Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка медичної системи обліку пацієнтів з використанням .Net

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Ільчій Б.В. (прізвище та ініціали)
Керівник	(підпис)	Петрик М.Р. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	Липак Г.І. (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота обсягом 59 сторінок містить 30 рисунків, 3 додатки, 15 джерел за списком літератури. Об’єктом дослідження є медичні інформаційні системи як інструмент цифрової трансформації закладів охорони здоров’я.

Ключові слова: медична система, WPF, C#, база даних, EF Core, реєстратор, лікар, електронна медична картка, прийом, епізод, ICPC2, ICD10.

Мета роботи – розробка десктопного застосунку для автоматизації процесів взаємодії між медичним персоналом у межах одного медичного закладу. У процесі розробки використано методи об’єктно-орієнтованого аналізу, структурне моделювання з використанням UML-діаграм, юніт- та інтеграційне тестування. Застосовано технології C#, платформа .NET 8.0, Windows Presentation Foundation (WPF), MySQL та Entity Framework Core.

Розроблений застосунок реалізує функціонал створення, редагування та перегляду прийомів пацієнтів, ведення епізодів захворювання, обробки діагнозів за класифікаціями ICPC-2 та ICD-10, взаємодії між лікарем і реєстратором. Система має модульну архітектуру, підтримує багаторівневі ролі користувачів і забезпечує зручну роботу в межах одного вікна з контекстними панелями.

Система підвищує ефективність ведення медичної документації, забезпечує швидкий доступ до історії пацієнта та зменшує навантаження на адміністративний персонал. Практична значимість полягає у створенні гнучкого інструменту, що може бути масштабований під потреби медичних закладів різного рівня. Подальший розвиток проєкту може включати веб-доступ до даних, мобільну версію та інтеграцію з eHealth.

ABSTRACT

The qualification paper consists of 59 pages and includes 30 figures, 3 appendices, and 15 sources listed in the bibliography. The object of the research is medical information systems as tools for the digital transformation of healthcare facilities.

Keywords: medical system, WPF, C#, database, EF Core, receptionist, doctor, electronic medical record, appointment, episode, ICPC2, ICD10.

The aim of the work is to develop a desktop application for automating workflows and interactions among medical personnel within a single healthcare institution. The development process included object-oriented analysis, structural modeling using UML diagrams, as well as unit and integration testing. Modern technologies were used, such as C#, .NET 8.0, Windows Presentation Foundation (WPF), MySQL, and Entity Framework Core.

The developed application provides functionality for creating, editing, and viewing patient appointments, managing episodes of illness, processing diagnoses based on ICPC-2 and ICD-10 classifications, and supporting interactions between doctors and receptionists. The system features a modular architecture, supports multi-level user roles, and ensures a streamlined user experience within a unified window interface with contextual panels.

The system improves the efficiency of medical documentation, provides quick access to patient histories, and reduces the workload on administrative staff. Its practical significance lies in delivering a flexible tool that can be scaled to meet the needs of various healthcare institutions. Future development may include a web-based version, mobile adaptation, and integration with national eHealth services.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації особливої актуальності набуває впровадження інформаційних технологій у сферу охорони здоров’я. Медичні інформаційні системи дозволяють не лише оптимізувати документообіг, а й забезпечують ефективну взаємодію між лікарями, пацієнтами та адміністративним персоналом. З огляду на значний обсяг даних, які щоденно обробляються в медичних закладах, виникає потреба у створенні програмного забезпечення, що забезпечувало б швидкий доступ до медичної інформації, її безпечне зберігання та обробку.

Огляд сучасних рішень, таких як Helsi та Medics, свідчить про те, що хоча ці платформи і охоплюють широкий функціонал, проте залишаються обмеження, зокрема щодо персоналізації ведення прийомів, гнучкого налаштування ролей працівників і деталізації медичних епізодів. Це обумовлює необхідність розробки власного рішення, орієнтованого на потреби конкретного закладу, з урахуванням можливостей масштабування, аналітики та гнучкої побудови прийомів лікарів.

Метою цієї дипломної роботи є проєктування та розробка десктопного застосунку для автоматизації роботи медичного персоналу, який дозволятиме здійснювати реєстрацію прийомів, ведення епізодів, перегляд медичних карток пацієнтів та ефективно взаємодіяти між підрозділами закладу. У ході реалізації було застосовано сучасні технології програмування, використано об’єктно-орієнтований підхід та розроблено ключові елементи системи згідно з принципами архітектурного моделювання.

РОЗДІЛ 1. ОГЛЯД СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз ринку та огляд аналогів

У контексті цифровізації медичної галузі в Україні все більшого поширення набувають системи електронного документообігу та електронної взаємодії між лікарями, пацієнтами й адміністративним персоналом. Серед провідних рішень на ринку варто відзначити два найбільш впливові сервіси – Helsi та Medics. Розглянемо короткий аналіз кожного з них, їх переваги та обмеження.

Аналог 1: Helsi

Helsi – це найбільша медична інформаційна система в Україні, яка працює з державними та приватними медичними закладами. Це незалежна компанія, яка працює без фінансування держави, але інтегрується з електронною системою охорони здоров'я, щоб забезпечити пацієнтам і лікарям доступ до всіх можливостей цифрової медицини в одному застосунку [1].

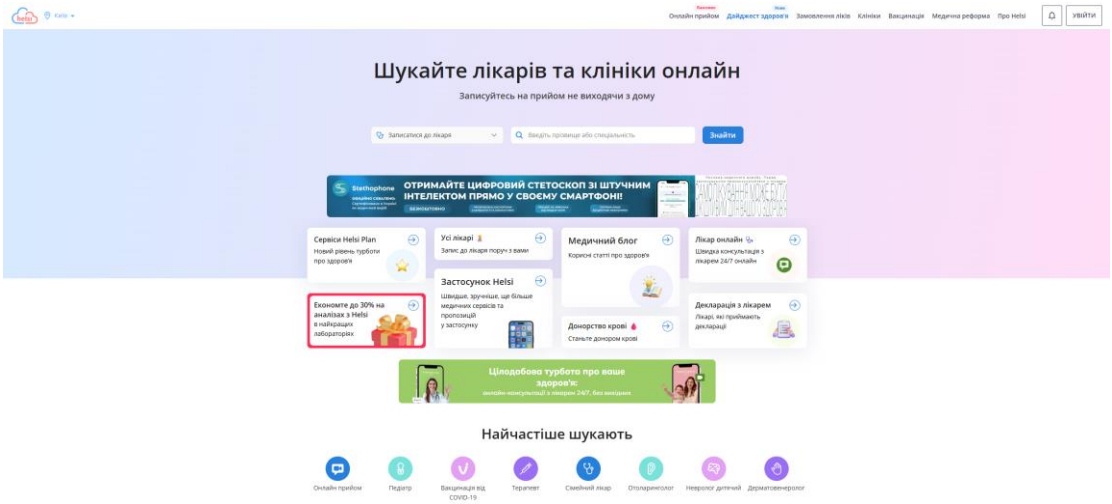

Рисунок 1.1 – Головна сторінка сервісу Helsi

Серед основних переваг Helsi слід виділити повну інтеграцію з eHealth, що забезпечує автоматизований документообіг між медичними установами. Крім того, система підтримує онлайн–запис пацієнтів, що знижує навантаження на

реєстратуру, а також має можливість формування електронних рецептів і направлень, що є суттєвою перевагою для лікарів.

Разом із цим система має низку обмежень. Зокрема, в Helsi майже відсутні засоби для гнучкого налаштування внутрішніх процесів медичного закладу, що ускладнює її використання в умовах невеликих приватних клінік або амбулаторій. Орієнтація системи переважно на великі установи створює додаткові бар'єри для адаптації. Окрім того, Helsi не забезпечує повноцінну підтримку ведення прийому на рівні епізодів, аналізу динаміки хвороби чи формулювання гіпотез – що обмежує клінічну гнучкість лікаря.

Аналог 2: Medics

Медична інформаційна система Medics – це сукупність практичних рішень, що роками формувались із спостережень за роботою медичних закладів. В основі лежить вільний доступ пацієнта до своїх медичних даних, а також вільний вибір лікаря, клініки чи послуги. У системі доступні відточені інструменти, що дозволять лікарям налагодити сервіс і підвищити рівень обслуговування пацієнтів [2].

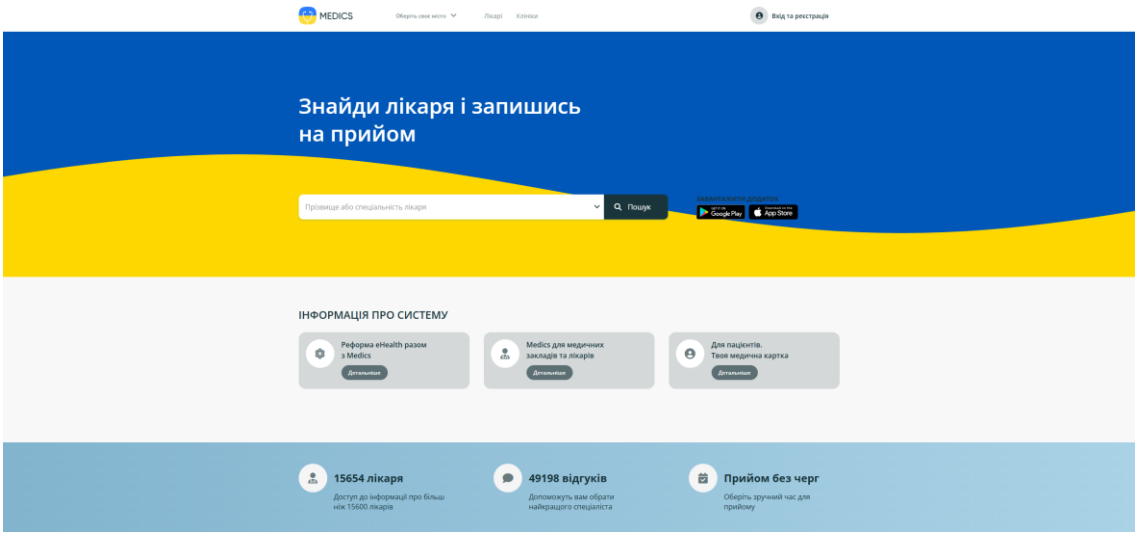

Рисунок 1.2 – Головна сторінка сервісу Medics

Серед переваг Medics варто відзначити зручний інтерфейс, адаптований під щоденну роботу лікаря, а також функціонал ведення електронних медичних карток, що полегшує облік історії хвороби. Окрему цінність становить доступ до

статистики прийомів та діяльності закладу, що може бути корисним для адміністраторів.

Втім, система має і суттєві обмеження. Medics не надає інструментів для глибокого клінічного аналізу – зокрема, в ній бракує засобів для дослідження динаміки захворювання, аналізу симптомів або епізодів лікування. Також відсутня підтримка спільної роботи між фахівцями різних напрямків у межах одного закладу, що обмежує функціональність у багатопрофільних лікарнях. Крім того, можливості кастомізації процесів дуже обмежені, що ускладнює адаптацію до специфічних потреб закладу.

Отже, існує потреба у розробці більш адаптивної системи, яка дозволить персоналізувати робочий процес, підтримувати гнучке ведення прийомів та забезпечити кращу інтеграцію даних між підрозділами медичного закладу.

1.2 Обґрунтування вибору напрямку дослідження

Проведений аналіз існуючих медичних інформаційних систем в Україні показав, що попри наявність потужних рішень, таких як Helsi та Medics, існують значні обмеження у контексті гнучкості, кастомізації та підтримки повного циклу лікарського прийому. Зокрема, більшість платформ фокусуються або на пацієнтському сервісі, або на адміністративному управлінні клінікою, залишаючи недоопрацьованими аспекти повсякденної роботи лікаря.

Сучасна практика надання медичних послуг вимагає більшої деталізації прийомів, можливості фіксації динаміки пацієнта, збереження діагностичних гіпотез, супутніх скарг і анамнезу, ведення серійних епізодів у межах одного випадку лікування. Також потребує покращення функціонал взаємодії лікаря з іншими медичними працівниками (реєстратором, лаборантом, асистентом), що особливо актуально в багатопрофільних або приватних медичних закладах.

З огляду на це, актуальним стає створення адаптивної, модульної медичної інформаційної системи, яка:

- підтримуватиме деталізовану роботу лікаря в межах одного прийому;
- інтегруватиметься з існуючими стандартами (наприклад, eHealth);
- включатиме механізми аналітики пацієнтів за епізодами;
- забезпечуватиме розмежування ролей у команді медичного персоналу.

Таким чином, вибір напряму дослідження обумовлений потребою в інноваційному підході до підтримки медичних процесів на рівні окремого прийому, епізоду, пацієнта та установи загалом. Такий підхід має забезпечити глибшу цифрову трансформацію української медицини, орієнтовану не лише на адміністрування, а й на якість клінічної взаємодії.

1.3 Специфікація вимог

Загальна характеристика

Розроблюване програмне забезпечення — це десктопний застосунок для медичних закладів України, що забезпечує автоматизацію процесів реєстрації пацієнтів, ведення прийомів лікарями, обліку медичних епізодів та взаємодії між персоналом.

Метою є підвищення ефективності та прозорості роботи медперсоналу, забезпечення зручного доступу до медичних даних і мінімізація людського фактору при введенні та обробці інформації.

Функціональні вимоги

Для Реєстратора:

- Можливість створювати, редагувати та видаляти інформацію про пацієнтів.
- Пошук пацієнтів за ПІБ або ідентифікатором.
- Перегляд списку лікарів.

- Формування розкладу прийомів лікарів.
- Запис пацієнтів на прийом у вільний часовий слот.
- Перегляд та фільтрація прийомів по лікарю та даті.

Для Лікаря:

- Перегляд розкладу прийомів на сьогодні.
- Початок прийому пацієнта.
- Завершення прийому з можливістю збереження результатів.
- Створення та ведення епізодів лікування.
- Призначення діагнозів (ICPC-2, ICD-10).
- Перегляд медичної картки пацієнта з хронологією епізодів.

Нефункціональні вимоги

Інтерфейс користувача:

- Застосунок повинен мати інтуїтивно зрозумілий інтерфейс з панеллю навігації.
- Основні сторінки: Робочий кабінет, Календар, Пацієнти, Прийом.
- Гнучка адаптація під потреби кожної ролі (лікар, реєстратор).

Продуктивність:

- Завантаження даних у списках має відбуватись менше ніж за 1 секунду.
- Оптимізована робота з великою кількістю записів у базі даних (1000+ пацієнтів).

Сумісність:

- Працює на ОС Windows 10 і вище.
- Розробка на WPF .NET, база даних — MySQL, ORM — Entity Framework Core.

Надійність та збереження даних:

- Дані мають зберігатися в захищеній базі даних із системою резервного копіювання.
- Усі критичні операції повинні мати підтвердження (наприклад, видалення запису).

Обмеження

- Система не підтримує інтеграцію з eHealth на першому етапі.
- Мультикористувацький режим з синхронізацією в реальному часі не реалізований у базовій версії.
- Відсутня веб-версія (тільки десктопний клієнт).

Припущення

- Заклад охорони здоров’я має стабільне локальне з’єднання із сервером бази даних.
- Користувачі проходять базове навчання перед роботою із системою.

1.4 Постановка задачі дослідження

З огляду на виявлені обмеження існуючих медичних інформаційних систем та актуальні потреби практичної охорони здоров’я, метою даного дослідження є розробка адаптивного програмного забезпечення для підтримки роботи лікаря в межах одного прийому, що дозволяє фіксувати симптоми, епізоди лікування, забезпечує пошук пацієнтів та взаємодію з іншими ролями медичного персоналу.

Для досягнення цієї мети поставлено такі основні задачі:

1. Розробити архітектуру і компоненти програмного забезпечення, що підтримують роботу з електронною карткою пацієнта, створення та ведення епізодів лікування, вибір діагнозів за класифікацією ICPC-2 та МКХ-10-АМ, гнучке керування розкладом лікаря, взаємодію між медичними ролями (реєстратор, лікар).
2. Реалізувати прототип системи у вигляді настільного додатку з використанням технологій WPF та EF Core.
3. Провести тестування програмного забезпечення на відповідність сформульованим вимогам.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Опис архітектури та технологій

Розробка сучасних медичних інформаційних систем вимагає не лише розуміння потреб користувачів, але й грамотного технічного проєктування. Для створення ефективного, зручного та масштабованого застосунку було обрано архітектуру клієнт-серверного типу, в якій клієнт (користувацький інтерфейс) взаємодіє із сервером (де зберігається та обробляється вся інформація) через встановлені правила взаємодії. Такий підхід дозволяє досягти розділення відповідальностей, забезпечує кращу підтримку, оновлення та потенційно – можливість інтеграції із зовнішніми службами, зокрема, державними системами охорони здоров’я.

Архітектура проєкту реалізована у вигляді настільного застосунку (desktop app), створеного з використанням Windows Presentation Foundation (WPF) – UI фреймворк, який не залежить від роздільної здатності та використовує векторний механізм рендерингу, створений для використання переваг сучасного графічного обладнання. WPF надає повний набір функцій для розробки додатків, які включають розширювану мову розмітки додатків (XAML), елементи керування, прив'язку даних, макет, 2D та 3D графіку, анімацію, стилі, шаблони, документи, медіа, текст та типографіку [3]. Однією з ключових переваг WPF є можливість розділення логіки програми та представлення даних (інтерфейсу) завдяки використанню шаблонів проєктування, зокрема MVVM (Model-View-ViewModel). Це дозволяє зменшити залежність між кодом UI та бізнес-логікою, що сприяє гнучкості при розширенні функціональності.

Бізнес-логіка, тобто основні правила обробки даних, обліку та валідації медичної інформації, реалізована мовою програмування C# – основною мовою для розробки під .NET. Це забезпечує високу продуктивність, доступ до широкого набору бібліотек, а також хорошу підтримку в середовищі розробки Visual Studio 2022.

Для роботи з базою даних у застосунку використовується Entity Framework Core (EF Core) – сучасний ORM-фреймворк, який забезпечує зручну абстракцію над SQL-запитами. EF Core – це легка, розширювана, з відкритим вихідним кодом і кросплатформна версія популярної технології доступу до даних Entity Framework [4]. Завдяки EF Core розробники можуть взаємодіяти з базою даних через об'єкти та колекції, що значно полегшує розробку, зменшує кількість ручного SQL-коду та підвищує безпеку.

Зберігання всіх даних про лікарів, пацієнтів, прийоми, медичні картки та діагнози виконується у реляційній базі даних MySQL, найпопулярніша система управління базами даних SQL з відкритим вихідним кодом, розроблена, розповсюджується і підтримується Oracle Corporation [5]. Цей механізм був обраний через свою швидкодію, доступність, широку підтримку з боку інструментів .NET і простоту у налаштуванні.

Інтеграція з GitHub дозволила ефективно керувати історією змін, створювати релізи та контролювати версії застосунку.

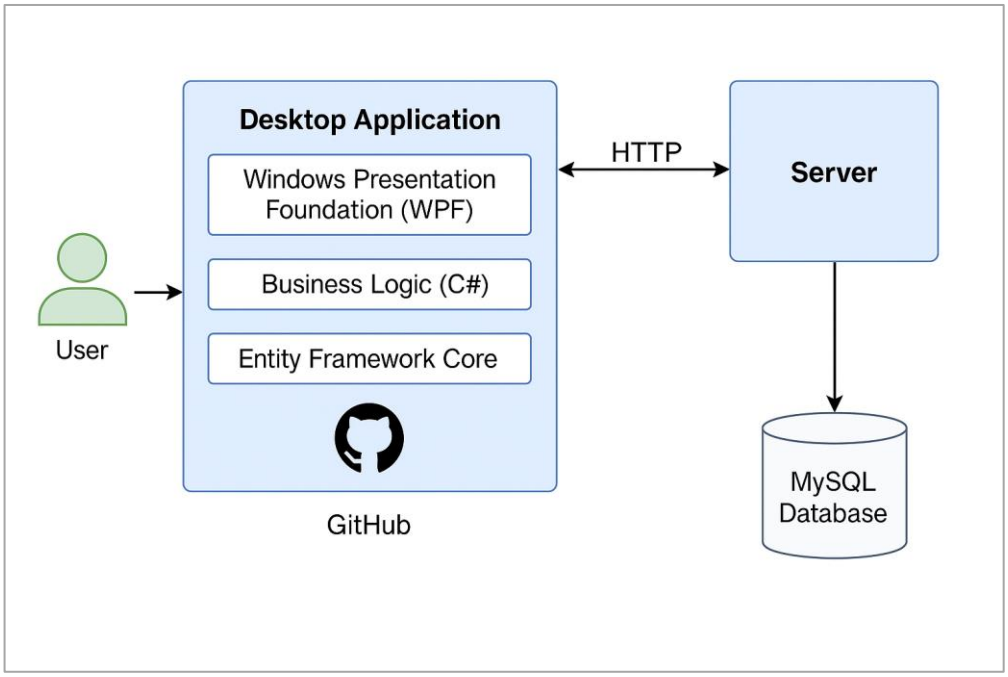

Рисунок 2.1 – Візуалізація архітектури проєкту

Таким чином, вибір технологій був зумовлений наступними критеріями: продуктивність, простота у підтримці, доступність документації, відповідність архітектурі клієнт-серверного типу та перспективність для майбутньої інтеграції з медичними державними сервісами, такими як eHealth.

2.2 Проєктування інформаційної системи

У процесі проєктування програмного забезпечення важливою складовою є моделювання структури та поведінки системи. Для цього широко використовуються нотації UML (Unified Modeling Language), які дозволяють уніфіковано представити функціональні можливості, взаємодії між об'єктами та їхні структури.

Існує дві підкатегорії діаграм UML: структурні діаграми та поведінкові діаграми. Структурні діаграми візуалізують компоненти, з яких складається система, і взаємозв'язок між ними – показують статичні аспекти системи. Поведінкові діаграми, з іншого боку, відображають те, що відбувається в системі, включаючи те, як компоненти взаємодіють один з одним і з іншими системами або користувачами [6].

2.2.1 Діаграма варіантів використання

Діаграма варіантів використання – (діаграма прецедентів, сценарій використання, use case) — дозволяє уявити типи ролей та їх взаємодію із системою. Проте не показує порядок виконання кроків. Зображує функціональні вимоги (те, що система може зробити) з точки зору користувача. Може описуватись текстом або у вигляді діаграми [7].

Діаграма використання для медичної інформаційної система наведена на рисунку 2.2

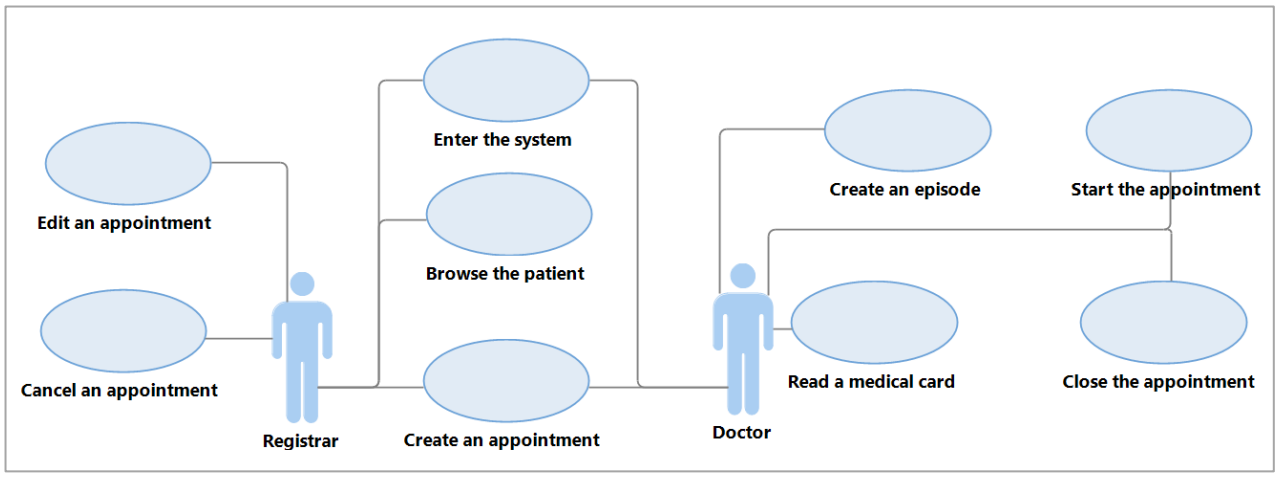

Рисунок 2.2 – Діаграма варіантів використання медичної системи

У нашому проєкті визначено дві основні ролі користувачів: Реєстратор та Лікар, які мають доступ до різних, а частково – спільних функцій системи. І Реєстратор, і Лікар починають свою роботу зі входу в систему, проходячи автентифікацію за логіном і паролем. Після авторизації обидва користувачі мають доступ до пошуку пацієнтів, використовуючи різні критерії: ім’я, прізвище, ідентифікатор тощо. Також спільним для обох ролей є сценарій створення прийому – тобто додавання в систему нового запису про медичну консультацію, з зазначенням пацієнта, лікаря, дати та часу.

Реєстратор має специфічні можливості, пов’язані з організаційною частиною обслуговування пацієнтів. Зокрема, він може редагувати прийоми, змінюючи їхні параметри в разі потреби, наприклад, якщо пацієнт просить перенести час консультації. Крім того, у випадку відміни візиту, Реєстратор має право скасовувати прийоми, видаляючи їх із розкладу або позначаючи як неактуальні.

Лікар, зі свого боку, володіє функціональністю, орієнтованою на медичну практику. Він розпочинає прийом, відкриваючи інтерфейс, у якому доступні всі дані про пацієнта та можливість заносити нову інформацію. У ході прийому лікар створює новий епізод – структуровану одиницю, яка включає в себе скарги

пацієнта, результати огляду, попередні діагнози, план лікування тощо. Після завершення консультації лікар фіксує результат і завершує прийом, зберігаючи всі дані в електронній картці. За потреби, він може переглядати повну історію пацієнта, включно з усіма попередніми епізодами, призначеннями, діагнозами й результатами лікування.

Загальна логіка побудови діаграми полягає в тому, що кожен користувач має зв’язки з відповідними сценаріями, які він може виконувати. Окремі дії можуть мати логічні залежності – наприклад, створення епізоду можливе лише після того, як лікар розпочав прийом. Така структура дозволяє чітко визначити повноваження кожного з користувачів, уникнути дублювання функцій та забезпечити логічну послідовність роботи з системою.

2.2.2 Діаграма послідовності

Діаграма послідовності описує взаємодію між об’єктами системи в певному сценарії, наприклад, створення епізоду прийому лікарем. Вона демонструє порядок викликів методів, передачу даних, послідовність запитів і відповідей. Sequence Diagram показує активність певних об’єктів у встановлених проміжках часу (як довготривалі періоди, так і короткі, наприклад, спринти розробки). І акцент в ній робиться на часі, а не на діяльності класів тощо. Окрім того, кожна діаграма орієнтована на єдиний Use Case [8].

Діаграма послідовності, що описує процес створення Епізоду пацієнта лікарем зображено на рисунку 2.3

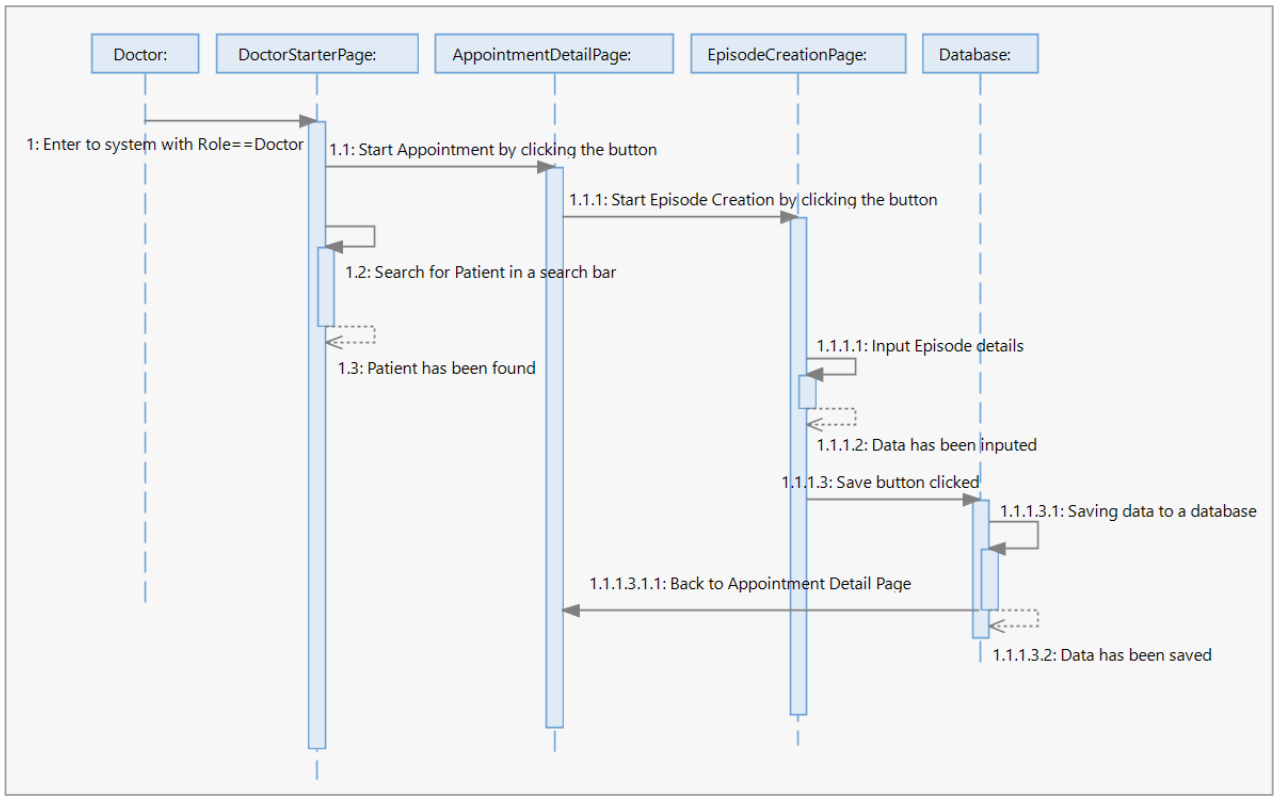

Рисунок 2.3 – Діаграма послідовностей процесу створення епізоду

Дана діаграма послідовності ілюструє процес створення нового медичного епізоду лікарем під час консультації пацієнта. У взаємодії беруть участь п’ять об’єктів: Doctor, DoctorStarterPage, AppointmentDetailPage, EpisodeCreationPage та Database. Цей процес відбувається в кілька етапів, кожен з яких відображає логічний та послідовний обмін повідомленнями між компонентами системи.

Процес починається з входу користувача в систему з роллю лікаря. Після авторизації, на сторінці DoctorStarterPage лікар натискає кнопку для початку консультації (Start Appointment), що ініціює перехід до сторінки AppointmentDetailPage. Там лікар використовує поле пошуку для вибору пацієнта, після чого система повертає відповідь, що пацієнта знайдено.

На цьому етапі лікар натискає кнопку для створення епізоду (Start Episode Creation), яка запускає сторінку EpisodeCreationPage. Тут лікар заповнює всі необхідні поля, що описують перебіг прийому: симптоми, діагнози, план лікування тощо. Після заповнення форми, натискається кнопка збереження (Save), і система надсилає запит на збереження даних до бази даних.

База даних приймає та обробляє інформацію про епізод, повертаючи підтвердження про успішне збереження. Після цього користувача автоматично повертає назад на сторінку AppointmentDetailPage, де можна продовжити роботу з прийомом або завершити його. Весь процес проходить послідовно, забезпечуючи логіку і цілісність запису медичної інформації.

Такий сценарій відображає типовий випадок взаємодії лікаря із системою в реальному медичному процесі, ілюструючи важливість послідовного обміну даними між компонентами застосунку, а також правильне розмежування відповідальностей між сторінками інтерфейсу та серверною логікою.

2.2.3 Діаграма класів

Діаграма класів відображає структуру системи: основні класи, їхні атрибути, методи, а також зв’язки між ними – асоціації, агрегації, композиції та наслідування. Сама діаграма класів являє собою набір статичних, декларативних елементів моделі. Вона дає нам найбільш повне і розгорнуте уявлення про зв'язки в програмному коді, функціональність та інформацію про окремі класи [9].

Реалізована діаграма класів медичної інформаційної системи показана на рисунку 2.4

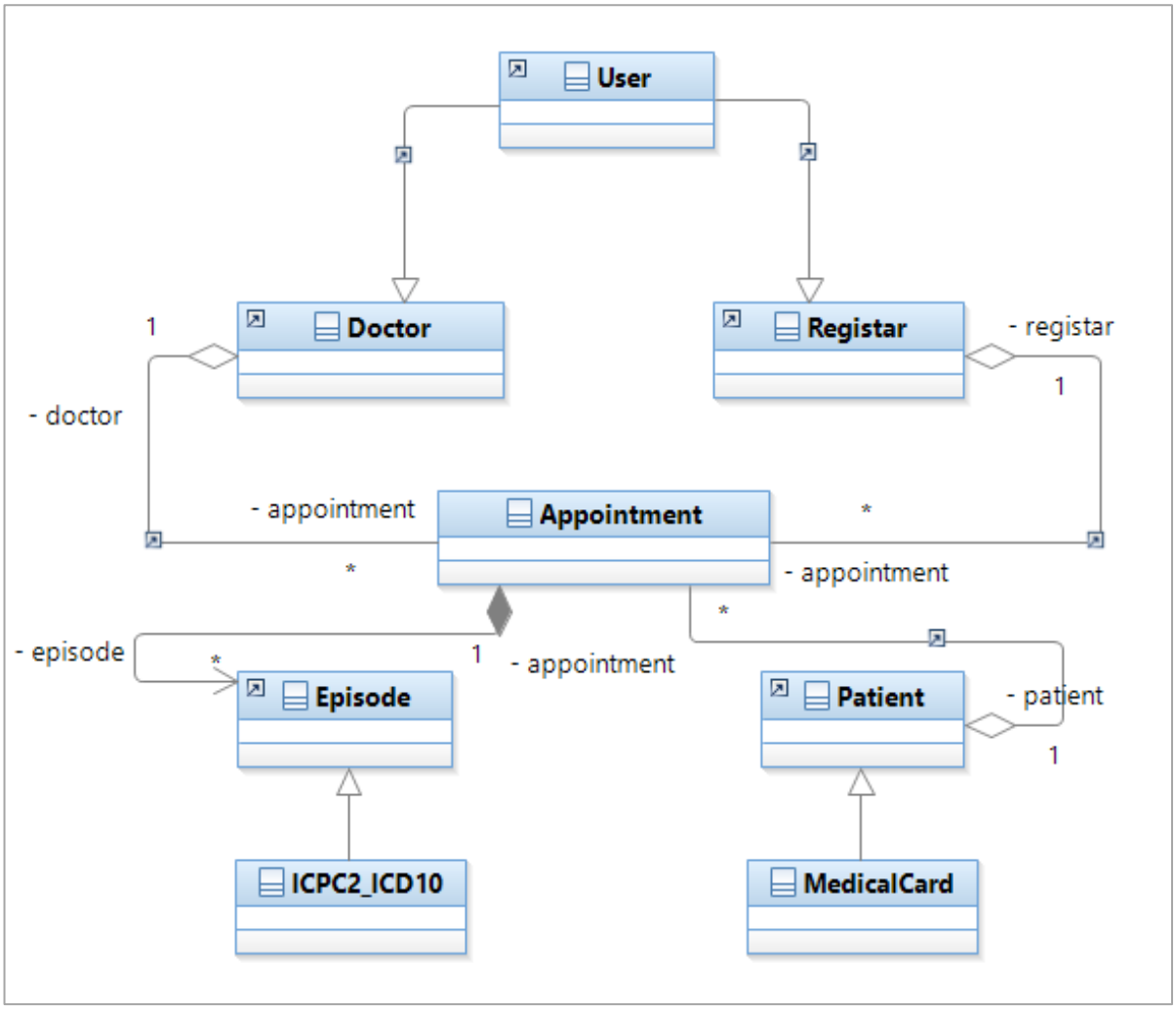

Рисунок 2.4 – Діаграма ієрархії класів медичної інформаційної системи

У нашій системі визначено основні класи:

- User, Doctor, Registrar, Patient, Appointment, Episode, MedicalCard, ICPC2_ICD10
- Встановлені зв’язки:
  - Doctor та Patient асоціюються з Appointment
  - Appointment може мати кілька Episode
  - Episode використовує класи ICPC2_ICD10
  - Patient має MedicalCard
  - Doctor та Registrar наслідують User

Ця діаграма дозволяє структурно спроєктувати архітектуру бази даних та логіку взаємодії об'єктів.

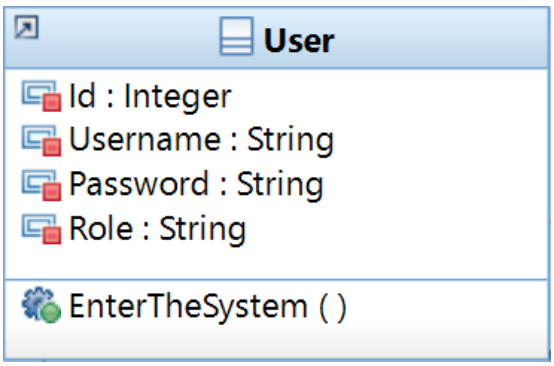

Рисунок 2.5 – Клас User

Клас User, як видно з рисунка 2.5, відповідає за збереження загальної інформації про користувачів системи. Зокрема, він містить поля Username, Password та Role, які дозволяють ідентифікувати користувача, перевіряти його повноваження в системі та розмежовувати функціональні можливості відповідно до ролі (наприклад, лікар або реєстратор). Цей клас є батьківським для класів Doctor та Registrar, що дозволяє реалізувати наслідування загальної функціональності.

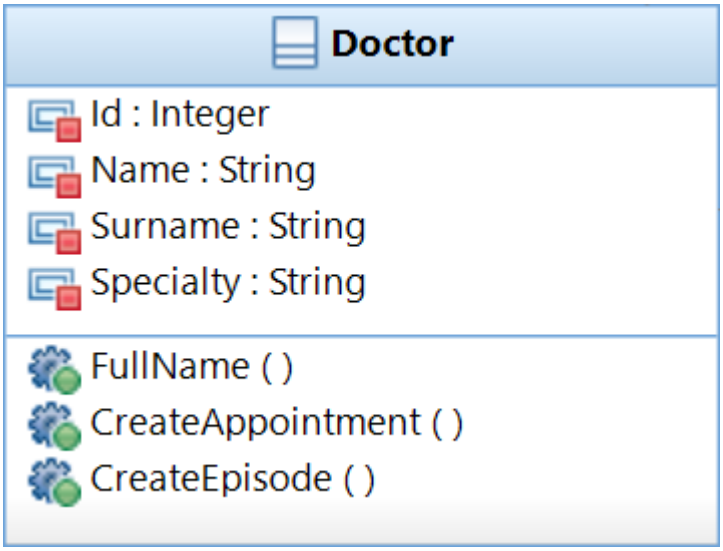

Рисунок 2.6 – Клас Doctor

Клас Doctor розширює клас User і зберігає додаткову інформацію, пов’язану з лікарем, таку як Name, Surname, Specialty. Він містить методи FullName() для об’єднання ПІБ лікаря та CreateAppointment() і CreateEpisode() для реалізації дій, які лікар може виконувати в системі: створення прийому або медичного епізоду.

Кожен лікар може мати декілька прийомів, що відображено у зв’язку один-до-багатьох з класом Appointment.

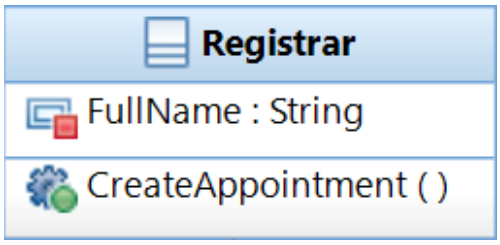

Рисунок 2.7 – Клас Registrar

Клас Registrar також є нащадком класу User і відповідає за збереження даних про реєстратора (поле FullName). Основна функція цього класу – створення прийомів пацієнтів, що реалізується методом CreateAppointment(). Це дозволяє реєстратору взаємодіяти з пацієнтами та лікарями, формуючи розклад прийомів.

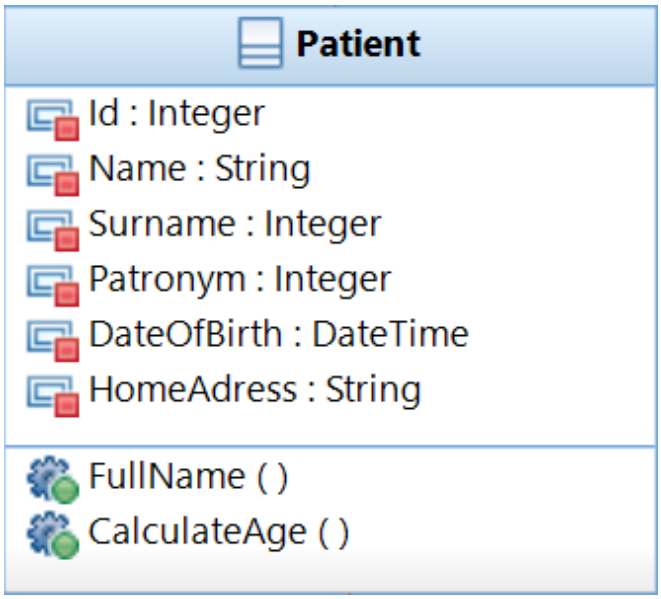

Рисунок 2.8 – Клас Patient

Клас Patient містить персональну інформацію про пацієнтів, зокрема: Name, Surname, Patronym, DateOfBirth, HomeAdress. Завдяки методам FullName() та CalculateAge() забезпечується зручне виведення повного імені пацієнта та обчислення його віку. Кожен пацієнт може мати багато прийомів (Appointment) і пов’язується один-до-одного з медичною картою (MedicalCard).

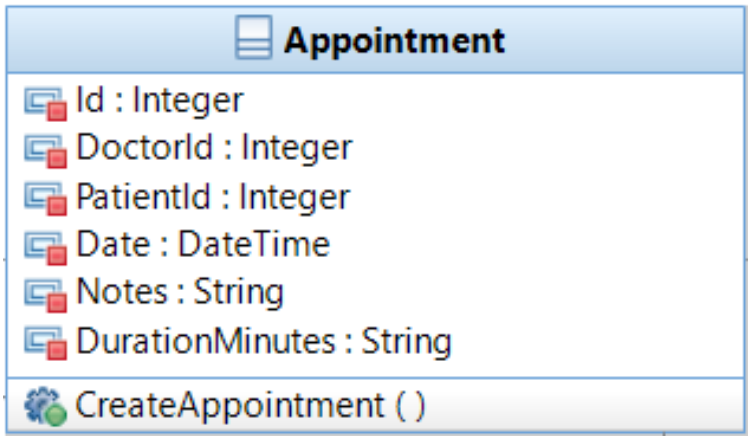

Рисунок 2.9 – Клас Appointment

Клас Appointment відповідає за збереження інформації про призначення на прийом між пацієнтом і лікарем. У ньому зберігається дата прийому (Date), його тривалість (DurationMinutes), а також нотатки (Notes). Зв’язки з класами Doctor та Patient є відношеннями "багато до одного", що означає: один лікар і один пацієнт можуть бути пов’язані з багатьма прийомами. Також клас пов’язаний з Episode, оскільки кожен прийом може мати декілька епізодів.

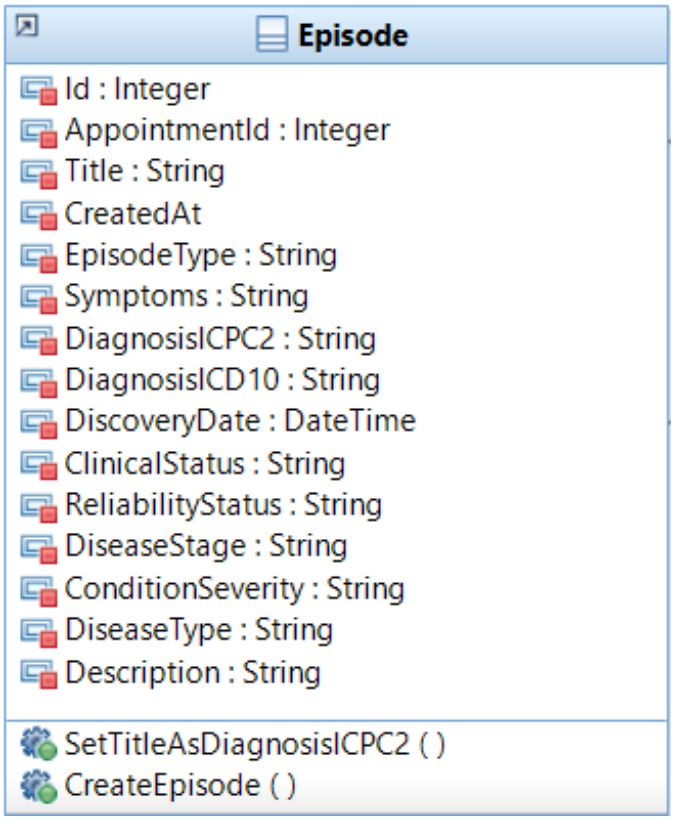

Рисунок 2.10 – Клас Episode

Клас Episode фіксує дані про окремий епізод медичного огляду чи лікування, пов’язаний із певним прийомом (AppointmentId). Поля Symptoms, DiagnosisICPC2, DiagnosisICD10, DiscoveryDate, DiseaseStage дозволяють деталізувати перебіг хвороби та діагноз. Завдяки політиці зберігання клінічного статусу, типу епізоду, достовірності діагнозу та опису (Description), система надає гнучке ведення історії лікування.

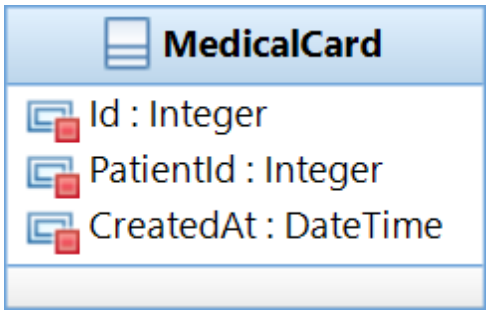

Рисунок 2.11 – Клас MedicalCard

MedicalCard є індивідуальною картою пацієнта, яка містить посилання на пацієнта (PatientId) та дату створення картки (CreatedAt). Один пацієнт має лише одну медичну картку, що реалізовано через зв’язок один-до-одного. Через цей клас реалізується централізоване зберігання всієї медичної інформації, пов’язаної з пацієнтом.

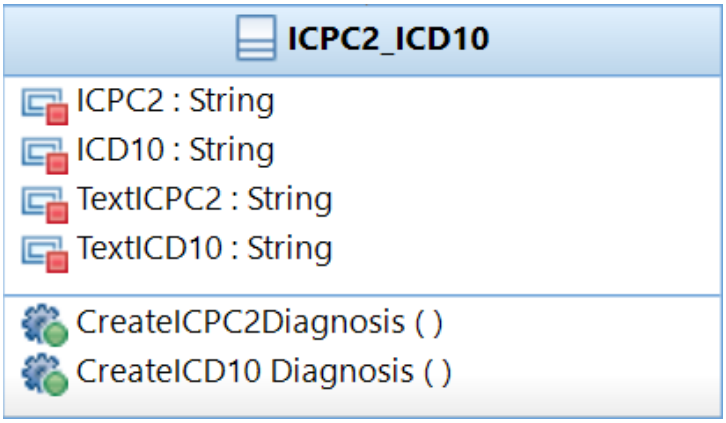

Рисунок 2.12 – Клас ICPC2-ICD10

Клас ICPC2_ICD10 використовується для зв’язку між двома міжнародними класифікаціями хвороб – ICPC-2 та ICD-10. Він містить коди та їх текстові описи (TextICPC2, TextICD10), а також функції для створення відповідних записів діагнозів. Зв’язок цього класу з Episode дозволяє зберігати медичні діагнози згідно з міжнародними стандартами.

РОЗДІЛ 3. КОНСТРУЮВАННЯ ТА ТЕСТУВАННЯ МЕДИЧНОЇ ІНФОРМАЦІЙНОЇЇ СИСТЕМИ

3.1 Структура бази даних

База даних відіграє ключову роль у функціонуванні системи, забезпечуючи збереження, організацію та доступ до всієї медичної інформації: даних пацієнтів, лікарів, прийомів, епізодів та діагнозів. Для розробки бази даних було використано MySQL, а для взаємодії з нею у застосунку — Entity Framework Core у режимі code-first.

Базу даних побудовано з урахуванням нормалізації до третьої нормальної форми (3НФ), що дозволяє мінімізувати надлишковість даних та підвищити цілісність інформації.

Основні таблиці бази даних:

- Users — містить загальну інформацію про користувачів системи (реєстраторів та лікарів). Зберігаються логіни, паролі, ролі.
- Patients — зберігає персональні дані пацієнтів, такі як ПІБ, дата народження, адреса проживання.
- Doctors — містить інформацію про лікарів (ПІБ, спеціалізація), пов’язану із загальним обліковим записом користувача.
- Appointments — таблиця прийомів. Містить дату, тривалість, пов’язаного пацієнта та лікаря, а також примітки.
- Episodes — зберігає медичні епізоди, створені під час прийому. Включає симптоми, діагнози, стадію захворювання, клінічний статус та інші атрибути.
- ICPC2_ICD10 — таблиця, що забезпечує зіставлення класифікацій ICPC-2 та ICD-10 із текстовими описами.
- MedicalCards — відображає історію пацієнта, пов’язану з його прийомами.

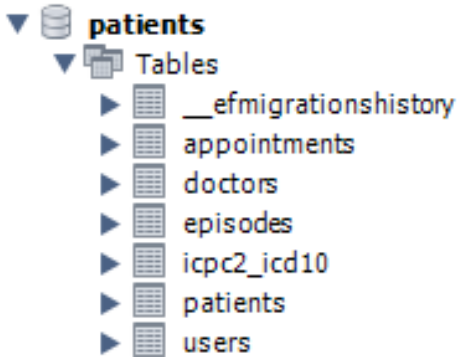

Рисунок 3.1 – База даних ‘patients’

Усі зв’язки між таблицями реалізовано за допомогою зовнішніх ключів (foreign key). Наприклад, таблиця Appointments має зовнішні ключі на Doctors та Patients, а Episodes — на Appointments.

Схему даних було реалізовано засобами Entity Framework Core за допомогою C#-моделей, які відповідають кожній сутності в базі даних. Конфігурація зв’язків, типів даних та правил цілісності також здійснюється через Fluent API.

3.2 Опис основних вікон програми

Програмний інтерфейс медичної інформаційної системи реалізований у вигляді вікон WPF-додатку, кожне з яких відповідає окремому процесу або функціональності, що виконується лікарем чи реєстратором. Інтерфейс побудовано відповідно до принципів UX-дизайну, з акцентом на зручність використання в умовах медичного закладу.

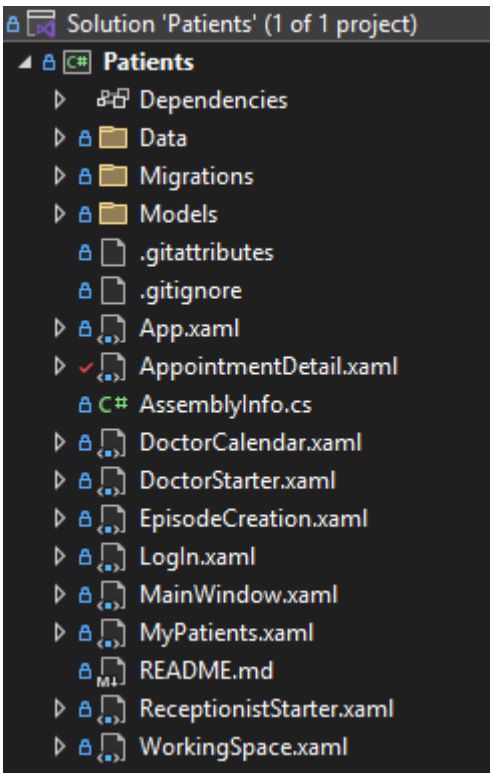

Рисунок 3.2 – Структура файлів проєкту

На рисунку 3.2 зображено структу проєкту Patients – це WPF-застосунок на C#, структурований за принципом розділення логіки та інтерфейсу користувача. Він містить директорії для моделей даних (Models), роботи з базою (Data, Migrations) і набір XAML-файлів, що представляють різні вікна програми, зокрема для лікаря, реєстратора, створення епізодів, перегляду пацієнтів та входу в систему. Центральним є MainWindow.xaml, а App.xaml керує запуском застосунку.

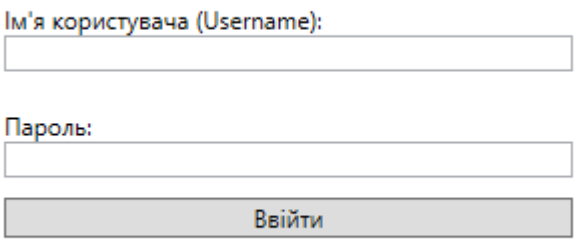

Рисунок 3.3 – Вікно автентифікації користувача

Головне стартове вікно забезпечує вхід до системи. Користувач повинен ввести логін та пароль, після чого система ідентифікує його роль (лікар або реєстратор) та перенаправляє до відповідного функціонального інтерфейсу.

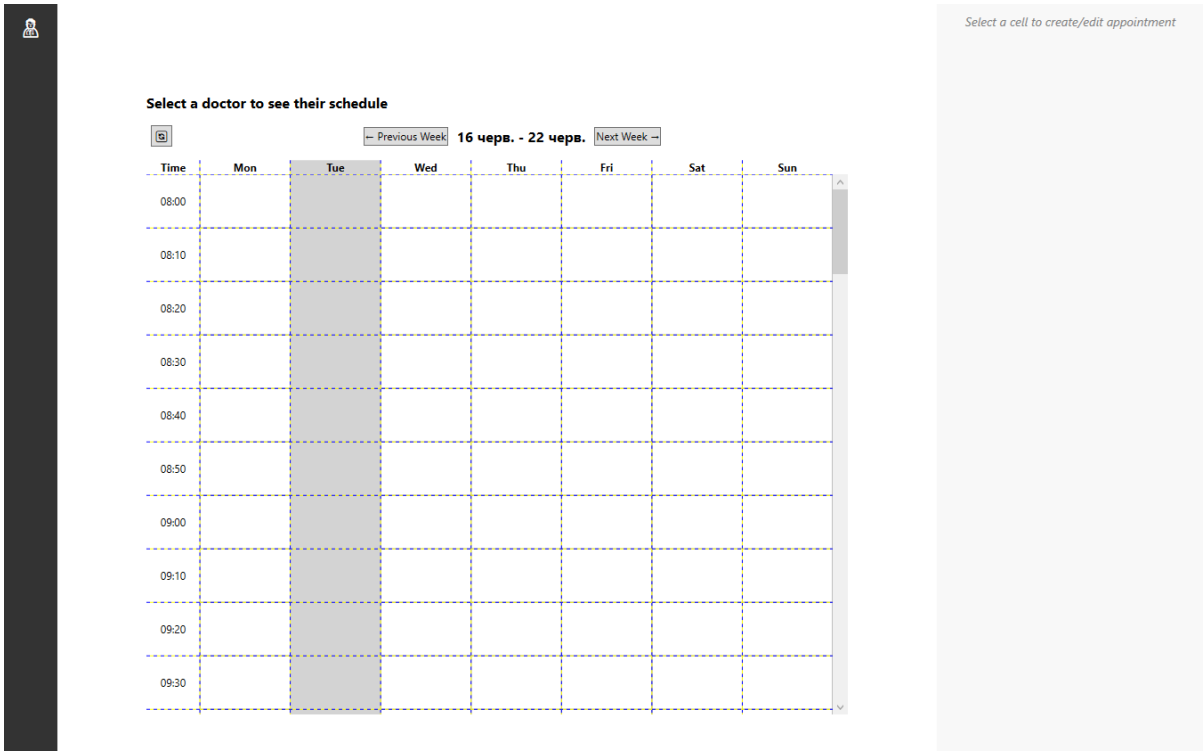

Рисунок 3.4 – Головне вікно реєстратора

Інтерфейс реєстратора включає:

- Бічну панель для вибору лікаря.
- Календар тижня, який показує розклад вибраного лікаря.
- Можливість створення нового прийому, натиснувши на порожню клітинку календаря.
- Праву інформаційну панель для перегляду/редагування обраного прийому або створення нового.

Реєстратор може скасовувати прийоми, переносити їх у часі, шукати пацієнтів у базі, а також створювати нові облікові записи пацієнтів.

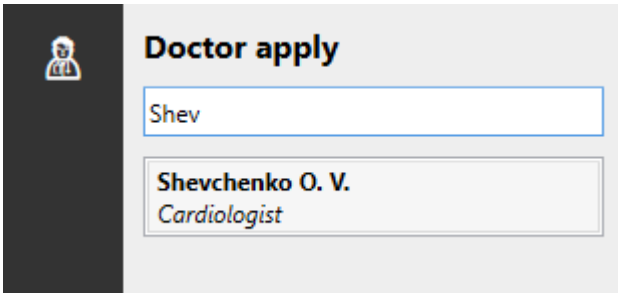

Рисунок 3.5 – Пошук лікаря

На старті календар, з яким може взаємодіяти Реєстратор, є неактивним, адже не вибраний лікар, якому будуть назначатися прийоми нових пацієнтів. Отже, для вибору лікаря було розроблено систему пошуку як за ім’ям, так і за спеціалізацією, що спрощує пошук потрібного спеціаліста.

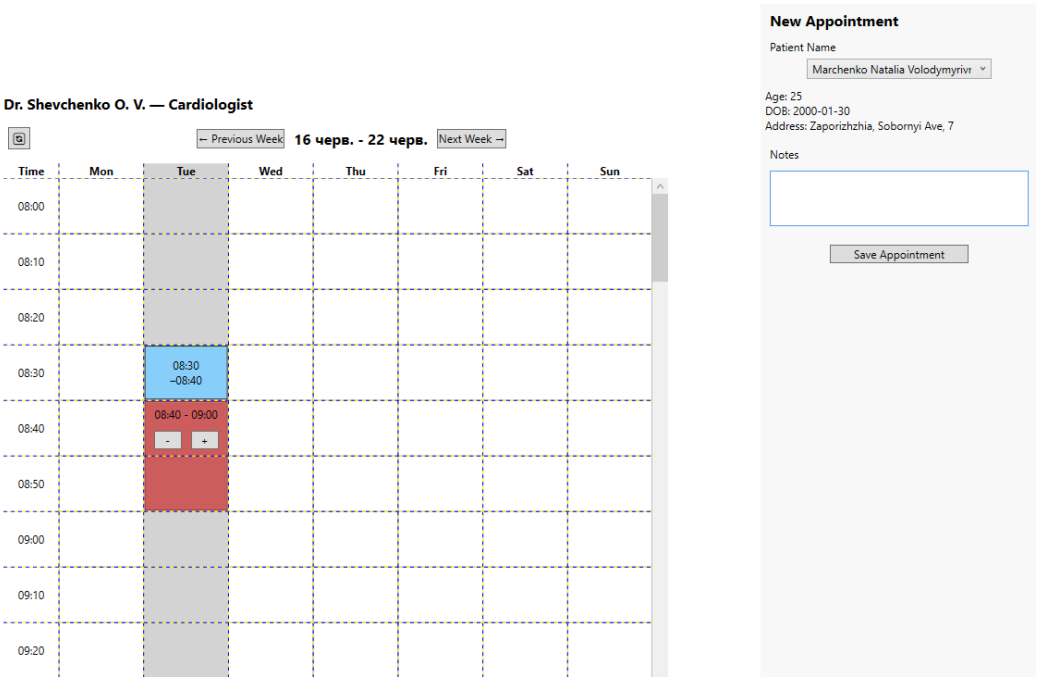

Рисунок 3.6 – Створення нового прийому в лікаря

Задля створення нового прийому потрібно обрати будь-яку пусту клітинку, натиснувши, відповідно, на пустому полі календаря. Надається можливість модифікувати час проводження прийому за допомогою клавіш «+» та «-» (від 10 хвилин до 1 години), що з’являються після обраного пустого поля. На панелі, що знаходиться праворуч з’являється форма створення прийому, що включає в себе

вибір/пошук пацієнта та вказання деталей прийому. При натисненні на вже існуючий прийом в цій панелі висвічуватиметься форма, що надає змогу змінити параметри прийому або видалити його.

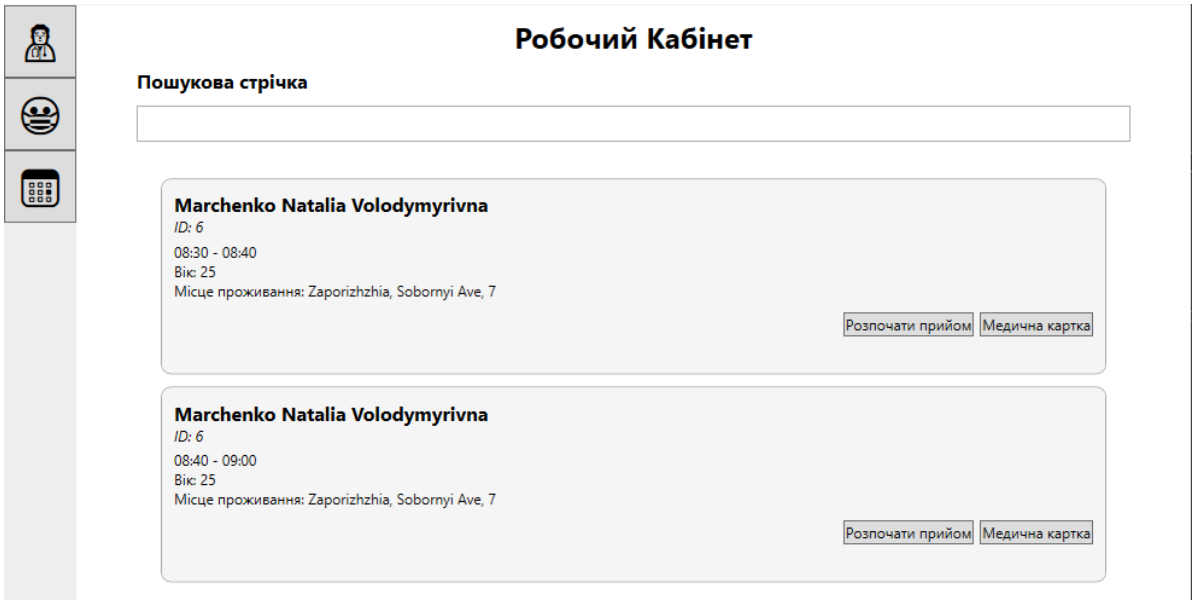

Рисунок 3.7 – Головне вікно лікаря (DoctorStarterWindow)

Після входу лікаря, відкривається головна сторінка, яка містить:

- Список прийомів на поточний день, що ще не були завершені.
- Панель пошуку пацієнтів.
- Бічне меню з навігацією (кнопки “Лікар” і “Календар”).

Лікар має змогу натиснути кнопку "Почати прийом", після чого відкривається детальне вікно прийому, де можна переглянути медичну карту пацієнта, створити новий епізод, завершити прийом.

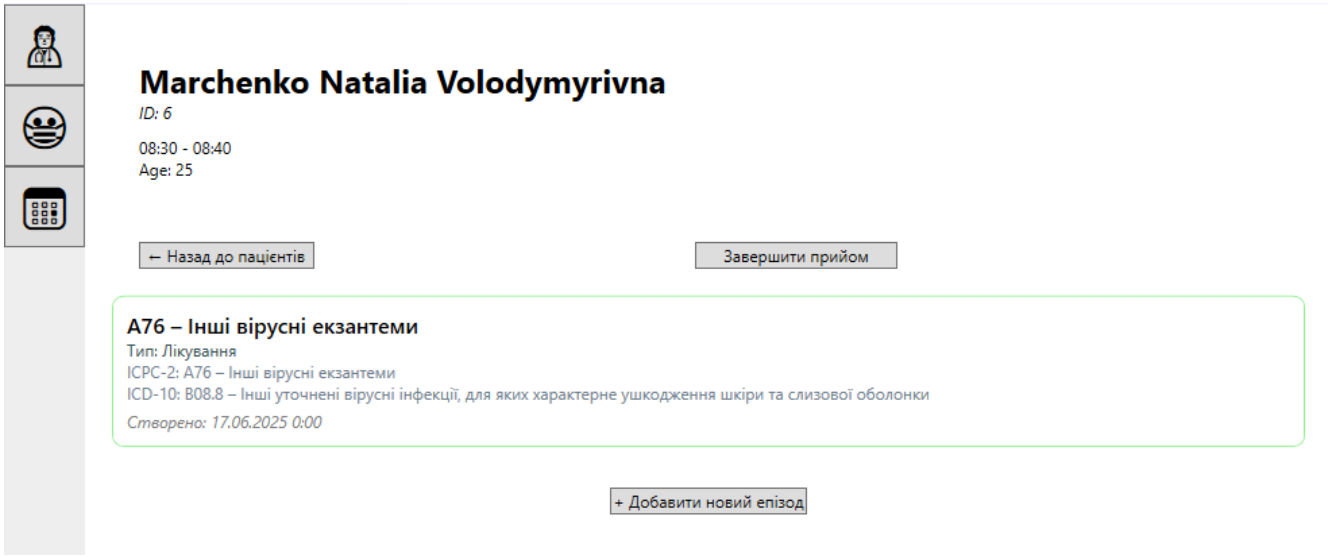

Рисунок 3.8 – Детальна сторінка пацієнта (AppointmentDetailPage)

Під час відкриття прийому користувача (лікаря) переносить до детальної сторінки пацієнта де перелічені відкриті та вже закриті епізоди. З основного функціоналу цієї сторінки, це можливість редагувати та створювати нові епізоди пацієнта, з яким розпочато прийом.

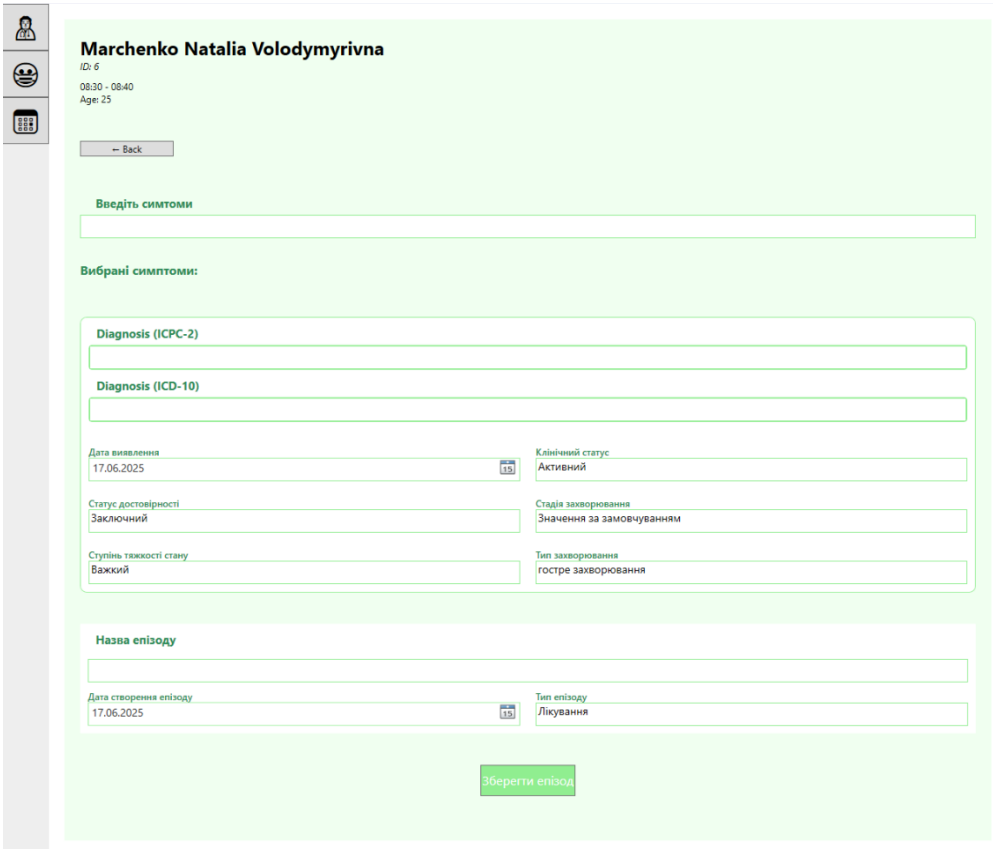

Рисунок 3.9 – Вікно створення епізоду (EpisodeCreationPage)

У цьому вікні лікар створює медичний епізод, вводячи:

- Заголовок епізоду.
- Тип епізоду, симптоми, діагнози.
- Клінічний та діагностичний статус.
- Ступінь надійності діагнозу, стадію хвороби, тощо.

Передбачено випадаючі списки ICPC-2 і ICD-10 (МКХ-10-АМ) з пошуком та фільтрацією для вибору діагнозів. Після заповнення всіх обов'язкових полів лікар натискає кнопку "Зберегти", після чого дані записуються в базу.

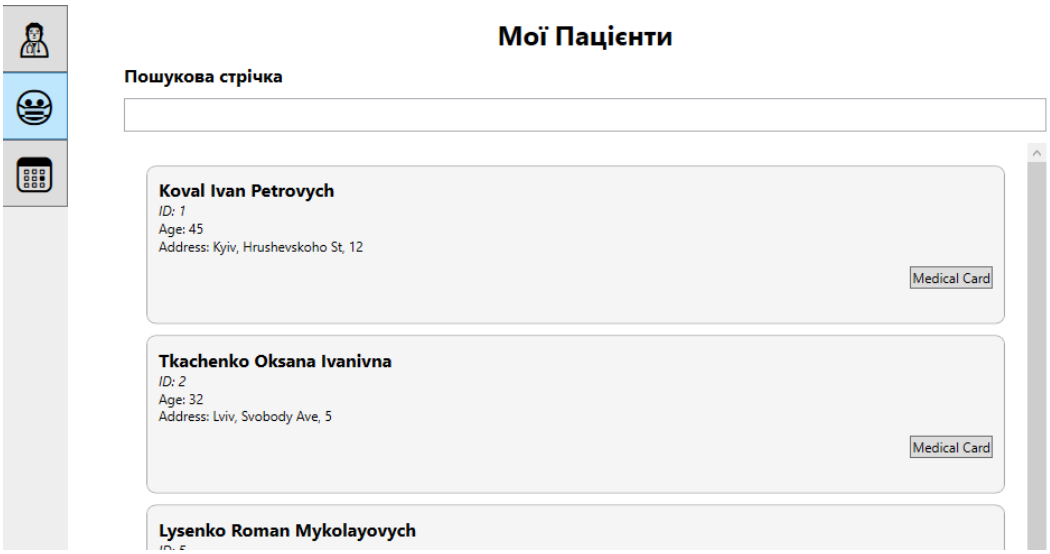

Рисунок 3.10 – Вікно «Мої пацієнти» (MyPatientsPage)

За допомогою вікна, що показано на рисунку 2.10, лікар має можливість переглянути дані про своїх пацієнтів, переглянути їхні медичні картки та закриті епізоди. Також, як можна побачити на панелі, існує схематична кнопка календаря, що відправляє лікаря на сторінку власного календаря (DoctorCalendarPage). Єдиною відмінністю між календарям Реєстратора та Лікаря в тому, що перший може призначити прийом будь-якому лікарю, в той час як другий обмежений призначенням прийомів лише собі. В усьому іншому календарі повністю повторюють функціонал один одного.

Додаткові функціональні вікна

- Медична карта пацієнта: відображає повний список епізодів, створених для конкретного пацієнта.
- Вікно пошуку пацієнта: дозволяє лікарю або реєстратору знайти пацієнта за ПІБ або ID.
- Редагування епізоду чи прийому: доступне через праву панель детальної інформації.

3.3 Реалізація взаємодії з базою даних через Entity Framework Core

Для зберігання, обробки та доступу до даних у системі використовується реляційна база даних MySQL, з якою взаємодія реалізована за допомогою ORM-технології Entity Framework Core. Цей інструмент дозволяє працювати з базою даних через об’єктно-орієнтований підхід, зменшуючи потребу в ручному написанні SQL-запитів і полегшуючи підтримку коду.

```
public class AppDbContext : DbContext
{
    1 reference
    public DbSet<User> Users { get; set; }
    3 references
    public DbSet<Doctor> Doctors { get; set; }
    16 references
    public DbSet<Appointment> Appointments { get; set; }
    2 references
    public DbSet<Patient> Patients { get; set; }
    2 references
    public DbSet<Episode> Episodes { get; set; }
    5 references
    public DbSet<Icpc2Icd10> Icpc2Icd10 { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<Icpc2Icd10>()
            .ToTable("icpc2_icd10")
            .HasNoKey();
    }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        var connectionString = "server=localhost;database=Patients;user=...;password=...;";
        optionsBuilder.UseMySql(connectionString, ServerVersion.AutoDetect(connectionString));
    }
}
```

Рисунок 3.12 – Налаштування контексту бази даних

У програмі створено клас ApplicationDbContext, який успадковує базовий клас DbContext. У ньому описано колекції DbSet<T>, які відповідають таблицям у базі даних. Цей клас також може містити логіку конфігурації зв'язків між сутностями (наприклад, HasMany, WithOne, HasForeignKey) або автоматичне заповнення (seed) бази на початковому етапі розробки.

Використовуючи Migrations, я послідовно оновлював структуру бази даних без втрати даних. Зокрема, я використовував такі команди:

```
Add-Migration InitialCreate
Update-Database
```

Їх використання допомогло мені у зберіганні змін в структурі проєкту, що значно полегшує відслідковування історії змін у проєкті.

Entity Framework Core дозволяє реалізувати запити до бази у вигляді C#-виразів. Наприклад, отримати всі прийоми для конкретного лікаря:

```
var appointments = context.Appointments
    .Where(a => a.DoctorId == doctor.Id &&
                a.Date >= _currentWeekStart &&
                a.Date < _currentWeekStart.AddDays(7))
    .Include(a => a.Patient)
    .ToList();
```

Рисунок 3.13 – Приклад використання запитів до бази даних

Завдяки визначенню ключів та зв’язків між сутностями (one-to-many, one-to-one) база забезпечує цілісність даних. Наприклад, при видаленні пацієнта система перевіряє наявність пов’язаних прийомів чи епізодів, і блокує або каскадно видаляє ці записи відповідно до конфігурації.

Переваги використання EF CoreЖ

- Висока швидкість розробки.
- Захист від SQL-ін’єкцій.
- Проста інтеграція з MySQL.
- Підтримка рефакторингу (перейменування класів/полів автоматично оновлює структуру БД).

- Зручна робота з навігаційними властивостями (наприклад, доступ до appointment.Patient.FullName).

3.4 Тестування застосунку

Процес тестування є критичним етапом життєвого циклу розробки інформаційної системи, особливо коли мова йде про медичне програмне забезпечення, що працює з чутливими персональними даними пацієнтів та забезпечує ключову підтримку в прийнятті рішень лікарем. Основна мета тестування полягає в перевірці коректності реалованої функціональності, стійкості системи до навантажень, а також зручності її використання з точки зору кінцевих користувачів — реєстраторів та лікарів.

У межах розробки цієї медичної інформаційної системи було застосовано кілька підходів до тестування:

3.4.1 Модульне тестування (Unit Testing)

Модульне тестування проводилося для перевірки роботи окремих логічних одиниць програми: сервісів, моделей та функцій. Зокрема, тестуванню піддавалися методи створення прийомів, пошуку пацієнтів, ведення епізодів, авторизації користувачів. Для цього використовувалась бібліотека xUnit/NUnit у поєднанні з мок-об’єктами (Moq) для моделювання залежностей.

```
using Moq;
using Patients.Models;

namespace TestProject
{
    0 references
    public class UnitTest1
    {
        0 references
        public class AppointmentServiceTests
        {
            [Fact]
            0 references
            public void CreateAppointment_ValidData_CallsAddMethod()
            {
                var mockRepo = new Mock<IAppointmentRepository>();
                var service = new AppointmentService(mockRepo.Object);

                var appointment = new Appointment
                {
                    Id = 30,
                    PatientId = 1,
                    DoctorId = 2,
                    Date = DateTime.Now
                };

                service.CreateAppointment(appointment);
                mockRepo.Verify(r => r.Add(It.IsAny<Appointment>()), Times.Once);
            }

            [Fact]
            0 references
            public void CreateAppointment_InvalidData_ThrowsException()
            {
                var mockRepo = new Mock<IAppointmentRepository>();
                var service = new AppointmentService(mockRepo.Object);

                var invalidAppointment = new Appointment
                {
                    Id = 2,
                    PatientId = 0,
                    DoctorId = 0
                };

                Assert.Throws<ArgumentException>(() => service.CreateAppointment(invalidAppointment));
            }
        }
    }
}
```

Рисунок 3.14 – Код тестування створення нового прийому

Було мною сформовано два тести, перший з яких вносить зміни у базу даних з коректними даними а другий – пробує внести некоректні дані. Результати проведених тестів пожна побачити на рисунку 3.16.

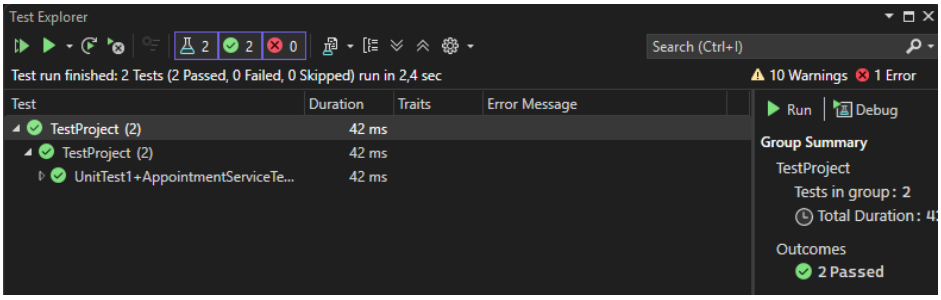

Рисунок 3.15 – Успішне проходження тестів створення нового прийому

Сформував ще один тест (рис. 3.16), що перевіряє, чи не виникає помилок при пошуку пацієнта у системі використовуючи як пошук за Id пацієнта, так і за його ім’ям.

```
0 references
public class PatientServiceTests
{
    [Fact]
    0 references
    public void Search_ByNameOrId_ReturnsCorrectResults()
    {
        var patients = new List<Patient>
    {
        new Patient { Id = 1, Surname = "Іваненко", Name = "Іван", Patronym = "Іванович" },
        new Patient { Id = 2, Surname = "Петренко", Name = "Петро", Patronym = "Петрович" }
    };
        var service = new PatientService(patients);

        var resultById = service.Search("1");
        var resultByName = service.Search("іван");

        Assert.Single(resultById);
        Assert.Equal(1, resultById.First().Id);

        Assert.Single(resultByName);
        Assert.Equal("Іваненко", resultByName.First().Surname);
    }
}
```

Рисунок 3.16 – Формування тесту, що перевіряє пошук пацієнта у системі

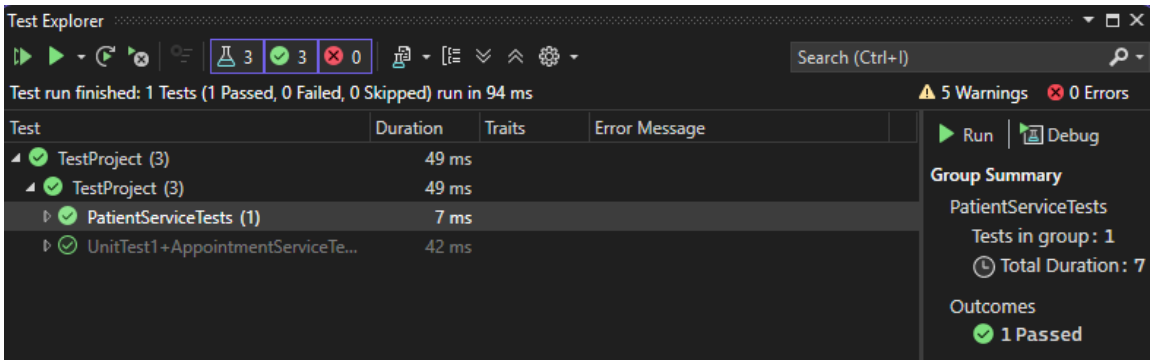

Рисунок 3.17 – Успішне проходження тестування пошуку пацієнта

Цей підхід дозволив на ранньому етапі виявити потенційні помилки у бізнес-логіці, не звертаючись до повноцінного запуску UI.

3.4.3 Тестування інтерфейсу (UI Testing)

UI-тестування проводилося вручну та охоплювало основні вікна програми: вікно реєстратора, лікаря, деталі прийому, сторінку медичної картки тощо. Кожен

функціональний елемент перевірявся на відповідність очікуваній поведінці: кнопки, списки, фільтри, індикатори, панелі.

Особливу увагу приділено сценаріям:

- Створення нового прийому;
- Перегляд прийомів по днях;
- Редагування та скасування записів;
- Перехід між вікнами (реєстратор → деталі прийому; лікар → епізод);
- Реакція інтерфейсу на невалідні дії (наприклад, створення прийому без пацієнта).

Перелік виконаних тестів над інтерфейсу розписані та перелічені в таблиці 3.1

Таблиця 3.1 – Виконання ручного тестування інтерфейсу

№	Об’єкт тестування	Що тестувалось	Проблеми, що виникли	Метод вирішення	Результат
1	Вікно реєстратора	Відображення календаря прийомів	Прийоми не завжди оновлювались при фільтрації	Оновлено логіку DisplayAppointments()	Фільтрація працює стабільно
2	Панель деталей прийому	Поява та зникнення правої панелі	Панель не ховалась після натискання «Назад»	Додано логіку приховування при відміні	Поведінка панелі коректна
3	Вікно лікаря	Відображення інформації про пацієнта	Не оновлювались поля після вибору іншого пацієнта	Прив’язано поля до DataContext	Дані оновлюють-ся динамічно
4	Пошук пацієнта	Фільтрація за ПІБ та ID	Пошук був нечутливим до пробілів/регістру	Додано Trim().ToLower() для введеного тексту	Пошук працює стабільно
5	Створення нового прийому	Створення прийому через календар	Не можна було створити прийом, якщо немає лікаря	Додано перевірку на selectedDoctor != null	Створення працює надійно

Продовження таблиці 3.1

6	Випадаючий список діагнозів	Автопошук та згортання після вибору	Список не зникав після вибору	Додано обробку LostFocus	UX став зрозумілі шим
7	Перетягування тривалості прийому	Зміна довжини т	Можливість перекривання інших прийомів	Додано перевірку на конфлікт з існуючими	Функція працює коректно

У результаті проведення тестування медичної інформаційної системи було підтверджено працездатність основних функціональних модулів, включаючи взаємодію між лікарем, реєстратором та пацієнтами в межах програмного забезпечення. Проведене юніт- та UI-тестування дозволило виявити й усунути низку помилок, пов’язаних із некоректною обробкою даних, порушенням логіки взаємодії компонентів, та забезпечити належний рівень зручності використання.

Зокрема, найбільша увага приділялася перевірці створення, редагування та відображення прийомів, а також роботі з електронною медичною карткою пацієнта. Було впроваджено необхідні перевірки даних, удосконалено логіку оновлення вікон та реалізовано запобігання логічним конфліктам, зокрема при накладанні прийомів у календарі.

Тестування підтвердило, що система демонструє стабільну роботу в типових умовах використання, має задовільну реакцію інтерфейсу та відповідає функціональним вимогам, визначеним на етапі проектування. Таким чином, розроблене програмне забезпечення може бути рекомендоване до експериментального впровадження у медичних закладах.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЕДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при шоку

Травматичний шок – складний патогенний процес, що виникає внаслідок важкої механічної травми, опіку і характеризується порушенням функцій життєво важливих органів та систем організму.

В розвитку травматичного шоку першочергову роль мають наступні фактори: втрата крові і біль, розлад дихання, порушення процесів метаболізму, інтоксикація організму недоокисленими продуктами обміну речовин внаслідок руйнування тканин.

Чинники, які сприяють розвитку шоку: запізніле і неповноцінне надання долікарської допомоги, вторинна травматизація в процесі транспортування в лікарню, повторна втрата крові, переохолодження або перегрівання, фізично-емоційне перенапруження, стреси, тривале недоїдання та зневоднення організму тощо.

При пораненнях зміни виникають у підкіркових утвореннях великого мозку та в системі периферійного кровообігу (перерозподіл крові, яка забезпечує життєдіяльність органів, передусім серця І мозку). Розвивається циркулярна гіпонія, судоми посткапілярних венул (випотіває плазма в позаклітинний простір), набряк і згущення крові. Знижується венозний тиск, слабнуть нирки, печінка, легені, відбувається тромбоутворення, розвиток незворотних змін в органах.

Діагностика шоку ґрунтується на визначенні показників, які характеризують загальний стан потерпілого. Найважливіший показник - рівень артеріального тиску. Що нижчим він є, то глибший розлад функцій організму, його життєдіяльності. Величина крововтрати – найоб'єктивніший показник ступеня важкості шоку.

Під час шоку усувають дію травмуючих факторів і факторів розвитку шоку, зупиняють кровотечу, перев'язують рани, усувають загрозу асфікції; вводять S-подібну трубку (повітропровід); при порушенні зовнішнього дихання в долікарську допомогою входить очищення порожнини рота і носоглотки, усунення западання

язика, відновлення прохідності дихальних шляхів; при пневмотораксі накладається пов'язка; проводиться інгаляція киснем, зупинка зовнішньої кровотечі; вводяться серцево-судинні і ана-лектичні засоби (виконує фельдшер); здійснюється іммобілізація кінцівок. Ввівши повторно знеболювальні засоби, дають гарячий чай та інші напої [10].

Ознаки шоку:

1) бліда, холодна і волога шкіра;
2) загальна слабкість;
3) неспокій, роздратованість;
4) сухість в роті, відчуття спраги;
5) часте або повільне дихання;
6) порушення свідомості, непритомність.

Причинами виникнення шоку можуть бути:

1) масивна зовнішня кровотеча;
2) внутрішня кровотеча;
3) травми різного ґенезу;
4) анафілаксія;
5) серцевий напад тощо.

Послідовність дій при наданні домедичної допомоги постраждалим при підозрі на шок:

1) перед наданням допомоги переконатися у відсутності небезпеки та за її відсутності перейти до наступного кроку;
2) заспокоїти постраждалого та пояснити свої подальші дії;
3) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику;
4) за можливості виявити та усунути причину виникнення шоку;
5) надати постраждалому протишокове положення:

а) перевести постраждалого в горизонтальне положення, якщо це не погіршує його дихання;

б) покласти під ноги постраждалого ящик, валик з одягу тощо таким чином, щоб ступні ніг знаходились на рівні його підборіддя;

в) підкласти під голову постраждалого одяг/подушку, якщо це не погіршує його дихання;

6) вкрити постраждалого термопокривалом/ковдрою;

7) забезпечити постійний нагляд за постраждалим до приїзду бригади екстреної (швидкої) медичної допомоги;

8) при погіршенні стану постраждалого до приїзду бригади екстреної (швидкої) медичної допомоги, повторно здійснити виклик екстреної медичної допомоги;

9) за можливості зібрати у постраждалого максимально можливу інформацію, стосовно обставин травми та механізму її отримання. Всю отриману інформацію передати працівникам бригади екстреної (швидкої) медичної допомоги або диспетчеру служби екстреної медичної допомоги [11].

4.2 Проведення інструктажів з охорони праці

Організацією навчання та перевірки знань з питань охорони праці працівників при підготовці, перепідготовці, підвищенні кваліфікації на підприємстві здійснюють працівники служби кадрів або інші спеціалісти, яким керівником підприємства доручена організація цієї роботи. Підготовка працівників для виконання робіт з підвищеною небезпекою здійснюється тільки в закладах освіти, які одержали в установленому порядку ліцензію Міносвіти та дозвіл Держпраці охорони праці на проведення такого навчання [12].

За характером і часом проведення інструктажів з охорони праці поділяються на вступний, первинний, повторний, позаплановий та цільовий.

Вступний інструктаж проводиться:

- усіма працівниками, яких приймають на постійну або тимчасову роботу, незалежно від їх освіти, стажу роботи та посади;
- з працівниками інших організацій, які прибули на підприємство і беруть безпосередню участь у виробничому процесі або виконують інші роботи для підприємства;
- з учнями та студентами, які прибули на підприємство для проходження виробничої практики;
- у разі екскурсії на підприємство;
- з усіма вихованцями, учнями, студентами та іншими особами, які навчаються в СЗО, ПЗО, ПТЗО, ВЗО, при оформленні або зарахуванні до ЗО.

Первинний інструктаж проводиться до початку роботи на робочому місці з працівником:

- новоприйнятим (постійно чи тимчасово) на підприємство;
- який переводиться з одного цеху виробництва до іншого;
- який буде виконувати нову для нього роботу;
- з відрядженим працівником, який бере безпосередню участь у виробничому процесі на підприємстві.

Проводиться з вихованцями, учнями та студентами СЗО, ПЗО, ПТЗО, ВЗО:

- на початку занять у кожному кабінеті, лабораторії де навчальний процес пов'язаний з небезпечними або шкідливими хімічними, фізичними, біологічними факторами, у гуртках, перед уроками трудового навчання, фізкультури, перед спортивними змаганнями, при проведенні заходів за межами території ЗО;
- перед виконанням кожного навчального завдання пов’язаного з використанням різних механізмів, інструментів, матеріалів тощо;
- на початку вивчення кожного нового предмета (розділу, теми), навчального плану (програми) – із загальних вимог безпеки, пов’язаних з тематикою і особливостями проведення цих занять.

Повторний інструктаж проводиться з працівниками на робочому місці в терміни, визначені відповідними чинними галузевими нормативними актами або керівником підприємства з урахуванням конкретних умов праці, але не рідше:

- на роботах з підвищеною небезпекою – 1 раз на 3 місяці;
- для решти робіт – 1 раз на 6 місяців.

Позаплановий інструктаж проводиться з працівниками на робочому місці або в кабінеті охорони праці:

- при введенні в дію нових або переглянутих нормативних актів про охорону праці, а також при внесенні змін та доповнень до них;
- при зміні технологічного процесу, зміні або модернізації устаткування, приладів та інструментів, вихідної сировини, матеріалів та інших факторів, що впливають на стан охорони праці;
- при порушенні працівником вимог нормативних актів про охорону праці, що можуть призвести до травм, аварій, пожеж тощо;
- при виявленні особами, які здійснюють державний нагляд і контроль за охороною праці незнання вимог безпеки стосовно робіт, що виконуються працівником;
- при перерві в роботі виконавця ,робіт більш ніж на 30 календарних днів – для робіт з підвищеною небезпекою, а для решти робіт понад 60 днів.

Цільовий інструктаж проводиться з працівниками:

- при виконанні разових робіт, непередбачений трудовою угодою;
- при ліквідації аварій, стихійного лиха;
- при проведенні робіт на які оформляються наряд-допуск, розпорядження або інші документи.

Проводиться з вихованцями, учнями, студентами 30 в разі організації масових заходів (екскурсії, походи, спортивні заходи тощо).

Вступний інструктаж проводить спеціаліст служби охорони праці первинний, повторний, позаплановий і цільовий інструктаж проводить безпосередній керівник робіт (начальник виробництва, цеху, дільниці, майстер) [13].

.

ВИСНОВКИ

У цій дипломній роботі було розроблено, спроєктовано та реалізовано програмне забезпечення для автоматизації роботи медичних працівників, зокрема лікарів та реєстраторів, в умовах сучасного медичного закладу. Основною метою дослідження стало створення функціональної медичної інформаційної системи, яка б відповідала сучасним вимогам до електронного документообігу, покращувала взаємодію між медичним персоналом та пацієнтами, і забезпечувала зручне ведення електронної медичної картки.

На етапі аналізу було проведено детальне вивчення існуючих рішень на ринку, зокрема систем Helsi та Medics. Було встановлено, що хоча ці платформи мають широкий функціонал, вони недостатньо охоплюють індивідуальні процеси роботи лікаря під час прийому, а також мають обмежені можливості щодо внутрішньої адаптації до специфіки конкретного закладу. Це стало підставою для вибору напрямку дослідження, орієнтованого на побудову більш гнучкої та функціонально розширюваної системи.

У процесі проєктування застосунку було створено низку UML-діаграм, які дозволили формалізувати вимоги до системи, структурувати її архітектуру та окреслити взаємозв’язки між її компонентами. Було розроблено діаграми варіантів використання, що описують дії користувачів системи, діаграми класів, які відображають внутрішню структуру програми, а також діаграми послідовності, що демонструють логіку обміну повідомленнями між об’єктами під час виконання окремих сценаріїв.

Розділ конструювання системи включав в себе вибір відповідних технологій, таких як C#, WPF, Entity Framework Core та MySQL, що забезпечили створення сучасного десктопного застосунку з розширеними можливостями взаємодії з базою даних. Було реалізовано низку ключових функцій: реєстрацію прийомів, обробку медичних епізодів, перегляд і редагування медичних карток, гнучкий календар прийомів, а також фільтрацію пацієнтів і лікарів.

Окрему увагу було приділено тестуванню програмного забезпечення. Було проведено юніт- та UI-тестування, що дозволило переконатися у коректності роботи окремих компонентів та взаємодії інтерфейсу з логікою додатку. Виявлені помилки були усунені, що сприяло підвищенню стабільності та зручності роботи з програмою.

Таким чином, у межах дипломної роботи було повністю реалізовано повний цикл розробки медичної інформаційної системи: від аналізу вимог до розгортання функціонального прототипу. Отримані результати можуть бути використані як основа для подальшого вдосконалення програмного забезпечення, інтеграції з eHealth, а також впровадження в умовах реальних медичних установ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Helsi.me. Офіційний сайт. Дата оновлення: 26.03.2025. URL: https://helsi.me/blog/helsi-vse-dlia-zdorovia-v-odnomu-zastosunku

2. Refactor.ua – Платформа відкритих інновацій. [Електронний ресурс]. URL: https://mind.ua/innovativeteams/474-medics

3. Windows Presentation Foundation Overview. Офіційний сайт. Дата оновлення: 07.05.2025. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/

4. Entity Framework Core. Офіційний сайт. Дата оновлення: 12.11.2024. URL: https://learn.microsoft.com/en-us/ef/core/

5. What is MySQL? Офіційний сайт. [Електронний ресурс]. URL: https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html

6. The ultimate guide to UML diagrams. [Електронний ресурс] URL: https://miro.com/diagramming/what-is-a-uml-diagram/

7. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. [Електронний ресурс]. Дата оновлення: 27.10.2022. URL: https://dou.ua/forums/topic/40575/

8. Sequence diagram досі жива?! [Електронний ресурс]. URL: https://e5.ua/uk/blogpost-2/sequence-diagram-dosi-zhiva/

9. UML для бізнес-моделювання: для чого потрібні діаграми процесів. [Електронний ресурс]. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html

10. ТЕМА 9. Медицина катастроф. Долікарська допомога при теплових і сонячних ударах. URL: https://dl.tntu.edu.ua/content.php?cid=299872

11. Про затвердження порядків надання домедичної допомоги особам при невідкладних станах: наказ Міністерства охорони здоров'я України від 28.03.2022 р. № 356/37692. [Електронний ресурс] URL: https://zakon.rada.gov.ua/laws/show/z0356-22#n543

12. ТЕМА 12. Організація охорони праці на підприємстві. Навчання з питань охорони праці. Профілактика травматизму. Інструктажі з охорони праці. URL: https://dl.tntu.edu.ua/content.php?cid=289134

13. Про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці та Переліку робіт з підвищеною небезпекою: наказ Державного комітету України з нагляду за охороною праці від 26.01.2005 р. № 231/10511. [Електронний ресурс]. URL: https://zakon.rada.gov.ua/laws/show/z0231-05#n119

14. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. URL: https://dl.tntu.edu.ua/bounce.php?course=5329

15. Методичні вказівки до виконання дипломної роботи освітнього рівня – бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

ДОДАТКИ

Додаток А. Тези конференції

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.4
Ільчій Б.–ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

АВТОМАТИЗОВАНА .NET-СИСТЕМА ДЛЯ ВЕДЕННЯ ЕЛЕКТРОННОЇ МЕДЧИНОЇ ДОКУМЕНТАЦІЇ ПАЦІЄНТІВ

Науковий керівник: д.ф.-м.н., професор Петрик М.Р.

Ilchii B.
Ternopil Ivan Puluj National Technical University

AUTOMATED .NET-SYSTEM FOR MAINTAINING ELECTRONIC MEDICAL DOCUMENTATION OF PATIENTS

Supervisor: Petryk M.R.

Ключові слова: електронна медична документація, .NET, облік пацієнтів
Key words: medical information system, .NET, patient records

Однією з найактуальніших проблем у сфері охорони здоров'я є неефективне ведення медичної документації, що часто базується на паперових носіях або застарілих цифрових рішеннях. Це ускладнює оперативний доступ до даних, створює ризики втрати або дублювання інформації, а також перешкоджає належному аналізу клінічної історії пацієнта. Відсутність єдиної системи управління електронними медичними записами негативно впливає як на якість обслуговування, так і на ефективність прийняття медичних рішень у динамічному середовищі лікарняних закладів [1].

Для вирішення зазначеної проблеми запропоновано створення сучасної автоматизованої системи обліку пацієнтів з використанням платформи .NET. Архітектура проєкту базується на технології ASP.NET MVC, що дозволяє створити гнучкий та інтуїтивно зрозумілий веб-інтерфейс для медичного персоналу. Для роботи з базою даних реалізовано ORM-рішення на основі Entity Framework Core, що забезпечує ефективну взаємодію з інформаційною моделлю та підтримку складних запитів до бази. Основна функціональність системи охоплює створення та редагування електронних медичних карток, фіксацію візитів, ведення історії хвороби, призначення лікування та облік лабораторних результатів. Особливу увагу приділено безпеці даних — використовується аутентифікація через ASP.NET Identity, шифрування персональної інформації, а також логування доступу відповідно до принципів HIPAA. Впровадження такої системи дозволяє оптимізувати адміністративні процеси, покращити точність ведення записів та зменшити навантаження на лікарів, надаючи їм швидкий доступ до повної інформації про стан пацієнтів [2, 3].

Література:

1. Islam M. N., Poly T. N., Yang H. C., et al. Electronic Health Records (EHR): A Review on Benefits and Challenges. *Journal of Medical Systems*, 2018. DOI: 10.1007/s10916-018-1030-6
2. HealthIT.gov. Benefits of Electronic Health Records (EHRs). URL: https://www.healthit.gov/topic/health-it-and-health-information-exchange-basics/benefits-ehrs
3. Microsoft Docs. Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/

Додаток Б. Основний код медичної інформаційної системи

Код створення календарю з ReceptionistStarter.xaml.cs:

```
      private void BuildScheduleGrid()
      {
          ScheduleGrid.Children.Clear();
          ScheduleGrid.RowDefinitions.Clear();
          ScheduleHeaderGrid.Children.Clear();

          ScheduleGrid.RowDefinitions.Add(new RowDefinition { Height =
GridLength.Auto });

          int todayCol = -1;
          if (DateTime.Today >= _currentWeekStart && DateTime.Today <
_currentWeekStart.AddDays(7))
          {
      todayCol = (DateTime.Today - _currentWeekStart).Days + 1; // +1
because 0 is the "Time" column
          }

          string[] days = { "Time", "Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun" };
          for (int col = 0; col < days.Length; col++)
          {
      Border headerBorder = new Border
      {
          Background = (col == todayCol) ? Brushes.LightGray :
Brushes.Transparent
      };

      var text = new TextBlock
      {
          Text = days[col],
          FontWeight = FontWeights.Bold,
          HorizontalAlignment = HorizontalAlignment.Center
      };
      headerBorder.Child = text;

      Grid.SetRow(headerBorder, 0);
      Grid.SetColumn(headerBorder, col);
      ScheduleHeaderGrid.Children.Add(headerBorder);
          }

          for (int i = 0; i < _timeSlots.Count; i++)
          {
      ScheduleGrid.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(60) });

      var timeText = new TextBlock
      {
          Text = _timeSlots[i],
          VerticalAlignment = VerticalAlignment.Center,
```

```
            HorizontalAlignment = HorizontalAlignment.Center
        };
        Grid.SetRow(timeText, i + 1);
        Grid.SetColumn(timeText, 0);
        ScheduleGrid.Children.Add(timeText);

        for (int day = 1; day <= 7; day++)
        {
            var cellBorder = new Border
            {
        BorderBrush = Brushes.Gray,
        BorderThickness = new Thickness(0.5),
        Background = Brushes.Transparent
            };

            if (day == todayCol)
            {
        cellBorder.Background = Brushes.LightGray;
            }
            cellBorder.MouseLeftButtonDown += CellBorder_MouseLeftButtonDown;
            Grid.SetRow(cellBorder, i + 1);
            Grid.SetColumn(cellBorder, day);
            ScheduleGrid.Children.Add(cellBorder);
        }
            }
        }


        private void ShowDoctorSchedule(Doctor doctor)
        {
            if (doctor == null) return;

            BuildScheduleGrid();

            using (var context = new AppDbContext())
            {
        var appointments = context.Appointments
            .Where(a => a.DoctorId == doctor.Id &&
        a.Date >= _currentWeekStart &&
        a.Date < _currentWeekStart.AddDays(7))
            .Include(a => a.Patient)
            .ToList();

        foreach (var appointment in appointments)
        {
            int dayCol = (int)appointment.Date.DayOfWeek;
            if (dayCol == 0) dayCol = 7;

            int row = _timeSlots.IndexOf(appointment.Date.ToString("HH:mm"))
+ 1;
            if (row < 1) continue;

            var border = new Border
            {
        Background = Brushes.LightSkyBlue,
        BorderBrush = Brushes.Black,
        BorderThickness = new Thickness(1),
        Margin = new Thickness(1),
```

```
      IsHitTestVisible = false // prevents click
          };
          var label = new TextBlock
          {
      Text = $"{appointment.Date:HH:mm}\n-
{appointment.Date.AddMinutes(10):HH:mm}",
      TextAlignment = TextAlignment.Center,
      HorizontalAlignment = HorizontalAlignment.Center,
      VerticalAlignment = VerticalAlignment.Center
          };
          border.Child = label;

          Grid.SetRow(border, row);
          Grid.SetColumn(border, dayCol);
          Grid.SetRowSpan(border, appointment.DurationMinutes / 10);
          ScheduleGrid.Children.Add(border);

      }
          }
      }
```

Код зберігання нового епізоду з EpisodeCreationPage.xaml.cs:

```
      private void SaveEpisodeButton_Click(object sender, RoutedEventArgs
e)
      {
          if (string.IsNullOrWhiteSpace(EpisodeNameBox.Text))
          {
      MessageBox.Show("Episode name cannot be empty.", "Error",
MessageBoxButton.OK, MessageBoxImage.Error);
      return;
          }
          if (selectedSymptoms.Count == 0)
          {
      MessageBox.Show("Please select at least one symptom.", "Error",
MessageBoxButton.OK, MessageBoxImage.Error);
      return;
          }
          var episode = new Episode
          {
      Title = EpisodeNameBox.Text,
      AppointmentId = _appointmentId,
      CreatedAt = EpisodeStartDatePicker.DisplayDate,
      EpisodeType = EpisodeTypeBox.Text,

      Symptoms = string.Join(", ", selectedSymptoms.Select(s => s.ICPC2)),
      DiagnosisICPC2 = ICPC2SearchBox.Text,
      DiagnosisICD10 = ICD10SearchBox.Text,

      DiscoveryDate = DiscoveryDatePicker.DisplayDate,
      ClinicalStatus = ClinicalStatusBox.Text,
      ReliabilityStatus = ReliabilityStatusBox.Text,
```

```
        DiseaseStage = DiseaseStageBox.Text,
        ConditionSeverity = ConditionSeverityBox.Text,
        DiseaseType = DiseaseTypeBox.Text,

            };
            _context.Episodes.Add(episode);
            _context.SaveChanges();
            MessageBox.Show("Episode saved successfully.", "Success",
MessageBoxButton.OK, MessageBoxImage.Information);
            // Navigate back to appointment details
            Back_Click(sender, e);
        }
```

Код, що керує випадаючими списками діагнозів ICPC-2 та МКХ-10-АМ (ICD-10) у формі створення епізоду, сторінка EpisodeCreationPage.xaml.cs:

```
        private void FilterICPC2Diagnoses()
        {
            string search = ICPC2SearchBox.Text.ToLower();

            var filtered = allICPC2Diagnoses
        .Where(d =>
            (d.TextICPC2 ?? "").ToLower().Contains(search) ||
            (d.ICPC2 ?? "").ToLower().Contains(search))
        .ToList();

            ICPC2ListBox.ItemsSource = filtered
        .Select(d => new { Display = $"{d.ICPC2} - {d.TextICPC2}", Original =
d })
        .ToList();

            ICPC2ListBox.DisplayMemberPath = "Display";
        }

        private void FilterICD10Diagnoses()
        {
            string search = ICD10SearchBox.Text.ToLower();

            var filtered = allICD10Diagnoses
        .Where(d =>
            (d.TextICD10 ?? "").ToLower().Contains(search) ||
            (d.ICD10 ?? "").ToLower().Contains(search))
        .ToList();

            ICD10ListBox.ItemsSource = filtered
        .Select(d => new { Display = $"{d.ICD10} - {d.TextICD10}", Original =
d })
        .ToList();

            ICD10ListBox.DisplayMemberPath = "Display";
        }

        private void ICPC2ListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
```

```
    {
        if (ICPC2ListBox.SelectedItem is null) return;

        var selected = ICPC2ListBox.SelectedItem as dynamic;
        var selectedICPC = selected?.Original?.ICPC2;

        if (selectedICPC == null) return;

         ICPC2SearchBox.Text = $"{selected.Original.ICPC2} -
{selected.Original.TextICPC2}";

        var linkedICD10s = _context.Icpc2Icd10
    .AsEnumerable()
    .Where(d => d.Type == "diagnosis" && d.ICPC2 == selectedICPC)
    .GroupBy(d => d.ICD10)
    .Select(g => g.First())
    .OrderBy(d => d.ICD10)
    .ToList();

        ICD10ListBox.ItemsSource = linkedICD10s
    .Select(d => new { Display = $"{d.ICD10} - {d.TextICD10}", Original =
d })
    .ToList();

        ICD10ListBox.DisplayMemberPath = "Display";

        ICPC2ListBox.Visibility = Visibility.Collapsed;
        icpc2DropdownOpen = false;
        ICPC2ListBox.SelectedItem = null;

        EpisodeNameBox.Text = selected.Display;
    }

    private void ICD10ListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        if (ICD10ListBox.SelectedItem is null) return;

        var selected = ICD10ListBox.SelectedItem as dynamic;
        var selectedICD10 = selected?.Original?.ICD10;

        if (selectedICD10 == null) return;

        ICD10SearchBox.Text = $"{selected.Original.ICD10} -
{selected.Original.TextICD10}";

        var linkedICPC2s = _context.Icpc2Icd10
    .AsEnumerable()
    .Where(d => d.Type == "diagnosis" && d.ICD10 == selectedICD10)
    .GroupBy(d => d.ICPC2)
    .Select(g => g.First())
    .OrderBy(d => d.ICPC2)
    .ToList();

        ICPC2ListBox.ItemsSource = linkedICPC2s
    .Select(d => new { Display = $"{d.ICPC2} - {d.TextICPC2}", Original =
d })
    .ToList();
```

```
    ICPC2ListBox.DisplayMemberPath = "Display";

    ICD10ListBox.Visibility = Visibility.Collapsed;
    icpc2DropdownOpen = false;
    ICPC2ListBox.SelectedItem = null;
}
```

Додаток В. Диск з роботою