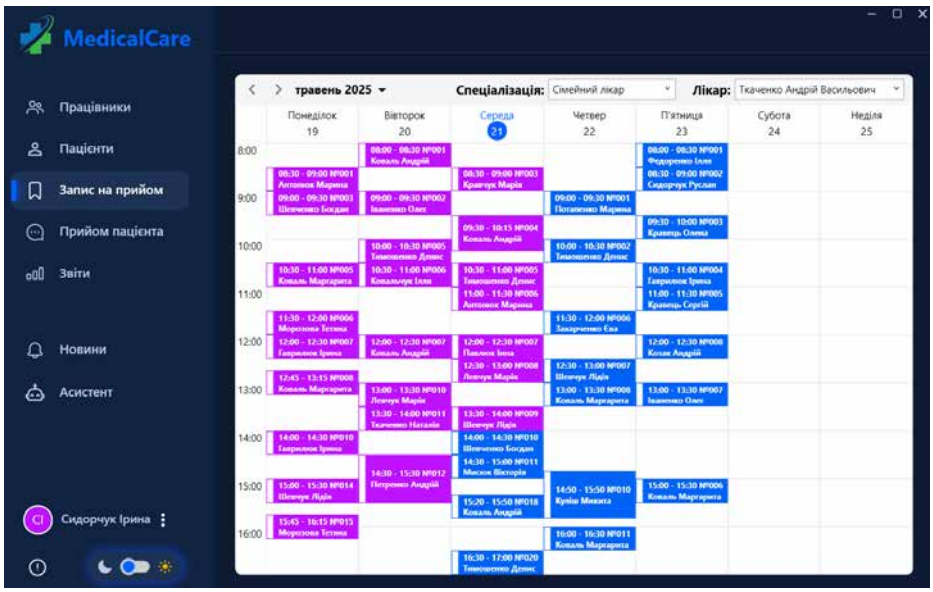


Рис. 38 Графік із доданим новим записом на прийом

- **Вхід під обліковим записом лікаря та заповнення результатів прийому**

Лікар авторизується у системі, що ілюструє рис. 39. Далі він переходить у розділ “Прийом пацієнта” та обирає “Результати прийому”, після чого відкривається вікно для заповнення даних, що показано на рис. 40. У вікні введення результатів лікар заповнює необхідні дані й натискає кнопку “Зберегти”. Після успішного збереження користувач отримує відповідне повідомлення, яке показано на рис. 41.

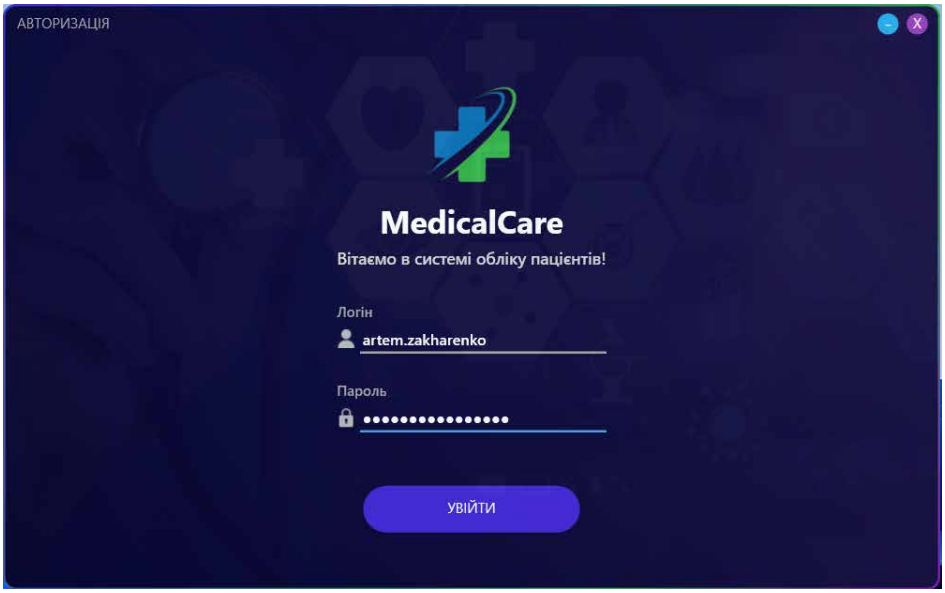


Рис. 39 Авторизація лікаря в системі

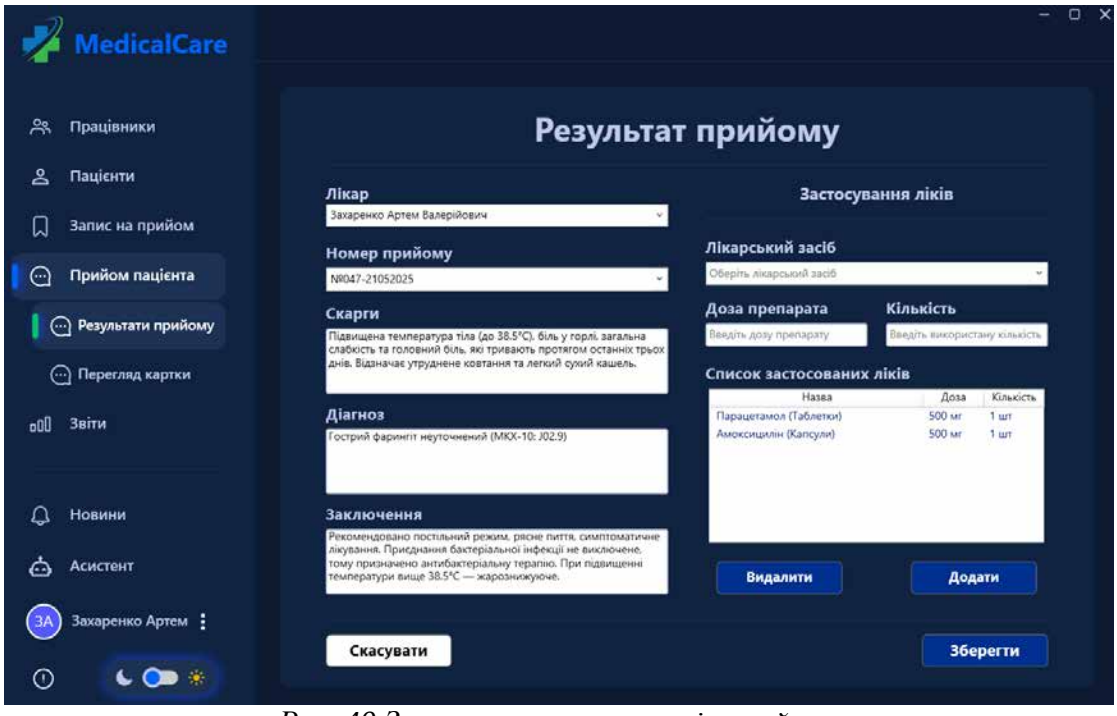


Рис. 40 Заповнення результатів прийому

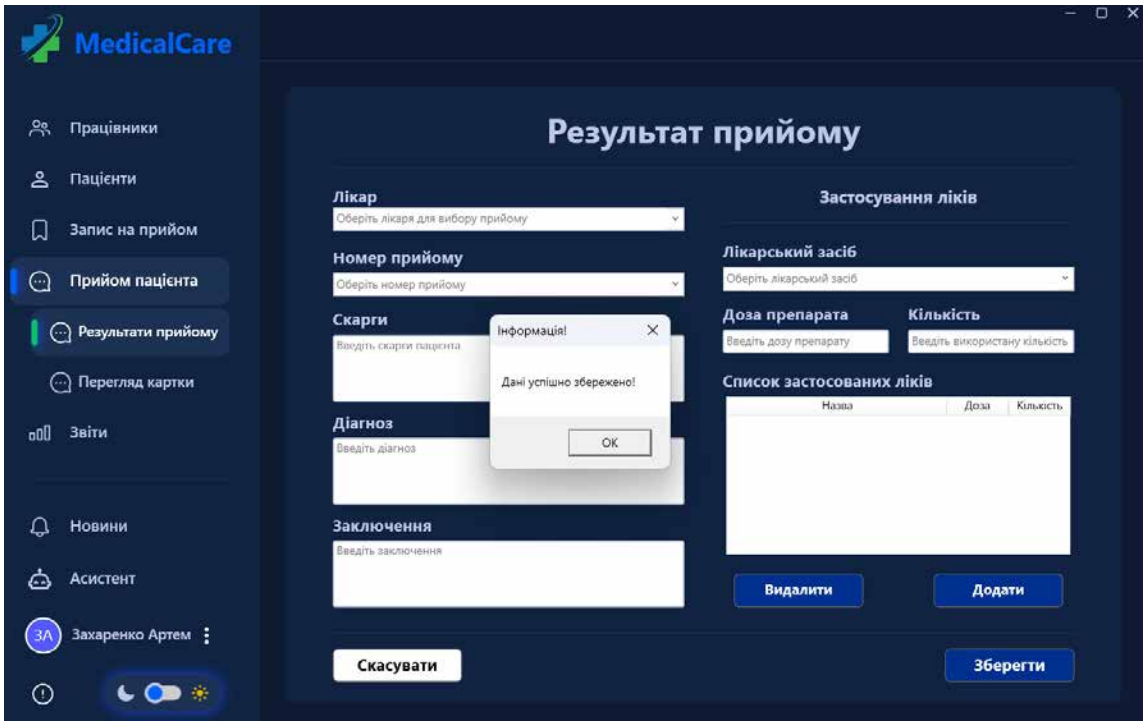


Рис. 41 Результат збереження даних

- **Генерація звіту за прийомами**

Користувач переходить до розділу “Звіти” й обирає пункт меню “Прийоми за період”. Далі з’являється форма вибору лікаря та періоду, як видно на рис. 42. Після підтвердження вибору система формує зведений звіт, приклад якого наведено на рис. 43.

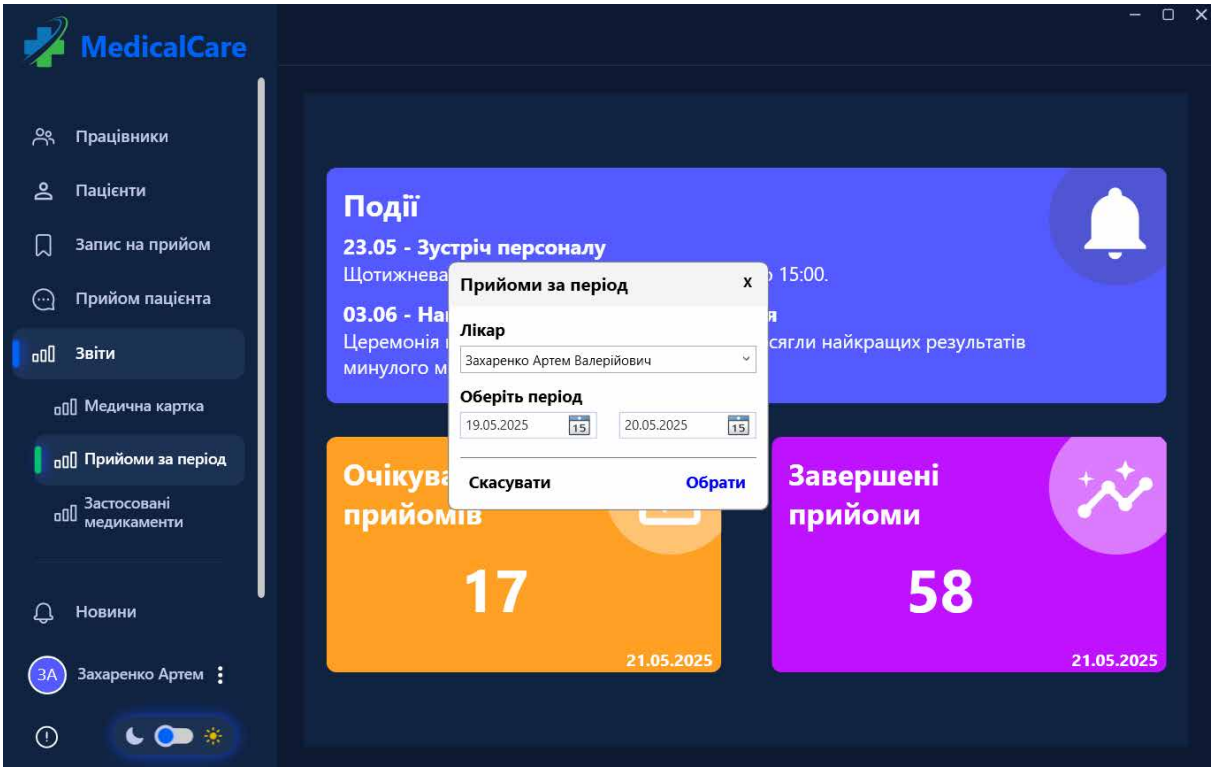
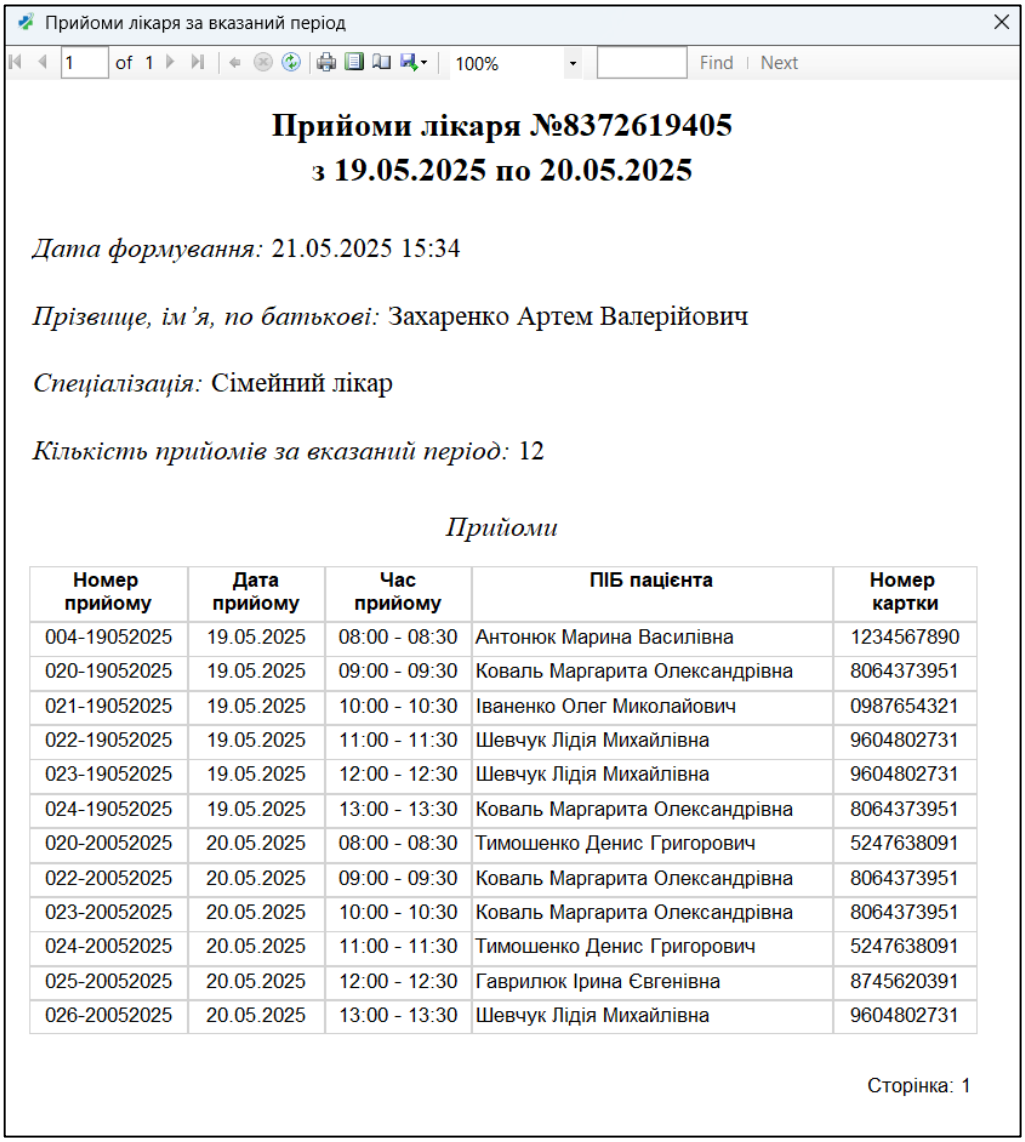


Рис. 42 Форма вибору лікаря та періоду для звіту



Номер прийому	Дата прийому	Час прийому	ПІБ пацієнта	Номер картки
004-19052025	19.05.2025	08:00 - 08:30	Антонюк Марина Василівна	1234567890
020-19052025	19.05.2025	09:00 - 09:30	Коваль Маргарита Олександрівна	8064373951
021-19052025	19.05.2025	10:00 - 10:30	Іваненко Олег Миколайович	0987654321
022-19052025	19.05.2025	11:00 - 11:30	Шевчук Лідія Михайлівна	9604802731
023-19052025	19.05.2025	12:00 - 12:30	Шевчук Лідія Михайлівна	9604802731
024-19052025	19.05.2025	13:00 - 13:30	Коваль Маргарита Олександрівна	8064373951
020-20052025	20.05.2025	08:00 - 08:30	Тимошенко Денис Григорович	5247638091
022-20052025	20.05.2025	09:00 - 09:30	Коваль Маргарита Олександрівна	8064373951
023-20052025	20.05.2025	10:00 - 10:30	Коваль Маргарита Олександрівна	8064373951
024-20052025	20.05.2025	11:00 - 11:30	Тимошенко Денис Григорович	5247638091
025-20052025	20.05.2025	12:00 - 12:30	Гаврилюк Ірина Євгенівна	8745620391
026-20052025	20.05.2025	13:00 - 13:30	Шевчук Лідія Михайлівна	9604802731

Рис. 43 Сформований звіт за обраний період

Результат тестування:

Система успішно пройшла всі кроки зазначеного сценарію без збоїв. Функції працювали відповідно до заданої логіки, введені дані коректно оброблялись, а результати формувались згідно з очікуванням на всіх етапах взаємодії.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректного функціонування системи обліку пацієнтів в районній поліклініці необхідно дотримуватися певних вимог до апаратного та програмного забезпечення, що забезпечить стабільність, швидкість роботи та

безпеку даних. Правильна організація технічної бази дозволяє системі ефективно обробляти запити користувачів, підтримувати одночасну роботу великої кількості працівників поліклініки та забезпечувати зручний інтерфейс для взаємодії з медичною інформацією [24].

Відповідно до діаграми розгортання системи, яка представлена на рис. 44, програмні компоненти на окремих апаратних вузлах, що відображає фізичну структуру реалізації.

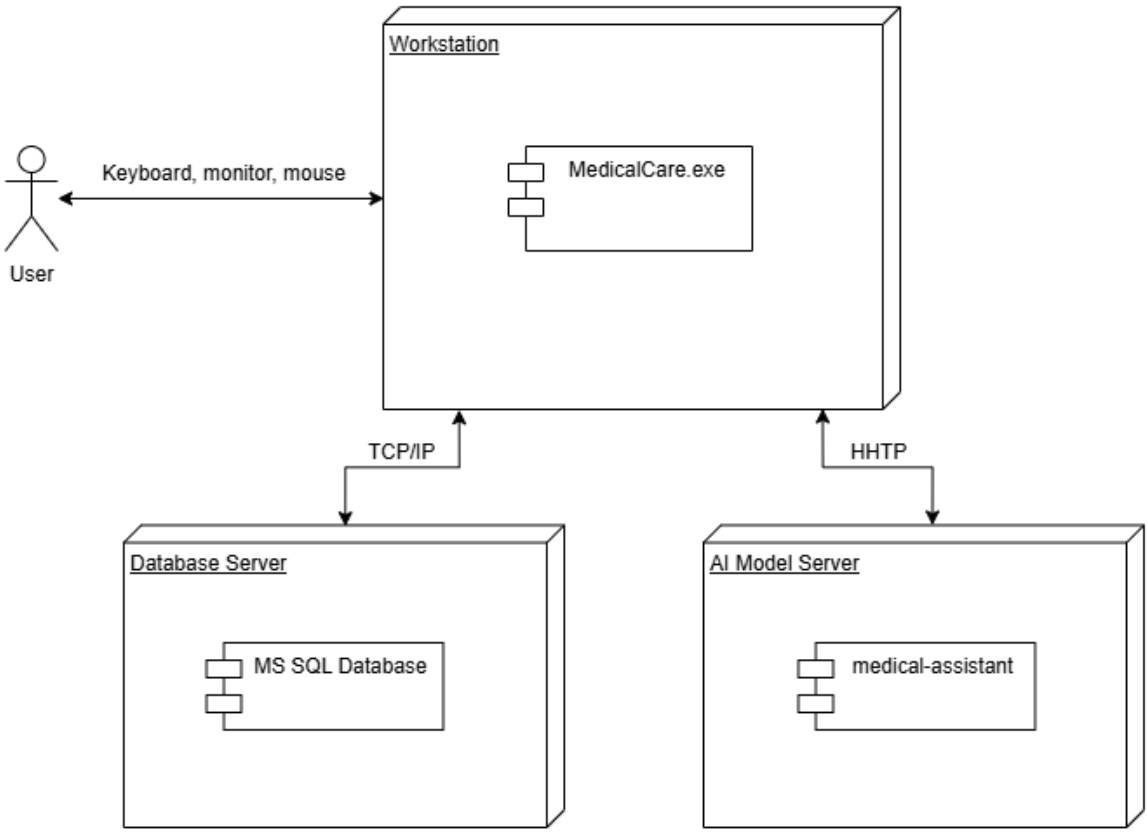


Рис. 44 Діаграма розгортання

Система реалізована за класичною схемою клієнт-сервер, де клієнтська частина представлена настільним застосунком MedicalCare.exe, що встановлюється на робочих станціях користувачів. Серверна частина складається з двох логічно розділених вузлів: Database Server, який містить компонент MS SQL Database для централізованого зберігання медичних даних, та AI Model Server, який містить компонент medical-assistant (AI-модель), розгорнутий на платформі Ollama для забезпечення функціональності інтелектуальної підтримки користувача.

Клієнтський застосунок взаємодіє з кожним із серверних компонентів через локальну мережу: з MS SQL Database за допомогою протоколу TCP/IP, а з medical-assistant – через HTTP-запити. Така організація забезпечує гнучке та надійне з'єднання з обома сервісами, які, хоча й можуть бути розгорнуті на одному фізичному сервері, логічно представлені як окремі вузли з незалежними інтерфейсами доступу.

Клієнтська частина забезпечує зручний графічний інтерфейс користувача, формує і надсилає запити до обох серверних вузлів, а також обробляє отримані результати. Вона відповідає за взаємодію з користувачем через засоби введення/виведення та реалізацію логіки користувацького інтерфейсу. Серверні вузли відповідно виконують свої ролі: Database Server виконує роль сховища даних, тоді як AI Model Server приймає відповідні запити, обробляє їх і повертає результати у зручному для користувача форматі.

Завдяки такій розподіленій архітектурі система є масштабованою, надійною та здатною забезпечувати швидку обробку даних.

Нижче наведені основні вимоги до апаратного та програмного забезпечення системи обліку пацієнтів в районній поліклініці.

Апаратне забезпечення

Клієнтська машина:

- процесор не нижче Intel Core i3 або аналогічний;
- оперативна пам'ять – 4 ГБ і більше;
- вільне місце на диску – не менше 500 МБ;
- мережевий адаптер для підключення до локальної мережі.

Серверна частина:

- процесор не нижче Intel Xeon або аналогічний;
- оперативна пам'ять – мінімум 16 ГБ;
- вільне місце на диску – мінімум 200 ГБ;
- надійне мережеве підключення з мінімальною затримкою.

Програмне забезпечення

Клієнтська машина:

- операційна система Windows 10 або новіша;
- клієнтський застосунок MedicalCare.exe;
- наявність Microsoft .NET Framework (.NET Framework 4.7.2 або новіший);
- налаштований доступ до локальної мережі з адресою сервера.

Серверна частина:

- операційна система Windows Server;
- MS SQL Server (версія 2019 і вище);
- платформа Ollama для розміщення та виконання AI-моделі;
- налаштовані служби резервного копіювання і безпеки.

Відповідне дотримання цих вимог забезпечить стабільну та ефективну роботу системи, зручність для користувачів і надійний захист даних.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення розроблений для швидкого та зручного розгортання як клієнтської, так і серверної частини системи. Пакет містить всі необхідні компоненти для повноцінної роботи.

До складу інсталяційного пакету входять наступні вказані компоненти.

- **Інсталятор клієнтського застосунку** – забезпечує автоматичне встановлення настільної програми MedicalCare разом із усіма необхідними бібліотеками, залежністю та компонентами середовища виконання .NET.
- **Конфігураційний файл** – містить параметри підключення до серверної частини системи.
- **SQL-скрипти для ініціалізації бази даних** – набір SQL-запитів для створення всіх необхідних об’єктів у MS SQL Server перед початком роботи системи.

- **AI-модель** – компонент, що встановлюється на серверну частину системи та забезпечує функціональність штучного інтелекту.
- **Документація** – включає інструкції з розгортання серверної частини та встановлення клієнтської частини.

Особливості інсталяції:

- інсталятор автоматично встановлює всі необхідні компоненти і бібліотеки, що забезпечує простоту та надійність розгортання клієнтської частини;
- серверна частина, яка складається з MS SQL Server та AI-моделі “medical-assistant”, має бути встановлена та налаштована перед початком роботи клієнтської програми;
- модель “medical-assistant” запускається через платформу Ollama, що забезпечуватиме обробку запитів від клієнтських застосунків;
- для адміністрування бази даних рекомендується використовувати SQL Server Management Studio (SSMS).

Таким чином, інсталяційний пакет забезпечує комплексне розгортання та налаштування як клієнтської, так і серверної частин системи, що гарантує їхню коректну взаємодію та готовність до експлуатації в робочому середовищі.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення системи обліку пацієнтів в районній поліклініці» було здійснено повний цикл розробки інформаційної системи, що охоплює етапи аналізу предметної області, проєктування бази даних, розробки прикладного програмного забезпечення та підготовки рекомендацій щодо впровадження і експлуатації.

Проведений аналіз діяльності районної поліклініки дозволив визначити ключові напрями автоматизації, які включають облік пацієнтів, запис на прийом, ведення медичних карток, збереження даних, а також інші супутні процеси медичного обслуговування. На основі виявлених потреб було сформульовано технічне завдання та розроблено концепцію програмного забезпечення.

Було побудовано логічну модель даних у вигляді ER-діаграми, спроєктовано структуру бази даних у середовищі MS SQL Server, а також реалізовано SQL-скрипти для автоматичної ініціалізації всіх необхідних об’єктів у базі. Це дозволило створити стійку та масштабовану основу для зберігання всієї необхідної медичної інформації.

Клієнтська частина програмного забезпечення реалізована мовою C# у середовищі Visual Studio з використанням технології WPF. Такий вибір забезпечив розробку сучасного, зручного та зрозумілого інтерфейсу для користувачів. Завдяки компоненту Microsoft Report Viewer у систему інтегровано функції генерації та перегляду звітів, що полегшує підготовку документації та аналітики.

Окремою інноваційною складовою розробленої системи є інтеграція елементів штучного інтелекту. На основі моделі LLaMA 3.1, локально розгорнутої через платформу Ollama, було реалізовано функціонал підтримки прийняття рішень лікарем під час діагностування. Система дозволяє вводити опис симптомів пацієнта та отримувати можливі варіанти діагнозів, що може значно прискорити та підвищити точність постановки попереднього діагнозу.

Такий підхід відкриває перспективи для подальшого розвитку системи у напряму інтелектуальної медичної підтримки.

Розроблене програмне забезпечення супроводжується інсталяційним пакетом, що забезпечує його швидке та зручне впровадження в умовах медичного закладу. Завдяки продуманій структурі та зрозумілим інструкціям, запуск системи не потребує значних технічних зусиль, що дозволяє оперативно розпочати її експлуатацію.

У результаті виконаної роботи всі поставлені цілі та завдання були повністю досягнуті. Розроблена система успішно автоматизує ключові процеси в районній поліклініці, поєднуючи зручність використання, стабільність роботи, ефективність обробки даних та можливості інтелектуальної підтримки прийняття рішень. Реалізоване рішення є практичним, функціональним та готовим до застосування в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Організація та зміст роботи амбулаторно-поліклінічних закладів різних рівнів, облік і аналіз їх діяльності – [Електронний ресурс] – режим доступу: https://medstudia.com/medviva/organizatsiya-ta-zmist-roboti-ambulatorno-poliklinichnih-zakladiv-riznih-rivniv-oblik-i-analiz-yih-diyalnosti-organizatsiya-dispansernogo-obslugovuvannya-naselennya#google_vignette (дата звернення: 07.04.2025)
2. The Patient Registration and Its Stages – [Електронний ресурс] – режим доступу: https://studycorgi.com/the-patient-registration-process-and-its-stages/ (дата звернення: 08.04.2025)
3. Томашевський В. М. Моделювання систем. – К.: Видавнича група BHV, 2005. – 352 с. – С. 9.
4. Що таке діаграма варіантів використання UML – [Електронний ресурс] – режим доступу: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram/ (дата звернення: 12.04.2025)
5. Як будувати UML-діаграми – [Електронний ресурс] – режим доступу: https://dou.ua/forums/topic/40575/ (дата звернення: 13.04.2025)
6. Helsi – [Електронний ресурс] – режим доступу: https://helsi.me/ (дата звернення: 15.04.2025)
7. МедІнфоСервіс – [Електронний ресурс] – режим доступу: https://www.infomed.ck.ua/ (дата звернення: 15.04.2025)
8. BAS Медицина (Лікарня та Поліклініка) – [Електронний ресурс] – режим доступу: https://a4.com.ua/bas-meditsina-ua/ (дата звернення: 15.04.2025)
9. Haux, R. Health information systems – past, present, future. International Journal of Medical Informatics, 2006. – С. 271-273
10. What is a Logical Data Model? – [Електронний ресурс] – режим доступу: https://www.tibco.com/glossary/what-is-a-logical-data-model (дата звернення: 19.04.2025)

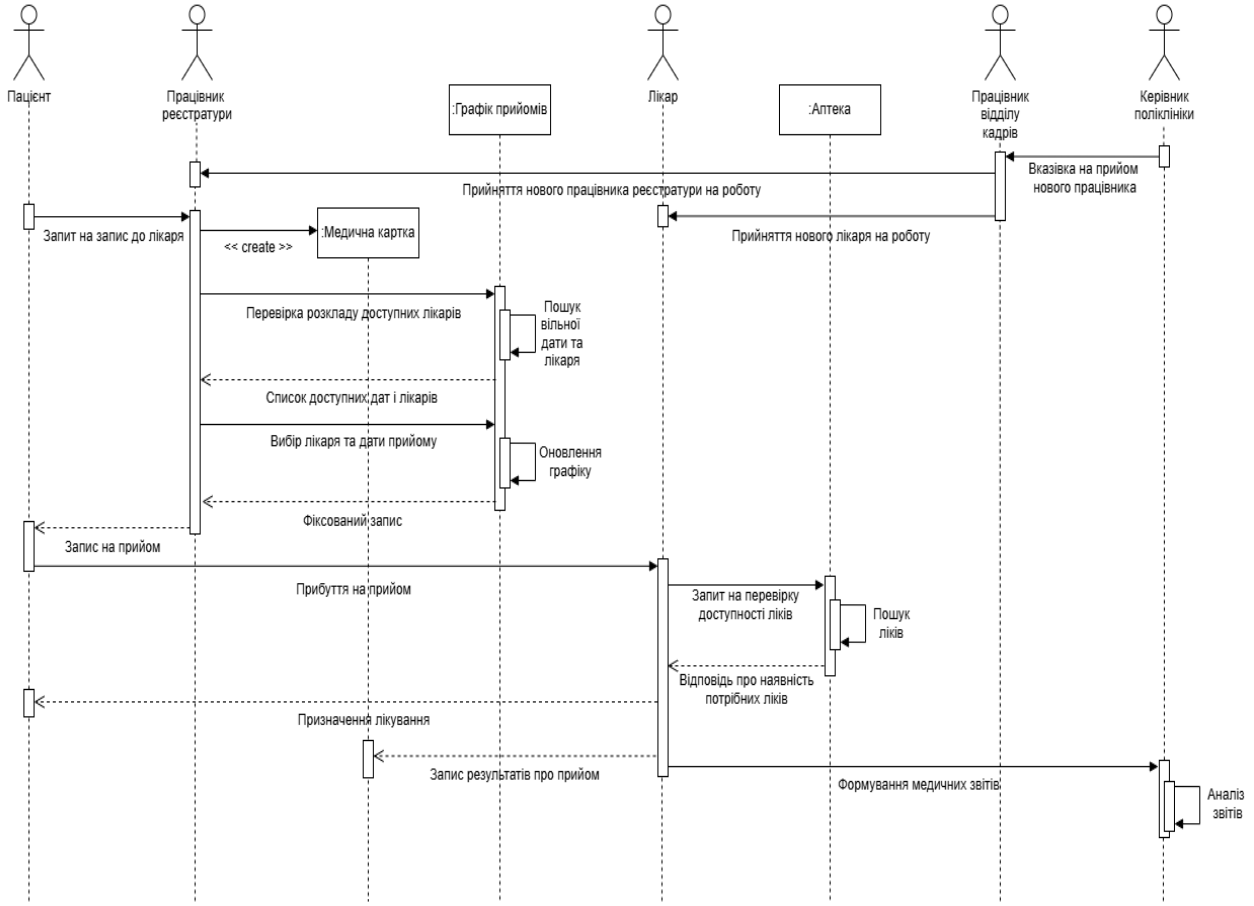
11. Лекція: Нормальні форми бази даних – [Електронний ресурс] – режим доступу: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture02 (дата звернення: 19.04.2025)
12. Нормалізація СУБД – [Електронний ресурс] – режим доступу: https://www.guru99.com/uk/database-normalization.html (дата звернення: 19.04.2025)
13. Файл-Серверні – [Електронний ресурс] – режим доступу: http://ni.biz.ua/14/14_2/14_28675_fayl-servernie.html#google_vignette (дата звернення: 20.04.2025)
14. 8 Top Database Management Systems (DBMS) – [Електронний ресурс] – режим доступу: https://www.pipedrive.com/en/blog/database-management-systems (дата звернення: 20.04.2025)
15. What is SQL Server – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/uk-ua/sql/sql-server/what-is-sql-server?view=sql-server-ver16 (дата звернення: 20.04.2025)
16. Основні об’єкти структури бази даних sql-серверу – [Електронний ресурс] – режим доступу: https://studfile.net/preview/14501229/page:5/ (дата звернення: 21.04.2025)
17. Фізичне проектування структури бази даних – [Електронний ресурс] – режим доступу: https://rdb.dp.ua/uk/chapter_04 (дата звернення: 21.04.2025)
18. Методологія класифікування предметної області – [Електронний ресурс] – режим доступу: https://stud.com.ua/86721/informatika/metodologiya_klassifitsirovaniya_predmetnoyi_oblasti (дата звернення: 25.04.2025)
19. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd Edition. Prentice Hall, 2004. – С. 197-225

20. Structural Diagrams – UML – GeeksforGeeks – [Електронний ресурс] – режимдоступу:https://www.geeksforgeeks.org/structural-diagrams-unified-modeling-languageuml/ (дата звернення: 27.04.2025)
21. C# development with Visual Studio – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/uk-ua/visualstudio/get-started/csharp/?view=vs-2022 (дата звернення: 09.05.2025)
22. What is Ollama? Introduction to the AI model management tool – [Електронний ресурс] – режим доступу: https://www.hostinger.com/tutorials/what-is-ollama (дата звернення: 10.05.2025)
23. Тестування ПЗ (види тестування) – [Електронний ресурс] – режим доступу: https://drukarnia.com.ua/articles/testuvannya-pz-vidi-testuvannya-JInS1 (дата звернення: 15.05.2025)
24. Hardware and software requirements: Overview, definition, and example – [Електронний ресурс] – режим доступу: https://www.cobrief.app/resources/legal-glossary/hardware-and-software-requirements-overview-definition-and-example/ (дата звернення: 17.05.2025)

ДОДАТКИ

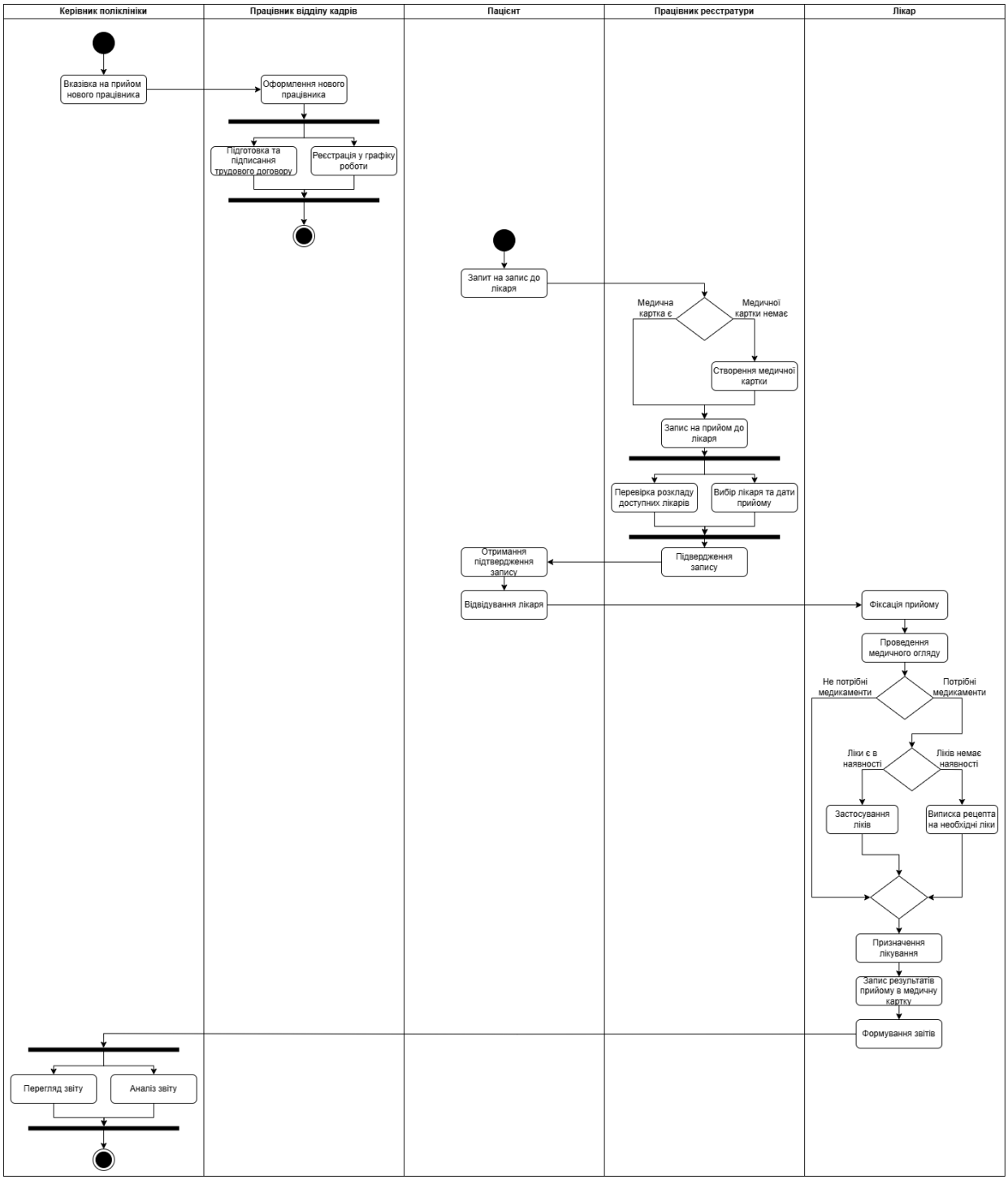
ДОДАТОК А

Діаграма послідовності



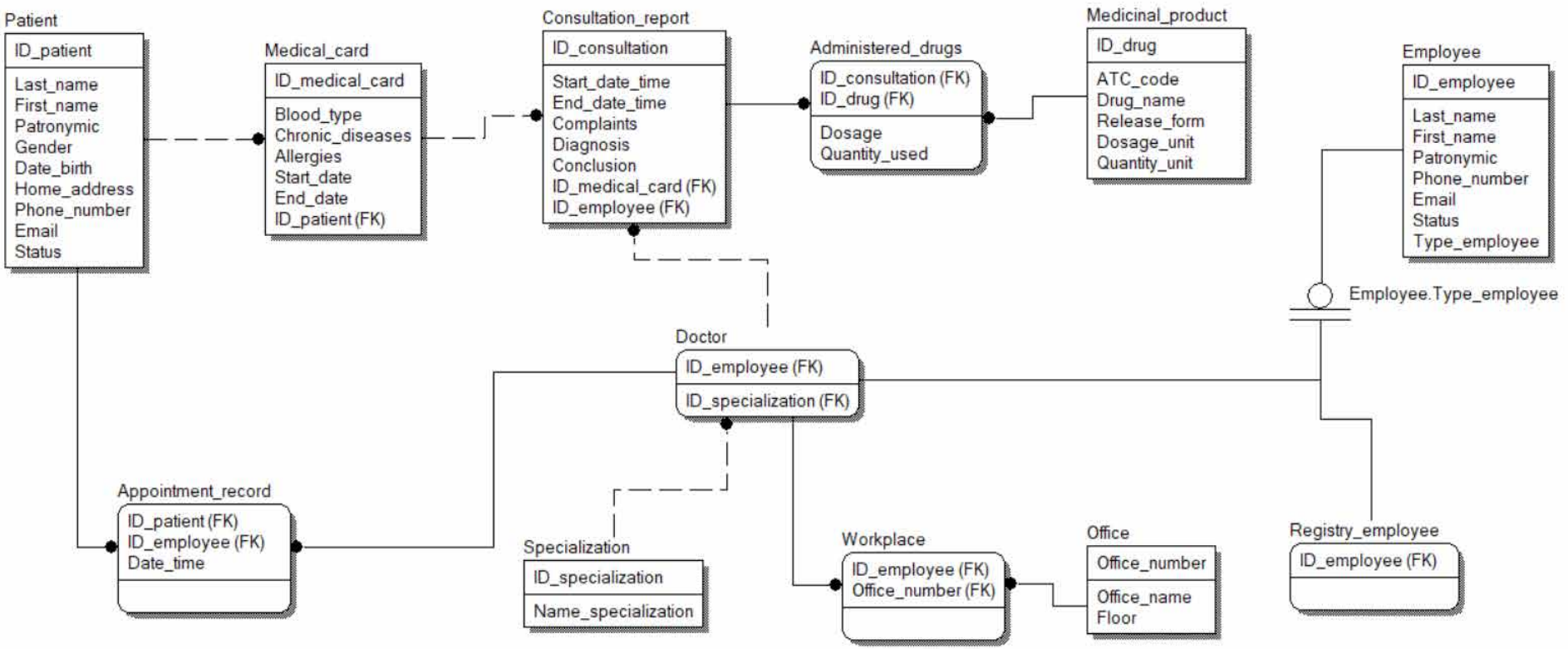
ДОДАТОК Б

Діаграма активності



ДОДАТОК В

Логічна модель бази даних



ДОДАТОК Д

Створення таблиць

```
USE DistrictPolyclinic
GO

IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Administered_drugs' AND type_desc =
'USER_TABLE')
    DROP TABLE Administered_drugs
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Consultation_report' AND type_desc =
'USER_TABLE')
    DROP TABLE Consultation_report
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Appointment_record' AND type_desc =
'USER_TABLE')
    DROP TABLE Appointment_record
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Workplace' AND type_desc =
'USER_TABLE')
    DROP TABLE Workplace
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Doctor' AND type_desc = 'USER_TABLE')
    DROP TABLE Doctor
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Registry_employee' AND type_desc =
'USER_TABLE')
    DROP TABLE Registry_employee
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Medical_card' AND type_desc =
'USER_TABLE')
    DROP TABLE Medical_card
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Specialization' AND type_desc =
'USER_TABLE')
    DROP TABLE Specialization
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Office' AND type_desc = 'USER_TABLE')
    DROP TABLE Office
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Medicinal_product' AND type_desc =
'USER_TABLE')
    DROP TABLE Medicinal_product
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Employee' AND type_desc =
'USER_TABLE')
    DROP TABLE Employee
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Patient' AND type_desc = 'USER_TABLE')
    DROP TABLE Patient
GO

-- Створення таблиць
CREATE TABLE Patient (
    ID_patient CHAR(10) PRIMARY KEY NOT NULL,
    Last_name NVARCHAR(50) NOT NULL,
    First_name NVARCHAR(50) NOT NULL,
    Patronymic NVARCHAR(50),
    Gender NVARCHAR(50),
    Date_birth DATE NOT NULL,
    Home_address NVARCHAR(255),
    Phone_number VARCHAR(20),
    Email VARCHAR(100),
    Status_patient NVARCHAR(50) NOT NULL
)

CREATE TABLE Employee (
    ID_employee CHAR(10) PRIMARY KEY NOT NULL,
    Last_name NVARCHAR(50) NOT NULL,
    First_name NVARCHAR(50) NOT NULL,
    Patronymic NVARCHAR(50),
```

```
    Phone_number VARCHAR(20),
    Email VARCHAR(100),
    Status_employee NVARCHAR(50),
    Type_employee NVARCHAR(50) CHECK (Type_employee IN ('Лікар', 'Працівник реєстратури'))
)

CREATE TABLE Medicinal_product (
       ID_drug CHAR(3) PRIMARY KEY NOT NULL,
    ATC_code NVARCHAR(7) NOT NULL,
    Drug_name NVARCHAR(100) NOT NULL,
    Release_form NVARCHAR(50) NOT NULL,
       Dosage_unit NVARCHAR(20) NOT NULL,
    Quantity_unit NVARCHAR(20) NOT NULL
)

CREATE TABLE Office (
    Office_number CHAR(3) PRIMARY KEY NOT NULL,
    Office_name NVARCHAR(100) NOT NULL,
    Floor_office INT NOT NULL
)

CREATE TABLE Specialization (
    ID_specialization CHAR(2) PRIMARY KEY NOT NULL,
    Name_specialization NVARCHAR(100) NOT NULL
)

CREATE TABLE Medical_card (
    ID_medical_card CHAR(10) PRIMARY KEY NOT NULL,
    Blood_type NVARCHAR(15),
    Chronic_diseases NVARCHAR(MAX),
    Allergies NVARCHAR(MAX),
    Start_date DATE NOT NULL,
    End_date DATE,
    ID_patient CHAR(10) NOT NULL,
    FOREIGN KEY (ID_patient) REFERENCES Patient(ID_patient)
)

CREATE TABLE Registry_employee (
    ID_employee CHAR(10) PRIMARY KEY NOT NULL,
    FOREIGN KEY (ID_employee) REFERENCES Employee(ID_employee)
)

CREATE TABLE Doctor (
    ID_employee CHAR(10) PRIMARY KEY NOT NULL,
    ID_specialization CHAR(2) NOT NULL,
    FOREIGN KEY (ID_employee) REFERENCES Employee(ID_employee),
    FOREIGN KEY (ID_specialization) REFERENCES Specialization(ID_specialization)
)

CREATE TABLE Workplace (
    ID_employee CHAR(10) NOT NULL,
    Office_number CHAR(3) NOT NULL,
    PRIMARY KEY (ID_employee, Office_number),
    FOREIGN KEY (ID_employee) REFERENCES Doctor(ID_employee),
    FOREIGN KEY (Office_number) REFERENCES Office(Office_number)
)

CREATE TABLE Appointment_record (
    ID_patient CHAR(10) NOT NULL,
    ID_employee CHAR(10) NOT NULL,
    Date_time DATETIME NOT NULL,
    PRIMARY KEY (ID_patient, ID_employee, Date_time),
    FOREIGN KEY (ID_patient) REFERENCES Patient(ID_patient),
    FOREIGN KEY (ID_employee) REFERENCES Doctor(ID_employee)
```

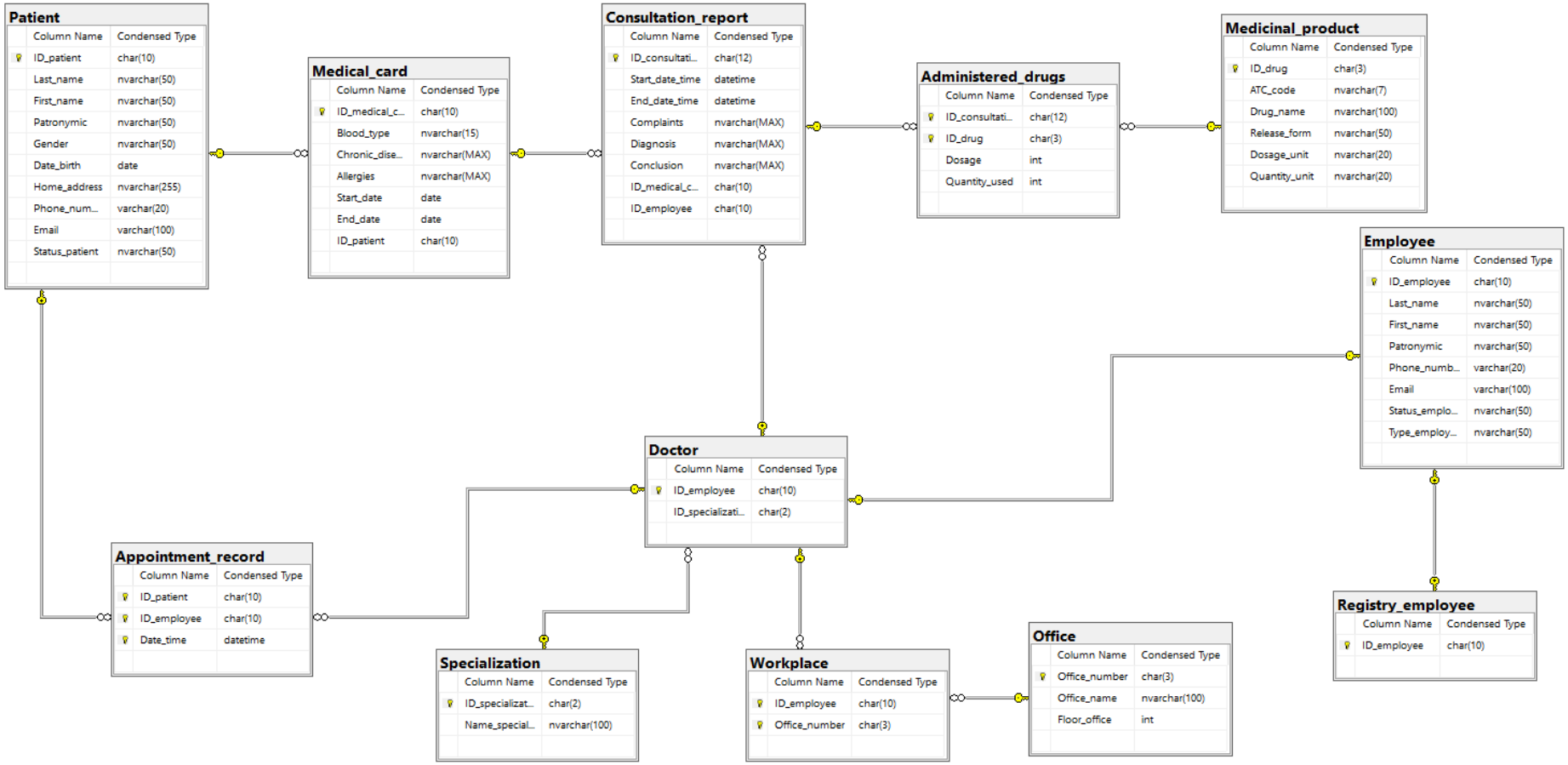
```
)

CREATE TABLE Consultation_report (
    ID_consultation CHAR(12) PRIMARY KEY NOT NULL,
    Start_date_time DATETIME NOT NULL,
    End_date_time DATETIME NOT NULL,
    Complaints NVARCHAR(MAX),
    Diagnosis NVARCHAR(MAX),
    Conclusion NVARCHAR(MAX),
    ID_medical_card CHAR(10) NOT NULL,
    ID_employee CHAR(10) NOT NULL,
    FOREIGN KEY (ID_medical_card) REFERENCES Medical_card(ID_medical_card),
    FOREIGN KEY (ID_employee) REFERENCES Doctor(ID_employee)
)

CREATE TABLE Administered_drugs (
    ID_consultation CHAR(12) NOT NULL,
    ID_drug CHAR(3) NOT NULL,
    Dosage INT NOT NULL,
    Quantity_used INT NOT NULL,
    PRIMARY KEY (ID_consultation, ID_drug),
    FOREIGN KEY (ID_consultation) REFERENCES Consultation_report(ID_consultation),
    FOREIGN KEY (ID_drug) REFERENCES Medicinal_product(ID_drug)
)
```

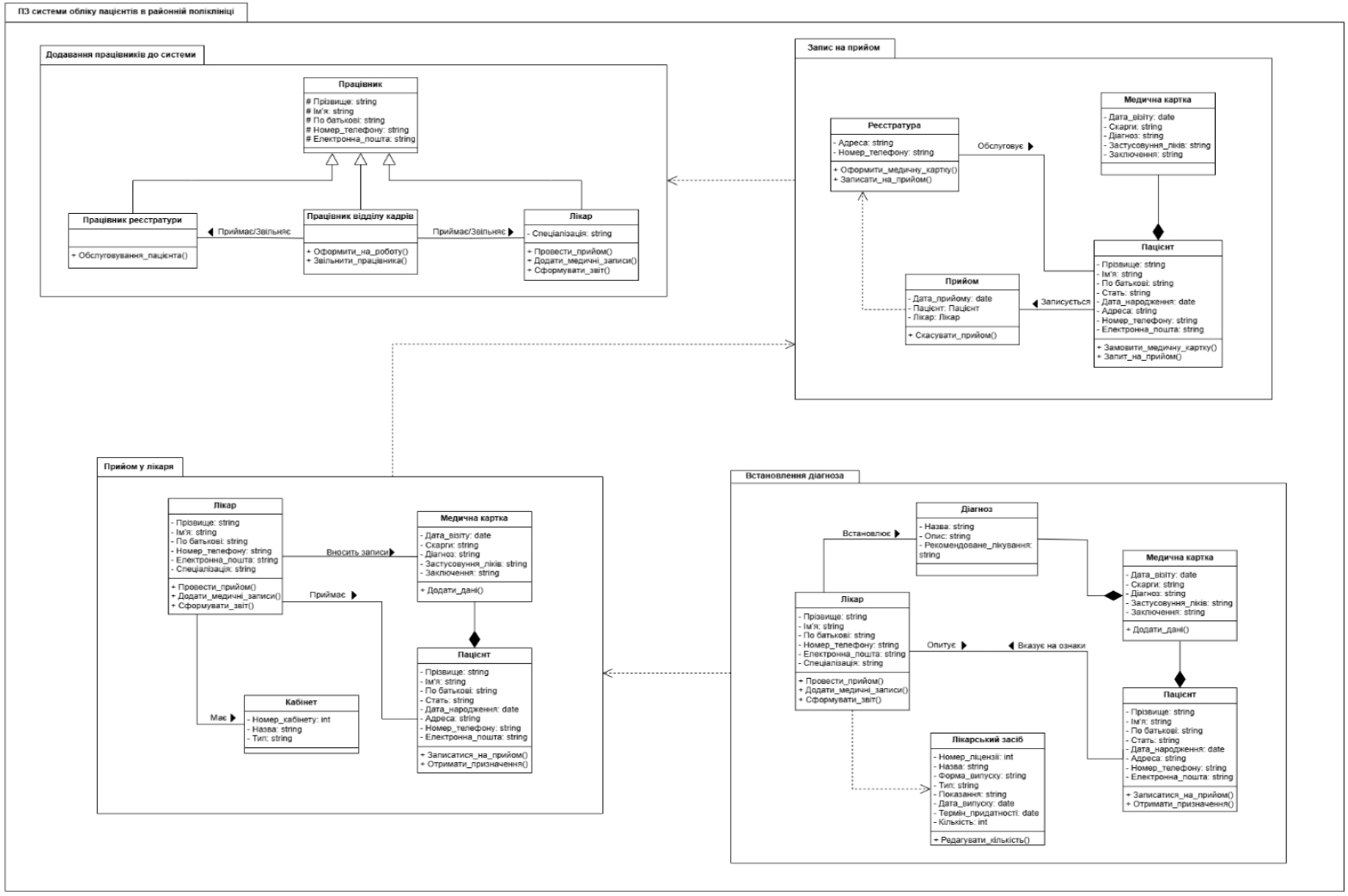
ДОДАТОК Е

Фізична модель бази даних



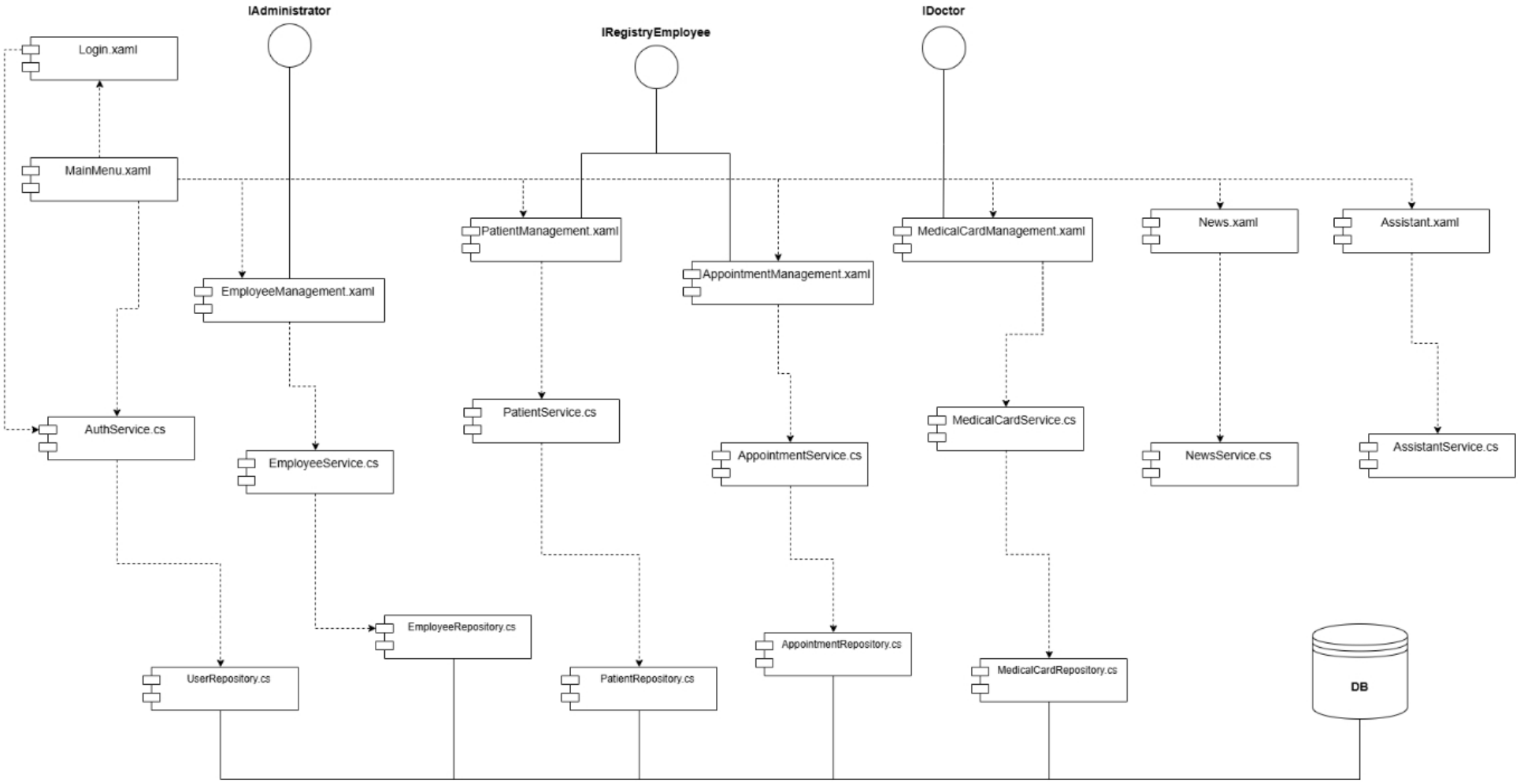
ДОДАТОК Ж

Діаграма пакетів



ДОДАТОК К

Діаграма компонентів



ДОДАТОК Л

Диплом І ступеня