НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення автоматизованого робочого місця працівників реєстратури медичного закладу

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Олійник С.А.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Олійнику Сергію Андрійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення автоматизованого робочого місця працівників реєстратури медичного закладу

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.06.02
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань що розглядаються:
1. Системний аналіз предметної області інформаційної системи
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Олійник С.А.
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП

У сучасних умовах розвитку цифрової медицини автоматизація адміністративних процесів у медичних закладах набула особливої актуальності. Одним із критичних підрозділів, де ефективність обслуговування пацієнтів залежить від швидкості обробки даних та організованості роботи персоналу, є реєстратура. Саме тут відбувається первинний контакт пацієнта із системою медичного обслуговування, здійснюється облік персональних даних, планування візитів, а також формування первинної медичної документації.

На практиці ручне ведення таких процесів часто супроводжується дублюванням даних, помилками у графіках, втратою записів, а також зниженням загальної якості обслуговування. Це зумовлює необхідність розробки спеціалізованого програмного забезпечення, яке дозволить автоматизувати діяльність працівників реєстратури, зменшити рутинне навантаження та підвищити надійність обробки медичної інформації.

Особливої важливості набуває інтеграція таких систем у загальнодержавну цифрову інфраструктуру охорони здоров’я, зокрема в частині обміну інформацією з платформами eHealth, підтримки єдиних форматів даних і забезпечення безпеки персональної інформації. Водночас система має бути придатною для розгортання в умовах обмеженого технічного ресурсу – у невеликих клініках чи амбулаторіях.

Об’єкт дослідження – автоматизована системах обліку та адміністрування в закладах охорони здоров’я.

Предмет дослідження – методи та програмні засоби автоматизації функцій реєстратури медичного закладу, зокрема – інтерфейс взаємодії, логіка обробки пацієнтських даних, управління розкладами лікарів та формування звітності.

Метою роботи є розробка функціонального, надійного та зручного у використанні програмного забезпечення, яке забезпечить автоматизацію основних процесів роботи працівників реєстратури медичного закладу. Програмний продукт має забезпечувати зручний інтерфейс для введення,

зберігання, пошуку та редагування даних пацієнтів, планування записів на прийом до лікаря, контроль за графіками роботи медичного персоналу та формування звітів.

У процесі реалізації застосовано об’єктно-орієнтоване моделювання (UML), середовище Visual Studio, мову програмування C#, технології WinForms, а також систему управління базами даних PostgreSQL з доступом через драйвер Npgsql. Для візуалізації інформації використано iTextSharp (PDF) та бібліотеки компонування інтерфейсу.

Окремі компоненти розробленої системи були продемонстровані на студентських науково-практичних конференціях і схвалені керівником як такі, що відповідають поставленим завданням. Робота виконана на кафедрі комп’ютерних наук факультету інформаційних технологій НУБіП України.

Проєктування додатку проєктування програмного забезпечення було використано модульний підхід до побудови архітектури додатку. Система розроблялася з урахуванням принципів розділення відповідальностей (SRP), що дозволяє підтримувати гнучкість і масштабованість. Ключовими модулями є: модуль авторизації, модуль керування пацієнтами, модуль запису на прийом, модуль керування лікарями, модуль адміністрування та звітності.

Усі форми взаємодіють із єдиним об'єктом підключення до бази даних (DBConnection), що інкапсулює логіку з'єднання та обробки запитів. Для візуалізації інтерфейсу обрано Windows Forms через швидкість розробки та зручність реалізації UI для настільних систем. Інтерфейс користувача орієнтований на простоту, доступність та логічність розташування елементів, що забезпечує швидке освоєння системи персоналом без спеціальної технічної підготовки

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область охоплює інформаційні та організаційні процеси, пов’язані з функціонуванням реєстратури медичного закладу, яка є одним із центральних елементів в обслуговуванні пацієнтів. Її діяльність полягає в обліку звернень, формуванні медичних записів, координації прийомів до лікарів і забезпеченні збереження актуальної медичної інформації.

У реаліях підвищеного навантаження, зростання кількості пацієнтів та переходу до електронного документообігу, зростає потреба у створенні інструментів, здатних забезпечити точність, швидкість і безпеку в роботі адміністративного персоналу.

Інформаційна система, що автоматизує діяльність реєстратури, повинна інтегрувати основні типи користувачів: пацієнтів, працівників реєстратури, лікарів та адміністратора. Кожна з цих категорій відіграє окрему функціональну роль. Пацієнти звертаються по медичну допомогу та фіксуються в базі даних; реєстратори виконують введення та оновлення інформації, забезпечують запис до лікарів; медичні працівники здійснюють прийоми відповідно до встановленого графіка; адміністратори налаштовують права доступу, ведуть контроль за системою та формують звітність.

У структурі предметної області передбачено логічну взаємодію між сутностями: кожен запис про прийом пацієнта є результатом вибору часу, лікаря та валідації розкладу, що формує основу для наступних звітів або дій з боку адміністратора. Таким чином, облікові операції є взаємопов’язаними, а сама система має підтримувати наскрізний облік усіх подій.

У табл. 1.1 наведено узагальнену модель типових функцій, що реалізуються в межах програмної системи реєстратури:

Таблиця 1.1

Функціональні можливості інформаційної системи реєстратури

№	Категорія функцій	Функціональний опис
1	Облік пацієнтів	Реєстрація, редагування, пошук, перегляд медичних даних
2	Запис на прийом	Бронювання часу, перевірка конфліктів, формування направлень
3	Керування розкладом	Створення та оновлення графіків роботи лікарів
4	Звітність	Формування друкованих звітів, експорт у PDF
5	Авторизація та доступ	Розмежування прав доступу за ролями: реєстратор, лікар, адміністратор
6	Контроль змін	Ведення журналу дій для аудиту та безпеки
7	Друк медичних документів	Генерація талонів та направлень для пацієнтів у друкованому або цифровому форматі

У технологічному аспекті реалізація системи базується на платформі .NET із використанням мови C# у середовищі Windows Forms [7]. У якості системи управління базами даних застосовано PostgreSQL із використанням бібліотеки Npgsql для з’єднання та виконання SQL-запитів [4]. Генерація звітів у форматі PDF реалізована через бібліотеку iTextSharp [3], а можливості інтерфейсної адаптації – через додаткові бібліотеки UI-компонентів (наприклад, Bunifu). Розробка ведеться у Visual Studio з використанням системи контролю версій Git [11].

Інформаційна система орієнтована на роботу з конфіденційною інформацією. Тому в її архітектурі реалізовано механізми автентифікації, шифрування та контролю доступу. Ролі користувачів визначають рівень доступу

до даних, а всі дії у системі логуються для забезпечення прозорості та можливості подальшого аудиту.

Предметна область інформаційної системи реєстратури визначається комплексом взаємопов’язаних дій, що забезпечують організацію медичного обліку та обслуговування пацієнтів. Автоматизація цих процесів дозволяє досягти ефективнішої координації, зменшити навантаження на персонал і підвищити точність обробки інформації при дотриманні високих стандартів безпеки та конфіденційності.

1.2 Аналіз вимог до програмної системи

Розробка програмного забезпечення автоматизованого робочого місця працівника реєстратури медичного закладу передбачає ретельне формулювання вимог до функціональності, продуктивності, безпеки та зручності використання системи. Аналіз вимог є базисом для подальшого моделювання, проєктування архітектури та впровадження програмного продукту.

Функціональні вимоги до інформаційної системи автоматизованого робочого місця працівника реєстратури медичного закладу відображають набір операцій, необхідних для ефективного виконання облікових і адміністративних процесів. Їх реалізація забезпечує зручну роботу з персональними та медичними даними пацієнтів, організацію записів на прийом, управління персоналом і формування звітності.

На рівні доступу до системи передбачено обов’язкову авторизацію користувачів із перевіркою облікових даних, що супроводжується визначенням ролі — реєстратора або адміністратора. Система дозволяє створювати нові профілі пацієнтів із введенням особистих даних, а також забезпечує можливість їх редагування та перегляду. При формуванні запису на прийом користувач обирає лікаря, дату та час, після чого система автоматично перевіряє доступність у розкладі, попереджаючи про можливі накладення.

Оформлення талона на прийом передбачає генерацію документа з даними про пацієнта, лікаря та прийом, який може бути роздрукований або збережений

у цифровому форматі. Адміністратор, у свою чергу, має розширені повноваження — від створення нових облікових записів і редагування графіків роботи медичного персоналу до ведення системного журналу дій користувачів. Такий журнал дозволяє здійснювати аудит змін, підвищуючи прозорість та безпечність роботи з даними.

Формування звітів є ще одним важливим аспектом функціональності. Адміністратор може створювати статистичні та операційні звіти на основі даних про пацієнтів, прийоми чи навантаження лікарів. При цьому система забезпечує перевірку коректності введених даних, включаючи унікальність логінів і запобігання дублюванню інформації у графіках.

У табл. 1.2 узагальнено основні функції програмної системи, користувачів, яким вони доступні, та їх функціональне призначення.

Таблиця 1.2

Специфікація функціональних вимог

№	Назва функції	Функціональне призначення	Основний користувач
1	Авторизація	Вхід до системи з перевіркою логіна й пароля, визначення ролі	Усі користувачі
2	Реєстрація пацієнта	Створення картки пацієнта з особистими та контактними даними	Реєстратор
3	Перегляд і редагування пацієнтів	Оновлення або перегляд збережених даних пацієнтів	Реєстратор
4	Запис на прийом	Створення нового запису на прийом із перевіркою розкладу лікаря	Реєстратор
5	Перевірка розкладу	Автоматичне виявлення конфліктів у записах	Система
6	Генерація та друк талонів	Формування документа про запис у форматі PDF або для друку	Реєстратор
7	Керування обліковими записами	Створення, видалення або оновлення даних користувачів	Адміністратор
8	Перегляд журналу дій	Відстеження дій користувачів у системі	Адміністратор
9	Створення звітів	Формування звітних документів за вибраними параметрами	Адміністратор
10	Перевірка унікальності логіна	Перевірка наявності логіна перед додаванням нового користувача	Адміністратор

Функціональні вимоги не лише визначають перелік допустимих дій користувача, а й формують архітектурну основу системи. Кожен елемент функціональності повинен взаємодіяти з іншими в межах узгодженої логіки, що забезпечує безперервність облікових операцій та узгодженість з базою даних.

У підсумку слід зазначити, що функціональні вимоги є визначальними для побудови логіки програмної системи. Їх правильне формулювання сприяє створенню ефективного та безпечного інструменту для автоматизації процесів реєстратури, адаптованого до умов реального медичного закладу. Саме ці вимоги забезпечують основу надійної й гнучкої роботи з медичною інформацією.

Нефункціональні вимоги визначають загальні характеристики програмної системи, які, хоча й не стосуються безпосередньо її прикладної функціональності, мають вирішальний вплив на якість, надійність та зручність її експлуатації. Вони охоплюють такі аспекти, як продуктивність, безпека, ергономіка інтерфейсу, стійкість до збоїв, масштабованість і відповідність сучасним технічним умовам.

У контексті автоматизованого робочого місця працівників реєстратури медичного закладу ці вимоги набувають особливої значущості через чутливість оброблюваних даних і необхідність безперервної роботи в умовах підвищеного навантаження. Програмне забезпечення повинно оперативно реагувати на дії користувача, забезпечуючи швидку обробку запитів і мінімізуючи час очікування. Для цього важливо застосовувати оптимізовані запити до бази даних, використовувати кешування та проєктувати інтерфейс без надлишкової складності.

Особливу увагу при формуванні нефункціональних вимог приділено надійності програмної системи, яка повинна забезпечувати стабільне функціонування в умовах тривалої експлуатації, коректно обробляти виняткові ситуації та фіксувати помилки для подальшого аналізу. Безпека та конфіденційність реалізуються через механізми автентифікації, хешування облікових даних і чітке розмежування прав доступу відповідно до ролей користувачів; при цьому критичні дані, зокрема персональна інформація

пацієнтів, мають бути захищені від несанкціонованого доступу. Інтерфейс системи повинен бути інтуїтивно зрозумілим і доступним для освоєння без спеціального навчання, що забезпечує зручність у роботі персоналу. Масштабованість передбачає можливість розширення функціональності шляхом додавання нових модулів без порушення цілісності системи, а портативність – підтримку роботи на більшості сучасних комп'ютерів у медичних установах, сумісних з операційними системами Windows.

Ще одним ключовим аспектом є резервне копіювання – система має підтримувати створення архівних копій бази даних для відновлення інформації у разі технічних збоїв. Логування дій користувачів забезпечує прозорість взаємодії із системою та слугує інструментом аудиту й аналізу ризиків.

У табл. 1.3 узагальнено основні нефункціональних вимог до системи.

Таблиця 1.3

Специфікація нефункціональних вимог до системи

№	Вимога	Призначення	Засоби реалізації
1	Продуктивність	Швидке реагування інтерфейсу на дії користувача	Оптимізація SQL-запитів, локальні кеші, спрощення UI
2	Надійність	Гарантія стабільної роботи без збоїв та втрати даних	Обробка винятків, журнал помилок, валідація введення
3	Безпека	Захист від несанкціонованого доступу та втрат інформації	Авторизація, шифрування, контроль прав доступу
4	Конфіденційність	Обмеження доступу до персональних даних пацієнтів	Авторизація за ролями, приховування полів у формах
5	Зручність (юзабіліті)	Полегшення використання системи без спеціального навчання	Інтуїтивна навігація, підказки, логічна структура інтерфейсу
6	Масштабованість	Можливість подальшого розширення функціональності	Модульна архітектура, незалежні сервіси
7	Портативність	Робота на більшості актуальних конфігурацій користувацьких пристроїв	Сумісність із Windows 10/11, використання .NET Framework / .NET Core
8	Резервне копіювання	Захист від втрати даних у разі збою	Експорт бази, створення бекапів через SQL-інструменти
9	Аудит дій	Контроль і фіксація активності користувачів	Логування подій із позначенням часу, користувача та типу операції

Загалом, нефункціональні вимоги формують основу для забезпечення довготривалої, безпечної та ефективної експлуатації програмного забезпечення.

У сфері медичних інформаційних систем особливо критичними є вимоги до захисту даних, стабільності в умовах інтенсивного навантаження та доступності інтерфейсу для персоналу, який не має спеціалізованої технічної підготовки. Врахування зазначених вимог дозволяє гарантувати відповідність системи практичним потребам медичного закладу, забезпечуючи її готовність до впровадження в реальне середовище експлуатації.

Таким чином, сукупність функціональних і нефункціональних вимог формує повноцінне уявлення про цілісність і призначення системи. Збалансоване поєднання зручності, надійності та безпеки є ключем до успішної експлуатації програмного забезпечення в умовах сучасного медичного закладу, де точність і швидкість обробки даних безпосередньо впливають на якість надання медичних послуг.

1.3 Моделювання предметної області

Моделювання предметної області є ключовим етапом розробки інформаційної системи, оскільки дозволяє формалізувати реальні процеси, що відбуваються в медичному закладі, й перетворити їх на структуру, придатну для реалізації у програмному середовищі. У контексті автоматизованого робочого місця працівника реєстратури предметною областю виступає діяльність, пов’язана з обліком пацієнтів, управлінням графіками лікарів, створенням записів на прийом, а також адмініструванням користувачів системи.

Формалізація цієї сфери включає визначення основних об’єктів (пацієнти, лікарі, реєстратори, адміністратори, прийоми), а також процесів взаємодії між ними. Моделювання дає змогу виявити логіку переходів між станами об’єктів, описати поведінку системи в типових ситуаціях і сформувати концептуальне уявлення про її функціональну структуру. Водночас важливо зберігати абстрактність моделі, фокусуючись лише на тих аспектах, що мають значення для реалізації програмного забезпечення [21].

Для створення формального опису використовуються засоби уніфікованої мови моделювання UML. UML надає набір стандартних діаграм, за допомогою

яких можна описати як структуру (через діаграми класів), так і поведінку системи (через діаграми прецедентів, активностей, станів, послідовностей). Кожен тип діаграм відповідає за окремий рівень деталізації й дозволяє моделювати як логічні залежності між сутностями, так і сценарії використання функціоналу.

Особливу роль відіграє діаграма прецедентів (use case diagram), яка фокусується на зовнішній взаємодії користувачів із системою. Вона демонструє, які дії доступні різним типам користувачів і як межі системи розділяють внутрішню логіку від зовнішнього інтерфейсу. У типовому випадку, що стосується системи реєстратури, акторами є реєстратор і адміністратор. Реєстратор виконує операції з пацієнтами, записами, друком талонів; адміністратор відповідає за управління персоналом, графіками та генерацію звітності.

На рис. 1.1 подано діаграму прецедентів для системи автоматизованого робочого місця працівників реєстратури медичного закладу. У ній відображено взаємозв’язки між акторами та діями, що реалізуються в системі.

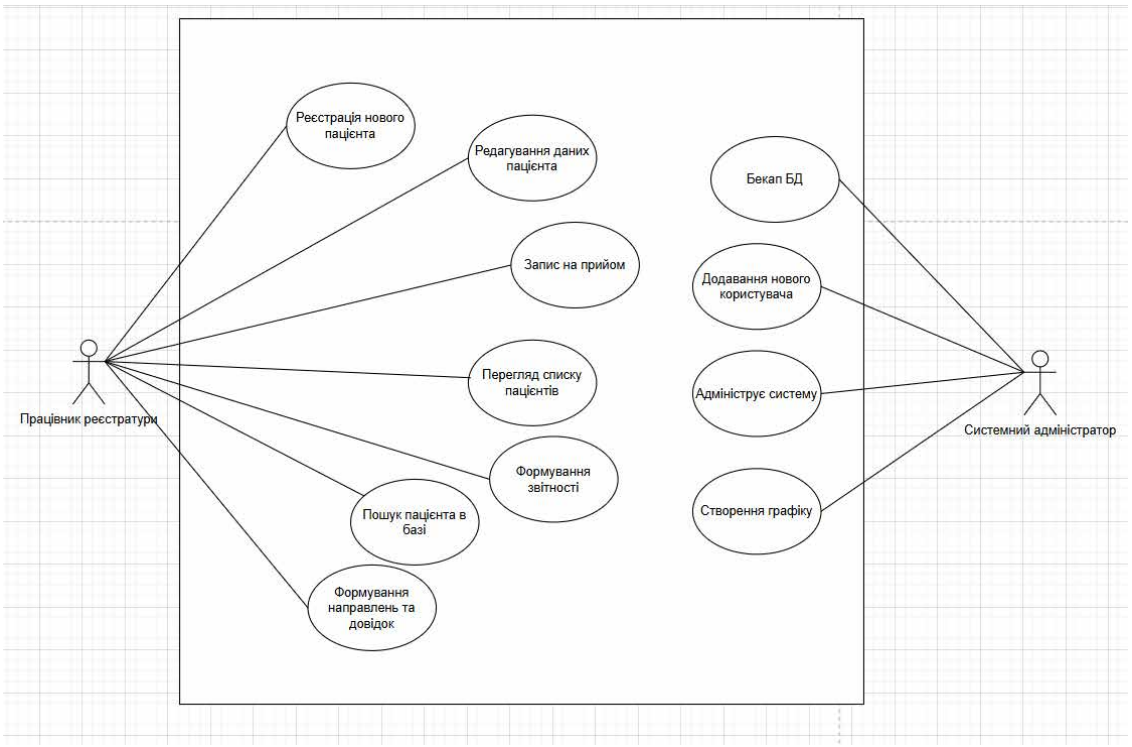


Рис. 1.1 Діаграма прецендентів для системи автоматизованого робочого місця працівників реєстратури медичного закладу

Прецеденти описують основні функції: авторизацію, додавання або редагування пацієнтів, запис на прийом, формування звітів тощо. Відношення типу <> або <> допомагають деталізувати залежності між окремими варіантами використання – наприклад, сценарій «Формування звіту» може включати підсценарій «Фільтрація записів».

Після аналізу діаграми доцільно деталізувати ролі користувачів та їхні функціональні можливості. У табл. 1.4 наведено основних акторів системи та відповідні дії, які вони можуть виконувати.

Таблиця 1.4

Актори та їх функціональні можливості

№	Актор	Функціональні можливості
1	Працівник реєстратури	Реєстрація, редагування, пошук і перегляд пацієнтів, запис на прийом, формування звітів, друк талонів і направлень
2	Системний адміністратор	Додавання користувачів, створення графіків лікарів, керування налаштуваннями, резервне копіювання БД

Для уточнення логіки виконання операцій і опису функціональних сценаріїв використано перелік прецедентів, що зведені в табл. 1.5.

Таблиця 1.5

Прецеденти та їх короткий опис

№	Назва прецеденту	Короткий опис
1	2	3
1	Реєстрація нового пацієнта	Додавання нового запису з ПІБ, датою народження, статтю, адресою
2	Редагування даних пацієнта	Оновлення помилкових або застарілих даних у картці пацієнта
3	Запис на прийом	Створення прийому з урахуванням розкладу лікаря
4	Перегляд списку пацієнтів	Виведення зареєстрованих пацієнтів з можливістю фільтрації
5	Формування звітності	Генерація аналітичних звітів за обраними параметрами
6	Пошук пацієнта в базі	Виявлення пацієнта за ПІБ, номером картки або іншими атрибутами

Таблиця 1.5 (продовження)

1	2	3
7	Формування направлень і довідок	Створення друкованих документів для пацієнтів
8	Додавання нового користувача	Створення облікового запису з параметрами доступу
9	Створення графіку	Побудова або редагування розкладу прийому
10	Адміністрування системи	Керування налаштуваннями, структурою та правами доступу
11	Резервне копіювання бази даних	Збереження актуальної копії бази з метою захисту від втрати інформації

Діаграма послідовності (Sequence Diagram) є важливим засобом динамічного моделювання систем у мові UML, який дає змогу представити хронологічний порядок виконання дій між об’єктами. Її застосування дозволяє наочно показати, як саме відбувається обмін повідомленнями між різними компонентами системи для реалізації певного сценарію використання.

Основою структури такої діаграми є вертикальні лінії життя (lifelines), що представляють учасників взаємодії (користувачів або внутрішні об’єкти), та горизонтальні стрілки, які відображають послідовність викликів методів, запитів або дій. Кожна подія на діаграмі впорядкована за часом зверху вниз: чим нижче розміщено стрілку, тим пізніше відбувається відповідна дія. Це дозволяє формалізувати логіку реалізації функціоналу, від інтерфейсу користувача до взаємодії з базою даних.

У межах програмного забезпечення для автоматизованого робочого місця реєстратора медичного закладу діаграма послідовності застосовується для відображення ключових сценаріїв, таких як створення нового пацієнта, запис на прийом, друк направлення або адміністрування користувачів і графіків лікарів. Таке візуальне представлення дозволяє зрозуміти порядок ініціалізації дій, черговість обробки запитів та участь окремих компонентів (інтерфейс, бізнес-логіка, база даних) у реалізації кожного процесу.

На рис. 1.2 зображено приклад діаграми послідовності, що демонструє типову взаємодію між користувачами системи (реєстратором та

адміністратором) і програмними модулями під час виконання операцій із введення пацієнтів, перегляду інформації та адміністрування.

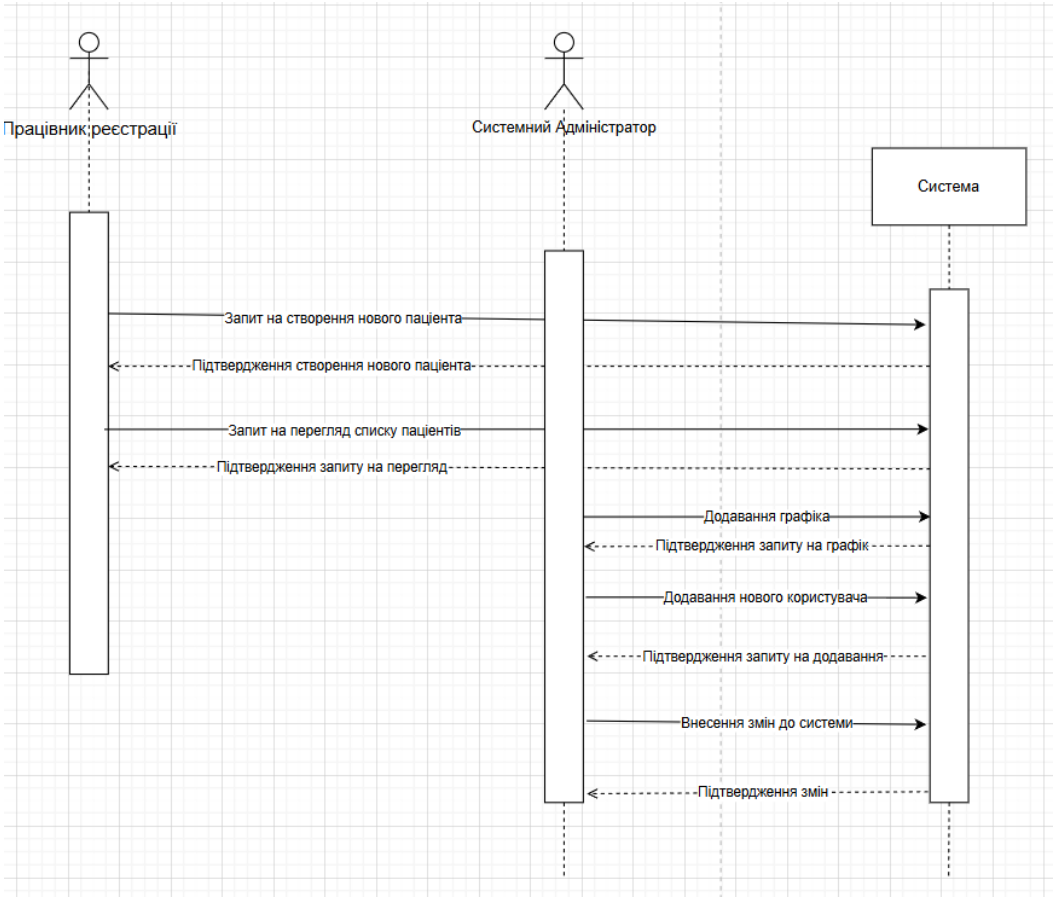


Рис. 1.2 Діаграма послідовності для системи автоматизованого робочого місця працівників реєстратури медичного закладу

З наведеного прикладу видно, як крок за кроком користувач ініціює дію, система обробляє запит, передає його через відповідні шари (UI → логіка → база даних), отримує відповідь і повертає результат. Це дозволяє не лише перевірити цілісність сценарію, а й проаналізувати, які модулі задіяні в кожній операції, де можливе виникнення помилки або затримки, та як можна оптимізувати взаємодію.

Діаграма ілюструє послідовну взаємодію між користувачами системи й об’єктами програмного забезпечення. Учасниками є працівник реєстратури, системний адміністратор і сама система, яка обробляє запити та генерує відповідні відповіді.

У табл. 1.6 узагальнено функціональні ролі кожного учасника та типові повідомлення, які передаються між компонентами.

Таблиця 1.6

Учасники та повідомлення в діаграмі послідовності

№	Учасник	Опис ролі	Типові повідомлення
1	Працівник реєстратури	Ініціює процеси, пов'язані з додаванням пацієнтів та переглядом даних	Запити: «Створити пацієнта», «Переглянути список пацієнтів»
2	Системний адміністратор	Керує користувачами, графіками, системними налаштуваннями	Запити: «Додати користувача», «Створити графік», «Змінити параметри системи»
3	Система	Приймає та обробляє запити, повертає відповіді	Відповіді: «Пацієнта створено», «Список пацієнтів оновлено», «Користувача додано»

Взаємодія між учасниками відбувається у формі синхронних викликів, що вимагають підтвердження, та асинхронних відповідей. Передача запитів представлена суцільними стрілками, що вказують на виклик дії, а відповіді відображаються пунктирними лініями, які показують повернення результату.

Послідовність подій розгортається згори вниз, від ініціації запиту працівником реєстратури до обробки його системою. Наприклад, після створення запиту на додавання нового пацієнта система виконує перевірку, додає запис у базу й повертає підтвердження. Далі користувач ініціює запит на перегляд списку пацієнтів і отримує результат. Адміністратор у той самий час надсилає запити на створення графіка лікаря або додавання користувача, кожен з яких також супроводжується підтвердженням з боку системи після завершення відповідної логіки обробки.

Діаграма активності (Activity Diagram) у мові UML є ефективним інструментом для візуалізації послідовності дій у рамках певного процесу. Вона відображає логіку поведінки системи, демонструючи переходи між етапами на основі умов, подій або результатів. За своєю структурою діаграма подібна до

блок-схеми, однак адаптована до потреб об’єктно-орієнтованого аналізу й системного проєктування. Вона дозволяє описати, як система змінює свій стан у відповідь на дії користувачів або зовнішні події.

На рис. 1.3 наведено приклад діаграми активності для інформаційної системи автоматизованого робочого місця працівника реєстратури медичного закладу. Ця діаграма моделює процес оформлення візиту пацієнта – від його прибуття до запису на прийом. Представлення подано у вигляді «доріжок» (Swimlanes), які розділяють відповідальність між двома учасниками: пацієнтом і реєстратором.

Пацієнт виступає ініціатором процесу, прибуваючи до медичного закладу. Подальша логіка визначається за умовою: якщо пацієнт приходить вчасно, він очікує перевірки у базі; у разі запізнення – система передбачає сценарій очікування дзвінка або скасування запису. Якщо пацієнт не знайдений у базі, йому пропонують заповнити анкету для первинної реєстрації.

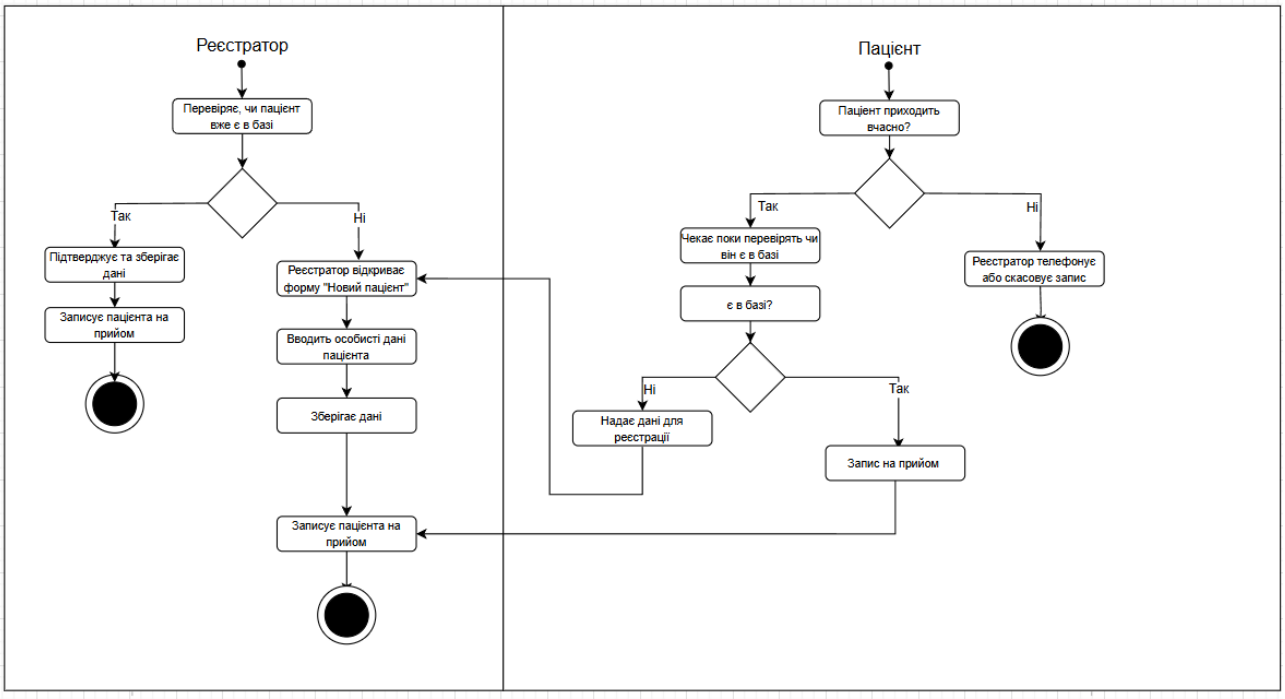


Рис. 1.3 Діаграма активності для системи автоматизованого робочого місця працівників реєстратури медичного закладу

Пацієнт виступає ініціатором процесу, прибуваючи до медичного закладу. Подальша логіка визначається за умовою: якщо пацієнт приходить вчасно, він очікує перевірки у базі; у разі запізнення – система передбачає сценарій очікування дзвінка або скасування запису. Якщо пацієнт не знайдений у базі, йому пропонують заповнити анкету для первинної реєстрації.

У той самий час реєстратор, як активний учасник, виконує перевірку наявності пацієнта в базі, ініціює створення нового запису в разі відсутності інформації, вводить персональні дані, а також зберігає їх у системі. Завершальним етапом є оформлення прийому, який фіксується в інформаційній системі.

У табл. 1.7 узагальнено дії учасників процесу та логіку переходів між етапами. Діаграма дозволяє побачити не лише стандартний сценарій, коли пацієнт уже зареєстрований, а й альтернативні варіанти – наприклад, реєстрація нового пацієнта або обробка ситуації, коли пацієнт запізнився.

Таблиця 1.7

Учасники та дії в діаграмі активності

№	Учасник	Дії
1	Пацієнт	Прибуття до закладу → перевірка часу → очікування / заповнення анкети / підтвердження
2	Працівник реєстратури	Перевірка в базі → відкриття форми → введення даних → збереження → оформлення прийому

Використання діаграми активності дає змогу наочно відобразити бізнес-процес взаємодії в реєстратурі та проаналізувати його на предмет ефективності, надійності та виняткових ситуацій. Така візуалізація є незамінним засобом для опису логіки функціонування системи, комунікації між користувачами й компонентами ПЗ та закладає основу для технічної реалізації робочих сценаріїв у медичному середовищі.

Використання UML-моделей у процесі проєктування є ефективним інструментом для перевірки логіки функціонування системи, пошуку слабких місць та забезпечення спільного розуміння між розробниками, аналітиками та замовником. Це особливо важливо в умовах медичних установ, де критичним є

не лише функціональний, а й організаційний аспект роботи програмного забезпечення.

Моделювання предметної області за допомогою засобів UML дозволяє сформувати цілісне уявлення про функціональну логіку системи, її структуру та взаємозв’язки між компонентами. Такий підхід забезпечує формальний опис основних процесів і користувацьких сценаріїв, що є основою для ефективного технічного проєктування та реалізації програмної системи у сфері медичного обслуговування.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед початком створення будь-якої інформаційної системи важливо провести ретельний аналіз існуючих джерел, програмних рішень та сучасних технологічних підходів, які вже використовуються у відповідній галузі. Такий аналіз дозволяє не лише ознайомитися з поточним станом справ у сфері медичних інформаційних систем, а й зрозуміти основні потреби кінцевих користувачів – працівників реєстратури, адміністраторів та пацієнтів.

У рамках розробки програмного забезпечення для автоматизації роботи реєстратури медичного закладу було проаналізовано низку вже реалізованих систем, як локальних, так і комерційних, зокрема: електронні медичні реєстри, амбулаторні облікові системи, онлайн-сервіси запису на прийом [23]. Особливу увагу приділено тому, як організовано взаємодію між користувачем і базою даних, наскільки інтерфейс є зручним, і як реалізовано захист персональних даних.

Такий порівняльний аналіз дозволяє виділити найкращі практики реалізації ключових модулів (наприклад, пошуку пацієнта, створення графіків лікарів, генерації звітів) та виявити недоліки, що зустрічаються у вже існуючих рішеннях. Наприклад, у деяких системах відсутній модуль логування дій або можливість формування резервної копії бази даних – ці функції доцільно врахувати в новій розробці.

Головна мета такого дослідження полягає в тому, щоб визначити ефективні компоненти, які необхідно включити до програмного продукту, та виявити аспекти, які можуть бути вдосконалені з урахуванням специфіки медичної установи. Це дозволить створити конкурентоспроможну, зручну, функціональну та захищену систему, яка відповідатиме сучасним вимогам і практичним очікуванням персоналу реєстратури.

Medstar HIS – це комплексна система управління лікарнею, яка охоплює всі аспекти медичного обслуговування, включаючи реєстратуру, електронні медичні записи та фінансовий облік [15]. Система забезпечує інтеграцію між різними відділеннями медичного закладу, що сприяє ефективному обміну інформацією та покращенню якості обслуговування пацієнтів (рис. 1.4). Її гнучка архітектура дозволяє адаптувати функціонал під специфічні потреби закладу.

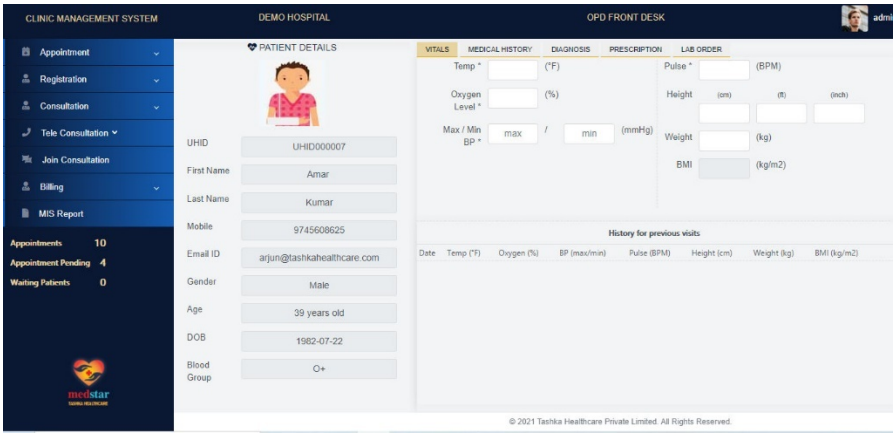


Рис. 1.4 Інтерфейс Medstar HIS

Doctor Eleks – українська медична інформаційна система, яка підтримує повний цикл роботи з пацієнтами, включаючи реєстрацію, ведення електронних медичних карток, управління графіками лікарів та облік медичних послуг [16]. Система відповідає вимогам національної електронної охорони здоров’я та забезпечує інтеграцію з eHealth. Її інтуїтивно зрозумілий інтерфейс сприяє швидкому навчанню персоналу та ефективному використанню в повсякденній роботі (рис. 1.5).

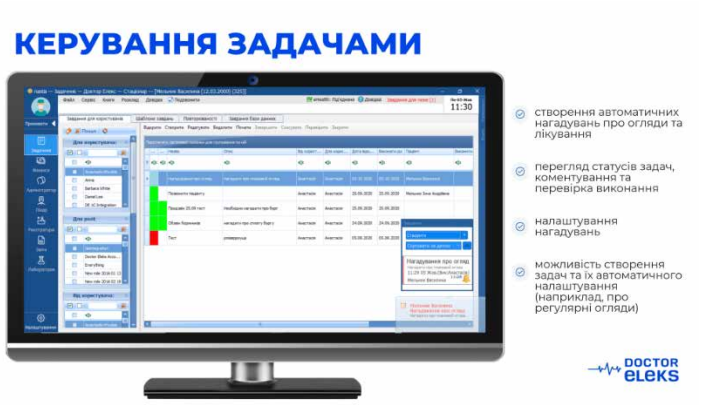


Рис. 1.5 Інтерфейс Doctor Eleks

OpenEMR – відкрита медична система, яка дозволяє вести облік пацієнтів, організовувати прийоми, формувати історії хвороб і виставляти рахунки [17]. Система має міжнародну спільноту розробників, активно підтримується та оновлюється. Перевагою є відкритий вихідний код, що дозволяє адаптувати функціонал під локальні потреби. З точки зору нашого проєкту особливо цікавим є реалізація процесу створення та редагування записів пацієнтів. Основна складність – англомовний інтерфейс та потреба в доопрацюванні української локалізації (рис. 1.6).

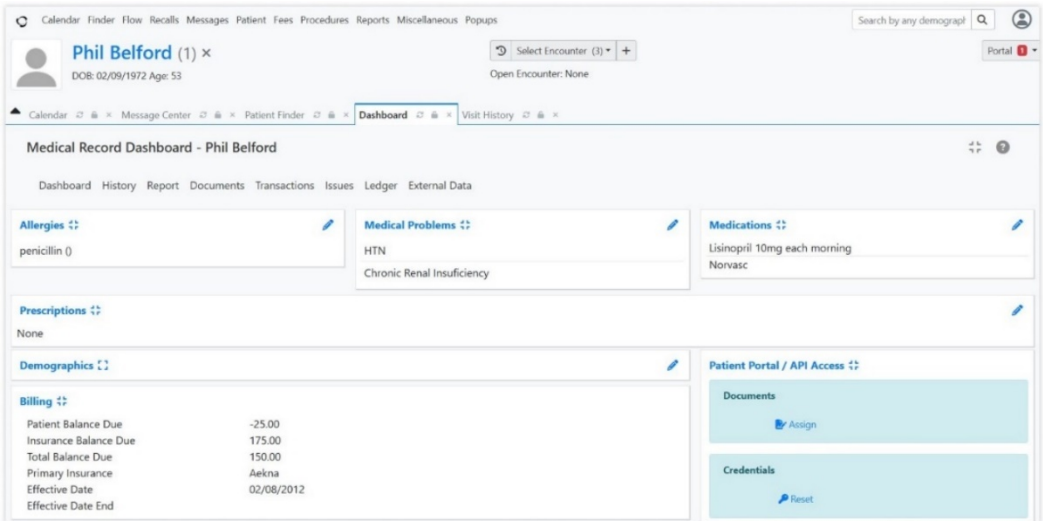


Рис. 1.6 Інтерфейс OpenEMR

SimplexMed – програмне забезпечення, призначене для автоматизації роботи медичних закладів, зокрема реєстратур [18]. Система підтримує функції електронного запису пацієнтів, управління графіками лікарів, ведення електронних медичних карток та формування звітності (рис. 1.7).

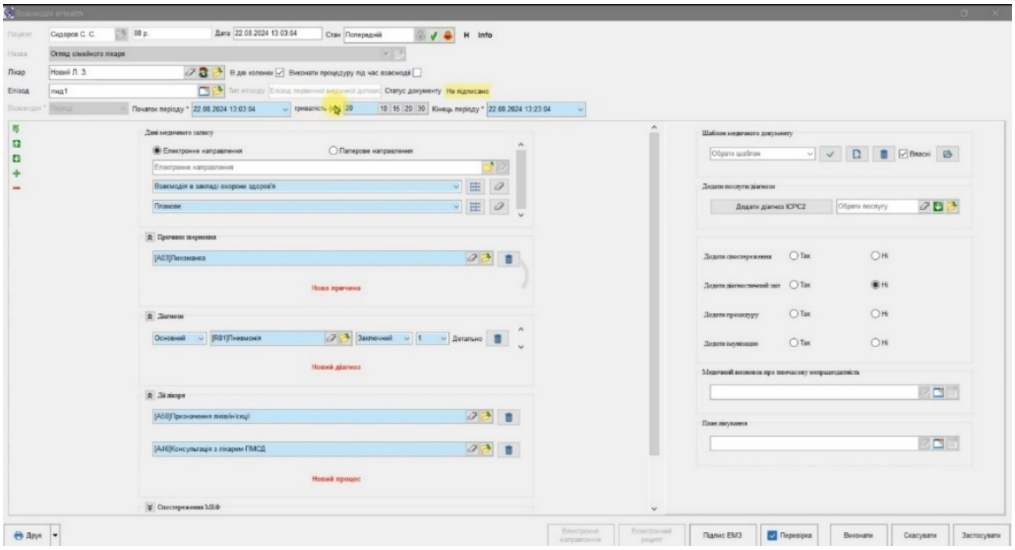


Рис. 1.7 Інтерфейс SimplexMed

Її простий та зручний інтерфейс дозволяє швидко освоїти систему навіть користувачам без глибоких технічних знань. SimplexMed також забезпечує інтеграцію з національною системою eHealth, що є важливим для відповідності сучасним вимогам охорони здоров’я в Україні.

eHealth Україна – національна електронна система охорони здоров’я, яка забезпечує централізоване зберігання та обробку медичних даних пацієнтів []. Система дозволяє медичним закладам інтегруватися з державними сервісами, такими як електронні рецепти, направлення та декларації. Для нашого проєкту важливо врахувати вимоги та стандарти eHealth при розробці програмного забезпечення, щоб забезпечити сумісність та відповідність національним нормативам (рис. 1.8).

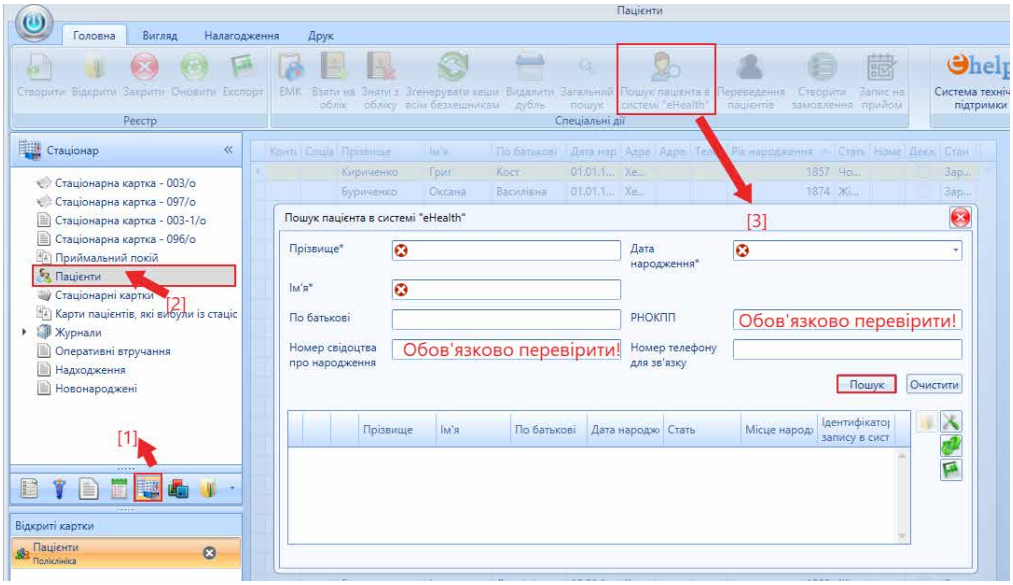


Рис. 1.8 Інтерфейс eHealth Україна

Огляд джерел та аналіз існуючих програмних рішень виявив необхідність створення адаптованої до умов локальних закладів охорони здоров’я системи з інтуїтивним інтерфейсом, можливістю локальної інсталяції та підтримкою повного циклу обробки даних. Така система має стати основою для цифрової трансформації роботи реєстратури, підвищуючи її ефективність і зменшуючи ризики помилок при обробці медичної інформації.

1.5 Постановка завдання

У межах системного аналізу предметної області було досліджено інформаційні потоки, характерні для діяльності працівника реєстратури медичного закладу. Особливу увагу приділено процесам реєстрації пацієнтів, планування медичних прийомів, ведення графіків роботи лікарів, а також підготовці звітної документації. На підставі результатів цього аналізу сформульовано завдання, яке визначає цілі, вимоги та функціональні обмеження майбутньої програмної системи.

Потреба у створенні такого програмного забезпечення обумовлена зростанням обсягів обробки медичних даних, необхідністю швидкого та безпечного доступу до них, а також актуальними вимогами до автоматизації

адміністративної діяльності в галузі охорони здоров’я. Впровадження інструменту автоматизованої реєстратури дозволить мінімізувати ручну працю, підвищити точність введення інформації, а також забезпечити цілісність і конфіденційність персоніфікованих записів.

В основі проєкту – розробка прикладного рішення, яке інтегрує функціонал реєстрації пацієнтів, організації прийомів відповідно до розкладу лікарів, створення та друку направлень і звітів. Крім того, система має передбачати підтримку авторизації з розмежуванням прав доступу, ведення журналу дій користувачів, збереження даних у форматі, придатному для резервного копіювання, а також забезпечення зручної експлуатації без потреби в спеціальній технічній підготовці персоналу.

Формалізована постановка завдання передбачає розробку масштабованої, сумісної з національними медичними інформаційними платформами архітектури, що поєднує простоту в користуванні з функціональною повнотою та високим рівнем надійності. Таким чином, проєкт спрямований на створення ефективного інструменту цифрової підтримки обліково-реєстраційних процесів у медичному закладі.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проєктування бази даних є ключовим етапом створення інформаційної системи. На цьому етапі формується логічна структура зберігання, обробки та взаємодії даних між різними компонентами системи. Основним інструментом, що використовується для формального опису взаємозв'язків між сутностями, є ER-діаграма (Entity–Relationship).

ER-моделювання дозволяє виявити основні сутності предметної області, їх атрибути та логічні зв'язки, що існують між ними. Це забезпечує уніфіковане бачення структури даних усіма учасниками процесу розробки: від розробників до замовників. У результаті підвищується якість реалізації, зменшується ризик логічних помилок у базі даних і полегшується подальше масштабування системи.

Особливу цінність ER-діаграма має в системах охорони здоров'я, де важливо забезпечити безперервний, логічно узгоджений і захищений облік пацієнтів, лікарів, записів на прийом, медичних карток тощо. Вона не лише структурує дані, а й задає логіку їх обробки, збереження та цілісності.

Серед найбільш ефективних інструментів для побудови ER-моделей варто виокремити CA ERwin Data Modeler – професійне CASE-середовище, яке дозволяє створювати концептуальні та логічні моделі з урахуванням бізнес-правил. Водночас для швидкого створення діаграм на ранніх етапах розробки зручно використовувати безкоштовний вебзастосунок Draw.io (diagrams.net), який дозволяє оперативно створювати графічні схеми з підтримкою збереження в різних форматах.

Вибір інструменту залежить від складності системи, вимог до документації та подальшої інтеграції з іншими засобами розробки.

На рис. 2.1 представлено графічну модель логічних зв’язків між ключовими сутностями системи:

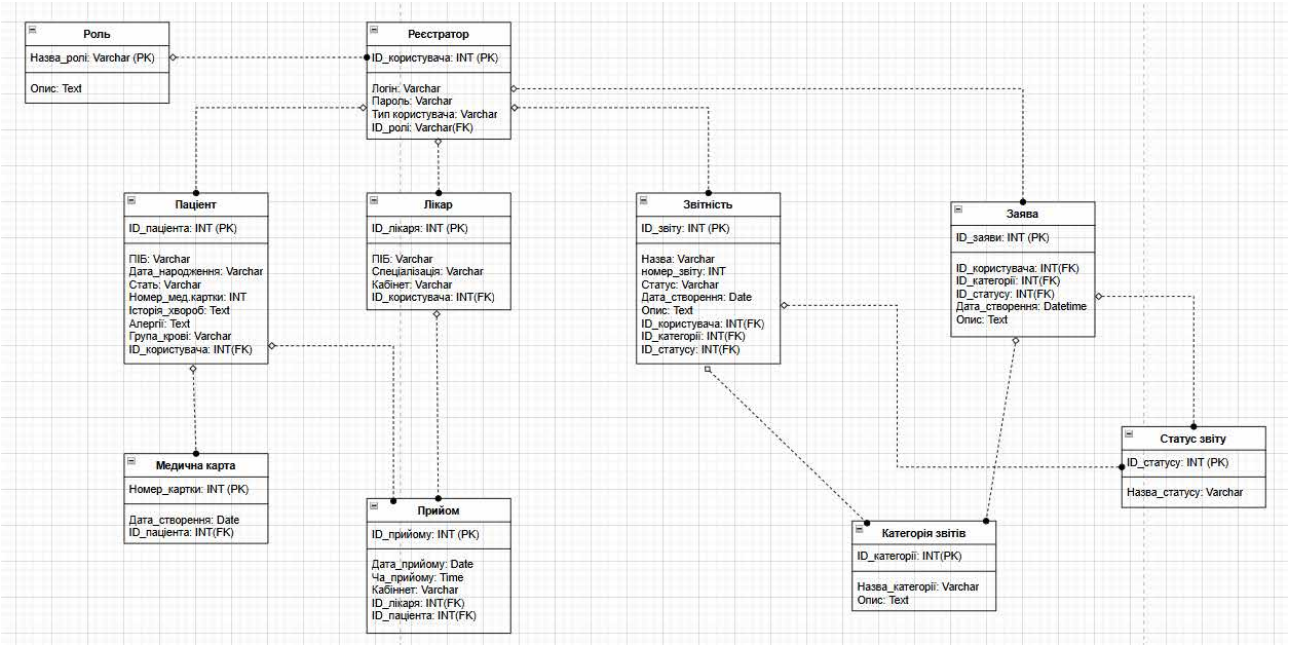


Рис. 2.1 ER-діаграма для системи автоматизованого робочого місця працівників реєстратури медичного закладу

З рисунка видно, що структура системи охоплює низку сутностей, зокрема Пацієнт, Лікар, Прийом, Медична картка, Користувач, Звітність, Заява тощо. Кожна сутність має відповідні атрибути та логічні зв’язки, які формалізують бізнес-процеси медичного закладу.

Для забезпечення цілісного уявлення про логічну модель даних, що використовується в системі автоматизованого робочого місця працівника реєстратури, доцільно узагальнити складові її структури у формі таблиці. Такий підхід дозволяє наочно відобразити основні параметри кожної сутності, включаючи первинні та зовнішні ключі, атрибути, що характеризують об’єкти предметної області, а також логічні зв’язки між ними. Представлена нижче табл. 2.1 слугує концептуальним відображенням логічної схеми бази даних, яка була сформована на основі вимог до функціонування інформаційної системи.

Таблиця 2.1

Опис сутностей логічної моделі даних

Сутність	Первинний ключ	Зовнішні ключі	Інші атрибути	Зв’язки з іншими сутностями
Роль	Назва_ролі	–	Опис	Пов’язана з Реєстратор
Реєстратор	ID_користувача	Назва_ролі	Логін, Пароль, Тип_користувача	Роль, Звітність, Заява
Пацієнт	ID_пацієнта	ID_користувача	ПІБ, Дата_народження, Стать, Телефон тощо	Прийом, Медична_картка
Медична картка	Номер_картки	ID_пацієнта	Дата_створення	Пацієнт
Лікар	ID_лікаря	ID_користувача	ПІБ, Спеціалізація, Кабінет	Прийом
Прийом	ID_прийому	ID_лікаря, ID_пацієнта	Дата, Час, Кабінет	Пацієнт, Лікар
Звітність	ID_звіту	ID_користувача, ID_категорії, ID_статусу	Назва, Номер, Дата, Опис	Реєстратор, Категорія_звіту, Статус_звіту
Категорія звіту	ID_категорії	–	Назва, Опис	Звітність, Заява
Статус звіту	ID_статусу	–	Назва_статусу	Звітність, Заява
Заява	ID_заяви	ID_користувача, ID_категорії, ID_статусу	Дата_створення, Опис	Реєстратор, Категорія_звіту, Статус_звіту

Представлена таблиця дозволяє простежити логічну послідовність формування структури бази даних, визначити залежності між ключовими сутностями системи та зрозуміти загальну архітектуру інформаційної моделі. Зокрема, кожна сутність має чітко визначене розташування в загальній схемі та підтримує конкретні відношення з іншими компонентами, що забезпечує логічну зв’язність і узгодженість даних у межах системи.

На підставі наведеного опису можна перейти до наступного етапу – оцінки відповідності логічної моделі вимогам нормалізації, зокрема перевірки дотримання першої, другої та третьої нормальних форм. Такий аналіз дозволяє переконатися, що структура бази даних є ефективною, позбавлена надлишкових залежностей та дублювань, а також відповідає принципам якісного проєктування реляційної моделі [5].

Однією з ключових вимог до логічної моделі бази даних є її відповідність нормалізованій структурі, що гарантує узгодженість, уникнення надмірного дублювання інформації та стабільну взаємодію компонентів системи в процесі експлуатації. Особливе значення в цьому контексті має третя нормальна форма (ЗНФ), яка передбачає дотримання ряду умов, спрямованих на усунення зайвих залежностей між даними.

Аналіз структури бази даних, побудованої для автоматизованого робочого місця працівника реєстратури, свідчить про відповідність основним етапам нормалізації. Перший рівень – перша нормальна форма – передбачає, що всі поля таблиць мають атомарну природу, тобто зберігають лише одне значення в кожному рядку без повторюваних груп. Усі таблиці моделі, зокрема «Пацієнт», «Прийом», «Лікар», «Звітність» тощо, структуровані таким чином, що їхні атрибути зберігають однозначні значення. Наприклад, поле ПІБ у таблиці пацієнтів призначене для зберігання одного повного імені, а поле дати прийому фіксує лише одну календарну дату.

Другий рівень – друга нормальна форма – передбачає, що всі неключові атрибути повністю залежать від усього первинного ключа, а не від його частини. Це особливо важливо в тих випадках, коли первинний ключ складений. У поточній моделі, однак, усі ключі є простими, що спрощує перевірку. Наприклад, у таблиці «Прийом» усі атрибути, включаючи дату, час і кабінет, мають однозначну залежність від ідентифікатора прийому. Аналогічна ситуація спостерігається в інших сутностях: у таблиці пацієнтів усі дані підпорядковані унікальному ідентифікатору пацієнта, а в таблиці заяв – ідентифікатору відповідної заяви. Усі залежності є повними та не містять часткових співвідношень.

Переходячи до третьої нормальної форми, слід переконатися у відсутності транзитивних залежностей, тобто таких, за яких один неключовий атрибут залежить від іншого неключового атрибута, а не безпосередньо від первинного ключа. Перевірка структури таблиць свідчить, що всі поля зберігають прямий зв’язок із ключовими ідентифікаторами. Наприклад, у таблиці «Пацієнт» дані

про стать, дату народження, групу крові не залежать один від одного, а лише від ID_пацієнта. У «Лікарях» поля спеціалізації чи кабінету не є похідними один від одного, а безпосередньо відносяться до ID_лікаря. У таблицях, що містять службові або довідкові дані – таких як «Категорія звіту», «Статус звіту», «Роль» – кожен атрибут пов’язаний винятково з відповідним первинним ключем.

Таким чином, уся структура моделі даних відповідає критеріям третьої нормальної форми. Всі атрибути є атомарними, залежності між ними чітко визначені й не перетинаються, а транзитивні зв’язки повністю виключено. Це не лише забезпечує логічну цілісність і чистоту даних, а й суттєво підвищує ефективність роботи з базою в процесі експлуатації, спрощує модифікації структури при її масштабуванні та зменшує ймовірність помилок при інтеграції з іншими модулями інформаційної системи.

2.2 Діаграма класів та кооперацій

Однією з ключових складових об’єктно-орієнтованого підходу до моделювання інформаційних систем є діаграма класів, що належить до основних типів уніфікованої мови моделювання UML (Unified Modeling Language). Вона виконує роль засобу графічного відображення внутрішньої структури програмного забезпечення, формалізуючи об’єкти (класи), їхні властивості, методи та логічні взаємозв’язки. Така візуалізація дає змогу описати архітектуру системи на концептуальному рівні, а також створити цілісне уявлення про логіку функціонування програмного продукту.

Для узагальнення функціонального призначення цього типу діаграм у табл. 2.2 наведено основні цілі їх використання, що охоплюють як технічну документацію, так і підтримку командної розробки. Як видно з наведених положень, діаграма класів виконує не лише описову функцію, а й слугує інструментом аналізу та валідації внутрішньої логіки майбутнього програмного продукту. У межах проєкту створення автоматизованого робочого місця реєстратора медичного закладу було сформовано набір основних класів, які відображають ключові сутності системи.

Таблиця 2.2

Основні цілі щодо використання діаграми класів

№	Ціль	Опис
1	Візуалізація структури системи	Наочно відображає класи, їх атрибути, методи та взаємозв’язки.
2	Формування об’єктно-орієнтованої моделі	Створює логічну архітектуру системи відповідно до принципів ООП.
3	Аналіз залежностей між компонентами	Дозволяє простежити взаємозв’язки та типи зв’язків між класами.
4	Документування архітектури	Використовується як структурна частина технічної документації.
5	Підтримка командної розробки	Забезпечує спільне бачення між розробниками, аналітиками та тестувальниками.
6	Виявлення логічних помилок	Сприяє ранньому виявленню надмірних залежностей або дублювання.

Щоб деталізувати склад логічних компонентів, доцільно звернутися до табл. 2.3, у якій представлено короткий опис структурних елементів класів – їх властивостей і методів. Ця інформація є основою для подальшої реалізації програмної логіки.

Таблиця 2.3

Структура діаграми класів

№	Клас	Властивості	Методи
1	2	3	4
1	Системний адміністратор	Додати користувача, Налаштувати систему, Управляти категоріями	–
2	Працівник реєстратури	Редагування даних пацієнта, Перегляд списку, Пошук пацієнта	Реєстрація пацієнта, Створення звітів, Запис на прийом
3	Пацієнт	ПІБ, Дата народження, Стать, Номер телефону, Номер мед. картки	–
4	Медична картка	Номер картки, Дата створення, Пацієнт	Додати запис, Отримати записи
5	Прийом	Дата прийому, Час, Кабінет, Пацієнт, Лікар	Записати на прийом, Скасувати прийом

Таблиця 2.3 (продовження)

1	2	3	4
6	Лікар	ПІБ, Спеціалізація, Кабінет	Переглянути записи, Додати діагноз
7	Звітність	Назва, Номер звіту, Статус, Дата створення, Опис	Зберегти звіт, Видалити звіт

Графічне зображення структури, сформоване на основі цієї таблиці, подано на рис. 2.2. Воно дозволяє узагальнити архітектуру класів і побачити логіку взаємозв’язків між об’єктами системи.

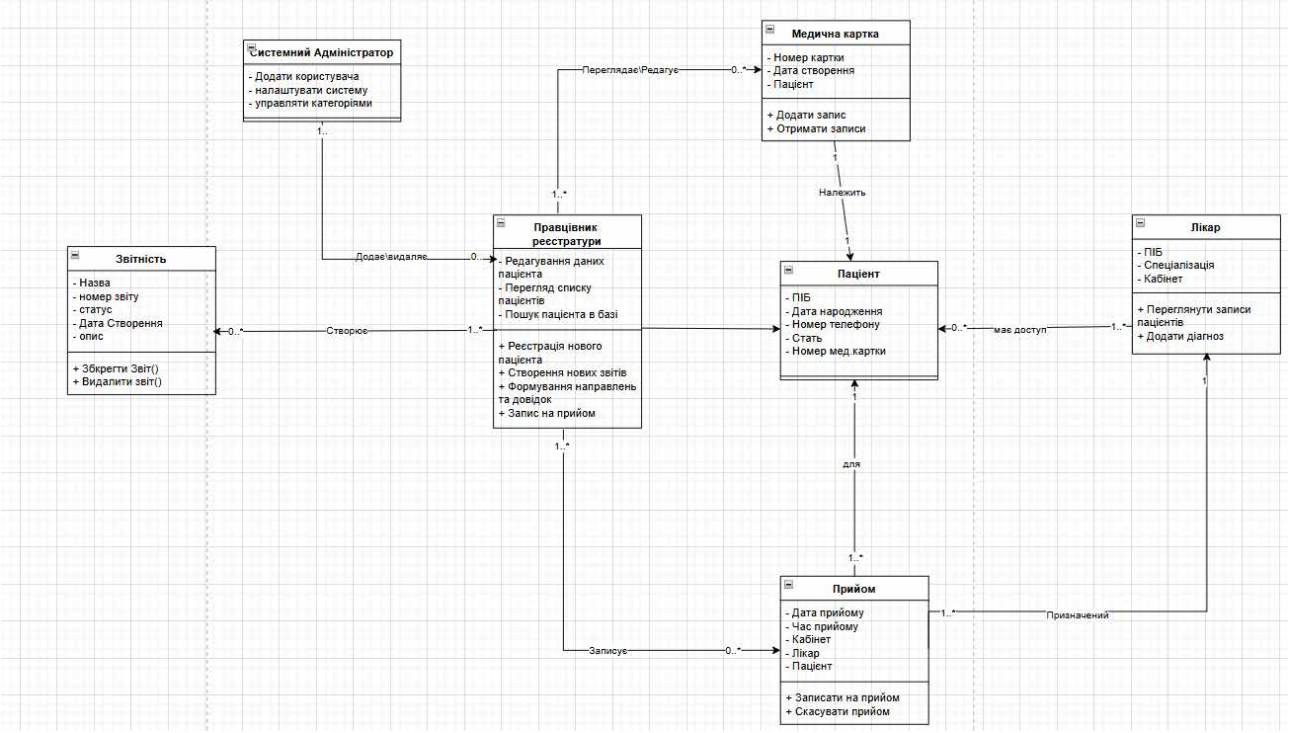


Рис. 2.2 Діаграма класів для системи автоматизованого робочого місця працівників реєстратури медичного закладу

Як видно з діаграми, центральне місце займають класи, пов’язані з обробкою пацієнтів, прийомів, звітів та адмініструванням користувачів. Клас «Системний адміністратор» відповідає за технічне налаштування системи, додавання облікових записів і керування категоріями, тоді як «Працівник реєстратури» виконує операції з пацієнтами, формує записи на прийом та забезпечує створення медичної документації. Клас «Пацієнт» зберігає особисті

дані, які надалі асоціюються з медичною карткою, що акумулює історію записів. Клас «Прийом» виступає сполучною ланкою між пацієнтом і лікарем, забезпечуючи фіксацію часу, кабінету й відповідального медичного працівника. Водночас «Лікар» має доступ до перегляду прийомів і внесення діагнозів. Нарешті, клас «Звітність» узагальнює інформацію для статистичного або адміністративного аналізу.

Побудована діаграма класів дозволила структурувати архітектуру системи на рівні логічного представлення класів, забезпечивши чітке розмежування функцій між елементами, підвищення зрозумілості проєкту та полегшення командної взаємодії. У подальшому це створює ґрунт для переходу до деталізації кооперaційної взаємодії класів і програмної реалізації функціональних модулів.

У процесі проєктування прикладного програмного забезпечення надзвичайно важливим є моделювання поведінки системи в умовах виконання конкретних сценаріїв. У цьому контексті ефективним засобом візуалізації взаємодії між об’єктами виступає діаграма кооперації (cooperation diagram), яка дозволяє відобразити динаміку обміну повідомленнями між окремими екземплярами класів у рамках певної функціональної задачі. На відміну від діаграми класів, що фокусується на статичній архітектурі системи, діаграма кооперації ілюструє саме поведінкову сторону – порядок дій, обмін сигналами та логіку реалізації сценаріїв у конкретних умовах.

У межах кожної кооперaційної діаграми демонструються лише ті об'єкти, які безпосередньо залучені до виконання відповідної дії. Комунікація між ними подається у вигляді підписаних стрілок, що вказують напрям, порядок та зміст передавання повідомлень. Таким чином, забезпечується структуроване уявлення про те, як саме взаємодіють частини системи на прикладі реального використання, що значно підвищує прозорість логіки програмної реалізації.

На наступних рисунках представлено фрагменти кооперaційної моделі системи автоматизованого робочого місця працівників реєстратури медичного закладу, відповідно до типових сценаріїв взаємодії користувача із програмою.

Рисунок 2.3 демонструє сценарій реєстрації пацієнта. Тут взаємодіють такі об'єкти, як працівник реєстратури, форма введення персональних даних та об’єкт доступу до бази даних. Послідовність дій включає ініціалізацію форми, заповнення обов’язкових полів, перевірку правильності введення та збереження даних у відповідну таблицю.

Дана діаграма відображає послідовну логіку внесення нового запису до бази з акцентом на валідацію вхідних даних і правильну координацію між інтерфейсом і механізмами збереження інформації.

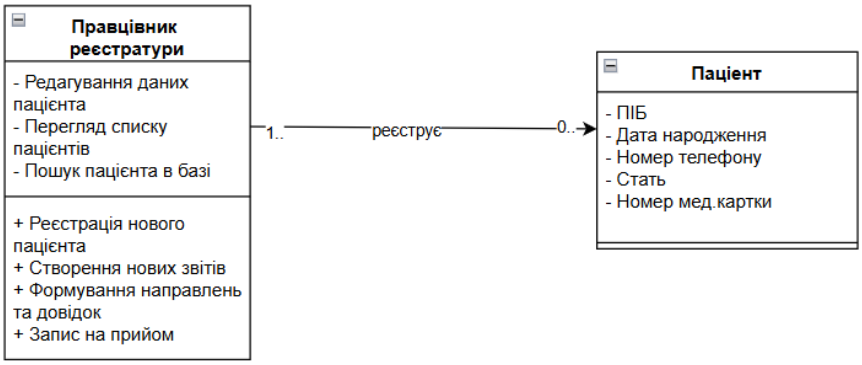


Рис. 2.3 Кооперaційна діаграма сценарію реєстрації пацієнта

Рисунок 2.4 ілюструє процес запису пацієнта на прийом. У моделі взаємодіють інтерфейсна форма, модуль перевірки доступності лікаря та об’єкти запису до бази. Комунікація відбувається шляхом ініціювання запиту, перевірки вільного часу лікаря й остаточного створення запису.

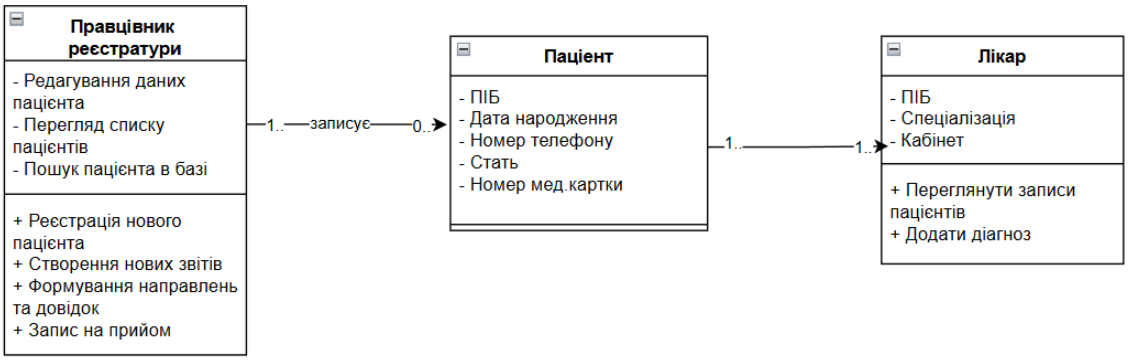


Рис. 2.4 Кооперaційна діаграма сценарію запису на прийом

Сценарій демонструє дотримання вимог логічної послідовності: перевірка доступності лікаря виконується до внесення запису, що унеможливлює накладання прийомів.

У рис. 2.5 представлено взаємодію при перегляді інформації про пацієнта. Об’єкти кооперації включають елемент інтерфейсу, контролер пошуку та репозиторій пацієнтів. Запит від користувача призводить до вибірки відповідного запису й відображення деталей у вигляді таблиці або форми.

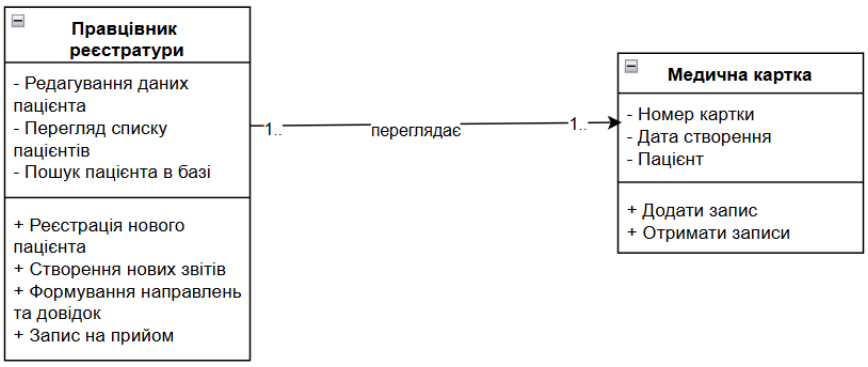


Рис. 2.5 Кооперaційна діаграма перегляду інформації про пацієнта

Дана схема демонструє механізм реалізації фільтрації й пошуку за заданими критеріями, що підвищує зручність користування системою в умовах великого обсягу даних.

Рисунок 2.6 зображає послідовність дій під час формування звітності. Взаємодія відбувається між модулем збору статистичних даних, інтерфейсом для введення параметрів звіту та об’єктом генерації результату у форматі PDF.

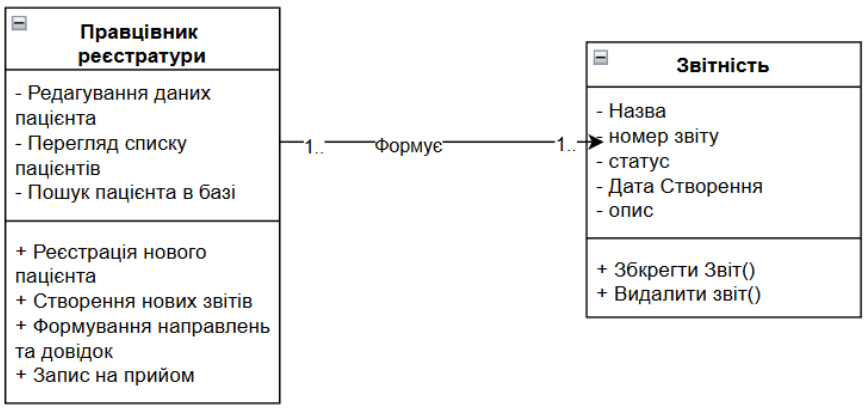


Рис. 2.6 Кооперaційна діаграма сценарію формування звітності

Завдяки цій діаграмі можна прослідкувати логіку побудови звітів із урахуванням обраних критеріїв, що дозволяє автоматизувати аналітичну функцію системи та мінімізувати людський фактор у розрахунках.

Рисунок 2.7 демонструє сценарій керування користувачами, реалізований адміністратором системи. Модель включає об’єкти інтерфейсу керування, модуль перевірки прав доступу, об’єкт створення/редагування записів та форму підтвердження дії.

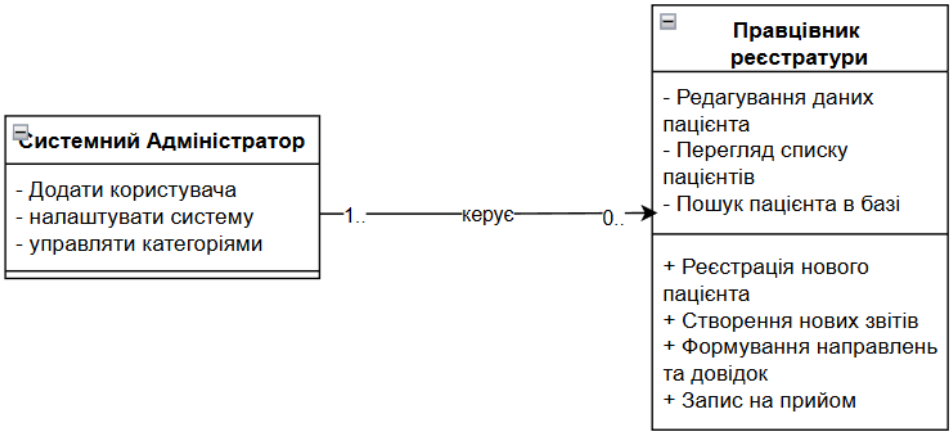


Рис. 2.7 Кооперaційна діаграма сценарію керування користувачами

Представлена діаграма акцентує увагу на ролі адміністратора у підтримці структурної безпеки системи, а також підкреслює важливість контролю за обліковими записами для забезпечення стабільного функціонування.

Загалом кооперaційні діаграми, подані в цьому підрозділі, дозволяють деталізувати логіку поведінки системи у відповідь на ключові дії користувачів. Їх використання сприяє підвищенню прозорості внутрішніх процесів, покращенню комунікації між елементами системи та спрощенню розробки й тестування окремих сценаріїв. У сукупності з діаграмами класів вони утворюють цілісну модель, що відображає як структуру, так і динаміку роботи інформаційної системи.

Таким чином, діаграми класів і кооперацій виступили фундаментальними інструментами для опису архітектури та поведінки інформаційної системи. Діаграма класів надала цілісне уявлення про логічну структуру об’єктів, їх

атрибути та взаємозв’язки, що дозволило сформувати концептуальну модель програмного забезпечення. У свою чергу, діаграми кооперацій деталізували механізми взаємодії між об’єктами під час реалізації основних сценаріїв, таких як реєстрація пацієнтів, запис на прийом, перегляд інформації, формування звітів та керування користувачами. Поєднання обох типів діаграм забезпечує всебічне розуміння як статичних, так і динамічних аспектів функціонування системи, сприяє підвищенню якості проєктування та полегшує подальшу реалізацію і супровід програмного продукту.

2.3 Діаграма пакетів

У проєктуванні сучасних інформаційних систем, особливо в медичній сфері, надзвичайно важливо забезпечити раціональну структуризацію архітектури програмного забезпечення. Системи, що містять численні функціональні модулі, вимагають логічної організації їхніх компонентів для досягнення цілісності, гнучкості та легкості супроводу. Одним із ефективних інструментів, який дозволяє реалізувати таку структурну декомпозицію, є **діаграма пакетів** (package diagram), що входить до складу уніфікованої мови моделювання UML [10].

Цей тип діаграми використовується для графічного подання взаємозв’язків між логічно пов’язаними частинами системи, які об'єднуються у вигляді так званих пакетів. Кожен пакет виступає своєрідним контейнером, що включає у свій склад класи, інтерфейси або підсистеми, які виконують схожі або взаємопов’язані функції. Такий підхід забезпечує не лише логічну ізоляцію елементів, а й сприяє модульності, що є критично важливою властивістю у великих програмних проєктах. У разі видалення пакета всі елементи, що належать до нього, автоматично виключаються з моделі, що дозволяє ефективно керувати залежностями між компонентами та підтримувати цілісність архітектури.

У рамках розробки системи автоматизованого робочого місця працівників реєстратури медичного закладу, діаграма пакетів відіграє роль своєрідної

логічної мапи. Вона дає змогу поділити загальну архітектуру системи на окремі підрівні відповідальності, що значно спрощує не лише початкову розробку, але й подальше супроводження та можливе масштабування програмного продукту.

Графічне подання такої архітектури наведено на рис. 2.8.

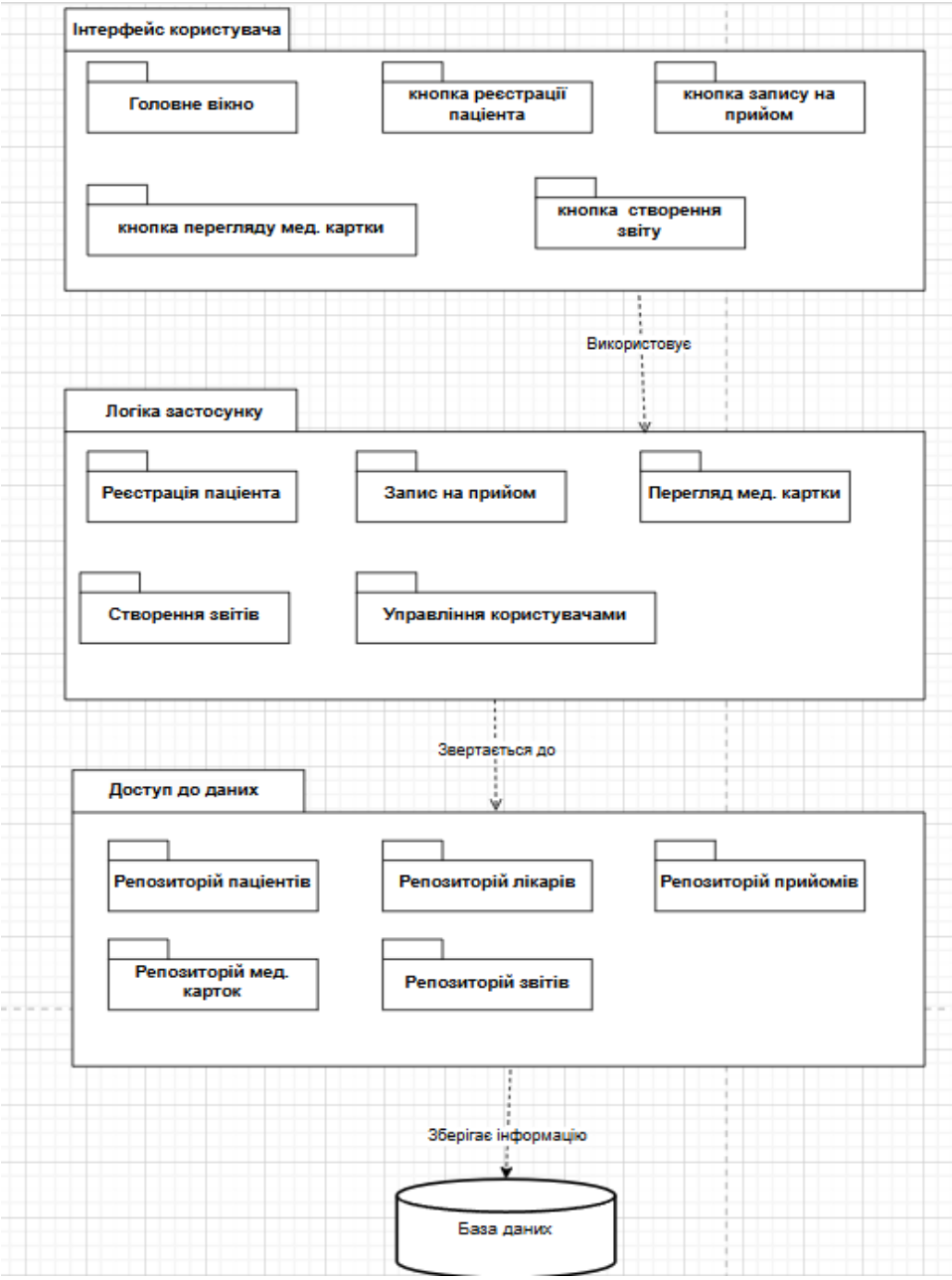


Рис. 2.8 Діаграма пакетів для системи автоматизованого робочого місця працівників реєстратури медичного закладу

Після розгляду діаграми пакетів (рис. 2.8), яка візуально структурує архітектуру програмного забезпечення автоматизованого робочого місця

реєстратури, доцільно подати узагальнений опис функціонального призначення кожного з її логічних рівнів. Такий підхід дозволяє краще усвідомити принципи взаємодії компонентів системи та сприяє побудові цілісного уявлення про її організацію. Нижче у табл. 2.4 наведено систематизовану характеристику архітектурних рівнів відповідно до їх функцій і складових елементів.

Таблиця 2.4

Функціональна структура архітектурних рівнів програмного забезпечення

Логічний рівень	Призначення	Основні елементи
Інтерфейс користувача (UI)	Забезпечує взаємодію користувача із системою, навігацію та ініціацію дій	Головне вікно, кнопки реєстрації, запису на прийом, перегляду картки, формування звітів
Логіка застосунку	Реалізує бізнес-процеси, обробляє запити з інтерфейсу, координує доступ до даних	Модулі реєстрації пацієнтів, запису, перегляду карток, звітності, управління ролями та користувачами
Рівень доступу до даних	Відповідає за звернення до бази даних, зчитування, оновлення та збереження	Репозиторії пацієнтів, лікарів, прийомів, медичних карток, звітів
База даних	Містить усю інформацію системи, до якої звертаються вищі рівні	Таблиці з даними пацієнтів, прийомів, медичних карток, користувачів, звітів

Подана структура чітко відображає багаторівневу архітектуру програмного забезпечення. Кожен рівень виконує окрему функціональну роль, що забезпечує розділення відповідальностей, полегшує підтримку, дозволяє масштабувати систему та підвищує надійність її роботи. Узгоджена взаємодія між рівнями гарантує стабільність процесів, а модульність – адаптивність до змін у вимогах або функціоналі.

Отже, побудова діаграми пакетів дозволила не лише візуалізувати логічну структуру програмного забезпечення, а й закласти архітектурну основу для подальшого розвитку системи. Її багаторівнева організація сприяє структурованому управлінню компонентами, полегшує супровід і розширення функціоналу, забезпечує гнучкість адаптації під специфіку медичного закладу. Така модульна архітектура є ключем до створення стабільного, надійного та масштабованого інформаційного середовища для роботи реєстратури.

2.4 Діаграма компонентів

У проєктуванні складних інформаційних систем, де особливого значення набуває модульність, масштабованість і підтримуваність архітектури, доцільним є використання діаграми компонентів у межах уніфікованої мови моделювання UML [10]. Така діаграма забезпечує візуалізацію високорівневої логічної структури програмного забезпечення, показуючи, з яких основних функціональних частин складається система, як саме ці частини взаємодіють одна з одною, а також через які інтерфейси відбувається обмін функціональністю та даними.

Компоненти, зображені на цій діаграмі, виступають самостійними логічними модулями, що реалізують окремі аспекти бізнес-логіки, обробки запитів або взаємодії з даними. Основна перевага такого підходу полягає у слабкому зв’язуванні між частинами системи, що дозволяє вносити зміни до одного компонента без суттєвого впливу на інші. Це сприяє гнучкості, повторному використанню функціоналу та легшому тестуванню.

На рис. 2.9 представлено діаграму компонентів, що реалізує логічну модель системи автоматизованого робочого місця працівників реєстратури медичного закладу.

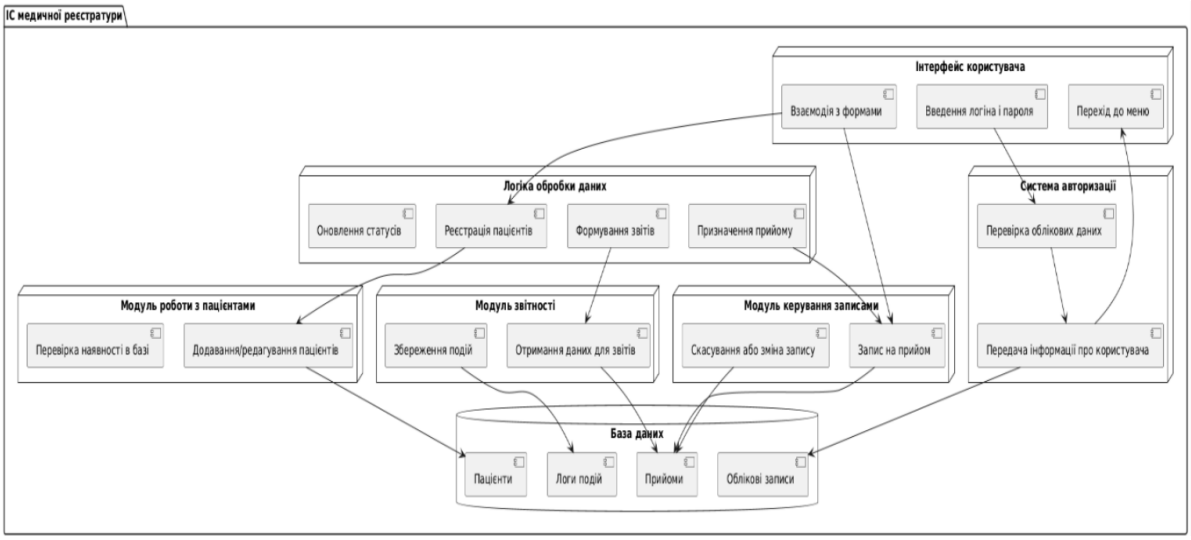


Рис. 2.9 Діаграма компонентів для системи автоматизованого робочого місця працівників реєстратури медичного закладу

Для розуміння представленої структури в табл. 2.5 узагальнено призначення кожного компонента системи відповідно до рівнів архітектури, які вони утворюють. Такий підхід дозволяє чітко простежити розподіл обов’язків, а також оцінити взаємозв’язки між логічними частинами.

Таблиця 2.5

Архітектурна структура системи за компонентами та їх призначенням

Рівень	Назва компонента	Функціональне призначення
Рівень представлення (Presentation Layer)	Інтерфейс користувача	Відображає дані, забезпечує введення інформації, взаємодію з формами, навігацію в системі
	Авторизаційна форма	Початковий доступ до системи через логін/пароль, перевірка прав користувача
Рівень прикладної логіки (Business Logic Layer)	Модуль реєстрації пацієнтів	Опрацювання персональних даних і створення записів у базі
	Модуль запису на прийом	Обробка запиту користувача, перевірка доступності лікаря, запис прийому
	Модуль перегляду карток	Пошук медичної історії пацієнта, формування відповідного запиту до репозиторію
	Модуль формування звітності	Побудова звітів на основі фільтрованих даних
	Модуль керування користувачами	Створення облікових записів, призначення ролей, контроль прав доступу
Рівень доступу до даних (Data Access Layer)	Репозиторій пацієнтів	Збереження та вибірка основних відомостей про пацієнтів
	Репозиторій лікарів	Операції над інформацією про медичних працівників
	Репозиторій прийомів	Зберігання історії прийомів пацієнтів
	Репозиторій медичних карток	Робота з історією обстежень та медичною документацією
	Репозиторій звітів	Ведення та вибірка аналітичної або медичної звітності
Рівень бази даних (Database Layer)	СУБД (база даних)	Фізичне зберігання всієї інформації, до якої звертаються вищі рівні системи

Подана структура чітко відображає багаторівневу архітектуру програмного забезпечення. Кожен рівень виконує окрему функціональну роль, що забезпечує розділення відповідальностей, полегшує підтримку, дозволяє масштабувати систему та підвищує надійність її роботи. Узгоджена взаємодія між рівнями гарантує стабільність процесів, а модульність – адаптивність до змін у вимогах або функціоналі.

Як видно з аналізу, архітектура системи є логічно цілісною та водночас модульною. Компоненти чітко поділені відповідно до функціональних обов’язків, що дозволяє забезпечити як незалежність їх розвитку, так і контрольовану взаємодію через інтерфейси. Такий підхід є ефективним для підтримки складних систем, які потребують гнучкої модернізації та стабільної роботи.

Отже, діаграма компонентів є важливим інструментом для опису архітектурної організації програмного забезпечення на високому рівні абстракції. Вона дає змогу структуровано представити систему як сукупність ізольованих, але взаємопов’язаних компонентів, що взаємодіють через чітко визначені інтерфейси. Завдяки цьому досягається узгодженість між модулями, спрощується управління змінами, поліпшується документація проєкту та забезпечується основа для ефективного командного розроблення. У медичних інформаційних системах така модульність відіграє критично важливу роль для підтримки безперервності, безпеки та точності обробки даних.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Ефективна робота будь-якої інформаційної системи неможлива без надійної системи управління базою даних (СУБД), яка забезпечує зберігання, обробку, захист і швидкий доступ до інформації. У розробці програмного забезпечення для автоматизованого робочого місця працівника реєстратури медичного закладу було обрано СУБД PostgreSQL – об’єктно-реляційне рішення з відкритим вихідним кодом, що поєднує гнучкість, стабільність і високу продуктивність [1].

PostgreSQL підтримує ACID-транзакції, складні типи даних, розширення, реплікацію, шифрування, політики доступу, індексацію та інші інструменти, що дозволяють створювати надійні та масштабовані бази даних для обробки критично важливої інформації [21]. Саме ці характеристики визначили вибір PostgreSQL як технологічної основи для реалізації даного проєкту.

Для обґрунтування вибору було проведено техніко-функціональний аналіз, результати якого наведено нижче у вигляді узагальненої табл. 3.1.

Таблиця 3.1

Переваги використання PostgreSQL у медичній інформаційній системі

Критерій	Характеристика PostgreSQL
Ліцензування	Відкрите ПЗ, безкоштовне використання, відсутність обмежень на розповсюдження
Транзакційність (ACID)	Підтримка повноцінних транзакцій, гарантія цілісності й узгодженості медичних записів
Безпека	Ролі користувачів, шифрування, політики доступу, перевірка автентичності
Розширюваність	Підтримка збережених процедур, тригерів, JSON, реплікацій, CTE, функцій на стороні сервера
Масштабованість	Стабільна робота з великими обсягами даних: від десятків до мільйонів записів
Сумісність та інтеграція	Сумісна з .NET через Npgsql, підтримує SQL-запити, транзакції, параметризовану обробку

PostgreSQL забезпечує баланс між надійністю, масштабованістю та гнучкістю, що робить її оптимальним вибором для проєктів у сфері охорони здоров’я, де важливі безпека, швидкість доступу та структурна цілісність інформації.

У створеній інформаційній системі база даних виконує роль єдиного джерела правди та ключового елементу взаємодії між усіма функціональними модулями. У табл. 3.2 наведено перелік основних таблиць бази даних, а також їх функціональне призначення.

Таблиця 3.2

Основні таблиці бази даних медичної інформаційної системи

№	Таблиця	Призначення
1	users	Зберігання облікових записів користувачів (логін, пароль, роль)
2	roles	Типи ролей (реєстратор, лікар, адміністратор) та їх описи
3	registrar	Розширена інформація про користувача з роллю реєстратора
4	doctor	Профілі лікарів, включаючи ПІБ, спеціалізацію, кабінет, статус
5	patient	Особисті дані пацієнтів, включаючи стать, групу крові, алергії, дату народження
6	medical_card	Зберігання номеру медичної картки, дати створення, посилання на пацієнта
7	appointment	Записи на прийом до лікаря (дата, час, пацієнт, лікар, кабінет, статус)
8	status	Довідник можливих статусів (для записів, запитів, звітів)
9	request	Обробка звернень, заяв, запитів від користувачів (дата, категорія, статус, автор)
10	report_category	Категорії звітів (адміністративні, медичні тощо)
11	report_status	Стан звітів (чернетка, підтверджено, скасовано тощо)
12	report	Основна таблиця звітності, зберігає зміст, авторство, категорію, дату створення, статус

Логічна структура бази забезпечує нормалізоване представлення даних, де кожна таблиця виконує чітко окреслену функцію, що сприяє підвищенню продуктивності запитів і підтриманню цілісності інформації.

Для підтримання надійної роботи системи PostgreSQL використовуються вбудовані механізми перевірки даних, оптимізації запитів та захисту від помилок, які узагальнено у табл. 3.3:

Таблиця 3.3

Засоби забезпечення цілісності та продуктивності в PostgreSQL

№	Інструмент	Призначення
1	Зовнішні ключі (FK)	Гарантують зв’язки між таблицями, виключають сирітські записи
2	Обмеження (Constraints)	NOT NULL, UNIQUE, CHECK – забезпечують валідацію даних при введенні або зміні
3	Індекси	Оптимізують вибірку при фільтрації, приєднанні, пошуку (наприклад, username, ID)
4	Тригери (Triggers)	Автоматизація логіки при зміні записів (наприклад, оновлення статусу при створенні звіту)
5	Write-Ahead Logging (WAL)	Гарантує відновлення даних після збою, підтримує транзакційну цілісність

PostgreSQL забезпечує комплексний захист даних та стабільну обробку запитів навіть за високих навантажень, що особливо важливо в критично чутливих медичних системах.

Застосунок розроблено на платформі .NET з використанням мови C# [8]. Для забезпечення зв’язку з базою даних використовується офіційна бібліотека Npgsql [4], що реалізує повноцінну взаємодію з PostgreSQL через SQL-інтерфейс. Така інтеграція дозволяє виконувати операції над базою даних безпосередньо з логіки застосунку (табл. 3.4).

Таблиця 3.4

Реалізовані можливості взаємодії з PostgreSQL через C#

№	Функція	Опис реалізації
1	SQL-запити	Виконання команд INSERT, SELECT, UPDATE, DELETE через підключення Npgsql
2	Параметризовані запити	Захист від SQL-ін’єкцій під час фільтрації, авторизації та обробки введених даних
3	Транзакції	Обробка взаємозалежних дій у рамках одного сеансу
4	Репозиторії/сервіси	Модульна структура для кожного бізнес-сценарію – окремий модуль доступу до даних

Застосування C# [8] у поєднанні з PostgreSQL забезпечує гнучку, безпечну та масштабовану реалізацію взаємодії між програмною логікою та базою даних, що відповідає вимогам до сучасної медичної інформаційної системи.

Таким чином, PostgreSQL виступає технологічною основою розробленої системи, забезпечуючи високу продуктивність, надійне зберігання даних та повну відповідність вимогам до безпеки й масштабованості. Поєднання структурованої моделі бази з функціональними механізмами цілісності й ефективною інтеграцією з .NET-архітектурою дозволило створити стабільну платформу для обробки чутливої медичної інформації, здатну до подальшого розширення і вдосконалення [7].

3.2 Розробка інформаційної бази

Створення інформаційної бази є одним із визначальних етапів проєктування програмної системи, оскільки вона формує основу для організації, зберігання та обробки ключових даних. У межах розробки інформаційної системи медичного реєстратура база даних виступає як централізоване сховище, що об'єднує всю інформацію про пацієнтів, лікарів, записи на прийом, звернення, звіти, користувачів і відповідні ролі доступу.

Інформаційна база розроблена відповідно до принципів нормалізації, що дозволяє зберігати дані у структурованому вигляді з мінімальним дублюванням і високим рівнем логічної узгодженості. Завдяки цьому система здатна підтримувати цілісність і достовірність інформації на всіх етапах її життєвого циклу і гарантує ефективний доступ до даних, спрощує побудову запитів і забезпечує гнучкість у випадку розширення функціональності.

З метою систематизації ключових переваг впровадження такої бази даних у медичному середовищі у табл. 3.5 узагальнено результати її структурного поділу за логічними блоками. Представлена структура бази даних демонструє раціональне розділення інформаційних блоків відповідно до їх функціонального призначення. Це дозволяє забезпечити логічну узгодженість, зручність доступу та гнучкість системи.

Таблиця 3.5

Розподіл таблиць бази даних

№	Структурний блок	Назва таблиці	Призначення таблиці
1	Користувачі та ролі	users	Зберігання логінів, паролів, ролей доступу та основної інформації про користувачів
		roles	Типи доступу до системи (реєстратор, лікар, адміністратор)
		registrar	Розширення для реєстратора, з додатковими авторизаційними полями
2	Медичні об'єкти	patient	Особисті дані пацієнтів: ПІБ, дата народження, стать, алергії, контакти
		doctor	Дані про лікарів, їхню спеціалізацію, кабінет, статус
		appointment	Записи на прийом: пацієнт, лікар, час, дата, статус, кабінет
		medical_card	Номер картки, дата створення, пов'язаний пацієнт
3	Звіти та звернення	report	Звіти з вказаними категоріями, статусом, автором, описом
		report_category	Категорії звітів (медичні, адміністративні тощо)
		report_status	Стан звітів (чернетка, підтверджено, скасовано тощо)
		request	Запити та звернення користувачів
		status	Узагальнені статуси для звернень, записів, звітів

У системі реалізовано розгалужені міжтабличні зв’язки за допомогою зовнішніх ключів (FOREIGN KEY), що дозволяють контролювати цілісність даних і запобігати помилкам під час виконання транзакцій. Завдяки такій моделі база гарантує узгодженість усіх записів: зв'язки між пацієнтами й прийомами, користувачами та звітами, зверненнями й категоріями чітко формалізовані, що виключає появу непов’язаних або логічно хибних даних.

SQL-код створення таблиць, представлений на рис. 3.1, відображає реалізацію логічної моделі у вигляді структурованої бази даних з урахуванням типів даних, первинних та зовнішніх ключів, індексів і обмежень.

```sql
CREATE TABLE Role (
 role_name VARCHAR PRIMARY KEY,
 description TEXT
);
CREATE TABLE Registrar (
 user_id INT PRIMARY KEY,
 login VARCHAR,
 password VARCHAR,
 user_type VARCHAR,
 role_id VARCHAR REFERENCES Role(role_name)
);
CREATE TABLE Patient (
 patient_id INT PRIMARY KEY,
 full_name VARCHAR,
 birth_date VARCHAR,
 gender VARCHAR,
 medical_card_number INT,
 medical_history TEXT,
 allergies TEXT,
 blood_group VARCHAR,
 user_id INT REFERENCES Registrar(user_id)
);
CREATE TABLE Doctor (
 doctor_id INT PRIMARY KEY,
 full_name VARCHAR,
 specialization VARCHAR,
 office VARCHAR,
 user_id INT REFERENCES Registrar(user_id)
);
CREATE TABLE Medical_Card (
 card_number INT PRIMARY KEY,
 creation_date DATE,
 patient_id INT REFERENCES Patient(patient_id)
);
CREATE TABLE Appointment (
 appointment_id INT PRIMARY KEY,
 appointment_date DATE,
 appointment_time TIME,
 office VARCHAR,
 doctor_id INT REFERENCES Doctor(doctor_id),
 patient_id INT REFERENCES Patient(patient_id)
);
CREATE TABLE Report_Category (
 category_id INT PRIMARY KEY,
 category_name VARCHAR,
 description TEXT
);
CREATE TABLE Report_Status (
 status_id INT PRIMARY KEY,
 status_name VARCHAR
);
CREATE TABLE Report (
 report_id INT PRIMARY KEY,
 report_name VARCHAR,
 report_number INT,
 status VARCHAR,
 creation_date DATE,
 description TEXT,
 user_id INT REFERENCES Registrar(user_id),
 category_id INT REFERENCES Report_Category(category_id),
 status_id INT REFERENCES Report_Status(status_id)
);
CREATE TABLE Request (
 request_id INT PRIMARY KEY,
 user_id INT REFERENCES Registrar(user_id),
 category_id INT REFERENCES Report_Category(category_id),
 status_id INT REFERENCES Report_Status(status_id),
```

Рис. 3.1 SQL-код створення таблиць бази даних

Представлені SQL-інструкції є формальним втіленням логічної моделі даних. Вони засвідчують нормалізовану структуру, підтримку референційної цілісності та оптимізацію доступу до даних.

Розроблена база даних інформаційної системи медичної реєстратури сформована з урахуванням вимог до масштабованості, адаптивності та чіткої структурної організації. Її архітектура дозволяє ефективно зберігати, підтримувати та обробляти як медичні, так і адміністративні дані, забезпечуючи комплексне керування всіма ключовими процесами в межах функціоналу реєстратури.

В основу побудови покладено принципи гнучкого логічного моделювання, що передбачають логічну ізоляцію окремих підсистем для підвищення гнучкості й розширюваності. У табл. 3.6 систематизовано основні особливості архітектури з урахуванням практичних аспектів її реалізації. Застосовані підходи до структурування бази дозволяють досягти балансу між технічною надійністю,

функціональною гнучкістю та продуктивністю, необхідною для ефективного використання системи в умовах медичного закладу.

Таблиця 3.6

Особливості архітектури бази даних медичної інформаційної системи

№	Особливість	Опис реалізації
1	Розширюваність довідників	Категорії звітів, типи статусів та ролі користувачів винесено в окремі таблиці для динамічного оновлення
2	Гнучке управління медичними об'єктами	Структура дозволяє додавати нових пацієнтів, призначати лікарів, зберігати історію карток без дублювання
3	Підтримка навантажень	Чітка структура з первинними ключами, індексами та зовнішніми зв'язками забезпечує продуктивність системи
4	Сумісність та інтеграція	Наявність зовнішніх ключів дає змогу легко інтегрувати додаткові сервіси, зберігаючи логічну послідовність
5	Типізація та нормалізація	Поля мають типи VARCHAR, INT, DATE тощо, з обмеженнями NOT NULL, що забезпечує цілісність і уникає дублювання

База даних побудована відповідно до принципів логічного моделювання: ключові об'єкти, такі як «Пацієнт», «Лікар» і «Реєстратор», об'єднані через взаємопов'язані таблиці, включаючи appointment, medical_card, report і request. Такий підхід забезпечує чітке розмежування відповідальностей між сутностями, спрощує супровід і оновлення системи, а також дозволяє масштабувати функціональність без порушення цілісності даних.

Узагальнюючи, розроблена інформаційна база повністю відповідає вимогам сучасних медичних інформаційних систем. Вона забезпечує стабільність, продуктивність і логічну узгодженість усіх процесів, слугуючи надійним фундаментом для довготривалої експлуатації та подальшого розвитку інформаційної платформи.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка прикладної частини інформаційної системи медичної реєстратури потребує не лише реалізації заявлених функціональних можливостей, але й обґрунтованого вибору технологічного середовища, що дозволить досягти оптимального балансу між продуктивністю, зручністю у підтримці, масштабованістю та стабільністю програмного продукту.

У межах реалізації програмного забезпечення було використано середовище Microsoft Visual Studio [6], яке є провідним інструментом для розробки десктопних застосунків. Це середовище забезпечує гнучкий і зручний інтерфейс для проєктування, налагодження та тестування програм, а також має високу інтеграцію з платформою .NET, що дає змогу реалізовувати масштабовані рішення з використанням сучасних технологій керування даними.

Для створення графічного інтерфейсу користувача обрано технологію Windows Forms (WinForms), яка дозволяє швидко та ефективно реалізовувати інтуїтивно зрозумілі форми, орієнтовані на зручність у роботі персоналу медичного закладу [2]. Інтерфейс передбачає роботу з формами авторизації, перегляду та редагування даних пацієнтів, запису на прийом, створення звітів і керування обліковими записами.

У табл. 3.7 узагальнено переваги обраного технологічного стеку з точки зору реалізації прикладної частини системи. Обрана сукупність інструментів забезпечує розробку застосунку з високим рівнем інтеграції, надійності та зручності в реалізації інтерфейсних і логічних компонентів.

Основні компоненти інтерфейсу включають форми для авторизації, запису пацієнта, створення прийому, внесення медичних карток і перегляду звітності. Розроблені форми логічно пов’язані між собою, що дозволяє забезпечити узгоджений користувацький досвід та швидкий доступ до основних функцій.

Таблиця 3.7

Переваги вибраного середовища розробки

№	Компонент	Перевага використання
1	Visual Studio	Професійне середовище з підтримкою проєктів, налагодження та інтеграції з СУБД
2	Windows Forms (WinForms)	Простий у реалізації інтерфейс, зручність графічного редагування форм, drag-and-drop
3	.NET Framework	Стабільна основа для масштабованих десктопних застосунків
4	Npgsql	Офіційна бібліотека для підключення до PostgreSQL із повноцінною підтримкою транзакцій

Реалізована архітектура також враховує можливість масштабування застосунку: у разі необхідності можна розширити функціонал без суттєвих змін у кодовій базі. Завдяки розділенню на окремі функціональні модулі (реєстрація, запис, звітність, доступ), система зберігає стабільність і гнучкість при подальшій модернізації.

Таким чином, застосування середовища Microsoft Visual Studio [6] у поєднанні з технологією WinForms та підтримкою PostgreSQL через бібліотеку Npgsql [3] дозволило створити стабільну, функціонально повну та зручну в користуванні прикладну частину медичної інформаційної системи. Така архітектура є адаптивною до майбутніх змін і відповідає сучасним вимогам до програмного забезпечення у сфері охорони здоров’я.

3.4 Алгоритмізація та програмування програмних модулів

У процесі розробки прикладного програмного забезпечення для інформаційної системи медичної реєстратури було реалізовано низку функціональних модулів, кожен з яких відповідає за окрему частину логіки взаємодії користувача із системою. Модульна побудова програмного продукту дала змогу суттєво спростити процес розробки, підвищити масштабованість, полегшити супровід та забезпечити стабільну роботу системи.

Усі модулі реалізовано з використанням технології Windows Forms на базі UserControl-компонентів. Кожен модуль включає як візуальну частину, так і

відповідну бізнес-логіку. Центральним елементом взаємодії виступає головна форма (Form2.cs), яка координує перемикання між модулями, обробку ролей користувачів і забезпечує логічну навігацію між функціональними панелями.

Для систематизації реалізованих модулів у табл. 3.8 подано їх опис разом із функціональними можливостями та відповідними класами.

Таблиця 3.8

Функціональні модулі системи

№	Назва модуля	Основні можливості	Відповідні класи
1	Реєстрація пацієнта	Введення ПІБ, дати народження, медичного номера, алергій; перевірка обов'язкових полів	NewPatientControl.cs
2	Запис на прийом	Вибір лікаря, дати й часу; перевірка конфліктів запису, збереження у таблицю	AppointmentControl.cs
3	Перегляд та редагування пацієнтів	Фільтрація, пошук, редагування записів, відображення медичної інформації	PatientListControl.cs, View/EditForms
4	Генерація звітів у PDF	Створення медичних/адміністративних звітів, збереження на диск, підпис і дата	GeneratePDF.cs
5	Додавання користувачів/лікарів	Створення облікових записів, прив'язка до ролі, визначення спеціалізації	AddUserControl.cs, AddDoctorControl.cs
6	Управління звітами	Формування звітів за категоріями і статусами, опис, фіксація дати	ReportControl.cs
7	Управління графіками лікарів	Встановлення днів/годин прийому, кабінетів, збереження розкладу	ScheduleControl.cs
8	Друк талонів	Генерація документів на прийом з QR/штрихкодом, друк через шаблони	PrintControl.cs

Представлена модульна структура дозволяє ізолювати функціональність кожного компонента, що спрощує оновлення, розширення та підтримку системи без впливу на її загальну стабільність [20].

Архітектура прикладного рівня реалізована за принципами поділу на чотири логічні шари. У табл. 3.9 наведено опис кожного з них.

Таблиця 3.9

Рівні архітектури прикладної частини

№	Рівень системи	Функціональне призначення
1	Рівень представлення (UI)	Форми Windows Forms та UserControl-компоненти для взаємодії з користувачем
2	Логічний рівень (Business Logic)	Обробка даних, перевірка вхідних значень, фільтрація, ізольовані методи логіки
3	Рівень доступу до даних	Реалізація SQL-запитів через Npgsql для роботи з таблицями системи
4	Рівень бази даних (Data Layer)	Постгрес-база з індексами, обмеженнями, зв'язками, що гарантує цілісність та масштабованість

Поділ на рівні забезпечує структурну впорядкованість, логічну узгодженість коду й можливість незалежного розвитку кожної компоненти системи (рис. 3.2).

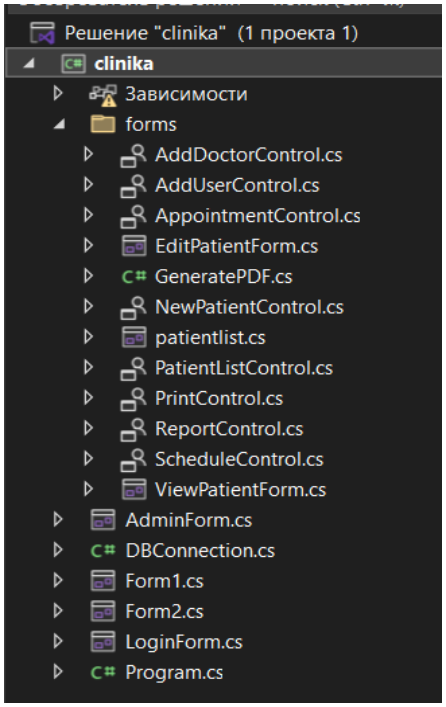


Рис. 3.2 Структурна схема модулів прикладної частини

Поділ на рівні забезпечує структурну впорядкованість, логічну узгодженість коду й можливість незалежного розвитку кожної компоненти системи.

Таким чином, архітектурно модульний підхід у розробці інформаційної системи медичної реєстратури дозволив створити ефективну, стабільну та розширювану платформу для обробки медичних і адміністративних даних. Поділ логіки за рівнями, ізольованість компонентів і чітка реалізація ролей забезпечують надійність та зручність у підтримці й розвитку системи.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Після завершення розробки прикладної частини інформаційної системи медичної реєстратури було проведено комплексне тестування, метою якого стало підтвердження стабільної роботи програми, відповідності функціональним вимогам та забезпечення надійної взаємодії з базою даних.

Основне завдання тестування полягало у виявленні логічних помилок, збоїв у роботі окремих модулів та перевірці коректності інтерфейсу користувача. Було сформульовано такі основні цілі перевірки системи, які узагальнено у табл. 4.1.

Таблиця 4.1

Цілі тестування інформаційної системи

№	Напрям перевірки	Суть перевірки
1	Валідація даних	Перевірка правильності введення персональних і медичних даних, їх збереження у БД
2	Інтерфейс користувача	Аналіз працездатності кнопок, форм, полів введення та відображення результатів
3	Обробка виключних ситуацій	Симуляція некоректного введення та перевірка реакції системи на порушення цілісності даних
4	Стабільність взаємодії з PostgreSQL	Перевірка виконання SQL-запитів на додавання, оновлення та видалення даних
5	Порівняння очікуваної і фактичної поведінки	Аналіз відповідності результатів дій користувача заданим вимогам у ключових модулях

Зазначені напрями дозволили всебічно перевірити роботу системи, включаючи її функціональну, логічну та технічну складову.

У ході тестування було застосовано кілька методик, наведені в узагальненій табл. 4.2.

Таблиця 4.2

Застосовані методи тестування

№	Метод тестування	Призначення
1	Модульне тестування	Перевірка окремих функцій: NewPatientControl, AppointmentControl, GeneratePDF
2	Інтеграційне тестування	Взаємодія між модулями: реєстрація пацієнта → запис на прийом
3	Системне тестування	Перевірка повного сценарію використання системи в умовах, наближених до реальних
4	UI-тестування	Аналіз зручності, інтуїтивності інтерфейсу, логіки форм, коректності валідації та повідомлень

Застосування різних методів тестування дозволило комплексно перевірити як окремі модулі системи, так і їхню взаємодію, що забезпечує всебічну оцінку стабільності, функціональності та зручності експлуатації програмного забезпечення в умовах, наближених до реального середовища медичного закладу.

З метою практичної перевірки ключових функцій системи були розроблені та проведені тестові сценарії, які охоплюють основні процеси роботи реєстратури. Нижче наведено опис кожного із сценаріїв разом із вхідними даними, очікуваним результатом, візуалізацією та висновком щодо виконання.

Сценарій 1. Реєстрація нового пацієнта. Однією з базових функцій інформаційної системи є реєстрація пацієнта, що реалізована у вигляді окремого модуля з перевіркою обов’язкових полів та збереженням даних у базі. У рамках тестування цього сценарію було здійснено введення стандартного набору

особистих даних: ім’я, прізвище, по батькові, адреса електронної пошти, номер телефону, дата народження, номер медичної картки та стать. Після заповнення всіх полів і підтвердження введеної інформації система автоматично зберегла дані в таблицю patient, що підтверджує коректну роботу механізмів валідації та взаємодії з базою даних.

На рисунку 4.1 зображено форму додавання нового пацієнта, яка була використана під час тестування. Інтерфейс модуля дозволяє зручно вводити інформацію, а всі поля супроводжуються підказками й валідаційними повідомленнями, що сприяє зменшенню ймовірності помилок під час введення.

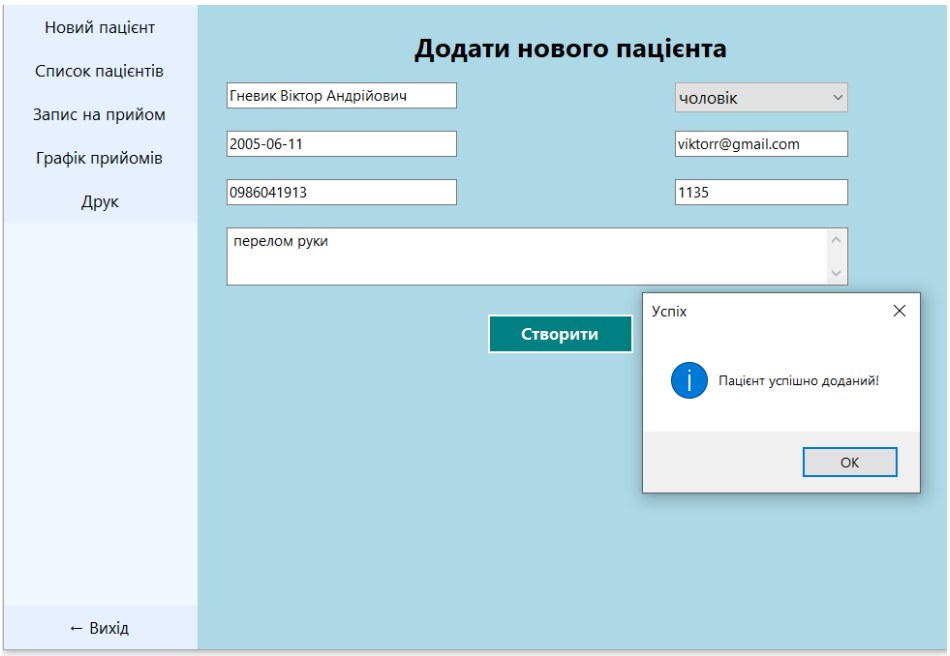


Рис. 4.1- Додаваня пацієнта

Тест успішно пройдено. Сценарій реалізовано успішно. Дані були збережені в базі, система відпрацювала стабільно, форма відображалася коректно, і користувач отримав повідомлення про успішну реєстрацію. Тест підтвердив функціональну готовність модуля до експлуатації в реальних умовах.

Сценарій 2. Робота зі списком пацієнтів. Наступним кроком тестування стало перевірення функціональності модуля перегляду, редагування та видалення пацієнтів. У рамках цього сценарію було імітовано стандартну роботу

реєстратора з базою: здійснено пошук пацієнта за такими критеріями, як унікальний ідентифікатор (ID), прізвище, ім’я, дата народження, стать та номер медичної картки. Після цього було проведено повний цикл взаємодії з обраним записом: відкриття для перегляду, редагування даних та їх подальше видалення.

На рисунку 4.2 представлено інтерфейс редагування даних пацієнта. У формі користувач має змогу змінювати основні персональні дані пацієнта, включаючи контактну інформацію, медичні відомості та пов’язану картку. Система перевіряє обов’язкові поля та автоматично зберігає зміни до бази даних після підтвердження.

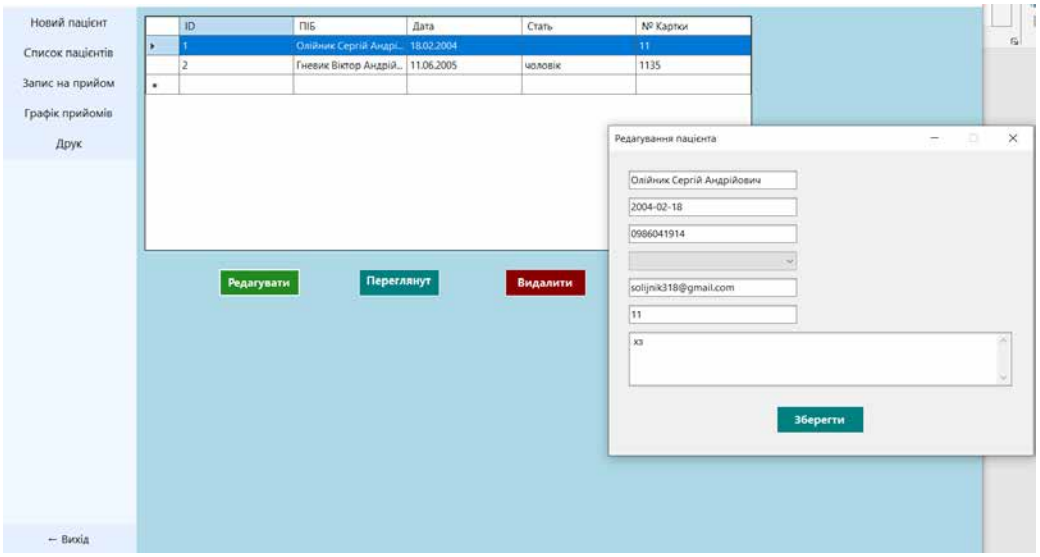


Рис. 4.2 – Редагування пацієнта

Редагування інформації проходить коректно, усі внесені зміни відображаються в таблиці patient, а перевірка на заповнення обов’язкових полів працює належним чином.

Далі на рисунку 4.3 продемонстровано форму перегляду повної картки пацієнта. Вона надає доступ до всієї зареєстрованої інформації без можливості внесення змін, що особливо корисно для адміністративного перегляду або лікарського консультування.

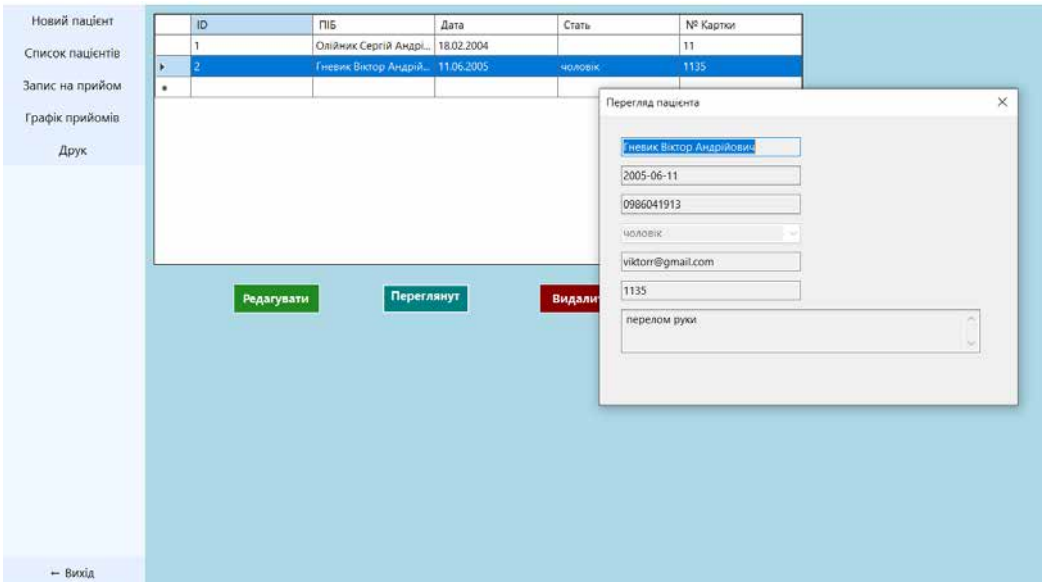


Рис. 4.3 – Перегляд пацієнта

Вивід інформації реалізований коректно, відображення усіх полів здійснюється повністю і з дотриманням форматування, що дозволяє зручно працювати з медичними даними.

На завершення сценарію, як видно на рисунку 4.4, було перевірено механізм видалення запису. Після підтвердження дії пацієнт був успішно вилучений із таблиці patient, а система відобразила повідомлення про успішну операцію.

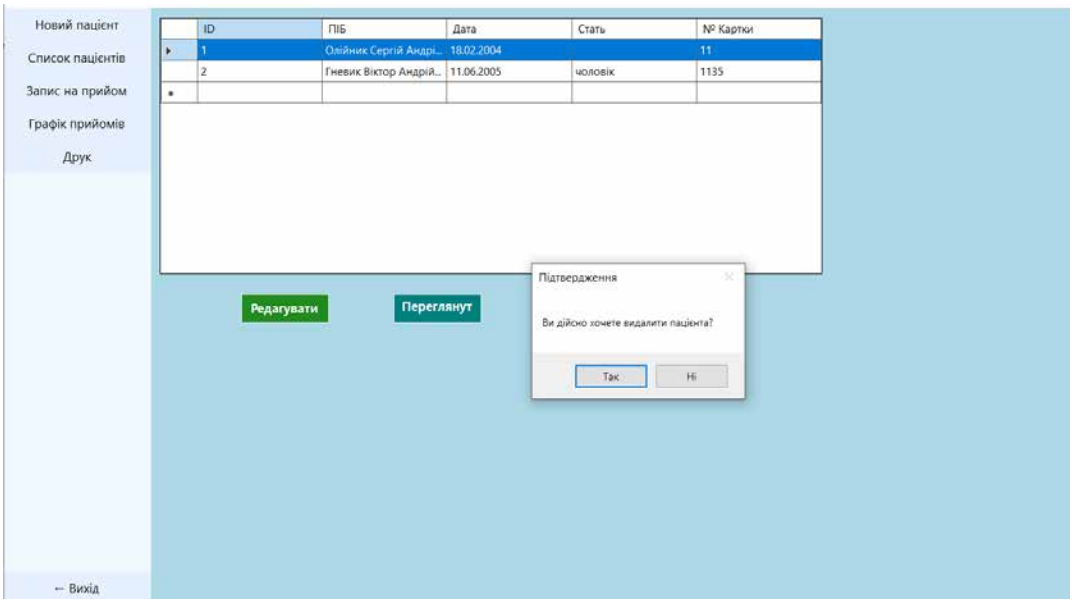


Рис. 4.4 – Видалення пацієнта

Функція видалення працює стабільно, об'єкти коректно видаляються з бази, а повторний перегляд запису вже неможливий, що підтверджує цілісність взаємодії з базою даних.

Усі ключові операції – перегляд, редагування та видалення інформації про пацієнтів – реалізовані повністю й без помилок. Інтерфейс модуля зручний у використанні, а система коректно обробляє всі дії, забезпечуючи стабільну взаємодію з базою даних.

Сценарій 3. Запис пацієнта на прийом. У межах тестування третього сценарію було перевірено функціональність модуля запису пацієнта на прийом до лікаря. Даний модуль є одним із ключових в інформаційній системі медичної реєстратури, оскільки він забезпечує щоденну роботу з формуванням медичних прийомів.

На початковому етапі тестування користувач вніс усі необхідні вхідні дані: прізвище, ім’я та по батькові пацієнта, вибраного лікаря, дату та час прийому, а також номер кабінету. Після цього була натиснута кнопка «Записати», яка ініціює обробку введених значень та передачу їх у відповідну таблицю бази даних.

На рисунку 4.5 наведено приклад реалізованої форми запису на прийом. Інтерфейс забезпечує зручне введення параметрів запису завдяки випадаючим спискам і календарному вибору дати. Також реалізовано валідацію на дублювання записів, що дозволяє уникнути конфліктів у розкладі лікарів.

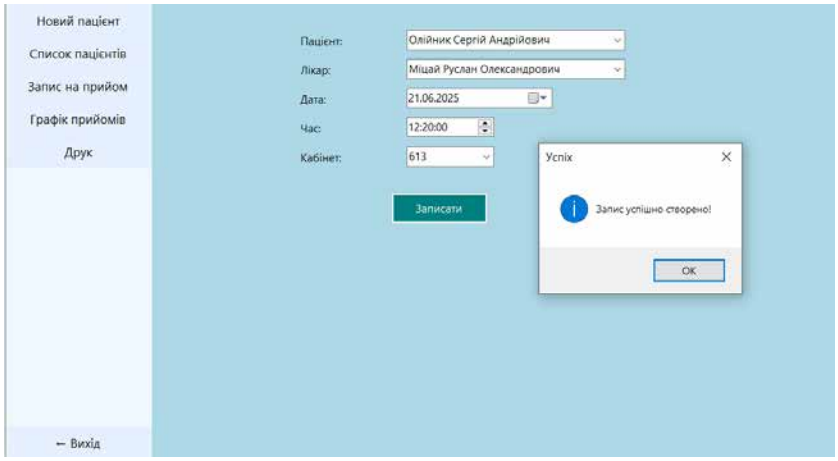


Рис. 4.5 – Форма запису на прийом

Модуль функціонує стабільно. Дані успішно передаються до таблиці appointment, конфлікти в графіку прийомів фіксуються і не допускаються, а інтерфейс чітко реагує на дії користувача. Усі перевірки на обов'язковість полів спрацьовують належним чином, що свідчить про коректну реалізацію бізнес-логіки процесу запису.

Сценарій 4. Створення PDF-направлення для прийому. Наступним етапом тестування стало перевірення модуля генерації направлень у форматі PDF. Цей функціонал має практичне значення для документального супроводу пацієнтів, оскільки дозволяє швидко сформувати офіційне направлення на медичний прийом.

На початку сценарію користувач виконав пошук пацієнта за прізвищем, ім'ям і по батькові через інтерфейс форми. Після ідентифікації потрібного запису була натиснута кнопка «Друкувати», яка ініціювала автоматичне формування PDF-файлу з інформацією про прийом – включаючи ПІБ пацієнта, дату, час, кабінет і лікаря.

На рисунку 4.6 продемонстровано інтерфейс формування направлення. У результаті тестування було підтверджено, що файл створюється коректно, зберігається у відповідну директорію на диску користувача, а також містить усі необхідні реквізити (дата створення, підпис лікаря, заголовок документу).

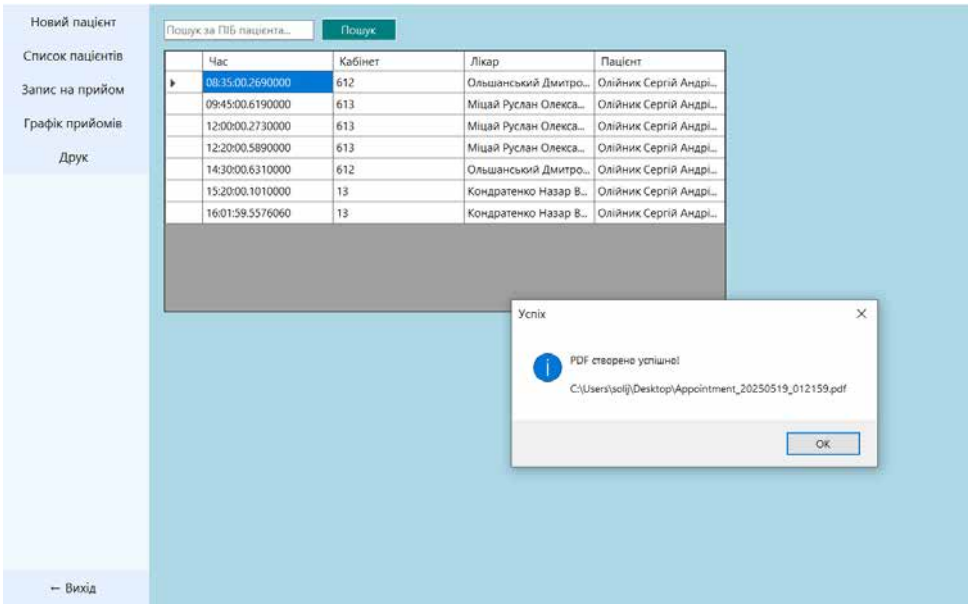


Рис. 4.6 – Створення PDF направлення

Генерація PDF-направлення реалізована успішно. Документ формується автоматично після натискання відповідної кнопки, зберігається у зазначене місце, а його вміст відповідає медичним стандартам оформлення. Модуль працює стабільно й забезпечує швидкий друк або архівування медичних документів.

Сценарій 5. Введення некоректного логіна або пароля. Завершальним етапом тестування стало моделювання ситуації некоректної авторизації користувача. Метою сценарію було перевірити надійність механізму автентифікації та його здатність коректно реагувати на помилки введення.

У межах цього тесту було змодельовано спробу входу до системи з помилковими обліковими даними: введено некоректний логін та/або пароль від імені користувача з роллю «реєстратор». Система обробила введені дані та перевірила їх на відповідність записам у базі.

Результати перевірки відображено на рисунку 4.7, де видно повідомлення про помилку входу. Система видала інформативне вікно з текстом "Невірний логін або пароль", що дозволяє користувачеві оперативно скоригувати введені дані.

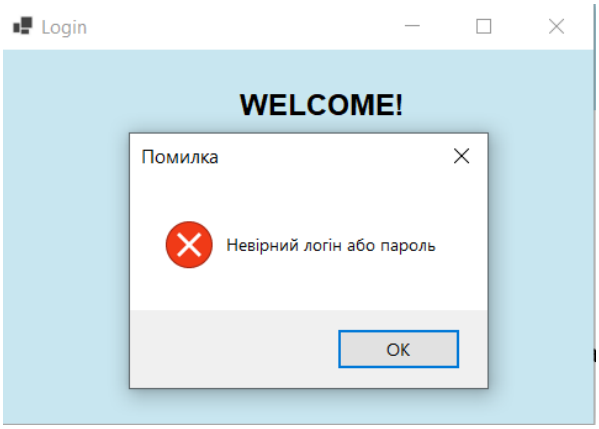


Рис. 4.7 – Повідомлення при спробі входу з невірним логіном або паролем

Механізм авторизації працює надійно. Неправильні спроби входу належним чином обробляються, система не допускає користувача до подальшої роботи, а повідомлення є чітким і зрозумілим. Це підтверджує функціональну безпеку та користувацьку зручність реалізованого механізму автентифікації.

Тестування проводилося з використанням доступного інструментарію, що наведено у табл. 4.3.

Таблиця 4.3

Інструменти, використані для тестування

№	Інструмент	Призначення
1	Visual Studio (режим налагодження)	Пошук логічних помилок, покрокове тестування модулів
2	MessageBox	Інформування користувача про помилки або успішність операцій
3	PostgreSQL Console	Перевірка змін у базі даних після виконання SQL-запитів
4	Консольне логування	Виведення повідомлень про хід виконання функцій, перевірка взаємодії з БД

Використані інструменти забезпечили глибоке тестування на всіх рівнях логіки програми та її взаємодії з базою даних.

Результати проведеного тестування підтвердили стабільну та надійну роботу інформаційної системи медичної реєстратури. Усі функціональні модулі виконують свої задачі відповідно до вимог, інтерфейс є інтуїтивно зрозумілим, а взаємодія з базою даних відбувається без помилок. Програмне забезпечення готове до впровадження в реальне середовище експлуатації та забезпечує надійну підтримку процесів медичної реєстрації.

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільного функціонування розробленої інформаційної системи медичної реєстратури, яка об'єднує облік пацієнтів, запис на прийом, формування звітів та керування медичними даними, важливо врахувати технічні характеристики середовища її експлуатації. Ретельне дотримання вимог до апаратного та програмного забезпечення дозволяє забезпечити безперебійну роботу, захищений доступ і високу продуктивність при виконанні повсякденних операцій.

Технічні вимоги умовно поділяються на два рівні: мінімальні (необхідні для встановлення та базового тестування системи) та рекомендовані (орієнтовані

на повноцінне використання в умовах медичних установ). У табл. 4.4 узагальнено вимоги до серверного обладнання, необхідного для запуску та підтримки роботи бази даних.

Таблиця 4.4

Вимоги до серверного обладнання

№	Компонент	Мінімальні вимоги	Рекомендовані параметри
1	Процесор	2 ядра, 2.0 ГГц	4 ядра, від 2.5 ГГц
2	Оперативна пам’ять	4 ГБ	8–16 ГБ
3	Накопичувач	20 ГБ SSD	50 ГБ SSD з урахуванням резервного копіювання
4	Мережа	100 Мбіт/с	1 Гбіт/с
5	ОС	Linux	Linux із налаштованим автоматичним бекапуванням

Рекомендовані параметри забезпечують оптимальну продуктивність системи в умовах медичного закладу зі щоденним навантаженням.

Наступним кроком є визначення вимог до клієнтських комп’ютерів, зокрема тих, які використовуються реєстраторами. У табл. 4.5 наведено мінімальні та рекомендовані характеристики для таких робочих станцій.

Таблиця 4.5

Вимоги до клієнтських ПК (реєстратори)

№	Компонент	Мінімальні вимоги	Рекомендовані параметри
1	Процесор	2 ядра, 1.8–2.0 ГГц	4 ядра, ≥2.4 ГГц
2	Оперативна пам’ять	4 ГБ	8 ГБ або більше
3	Накопичувач	20 ГБ HDD/SSD	SSD ≥50 ГБ
4	Екран	1280×768	Full HD (1920×1080)
5	ОС	Windows 10+	Windows 10 Pro / 11 з оновленнями
6	Мережа	Ethernet / Wi-Fi 100 Мбіт/с	Ethernet 1 Гбіт/с
7	ПЗ	.NET Framework 4.7+, Npgsql	Повний пакет .NET + офісне ПЗ (MS Office або LibreOffice)

Використання рекомендованої конфігурації ПК дозволяє забезпечити швидкий запуск, комфортну взаємодію з програмою та мінімальне навантаження на системні ресурси.

Окрім апаратної складової, важливо врахувати вимоги до програмного забезпечення, яке необхідне як на стороні сервера, так і на клієнтських машинах. У табл. 4.6 та 4.7 наведено перелік необхідних компонентів.

Таблиця 4.6

Програмне забезпечення серверної частини

№	Компонент	Призначення / Версія
1	PostgreSQL	Основна СУБД (версія 17 або новіша)
2	pgAdmin	Графічний інтерфейс адміністрування БД (v6+)
3	Npgsql	ADO.NET-драйвер для зв'язку з PostgreSQL
4	SSH-сервер	Безпечне підключення до віддаленого сервера
5	Cron	Автоматизація резервного копіювання [9]

Таблиця 4.7

Клієнтське програмне забезпечення

№	Компонент	Призначення / Мінімальна версія
1	Windows 10/11	ОС для запуску застосунку
2	.NET Framework 4.7+	Платформа для виконання WinForms-застосунків
3	Npgsql (клієнт)	Драйвер для взаємодії з PostgreSQL
4	iTextSharp / PDFSharp	Генерація звітів у PDF-форматі
5	Антивірусне ПЗ	Захист робочих станцій (наприклад, Windows Defender)

Для стабільного доступу до системи важливо забезпечити надійне мережеве з'єднання між клієнтськими ПК та сервером. У табл. 4.8 наведено ключові вимоги до мережевого середовища.

Таблиця 4.8

Мережеве забезпечення

№	Компонент	Вимоги
1	2	3
1	Доступ до сервера	Стабільне інтернет-з’єднання з можливістю віддаленого доступу до СУБД
2	Пропускна здатність	Рекомендовано: ≥10 Мбіт/с

Таблиця 4.8 (продовження)

1	2	3
3	Надійність каналу	Бажаний uptime – не менше 99% протягом робочого часу
4	Захист з’єднання	Наявність фаєрволів, обов’язкова автентифікація, шифрування з’єднання (SSL/TLS)

Виконання наведених вимог гарантує безпечний обмін даними, надійне підключення до бази даних та збереження її цілісності.

Для гарантування збереження медичних даних та запобігання несанкціонованому доступу передбачено низку заходів, наведених у табл. 4.9.

Таблиця 4.9

Комплекс заходів безпеки

№	Захід	Призначення
1	Резервне копіювання	Відновлення даних у разі збою
2	Антивірусний захист	Виявлення шкідливих програм, захист робочих станцій
3	Шифрування мережевого з’єднання	Безпечна передача даних через SSL/TLS
4	Рольовий контроль доступу	Обмеження функціоналу відповідно до ролі користувача (реєстратор, адміністратор)

Впровадження наведених заходів дозволяє суттєво знизити ризики порушення безпеки, втрати даних і забезпечити відповідність вимогам медичних інформаційних систем.

Технічні вимоги до інформаційної системи розроблені з урахуванням особливостей медичного середовища, необхідності захисту даних і зручності в користуванні. Помірні апаратні параметри, підтримка популярних ОС і мережевих протоколів роблять систему доступною до впровадження навіть в умовах обмежених ресурсів, зберігаючи при цьому високу надійність і безпечність експлуатації.

4.3 Склад інсталяційного пакету

Для забезпечення ефективного розгортання інформаційної системи медичної реєстратури на робочих місцях було сформовано інсталяційний пакет, створений за допомогою інструментів Visual Studio. Його структура передбачає автоматизоване встановлення з усіма необхідними бібліотеками, компонентами та конфігураційними файлами. Це мінімізує ризик помилок при розгортанні, полегшує оновлення й не потребує спеціальної підготовки персоналу.

Застосування шаблону публікації у Visual Studio, зокрема технології ClickOnce, дозволило згенерувати інсталятор, який підтримує автоматичне оновлення, створення ярликів і перевірку наявності необхідних компонентів середовища виконання. У результаті було сформовано кілька основних елементів пакету:

- виконуваний файл clinika.exe,
- інсталятор setup.exe,
- каталог Application Files з усіма залежностями.

Процес підготовки пакету включав налаштування параметрів публікації, вибір шляху розгортання, автоматичну генерацію структури й, за потреби, підпис сертифікатом безпеки (табл.4.10).

Таблиця 4.10

Основні компоненти інсталяційного пакету

№	Компонент	Призначення
1	setup.exe	Ініціалізація процесу встановлення, перевірка залежностей, створення ярликів
2	clinika.exe	Основний виконуваний файл системи, запуск інтерфейсу для роботи з пацієнтами та лікарями
3	Application Files	Каталог зі всіма бібліотеками, ресурсами, конфігураціями та підпапками для кожної версії
4	*.deploy-файли	Файли з обмеженим прямим запуском, створені автоматично системою ClickOnce
5	*.manifest	XML-документ з інформацією про структуру пакету, версію, залежності та дозволи
6	Файли ресурсів	Іконки, зображення, шаблони звітів, текстові файли, необхідні для UI

Усі складові пакету взаємодіють між собою як єдиний інструмент інсталяції та подальшого оновлення системи без участі користувача в технічних налаштуваннях.

До складу інсталяційного дистрибутива також входять службові файли конфігурації, що містять параметри підключення до бази даних, правила логування та маршрути оновлень. Зокрема:

- clinika.application – документ, який описує правила інсталяції, дозволи доступу, джерело оновлень;
- app.config / clinika.exe.config – зберігає параметри з'єднання з PostgreSQL, шляхи до шаблонів, локалізацію;
- лог-файли інсталяції – журнал подій для діагностики у разі збоїв.

Для забезпечення безпомилкової установки рекомендовано запускати файл setup.exe від імені адміністратора. Це дозволить правильно створити ярлики, записати параметри в системний реєстр та надати програмі всі необхідні дозволи. Попередньо слід переконатися, що на ПК встановлено .NET Framework 4.7.2 або новішу версію, а також, за потреби, Visual C++ Redistributable.

ClickOnce як технологія інсталяції надає низку переваг:

- мінімізація дій користувача: запуск в один клік,
- автоматичне включення всіх необхідних бібліотек,
- підтримка централізованого оновлення,
- перевірка цілісності файлів,
- можливість цифрового підпису пакету.

Сформований інсталяційний пакет повністю відповідає вимогам до зручного, швидкого та безпечного розгортання програмного забезпечення на робочих місцях медичних закладів. Його структура дозволяє виконувати встановлення без залучення IT-спеціалістів, підтримує автоматичне оновлення й забезпечує надійність взаємодії з усіма компонентами системи.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було повністю реалізовано повний цикл розробки програмного забезпечення для автоматизованого робочого місця працівника реєстратури медичного закладу. Від вихідного аналізу предметної області до практичної реалізації прикладного застосунку та проведення тестування, результати роботи продемонстрували відповідність заявленим цілям, функціональним вимогам і технічним стандартам.

У рамках системного аналізу було досліджено поточний стан організації облікових і адміністративних процесів у медичних установах, виявлено типові інформаційні потоки, функціональні ролі користувачів і загрози, пов’язані з ручним введенням даних. Побудовано UML-моделі (діаграми прецедентів, послідовності, активностей, класів і кооперацій), що стали основою для архітектурного та програмного проєктування системи [10].

Було створено нормалізовану базу даних із підтримкою зв’язків, цілісності та масштабованості. Для її реалізації обрано PostgreSQL як стабільну та надійну СУБД. Архітектура програмного забезпечення побудована на основі модульного принципу з поділом на рівні: інтерфейс, бізнес-логіка, доступ до даних і фізичне зберігання. Основними інструментами розробки виступили Visual Studio, C#, WinForms, бібліотеки Npgsql та iTextSharp.

У прикладній частині реалізовано повний набір функціональних модулів: реєстрація та облік пацієнтів, створення записів на прийом, керування користувачами та графіками, формування звітності у PDF-форматі, друк талонів і журналювання дій користувачів. Проведене комплексне тестування підтвердило працездатність системи, її відповідність вимогам та стійкість до типових помилок користувача.

Сформовано інсталяційний пакет із підтримкою автоматичного встановлення та оновлення. Програмне забезпечення успішно адаптоване до

вимог типового клієнт-серверного середовища, з можливістю подальшої інтеграції з іншими медичними платформами (зокрема eHealth).

Результатом проєкту стала стабільна, функціональна та масштабована інформаційна система, яка дозволяє автоматизувати роботу працівників реєстратури, підвищити ефективність обслуговування пацієнтів, зменшити навантаження на персонал і забезпечити безпечну обробку чутливої медичної інформації. Робота підтверджує набуті компетентності у сфері розробки прикладного ПЗ, моделювання систем і побудови баз даних, що дозволяє розглядати її як готову до впровадження в реальних умовах медичного закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

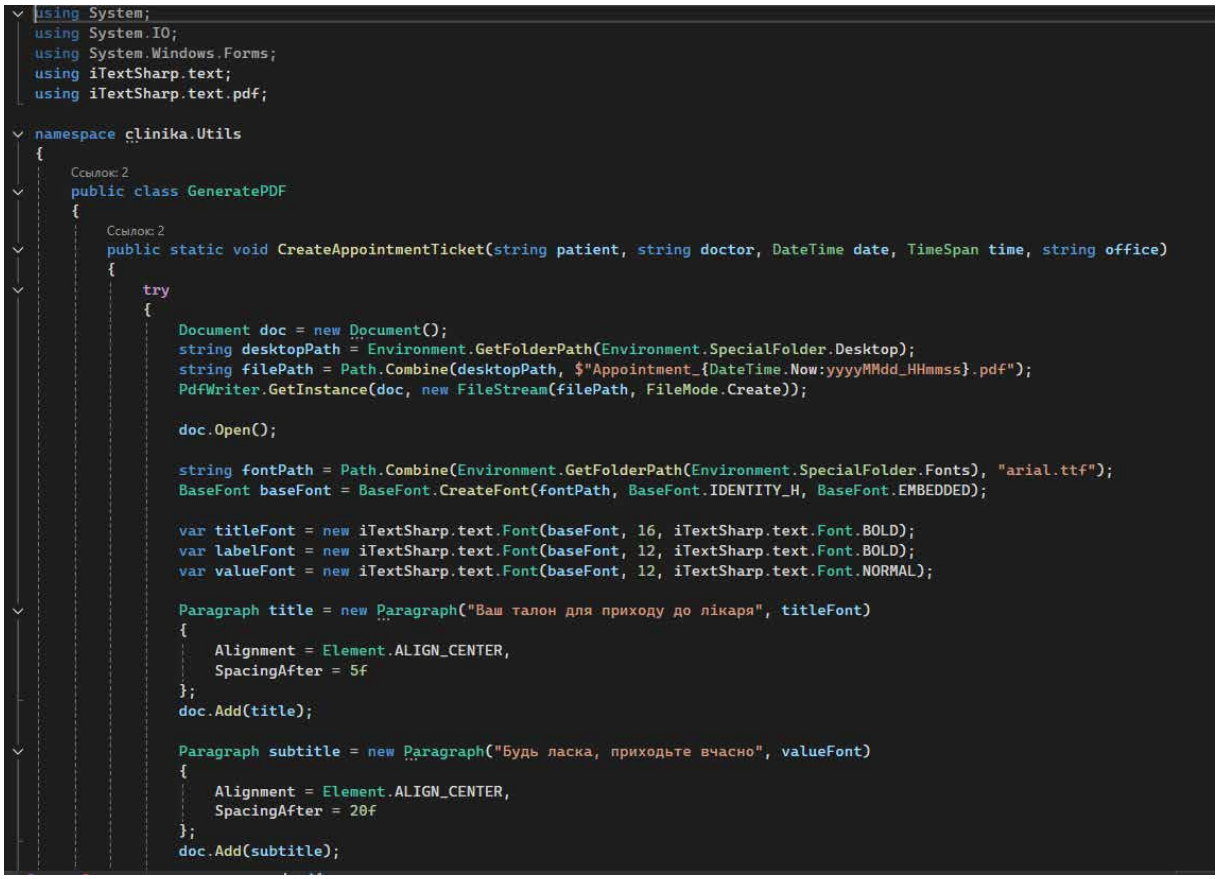
1. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ – Дата звернення: 05.05.2025.
2. Microsoft Docs. WinForms documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/ – Дата звернення: 05.05.2025.
3. iTextSharp PDF Library Documentation [Електронний ресурс]. – Режим доступу: https://itextpdf.com/en/resources/documentation – Дата звернення: 25.04.2025.
4. Npgsql: .NET Data Provider for PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.npgsql.org/ – Дата звернення: 07.05.2025.
5. Реляційні бази даних: структурні засади та сучасні підходи [Електронний ресурс]. – Режим доступу: https://rdb.dp.ua/uk/chapter_03 – Дата звернення: 06.05.2025.
6. Microsoft Visual Studio. Documentation [Електронний ресурс]. – Режим доступу: https://visualstudio.microsoft.com/ – Дата звернення: 05.05.2025.
7. .NET Framework Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/framework/ – Дата звернення: 15.04.2025.
8. C# Language Reference [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/ – Дата звернення: 25.04.2025.
9. Cron job scheduling [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Cron – Дата звернення: 25.04.2025.
10. UML 2.5 Specification [Електронний ресурс]. – Object Management Group. – Режим доступу: https://www.omg.org/spec/UML/2.5 – Дата звернення: 20.04.2025.
11. GitHub. Sample Medical Registration System in C# [Електронний ресурс]. – Режим доступу:

https://github.com/search?q=medical+registration+system+csharp – Дата звернення: 12.04.2025.

12. Закон України «Про захист персональних даних» №2297-VI від 01.06.2010 (в редакції 2024 р.) [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17 – Дата звернення: 04.05.2025.
13. ISO/IEC 27001:2022 Information security management systems – Requirements. – International Organization for Standardization. – Женева, 2022.
14. OWASP Foundation. Top 10 Security Risks for Web Applications [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ – Дата звернення: 12.05.2025.
15. Medoc. Інструкція користувача [Електронний ресурс]. – Режим доступу: https://medoc.ua/ua/support/manual – Дата звернення: 10.05.2025.
16. Doctor Eleks. Офіційна документація [Електронний ресурс]. – Режим доступу: https://eleks.com/products/doctor-eleks/ – Дата звернення: 10.05.2025.
17. MedWork. Модульна система медичного обліку [Електронний ресурс]. – Режим доступу: https://medwork.com.ua – Дата звернення: 11.05.2025.
18. eHealth. Офіційний портал [Електронний ресурс]. – Режим доступу: https://portal.ehealth.gov.ua/ – Дата звернення: 11.05.2025.
19. ISO/IEC/IEEE 42010:2011. Systems and software engineering – Architecture description. – Женева: ISO, 2016.
20. Ерік Ґамма, Річард Гелм, Ральф Джонсон, Джон Вліссідес. Шаблони проектування. Елементи повторно використовуваного об’єктно-орієнтованого програмного забезпечення. – К.: Діалектика, 2018. – 395 с.
21. Пономаренко В. Сучасні підходи до побудови інформаційних систем. // Інформаційні технології. – 2019. – №4. – С. 33–42.
22. Бублик Н. Програмна інженерія: базові поняття, процеси, стандарти. – К.: КНЕУ, 2021. – 272 с.
23. Кравченко І. Програмні продукти для автоматизації медичних реєстратур. // Системи управління, навігації та зв’язку. – 2020. – №1. – С. 98–104.

ДОДАТОК А

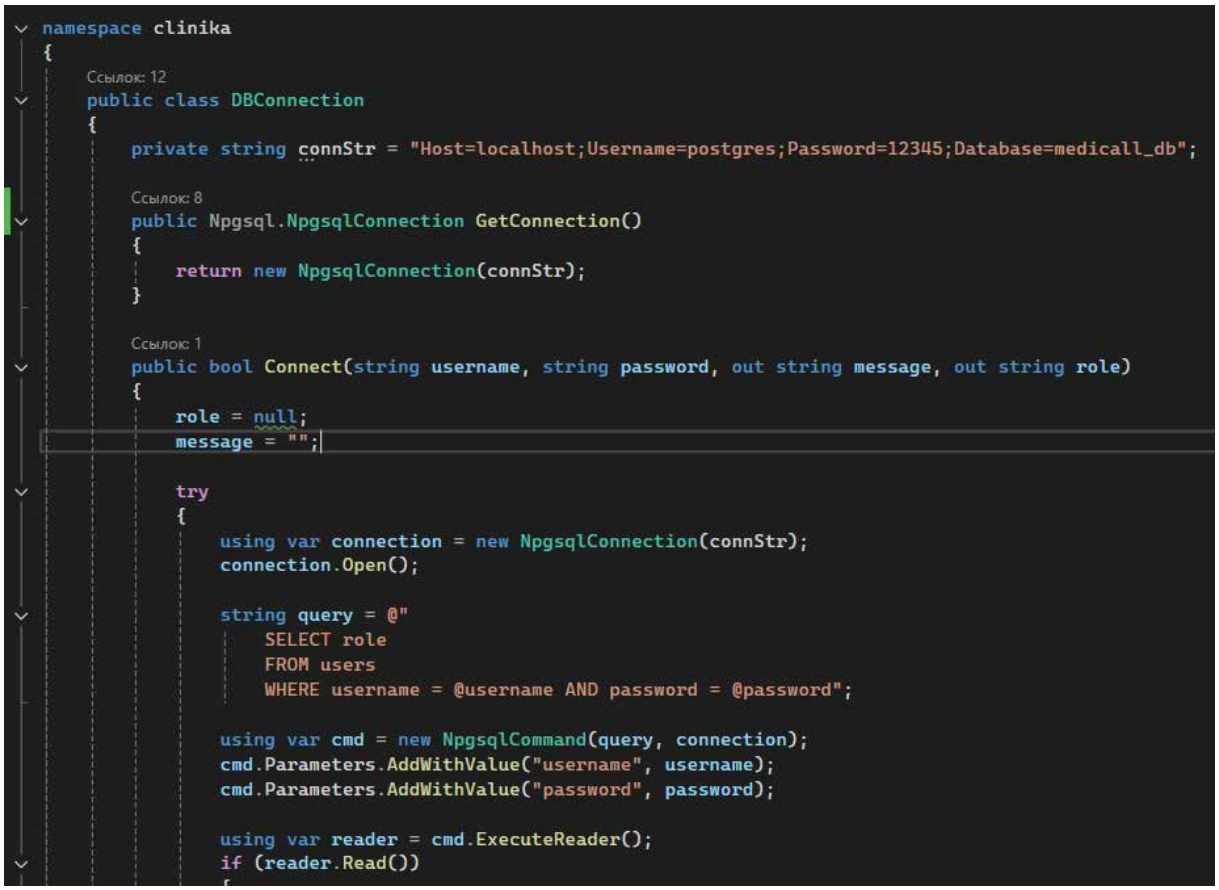
Фрагмент коду генератора ПДФ



```
using System;
using System.IO;
using System.Windows.Forms;
using iTextSharp.text;
using iTextSharp.text.pdf;

namespace clinika.Utils
{
    Ссылок: 2
    public class GeneratePDF
    {
        Ссылок: 2
        public static void CreateAppointmentTicket(string patient, string doctor, DateTime date, TimeSpan time, string office)
        {
            try
            {
                Document doc = new Document();
                string desktopPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
                string filePath = Path.Combine(desktopPath, $"Appointment_{DateTime.Now:yyyyMMdd_HHmmss}.pdf");
                PdfWriter.GetInstance(doc, new FileStream(filePath, FileMode.Create));

                doc.Open();

                string fontPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Fonts), "arial.ttf");
                BaseFont baseFont = BaseFont.CreateFont(fontPath, BaseFont.IDENTITY_H, BaseFont.EMBEDDED);

                var titleFont = new iTextSharp.text.Font(baseFont, 16, iTextSharp.text.Font.BOLD);
                var labelFont = new iTextSharp.text.Font(baseFont, 12, iTextSharp.text.Font.BOLD);
                var valueFont = new iTextSharp.text.Font(baseFont, 12, iTextSharp.text.Font.NORMAL);

                Paragraph title = new Paragraph("Ваш талон для приходу до лікаря", titleFont)
                {
                    Alignment = Element.ALIGN_CENTER,
                    SpacingAfter = 5f
                };
                doc.Add(title);

                Paragraph subtitle = new Paragraph("Будь ласка, приходьте вчасно", valueFont)
                {
                    Alignment = Element.ALIGN_CENTER,
                    SpacingAfter = 20f
                };
                doc.Add(subtitle);
```

ДОДАТОК Б

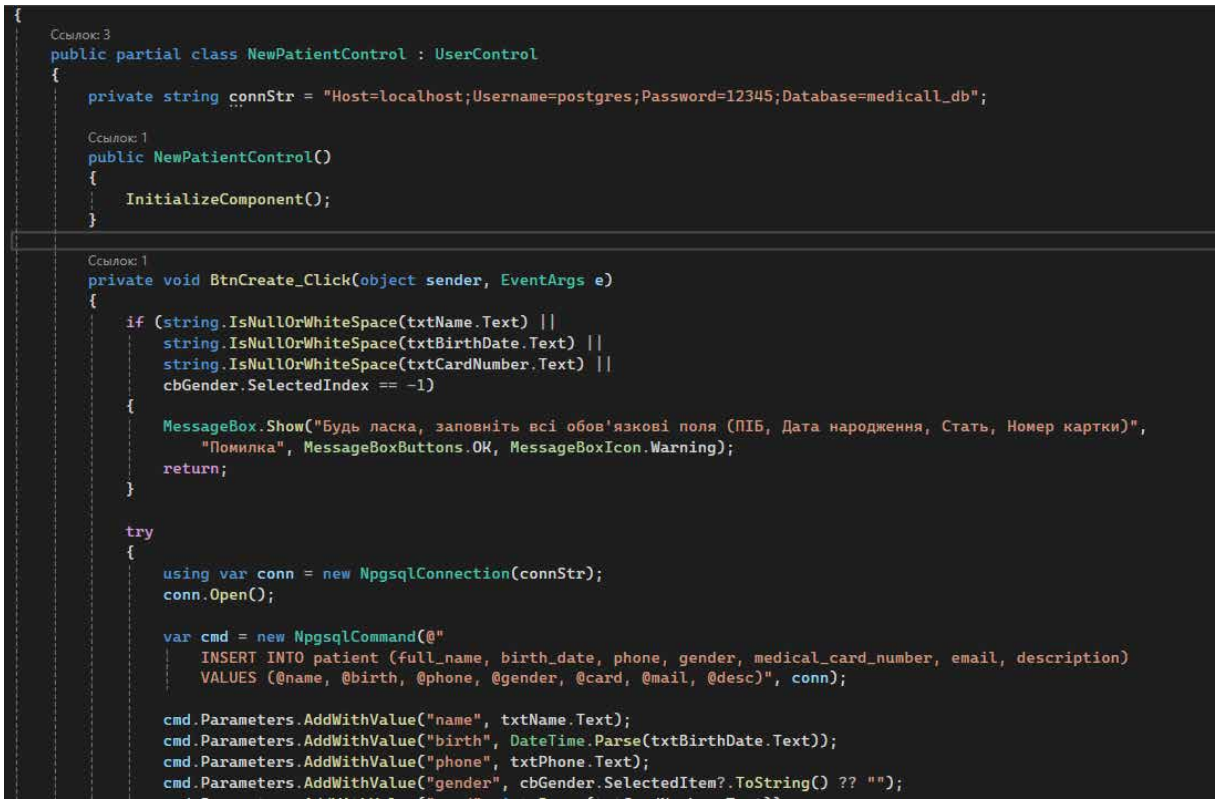
Фрагмент коду підключення до БД



```
namespace clinika
{
    Ссылок: 12
    public class DBConnection
    {
        private string connStr = "Host=localhost;Username=postgres;Password=12345;Database=medicall_db";

        Ссылок: 8
        public Npgsql.NpgsqlConnection GetConnection()
        {
            return new NpgsqlConnection(connStr);
        }

        Ссылок: 1
        public bool Connect(string username, string password, out string message, out string role)
        {
            role = null;
            message = "";

            try
            {
                using var connection = new NpgsqlConnection(connStr);
                connection.Open();

                string query = @"
                    SELECT role
                    FROM users
                    WHERE username = @username AND password = @password";

                using var cmd = new NpgsqlCommand(query, connection);
                cmd.Parameters.AddWithValue("username", username);
                cmd.Parameters.AddWithValue("password", password);

                using var reader = cmd.ExecuteReader();
                if (reader.Read())
```

ДОДАТОК В

Фрагмент коду додавання нового пацієнта



```
{
    Ссылок: 3
    public partial class NewPatientControl : UserControl
    {
        private string connStr = "Host=localhost;Username=postgres;Password=12345;Database=medicall_db";

        Ссылок: 1
        public NewPatientControl()
        {
            InitializeComponent();
        }

        Ссылок: 1
        private void BtnCreate_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(txtName.Text) ||
                string.IsNullOrWhiteSpace(txtBirthDate.Text) ||
                string.IsNullOrWhiteSpace(txtCardNumber.Text) ||
                cbGender.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, заповніть всі обов'язкові поля (ПІБ, Дата народження, Стать, Номер картки)",
                    "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            try
            {
                using var conn = new NpgsqlConnection(connStr);
                conn.Open();

                var cmd = new NpgsqlCommand(@"
                    INSERT INTO patient (full_name, birth_date, phone, gender, medical_card_number, email, description)
                    VALUES (@name, @birth, @phone, @gender, @card, @mail, @desc)", conn);

                cmd.Parameters.AddWithValue("name", txtName.Text);
                cmd.Parameters.AddWithValue("birth", DateTime.Parse(txtBirthDate.Text));
                cmd.Parameters.AddWithValue("phone", txtPhone.Text);
                cmd.Parameters.AddWithValue("gender", cbGender.SelectedItem?.ToString() ?? "");
```