КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кібербезпеки та комп'ютерної інженерії
(назва випускної кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему:

Система управління доступом в медичних інформаційних технологіях

Балобольченкова Марія Ігорівна
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кібербезпеки та комп'ютерної інженерії
(назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ
„ ___ ” _____ 20 25 року

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Система управління доступом в медичних інформаційних технологіях
(назва)

Здобувач Балобольченкова Марія Ігорівна
(прізвище, ім'я та по батькові повністю)
125 «Кібербезпека та захист інформації»
(спеціальність)
Безпека інформаційних і комунікаційних систем
(освітня програма)
Група БІКСм-24
Керівник Ізмайлова О. В.
(прізвище та ініціали)
Кандидат технічних наук, доцент
(вчене звання, науковий ступінь)
Рецензент
(прізвище та ініціали)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій
Кафедра: Кібербезпеки та комп'ютерної інженерії
Ступінь вищої освіти: Магістр
Спеціальність: 125 «Кібербезпека та захист інформації»
ОПП: Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри

к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ
„ ___ ” _____ 20 25 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА

СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Балобольченкової Марії Ігорівни

(прізвище, ім'я та по батькові здобувача)

1. Тема роботи «Система управління доступом в медичних інформаційних технологіях» затверджено наказом ректора КНУБА №1635/23.2/25 від « 30 » вересня 2025 року
2. Керівник роботи к.т.н. Ізмайлова Ольга Василівна, доцент кафедри кібербезпеки та комп'ютерної інженерії

(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту 15 грудня 2025 року.
4. Зміст пояснювальної записки за розділами:

Р.1. Особливості управління доступом у медичних інформаційних технологіях
Р.2. Методи та моделі управління доступом у медичних ІТ: аналіз,вибір та обґрунтування
Р.3. Моделювання компонентів управління та контролю доступу МІС
Р.4. Програмне забезпечення компонентів системи

5. Графічний матеріал за розділами:

С. 2 Вступ

С. 3 Актуальність

С. 4 Аналіз предметної області

С. 8 RBAC. SWOT-аналіз

С. 13 UML: діаграма прецедентів для користувачів

С. 21 Засоби реалізації та структура проєкту

6. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		Дата	підпис
Розділ 1.	Шабала Є.Є., к.т.н., доцент		
Розділ 2.	Делембовський М.М., к.т.н., доцент		
Розділ 3.	Шабала Є.Є., к.т.н., доцент		
Розділ 4.			

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Аналіз предметної області та огляд методів і моделей управління доступом	15.10.2025 р.
Моделювання компонентів управління та контролю доступу	27.10.2025 р.
Розробка програмного забезпечення	30.11.2025 р.
Остаточне оформлення роботи	08.12.2025 р.
Направлення роботи на рецензування, перевірку на плагіат	12.12.2025 р.
Попередній захист роботи на кафедрі	15.12.2025 р.

8. Дата видачі завдання: 30 вересня 2025 року.

Керівник _____ _____
(підпис) (прізвище та ініціали)

Здобувач _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Балобольченкова М.І. «Система управління доступом в медичних інформаційних технологіях».

Атестаційна випускова робота магістра за спеціальністю: 125 «Кібербезпека та захист інформації», освітня програма: «Безпека інформаційних і комунікаційних систем». – Київський національний університет будівництва і архітектури. – Київ, 2025.

Робота присвячена аналізу та реалізації методів автентифікації, авторизації й контролю доступу в медичних інформаційних системах. У роботі проаналізовано особливості функціонування медичної інформаційної системи, бізнес-процеси медичних установ, загрози безпеці медичних даних та чинну нормативно-правову базу. Розглянуто сучасні моделі управління доступом та проведено їх порівняння за допомогою SWOT-аналізу. Обрано метод, що забезпечує необхідний рівень безпеки для модулю «Медична карта».

У роботі побудовано сценарії взаємодії користувачів, політики доступу, алгоритми надання прав та діаграми моделювання, що описують логіку функціонування системи. Практичний розділ містить розробку модуля управління доступом, включно з засобами забезпечення інформаційної безпеки. Результатом є функціональний програмний компонент «Електронна медична карта», у якому реалізовано механізми управління доступом користувачів.

Ключові слова: медична інформаційна система, права доступу, управління доступом на основі ролей, управління доступом на основі атрибутів.

ABSTRACT

Balobolchenkova M.I. "Access control system in medical information systems".

Attestation work of a master's degree on the specialty 125 " Cybersecurity and information protection", educational program: "Security of information and communication systems". - Kyiv National University of Construction and Architecture. - Kyiv, 2025.

The work is devoted to the analysis and implementation of authentication, authorisation and access control methods in medical information systems. The work analyses the peculiarities of the functioning of medical information systems, the business processes of medical institutions, threats to the security of medical data and the current regulatory framework. Modern access management models are considered and compared using SWOT analysis. A method is selected that provides the necessary level of security for the 'Medical Record' module.

The paper develops user interaction scenarios, access policies, rights granting algorithms, and modelling diagrams that describe the logic of the system's functioning. The practical section contains the development of an access control module, including information security measures. The result is a functional software component, 'Electronic Medical Record,' which implements user access control mechanisms.

Keywords: medical information system, access rights, role-based access control, attribute-based access control.

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ Балобольченкова Марія Ігорівна Balobolchenkova Mariia		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Система управління доступом в медичних інформаційних технологіях		
	Access control system in medical information systems		
Освітній ступінь	Магістр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерії		
Спеціальність	125 «Кібербезпека та захист інформації»		
Освітня програма	Безпека інформаційних і комунікаційних систем		
Керівник	Ізмайлова Ольга Василівна		
Обсяг роботи:	*Пояснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	164(122 без додатків)	4	27
Розділ 1	Особливості управління доступом у медичних інформаційних технологіях		
Розділ 2	Методи та моделі управління доступом у медичних ІТ: аналіз, вибір та обґрунтування		
Розділ 3	Моделювання компонентів управління та контролю доступу МІС		
Розділ 4	Програмне забезпечення компонентів системи		
Висновки по роботі	Досліджено безпеку медичних інформаційних систем і запропоновано гібридну модель доступу RBAC+ABAC+PBAC. На практиці вдосконалено модуль електронної медичної картки на Django 5.2.		
Ключові слова: Keywords:	медична інформаційна система, права доступу, управління доступом на основі ролей, управління доступом на основі атрибутів medical information system, access rights, role-based access control, attribute-based access control		

Здобувач _____/ _____

Керівник _____/ _____

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ

ABAC – Attribute-Based Access Control

API – Application Programming Interface

CBAC – Context-Based Access Control

CQRS – Command Query Responsibility Segregation

DFD – Data Flow Diagram

FK – Foreign Key

GDPR – General Data Protection Regulation

HIPAA – Health Insurance Portability and Accountability Act

IDEF0 – Integration Definition for Function Modeling

IT – Information Technology

MAC – Mandatory Access Control

MFA – Multi-Factor Authentication

MITM – Man-In-The-Middle (attack)

MTV – Model–Template–View

NIST – National Institute of Standards and Technology

ORM – Object-Relational Mapping

OWASP – Open Web Application Security Project

PBAC – Policy-Based Access Control

PK – Primary Key

RBAC – Role-Based Access Control

SOA – Service-Oriented Architecture

SoD – Separation of Duties

SWOT – Strengths, Weaknesses, Opportunities, Threats

TLS/SSL – Transport Layer Security / Secure Sockets Layer

UML – Unified Modeling Language

XSS — Cross-Site Scripting

Адмін.панель – адміністративна панель

ЕМК – електронна медична карта

ЕСОЗ (eHealth) – електронна система охорони здоров’я

ІС – інформаційні системи

ІТ – інформаційні технології

МІС – медична інформаційна система

ММП – молодший медичний персонал

МКД – моделі контролю доступу

ОК – особистий кабінет

ЦБД – центральна база данихпервинний

ВСТУП

Розвиток медичних інформаційних технологій та активне впровадження інформаційних технологій в сферу охорони здоров'я вимагає створення надійних механізмів захисту доступу до медичних даних. Медичні інформаційні системи містять складні модулі, які відповідають за обробку, зберігання та передачу конфіденційної інформації, що робить їх особливо вразливими до кіберзагроз. Неефективне управління доступом може стати причиною для таких проблем як витік персональних даних, несанкціонована модифікація медичних записів, збій у функціонуванні медичних установ, репутаційні та економічні збитки тощо. У цьому контексті підтверджується ***актуальність*** дослідження та впровадження надійних і стандартизованих механізмів автентифікації, авторизації та контролю доступу, що забезпечують належний рівень безпеки медичної інформації.

Об'єктом дослідження є медичні інформаційні системи як комплексні програмні засоби, що забезпечують збір, зберігання, обробку та захист медичних даних у закладах охорони здоров'я.

Предметом дослідження є моделі та методи управління доступом в медичній інформаційній системі, включно з процесами автентифікації, авторизації та реалізації політик доступу.

Метою роботи є дослідження моделей та методів управління доступом для медичної інформаційної системи з урахуванням сучасних вимог до інформаційної безпеки, а також покращення програмного модулю медичної інформаційної системи «Електронна медична карта» відповідно до цих вимог.

Для досягнення поставленої мети необхідно виконати такі ***завдання:***

- провести аналіз предметної області та дослідити нормативно-правову базу, що регламентує захист медичних даних;
- дослідити існуючі моделі та методи автентифікації, авторизації та контролю доступу;
- проаналізувати загрози інформаційній безпеці, притаманні медичним інформаційним системам;

- виконати порівняння моделей контролю доступу за допомогою SWOT-аналізу та обґрунтувати вибір методів, що найбільш відповідають вимогам модулю «Електронна медична карта»;
- здійснити моделювання процесів управління доступом за допомогою UML та IDEF-діаграм;
- запропонувати алгоритми функціонування компонентів системи контролю доступу;
- розробити компонент «Електронна медична карта, використовуючи фреймворк Django, з акцентом на управління доступом та оцінити результати його роботи.

Під час написання магістерської роботи буде вдосконалено архітектуру та програмну реалізацію раніше розробленої електронної медичної карти. Поліпшення передбачає створення практичної програмної реалізації комбінованої моделі контролю доступу RBAC+ABAC+PBAC на основі оновленої версії Django 5.2. Це дозволить підвищити рівень безпеки системи та забезпечити відповідність сучасним міжнародним вимогам у сфері захисту медичних даних.

Дослідження має також вагоме практичне значення. Воно полягає в можливості застосування запропонованого підходу для створення або модернізації систем керування доступом у медичних інформаційних системах. Розроблений модуль може бути використаний як макет для контролю доступу у медичних інформаційних системах закладів охорони здоров’я або для розробки платформ електронного медичного документообігу. Методологічні рішення та програмні компоненти можуть бути адаптовані для інших галузей, що потребують підвищених вимог до захисту персональних даних. Також за допомогою подальшого аналізу моделей та методів контролю доступу буде можливість обрати найбільш доцільний варіант захисту систем, не тільки медичних.

1. ОСОБЛИВОСТІ УПРАВЛІННЯ ДОСТУПОМ У МЕДИЧНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЯХ

1.1 Медичні інформаційні системи як об’єкт управління доступом

1.1.1 Бізнес-процеси медичних установ

Розвиток інформаційних технологій (ІТ) сприяв покращенню всіх сфер життя, в яких вони були задіяні. Сфера охорони здоров’я не стала винятком. Завдяки ІТ були відкриті нові можливості лікування людей та покращилась якість надання адміністративних послуг. Роль провідних технологій не зосереджена лише на якомусь конкретному напрямі, а розповсюджується на більшість процесів, що притаманні медицині.

Процес медичної діагностики один з аспектів, що кардинально змінився за допомогою новітніх систем та приладів. Технології не тільки вдосконалили існуючі методи діагностики. З їх допомогою з’явились принципово нові методики такі як електрокардіограма, томографія, ультразвукове дослідження та інші. Без якісного створення та обробки зображень, що проводяться за допомогою апаратно-комп’ютерних систем, дані дослідження є або малоефективними або взагалі неможливими [1,2].

Автоматизація управлінських процесів медичних закладів. ІТ дозволяють автоматизувати багато аспектів управління медичними закладами, включаючи реєстрацію пацієнтів, призначення лікарських засобів, планування ресурсів та фінансовий облік.

Робота з даними. Технологічні засоби обробки інформації дозволяють збирати, зберігати та аналізувати великі обсяги медичної інформації, що допомагає виявляти тенденції, покращувати протоколи лікування та приймати більш обґрунтовані медичні рішення.

Телемедицина та обмін інформацією. ІТ дозволяють створювати та зберігати електронні версії медичних карток пацієнтів та інші важливі дані, що полегшує

доступ до інформації для лікарів та пацієнтів. Системи що впроваджуються дозволяють не тільки обмінюватись інформацією в текстовому форматі. За допомогою налагодженої передачі каналами зв’язку проводяться консультації та моніторинг пацієнтів на відстані, що особливо корисно для мешканців віддалених регіонів або тих, хто не може легко отримати доступ до медичної допомоги [1,2].

З огляду на різноманітність завдань, які постають перед сферою надання медичних послуг, автоматизація бізнес-процесів є неможлива без залучення сучасних інформаційних технологій, що забезпечують швидкий доступ до даних, їх обробку та обмін між структурними одиницями.

Водночас значна частина цих процесів пов’язана з обробкою медичних та персональних даних пацієнтів, які є чутливими та підлягають захисту згідно з чинним законодавством. Без належного налаштування політик інформаційної безпеки виконання процесів пов’язане з високим ризиком для конфіденційності та цілісності медичних даних. Саме тому інформаційна безпека є ключовим важелем для якісної роботи інших модулів.

1.1.2 Інфмаційні системи (ІС) як об’єкт управління безпекою. Класифікація ІС.

Заклади охорони здоров’я у своїй діяльності активно впроваджують комплексні інформаційні технології, що об'єднують декілька груп бізнес-процесів за допомогою широкого спектру інструментів – від телемедичних платформ і мобільних застосунків для пацієнтів до автоматизованих лабораторних систем і сховищ медичних зображень. Використання комплексного підходу зумовлено такими факторами: плавна інтеграція з іншими підсистемами та налаштована взаємодія їх між собою, економія часу та грошей на розробку, спільне використання функціональної складової та ресурсів, зручне управління даними. Для користувачів і їх взаємодії між собою є більш зручним варіантом саме комплекс із менших підсистем, аніж використання їх як окремих сутностей. Також такий підхід гарантує інтеграцію засобів безпеки на всіх рівнях функціонування.

Такий комплексний підхід в Україні реалізується в державних інформаційних ініціативах, зокрема в Електронній системі охорони здоров’я.

Електронна система охорони здоров’я (ЕСОЗ, eHealth) – інформаційно-комунікаційна система в Україні, що затверджена на законодавчому рівні ще у 2018 році. До даної системи мали підключитись всі заклади, що проводять медичну практику. Законом передбачено обов'язкову умову щодо того, щоб всі медичні заклади зареєструвалися у центральній базі даних ЕСОЗ та здійснювали ведення електронних медичних записів.

У центрі системи знаходиться центральна база даних, яка взаємодіє з різними медичними інформаційними системами (МІС), забезпечуючи автоматизований обмін даними через відкритий програмний інтерфейс (API). Як показано на рисунку 1.1, концептуальна архітектура української eHealth-системи відображає складну мережу взаємопов’язаних МІС, що на сьогодні включає понад 30 систем. Серед них є спеціалізовані підсистеми: лабораторні інформаційні системи, радіологічні інформаційні системи, системи архівування та обміну медичними зображеннями та різноманітне медичне обладнання й програмні рішення[3].

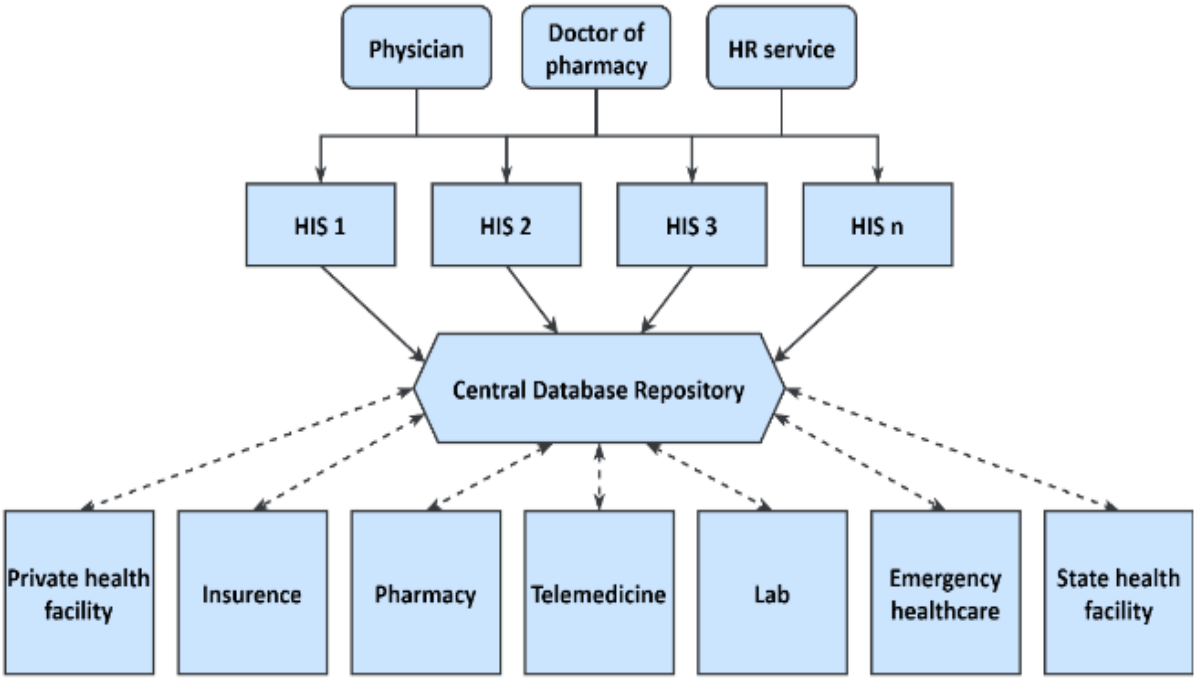


Рисунок 1.1 – Архітектура ЕСОЗ

У центрі eHealth-системи знаходиться центральна база даних (ЦБД), яка виконує роль єдиного сховища медичної інформації та забезпечує інтеграцію з різними медичними інформаційними системами закладів охорони здоров'я. ЦБД накопичує та структурує дані пацієнтів, забезпечує їхню доступність і обмін між підсистемами, проте безпосередньо не взаємодіє з операційними процесами лікарень. Через це захист ЦБД є важливим бо будь-які порушення конфіденційності або цілісності даних можуть вплинути на роботу всієї системи та на безпеку пацієнтів.

Ключовим елементом інтегрованої системи на рівні конкретного закладу МІС, що відтворюють функціональний блок, що необхідний для лікарні.

Медична інформаційна система – це програмне забезпечення (ПЗ), за допомогою якого реалізується збір, зберігання, обробка та передача медичних даних. Головною метою МІС є полегшення роботи медичних установ, управління медичними процесами, покращення доступу до даних. А також покращення якості надання медичних послуг та підвищення ефективності медичного обслуговування пацієнтів. МІС реалізують контроль доступу, часто за допомогою управління ролями користувачів та політик безпеки, а також координують роботу локальних підсистем (лабораторії, діагностика, телемедицина). Саме через МІС забезпечується безпечний та ефективний обмін даними з ЦБД та іншими підсистемами.

Тому в подальшому дослідженні фокус буде спрямований саме на захист МІС, оскільки ці системи є безпосередньо залученими до процесів управління доступом в медичних закладах, впливають на безпеку пацієнтів та реалізують політики доступу до даних, тоді як ЦБД виконує більше роль централізованого сховища. Відповідно, ***об'єктом*** даного дослідження виступають медичні інформаційні системи як комплексні програмні засоби, що забезпечують збір, зберігання, обробку та захист медичних даних у закладах охорони здоров'я.

Медичні інформаційні системи у закладах охорони здоров'я реалізуються через інтегровані функціональні модулі, кожен з яких відповідає за окремий аспект діяльності.

Основні функціональні модулі МІС для лікарні: [4,5]

- Адміністративний модуль – управління медичними записами, робота адміністратора, планування процедур, управління пацієнтопотоком.
- Кабінет пацієнта – надає пацієнтові доступ до його медичних даних, запису на прийом, комунікації з лікарем, нагадувань та повідомлень.
- Робочі місця медичного персоналу – лікаря (первинної та вторинної допомоги, ведення історій хвороби, клінічна оцінка, план лікування), середнього медичного персоналу (спостереження, допомога у процедурах), медичного координатора (координація процесів, електронні направлення).
- Лабораторний та діагностичний модуль – робоче місце лаборанта, ведення результатів аналізів, обмін даними з лабораторними системами.
- Модуль електронних медичних висновків та рецептів – оформлення висновків (народження, тимчасова непрацездатність), електронних рецептів на ліки та медичні вироби.
- Модуль аптечного закладу/фармацевта – доступ до електронних рецептів, управління видачею ліків, контроль залишків.

Перераховані модулі показую з чого зазвичай складається МІС. Вони мають бути орієнтовані на роботу в умовах інтеграції з ЕСОЗ. Однак, інформаційні системи, в тому числі і медичні, можуть суттєво відрізнятися між собою за масштабом, технічною реалізацією та рівнем інтеграції з іншими системами. Деякі автори класифікують інформаційні системи за рівнем управління, інші – за сферою застосування (банківські, медичні, освітні), а сучасні дослідники додають критерій архітектури (on-premise, cloud, hybrid) [6,7].

Для цілей цього дослідження, окрім розподілу за функціональним призначенням що наведена вище, доцільно виділити класифікацію МІС за типом користувачів, рівнем охоплення, архітектурою та способом розгортання. Це зумовлено тим, що саме ці характеристики визначають кількість потенційних точок доступу, розподіл відповідальності за безпеку та обсяг чутливих даних, до яких можуть отримати доступ користувачі.

За типом користувачів і ролей ІС в охороні здоров'я орієнтовані на:

- лікарів та медичний персонал;
- адміністраторів;
- пацієнтів;
- державні органи чи страхові компанії.

Кожен тип користувачів у цій класифікації має свої вимоги до безпеки: лікарям важливо гарантувати цілісність і актуальність медичних записів, пацієнтам – захист персональних даних, адміністраторам – контроль привілеїв, аудит та запобігання зловживанням доступу до системи, а державним органам і страховим компаніям важливі достовірність і повнота переданої інформації. Також треба зазначити що без чіткої моделі контролю доступу неможливо забезпечити ні конфіденційності, ні цілісності, ні звітності даних у роботі МІС.

За рівнем охоплення (який масштаб користувачів і організацій охоплює інформаційна система) ІС можна розділити на локальні, регіональні та глобальні.

Локальні (внутрішньолікарняні) системи фокусуються на діяльності конкретного медичного закладу. Основні ризики тут – внутрішні: неправильний доступ персоналу, суміщення ролей (наприклад, лікар одночасно виконує функції адміністратора), відсутність журналювання дій. Захист в таких системах будується на суворому контролі доступу, фізичному захисті серверів, регулярному резервному копіюванні. Тут управління безпекою часто більш гнучке, але залежить від ресурсів і персоналу.

Регіональні системи об'єднують кілька закладів у межах області чи регіону. Особливістю даних систем є необхідність координації політик безпеки між різними установами з урахуванням різного рівня технологічного устаткування закладів: лікарні можуть мати різні технічні можливості, але працювати із загальними даними. Щоб захистити такі системи необхідно зосередитись на уніфікованих протоколах обміну, стандартизованих API, а також міжмережевих екранах і VPN для безпечного з'єднання.

Глобальні/національні системи (наприклад, ЕСОЗ в Україні) зберігають дані мільйонів пацієнтів та інтегрують десятки підсистем. Основна проблема – масштаб: чим більше точок підключення (МІС, лабораторії, аптеки), тим вищий ризик витоку чи кібератаки. Тому акцент робиться на централізованій політиці безпеки та моніторингу. Управління безпекою орієнтоване на стандарти міжнародного рівня (ISO/IEC 27799, GDPR, HIPAA), що передбачає використання багаторівневого захисту, а саме: шифрування даних при їх передачі і під час розміщення в базі даних, централізований аудит усіх доступів, складні механізми авторизації через державні реєстри тощо.

Спосіб розгортання описує де й як фізично встановлюється система. Основними варіантами є локальне, хмарне та гібридне розгортання.

У випадку локального розгортання система повністю знаходиться в межах медичного закладу: і база даних, і сервери, і адміністрування. Це дає закладу максимальний контроль над тим, хто і як має доступ до інформації. Однак усі заходи захисту – від фізичного доступу до серверної кімнати до налаштування міжмережевих екранів і резервного копіювання – лежать на самому закладі. Загалом даний спосіб розгортання ідентичний до локального рівня охоплення. Тож ризики та управління безпекою ті ж самі.

Хмарні рішення побудовані на інфраструктурі стороннього провайдера і користувачі отримують доступ до системи через інтернет. У такому випадку значна частина відповідальності за захист даних перекладається на постачальника послуг. Важливим елементом стає шифрування даних, захист каналів зв’язку та контроль довіри до провайдера. Основний виклик тут – гарантії збереження конфіденційності, адже у медичній сфері дані особливо чутливі і потребують захисту від сторонніх осіб.

Гібридний варіант розгортання – це об’єднання переваг обох підходів: критично важливі або особливо конфіденційні дані можуть залишатися на локальних серверах, а менш ризикові сервіси (наприклад, статистика чи аналітика) працюють у хмарі. Це дозволяє знайти баланс між контролем і гнучкістю, однак потребує ретельної політики управління доступом і синхронізації даних.

Для дослідження найбільш важливими є ***класифікація архітектурних підходів***, адже від них залежить, як організовано зберігання й обробку даних, які точки доступу виникають у системі та які механізми захисту необхідні. Одним із базових історичних рішень, на якому ґрунтувалося багато медичних систем, була клієнт-серверна архітектура.

Клієнт-серверна архітектура – класичний фундаментальний підходів, на якому свого часу будувались більшість МІС. Суть її полягає в тому, що є **клієнт** (користувацький застосунок або вебінтерфейс), який звертається до **сервера**, де зберігаються дані і виконується бізнес-логіка. Це був великий крок уперед у порівнянні з «монолітними» локальними програмами, адже дозволив централізовано зберігати медичні записи, контролювати доступ до них і підключати декілька робочих місць лікарів.

Водночас клієнт-серверна модель має і обмеження. При зростанні кількості користувачів виникає ефект централізованого перевантаження – усі користувачі конкурують за ресурси одного сервера, а без додаткових механізмів масштабування та сегментації така архітектура вразлива до атак на єдиний сервер. Саме ці недоліки поступово підштовхнули розробників до більш гнучких архітектур, які спираються на клієнт-серверну модель. Такими сучасними архітектурами є [8]:

- монолітна – всі модулі (база даних, логіка, інтерфейс) об’єднані в один застосунок. Це спрощує розгортання та підтримку, проте ускладнює масштабування й робить систему вразливою до збоїв: помилка в одному модулі впливає на всю програму. Сьогодні такий підхід рідко застосовується в МІС, але ще трапляється у невеликих закладах через простоту й дешевизну;
- сервісно-орієнтована (SOA) – система складається з окремих сервісів (програмних компонентів), які виконують вузькі функції й взаємодіють через стандартизовані протоколи. Це забезпечує гнучкість, масштабованість та повторне використання компонентів, адже сервіси можна розгортати незалежно. Водночас SOA може призводити до труднощів у підтримці

цілісності системи та узгодженні даних. Такий підхід поширений у корпоративних МІС, де важлива інтеграція й можливість розширення;

- мікросервісна – це підхід, коли застосунок складається з багатьох невеликих сервісів, що працюють незалежно та взаємодіють через API. Це дає високу масштабованість і гнучкість: кожен сервіс можна розробляти, оновлювати чи масштабувати окремо. Для МІС такий підхід зручний, бо дозволяє розподіляти модулі (наприклад, електронні рецепти, лабораторні результати, страхові дані) між різними сервісами. Водночас він ускладнює адміністрування й безпеку, оскільки потрібно захищати велику кількість точок взаємодії та гарантувати узгодженість даних;
- багатошарова – ділить застосунок на окремі шари: представлення, бізнес-логіку та зберігання даних. Такий поділ спрощує підтримку, дозволяє незалежно змінювати або оновлювати рівні та робить систему більш модульною. Для медичних ІС це важливо, бо можна окремо посилювати захист на рівні бази даних чи доступу користувачів. Водночас така архітектура складніша у проектуванні й вимагає чіткої взаємодії між рівнями, інакше зростає ризик вразливостей;
- архітектура, керована подіями – це архітектура, в якій сервіси або компоненти взаємодіють та координують свою роботу на основі створення, виявлення та споживання подій;
- розділення відповідальності за команди та запити (CQRS) - це шаблон, який розділяє відповідальність за команди (дії, що змінюють стан програми) та запити (дії, що отримують дані з програми) на окремі компоненти, що забезпечує незалежне масштабування та оптимізацію для великих обсягів даних. Для сучасних МІС такі підходи актуальні, оскільки дозволяють швидко обробляти медичні дані, координувати взаємодію між підсистемами та підтримувати високий рівень безпеки та продуктивності.

У монолітних системах управління безпекою зводиться до централізованого контролю: усі користувачі працюють з однією базою даних і однією бізнес-

логікою, тому доступ і політики безпеки налаштовуються «єдиним блоком». Це може здаватися простим, але створює ризик: будь-яка вразливість або компрометація дає зловмиснику доступ до всієї системи одразу. Захист у таких випадках базується на жорсткому контролі фізичного доступу, ролях користувачів та таких інструментів як фаєрвол, обмеження мережевих з'єднань.

У сервісно-орієнтованій архітектурі безпека будується навколо кількох ключових моментів. По-перше, важливим є контроль доступу до кожного окремого сервісу, адже вони часто виконують критичні функції (зберігають історії хвороб або передають результати обстежень). По-друге, потрібне наскрізне шифрування обміну даними, щоб виключити ризики перехоплення. По-третє, особливу роль відіграє моніторинг і аудит: необхідно відслідковувати, які сервіси взаємодіють один з одним та з користувачем, і чи не відбуваються підозрілі запити.

У мікросервісній архітектурі управління безпекою ускладнюється тим, що замість одного великого застосунку з'являється десятки чи навіть сотні незалежних сервісів. Кожен із них має власні точки доступу, які треба захищати. Тому ключем до безпеки стають захищені API, шифрування трафіку між сервісами, централізована автентифікація та авторизація. Важливо також відстежувати взаємодію сервісів і вчасно виявляти аномалії, адже атака може початися з найслабшого модуля. У результаті головний виклик полягає не лише у захисті даних, а й у підтримці цілісності та узгодженості всієї системи.

У багатошаровій архітектурі безпека реалізується через чітке розділення обов'язків між шарами. Рівень представлення відповідає за аутентифікацію користувачів та захист інтерфейсів, бізнес-логіка – за реалізацію політик доступу та перевірку прав на операції з даними, а рівень зберігання – за цілісність та шифрування інформації. Такий поділ дозволяє ізолювати критичні функції, спрощує аудит і мінімізує ризики несанкціонованого доступу.

У архітектурі, що керована подіями, безпека орієнтована на контроль та захист потоків подій. Оскільки система функціонує через обмін повідомленнями між компонентами, важливо гарантувати автентичність джерел подій і цілісність

повідомлень, щоб не допустити підробки чи втрати даних, що можуть вплинути на роботу всієї МІС.

Шаблон CQRS, що розділяє операції читання та запису, дозволяє більш гнучко управляти безпекою: доступ до команд (записів) контролюється окремо від запитів (читання), що дозволяє обмежити можливість модифікації даних лише для авторизованих користувачів і водночас забезпечити масштабованість системи при високих навантаженнях.

У будь-якій з цих архітектур чи інших типів класифікації ключовим компонентом безпеки залишається управління доступом. Саме через нього визначається, які користувачі та ролі можуть працювати з певними модулями МІС, які дії дозволені, а які заборонені, що забезпечує конфіденційність, цілісність і доступність медичних даних, одночасно мінімізуючи ризики для пацієнтів і закладу. Саме тому ***предметом*** дослідження даного проєкту стануть моделі та методи управління доступом в МІС.

1.2 Аналіз існуючих рішень контролю доступу в МІС

На сучасному ринку медичних інформаційних систем в Україні домінують кілька платформ, серед яких особливе місце займає Helsi як одна з найбільш масштабних і функціонально розвинених систем. Станом на 2022 рік, понад 23 млн. українців зареєстровано в Helsi, більше 50 тис. лікарів та ~1300 медзакладів підключено до цієї системи [9].

У попередній кваліфікаційній роботі вже було проведено функціональний аналіз, де було відзначено широкі можливості для пацієнтів, лікарів та адміністрацій медичних закладів. Проте в рамках даного дослідження основну увагу зосереджено на механізмах контролю доступу та захисту персональних даних. Для пацієнтів є можливим авторизуватися у власний кабінет для перегляду електронної медичної картки, надається доступ до результатів лабораторних досліджень та призначень лікаря в режимі реального часу. Також додали гнучку системи налаштування доступу до даних членів сім’ї. Для лікарів Helsi надає

персоналізований доступ до кабінету з обмеженнями відповідно до ролі (сімейний лікар, вузький спеціаліст, лаборант). А медичні заклади можуть централізовано керувати ролями і правами доступу. Для закладів охорони здоров'я також є можливість призначення рівнів доступу для адміністратора, лікаря, реєстратора, бухгалтера тощо.

Щодо безпеки інформації, то на сайті ІС зазначено, що усі дані зберігаються у спеціалізованому дата-центрі, який має сертифікат комплексної системи захисту інформації, виданий Державною службою спеціального зв'язку та захисту інформації України [10].

Попри високий рівень впроваджених механізмів захисту, користувачі системи Helsi відзначають низку недоліків. По-перше, система має обмежені способи автентифікації. У більшості випадків доступ здійснюється через номер телефону та пароль або SMS-підтвердження (Рис.1.2). Відсутність альтернативних методів (BankID, електронний підпис, двофакторна автентифікація через додаток) знижує зручність та безпеку.

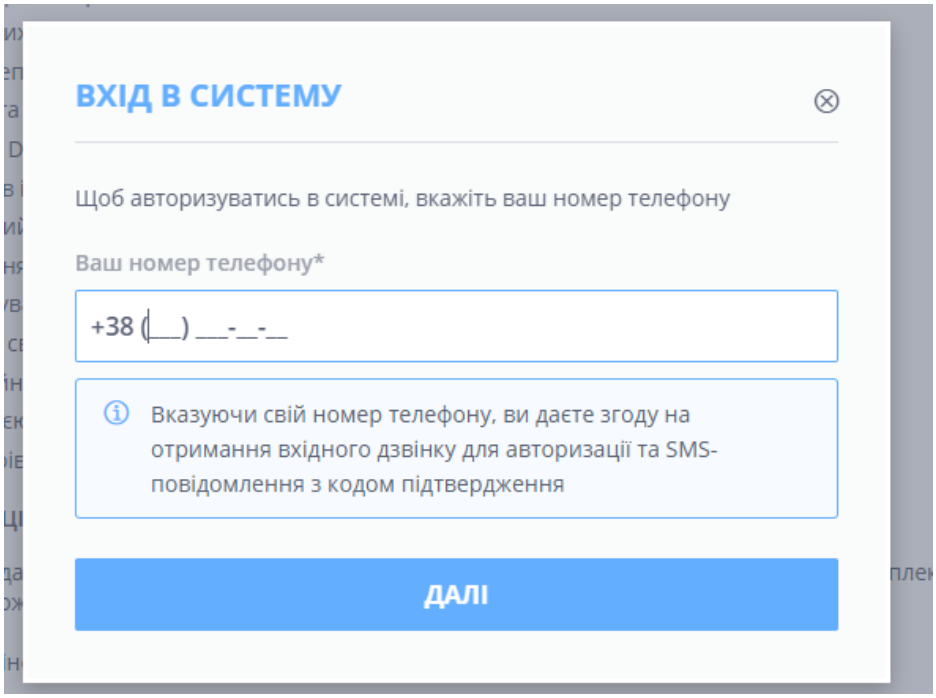


Рисунок 1.2 – Форма входу в систему Helsi (веб-інтерфейс)

По-друге, оскільки дані розміщуються централізовано в єдиному дата-центрі, у разі технічних проблем доступ до інформації може бути втрачений усіма користувачами одночасно. Це є вагомим недоліком безпеки цілісності даних.

По-третє, у системі спостерігається недостатня прозорість щодо доступу до медичних даних. Пацієнт не має можливості переглянути, які саме медичні працівники здійснювали доступ до його електронної медичної картки, хоча відповідні дії фіксуються в журналах аудиту. Надання користувачу такої інформації підвищило б рівень довіри та забезпечило додатковий контроль з боку пацієнта.

До того ж, багато користувачів скаржаться на незручність мобільного додатку. Хоча система має широкий функціонал, інтерфейс не завжди є інтуїтивно зрозумілим, а деякі операції вимагають зайвих кроків, що ускладнює швидке отримання необхідної інформації або запис на прийом. Через незручність, а подекуди власну халатність, адміністрація закладів пропускає етапи ідентифікації особи при реєстрації, нехтуючи безпекою.

Система «Helsi» є приватним сервісом, який не належить державі, проте фактично виконує функції державного рівня: забезпечує доступ до електронних медичних записів, запис до лікаря, обмін даними між закладами та лікарями, а також взаємодію з Національною службою здоров'я України. Такий статус створює низку ризиків:

- відсутність повного державного контролю – держава не володіє системою, тому не може гарантувати безперервність роботи та збереження даних у разі зміни власника або політики компанії;
- можливість комерційного використання даних – існує ризик передачі або аналізу медичної інформації з метою маркетингу, страхування чи фармацевтичного впливу;
- залежність від приватної інфраструктури – критично важливі державні функції виконуються через приватний сервіс, що створює ризики у випадку кіберінциденту, банкрутства або санкцій. Також це дає компанії право робити критично необхідні послуги платними. Через те, до системи

під’єднано багато користувачів та налагоджені процеси в закладах більшість лікарень змушені платити, хоча не розраховували на це.

При розгляді міжнародних аналогічних рішень було виявлено, що більшість з них велика корпоративна розробка. Станом на червень 2025 року понад 3600 лікарень у Сполучених Штатах використовують Epic EHR, що становить трохи менше 38% ринку стаціонарних МІС у країні [11,12].

Epic – це хмарне рішення для електронних медичних карток, що обслуговує низку спеціальностей. Epic EHR використовується в широкому спектрі практик, від громадських лікарень та незалежних практик до багатопрофільних лікарняних груп та постачальників хоспісної допомоги. Він пропонує стандартний набір «основних» функцій електронної медичної картки, а клініки можуть додавати модулі залежно від спеціалізації (Рис.1.3). Компанія, що розробила даний продукт, пропонує окремий портал для пацієнтів. MyChart – це портал для пацієнтів Epic, який дозволяє пацієнтам отримувати доступ до своєї медичної інформації, записуватися на прийом та спілкуватися з медичними працівниками [12].

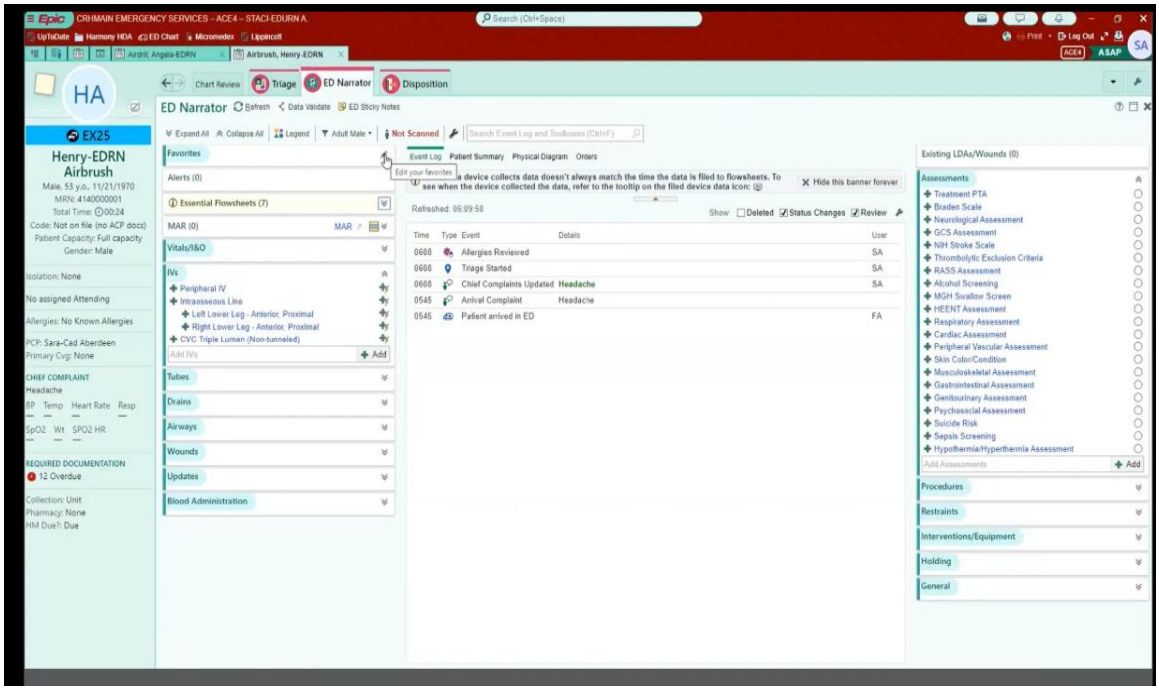


Рисунок 1.3 – Робоче місце лікаря в системі Epic

У Європі, наприклад, Dedalus ORBIS U (Рис.1.4) – провідна система стаціонарного EPR установлена в понад 1000 лікарнях та містить більше 68 інтегрованих модулів і поєднує всі клінічні/адміністративні процеси [13].

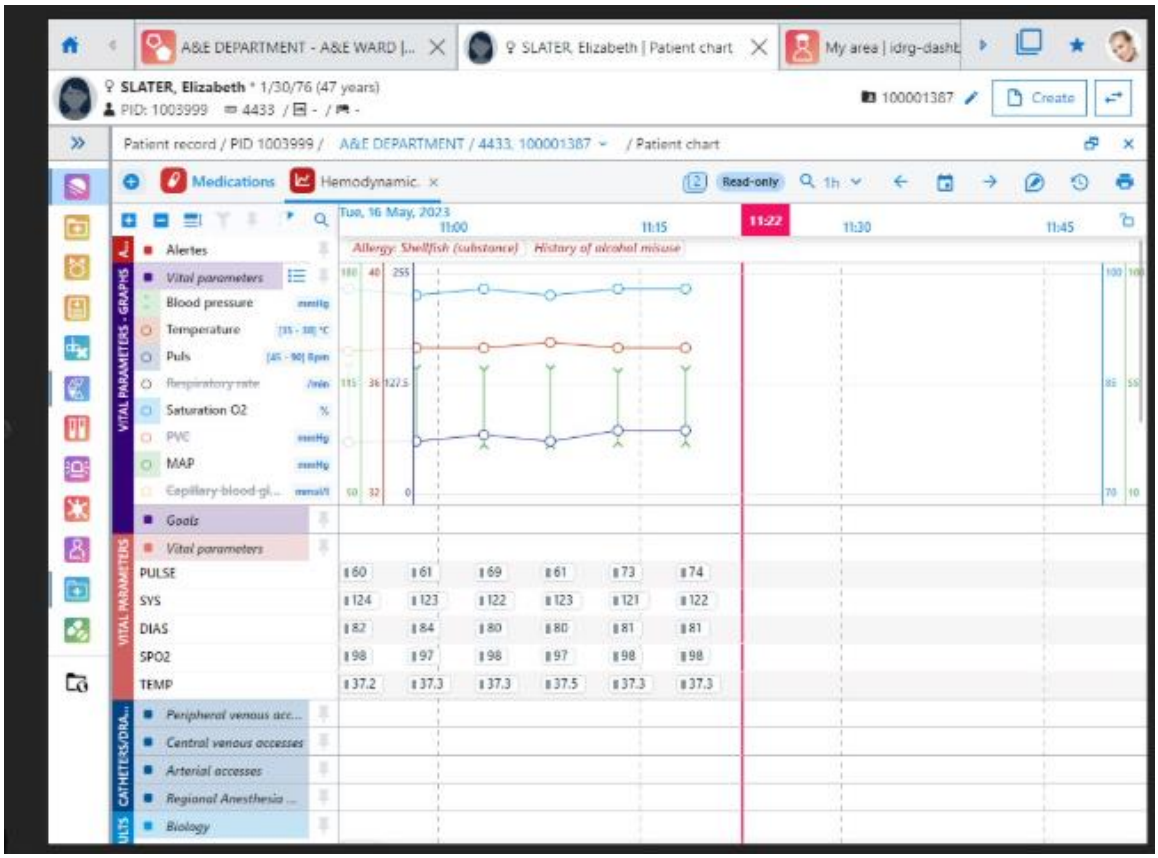


Рисунок 1.4 – Робоче місце лікаря в системі Dedalus ORBIS U

Згадані системи пропонують багаторівневий контроль доступу (різні ролі лікаря, медичної сестри, адміністратора та пацієнта), журнали аудиту змін і шифрування даних. Наприклад, у хмарному рішенні Epic, реалізовано TLS/SSL-з'єднання та шифрування даних «на місці зберігання», а доступ – за логіном/паролем і часто з двофакторною аутентифікацією. Dedalus підкреслює, що має чітку стратегію кібербезпеки з регулярними аудитами та пен-тестами.[14]

У міжнародній практиці МІС тісно інтегровані з системою медичного страхування. Саме страхові компанії є ключовими учасниками процесу обробки медичних даних, оскільки на основі електронних записів здійснюється підтвердження наданих медичних послуг та їх фінансування. Таким чином, доступ

до МІС має не лише лікар, але й страхова організація. Через це є потреба у високих вимогах до автентифікації, логування дій користувачів та шифрування даних.

Більшість зарубіжних медичних систем є локальними, тобто розгорнутими безпосередньо в медичному закладі. Це дозволяє контролювати фізичний доступ до серверів і зменшити потенційний витік інформації на сторонні ресурси. Проте такі системи орієнтовані переважно на лікарів та адміністративний персонал, тоді як прямий доступ пацієнтів є вкрай обмеженим. Це пов'язано з тим, що головною метою є забезпечення ефективності лікарів та юридичної відповідальності закладу, а не зручністю для пацієнтів.

Варто зазначити, що суворе законодавство у сфері захисту персональних даних (GDPR у ЄС, HIPAA у США) істотно впливає на функціональність таких систем та швидкість впровадження оновлень. Це відбувається тому що будь-яка додаткова функція потребує ретельної перевірки на предмет безпеки та відповідності чинного законодавства обробки персональних даних. Основний пріоритет – захист конфіденційності та недопущення несанкціонованого доступу, тоді як спрощення інтерфейсу або максимальна зручність користувачів не завжди є головною метою.

Таким чином, на відміну від України, де МІС часто мають широкий функціонал в тому числі і для пацієнтів, міжнародні системи будуються за принципом «безпека понад усе». В порівнянні з населенням України, в ЄС та США більше розповсюджена практика мати страховий поліс. Тому МІС краще інтегруються із страховими структурами, що посилює вимоги до автентифікації, контролю доступу та аудиту дій користувачів.

1.3 Загрози для управління доступом в МІС та способи їх реалізації

Безпека МІС значною мірою визначається тим, наскільки ефективно реалізовано контроль і розмежування доступу, що забезпечує роботу з персональними медичними даними лише уповноваженим особам. Проте процеси управління доступом можуть піддаватися різноманітним загрозам з боку

зловмисників, які намагаються обійти або порушити засоби захисту. У сфері інформаційної безпеки під загрозою розуміють потенційну подію або дію, здатну завдати шкоди системі та порушити ключові принципи:

- Конфіденційність означає, що доступ до інформації мають лише уповноважені користувачі, а сторонні особи не можуть ознайомитися з даними.
- Цілісність передбачає збереження точності та повноти інформації, недопущення її несанкціонованої зміни чи пошкодження.
- Доступність гарантує, що інформація та системні ресурси залишаються доступними для легальних користувачів у потрібний час.

Найочевиднішою загрозою для даних принципів та роботи системи в цілому є слабкі паролі або токени. Це вразливості в безпеці системи, що пов’язані з ненадійними методами автентифікації: використанням простих паролів, повторним застосуванням токенів чи відсутністю багатофакторного захисту. Це дозволяє зловмиснику легко отримати доступ до облікових записів і конфіденційних медичних даних.

Під час несанкціонованого доступ у МІС зловмисники можуть використовувати технічні вразливості чи людські помилки для спроби або успішного отримання доступу до системи, мережі, застосунку або бази даних без згоди адміністратора безпеки. Такі дії здатні призвести не лише до витоку інформації про пацієнтів, але й до модифікацій електронних записів, що безпосередньо впливає на якість лікування. Окрім цього, наслідками можуть стати фінансові втрати закладу, юридична відповідальність та суттєве погіршення репутації [15].

З несанкціонованого доступу випливає ще одна важлива загроза – крадіжка даних. У медичних інформаційних системах це може означати викрадення персональних відомостей пацієнтів, результатів обстежень, страхових даних чи облікових записів. Отримана інформація використовується для шахрайських дій: незаконного отримання медичних послуг або страхових виплат, що несе значні фінансові та репутаційні ризики для закладів охорони здоров’я.

Отримання зайвих прав/неправильні ролі – ситуація, коли користувач має ширший доступ, ніж вимагає його роль. Це може бути наслідком неправильно налаштованих політик доступу і призводити до небажаних дій з боку користувачів системи. До прикладу пацієнт може не тільки переглядати, а й змінювати дані про виписку рецептів.

Викрадення сесій/підміна особистості – несанкціоноване використання активної сесії легального користувача для виконання дій від його імені. Навіть без знання пароля зловмисник може отримати доступ до тих самих можливостей, що і авторизований користувач. Через це неможливо коректно відрізнити дії зловмисника від дій користувача й ускладнює реагування.

Окрім вище перечислених ризиків, МІС піддаються й іншим значним загрозам: відмовам у наданні послуг, підключення вразливого стороннього ПЗ та медичних пристроїв, витокам інформації через неправильну конфігурацію хмарних сховищ тощо. Ці загрози можуть реалізовуватися різними методами: розповсюдження шкідливого ПЗ, використання поширених вразливостей та загроз, фішинг та соціальна інженерія, помилки в конфігурації й недбалість персоналу. Проте в межах цієї роботи основна увага приділяється саме питанням управління та контролю доступу.

Таким чином виділяють декілька найбільш розповсюджених способів реалізації загроз, що пов’язані з управлінням доступу:

- Атаки грубою силою (brute-force): автоматизовані спроби підбрати логін/пароль шляхом перебору варіантів. Зловмисник запускає скрипт, який підбирає паролі до облікового запису лікаря та після вдалої спроби хакер заходить у систему, маючи доступ до електронних медичних карток. Такому виду атак можна запобігти шляхом блокування облікового запису після кількох невдалих спроб входу, використання багатофакторної автентифікації та надсилання сповіщень адміністратору безпеки про невдалу спробу входу.
- Фішинг та соціальна інженерія: обман користувачів з метою отримання конфіденційної інформації, такої як логіни, паролі або інші дані для доступу

до системи. Наприклад, співробітник отримує електронний лист, нібито від IT-відділу, із проханням оновити пароль і вводить свої дані на підробленому сайті. Зловмисник у результаті отримує доступ до облікового запису молодшого медичного персоналу або лікаря. Щоб перешкодити такому виду атак треба навчати персонал правилам кібергігієни та використовувати системи фільтрації підозрілих листів чи запитів.

- Спуфінг – це атака, за якої зловмисник видає себе за користувача системи для обходу контролю доступу. Спуфінг можна реалізувати за допомогою підміни IP-адреси, електронної пошти або сертифіката сервера, що дозволяє отримати доступ до електронних медичних карток або адміністративних функцій. Для захисту використовують багатофакторну автентифікацію, перевірку сертифікатів, шифрування мережевого трафіку та моніторинг аномальної поведінки.
- Міжсайтовий скриптинг (XSS): вразливість веб-застосунків, яка дозволяє зловмиснику вставляти шкідливий код у сторінки, якими користуються користувачі. Часто дані типи атак застосовують на формах авторизації в систему. Запобігти XSS можна шляхом правильного очищення та кодування всіх вхідних даних і регулярного тестування веб-застосунків на наявність вразливостей.
- Атаки за допомогою фізичного доступу означають, що зловмисник отримує фізичний доступ до пристроїв, що містять чутливу медичну інформацію. Запобігання включає шифрування даних в системі, контроль фізичного доступу, використання замків, біометрії і систем відеоспостереження.
- Загроза атаки MITM (людина посередині) полягає в тому, що зловмисник перехоплює та змінює трафік між користувачами та системою без їхнього відома. Перешкодити зловмисникам можна шляхом шифрування трафіку (TLS/HTTPS), використання VPN та моніторингу аномальної активності.

- SQL-ін’єкції це вид загроз, коли зловмисник вводить шкідливі SQL-команди у форми або запити системи для отримання несанкціонованого доступу до бази даних. Захист включає валідацію та екранування введених даних.
- Атаки на логіку застосунку означають що зловмисник використовує особливості роботи системи для обходу правил і отримання надмірних прав або доступу до даних. Запобігання включає ретельне тестування бізнес-логіки, перевірку валідації даних і обмеження прав користувачів відповідно до їхньої ролі.

1.4 Нормативно-правова база у сфері безпеки медичних даних

Обробка персональних даних в Україні, зокрема чутливої медичної інформації, суворо регламентована законодавчою базою, що забезпечує реалізацію конституційного права на невтручання в особисте життя. Ключовим регулятором є Закон України «Про захист персональних даних» (2010). Він встановлює загальні вимоги до обробки персональних даних (збирання, зберігання, використання), визначає відносин між суб'єктами даних (пацієнт) та суб'єктами обробки (медичний заклад, МІС) та вимагає чіткої згоди пацієнта. Закон прямо відносить дані про здоров’я до категорії чутливих (Стаття 7), забороняючи їх обробку без згоди, за винятком випадків, необхідних для лікування та функціонування ЕСОЗ. Медичні дані про особу Закон «Про інформацію» відносить до інформації з обмеженим доступом, а саме до її виду — конфіденційної інформації (Стаття 21).

При цьому, традиційна Комплексна система захисту інформації з її атестацією, яка раніше була обов'язковою для МІС, що інтегровані з ЕСОЗ, поступово відходить від класичної моделі. Держава, в особі Держспецзв’язку, активно впроваджує нову модель захисту, засновану на оцінці ризиків та міжнародних стандартах (до прикладу NIST). Цей перехід затверджений постановою Кабінету Міністрів від 18 червня 2025 року про «Деякі питання захисту інформаційних, електронних комунікаційних, інформаційно-комунікаційних, технологічних

систем». Ці зміни відповідають курсу на адаптацію національного законодавства до вимог Загального регламенту ЄС про захист даних (GDPR), що є стратегічно важливим для євроінтеграції.

Цей новий підхід передбачає декларування відповідності систем захисту інформації на основі цільових профілів безпеки. За наказом Адміністрації Держспецзв'язку від 30.06.2025 «Про затвердження базового профілю безпеки системи, де обробляється відкрита або конфіденційна інформація» був сформований базовий профіль безпеки системи, де обробляється відкрита або конфіденційна інформація. Ключові вимоги зосереджені на запобіганні несанкціонованому доступу та забезпеченні цілісності даних:[16]

- Мінімізація повноважень (№5): надання користувачам лише авторизованого, мінімально необхідного доступу для виконання завдань.
- Розмежування обов'язків (№4): Визначення обов'язків, які потребують розмежування, та встановлення правил авторизації.
- Блокування пристрою, сеансу (№10, №11): Автоматичне блокування пристрою після періоду бездіяльності, що не перевищує певного короткотривалого проміжку часу, або завершення сеансу.
- Багатофакторна автентифікація (№35): Упровадити багатофакторну автентифікацію для доступу до облікових записів системи.
- Парольна політика (№38, №40): Паролі мають бути 12-символьні та складатися з різних регістрів, цифр та спецсимволів, захищені криптографічними ключами, а їх зміна має відбуватися не рідше одного разу на 180 днів.
- Криптографічний захист (№72): Реалізація механізмів криптографічного захисту для запобігання несанкціонованого доступу та розкриття конфіденційної інформації.
- Реєстрація та аналіз (№19, №23): Визначення, які події реєструються, та обов’язок щонайменше щотижня переглядати записи аудиту на предмет незвичної діяльності.

- Захист аудиту (№26): Захист інформації аудиту від несанкціонованого доступу, зміни та видалення.
- Антивірусний захист (№80): Впровадження та щотижневе оновлення механізмів захисту від шкідливого коду.
- Виправлення дефектів (№79): Встановлення оновлень безпеки протягом 30 діб після їх виходу.

GDPR поширюється на будь-яку організацію, що обробляє персональні дані громадян ЄС, і передбачає низку ключових принципів. Зокрема, мінімізація даних: обсяг зібраної інформації має бути «адекватним, релевантним і обмеженим тим, що необхідно» для мети обробки. Обробка «спеціальних» даних (зокрема відомостей про здоров'я) заборонена за замовчуванням (ст. 9 GDPR) і може відбуватися лише за винятковими умовами, наприклад, на основі чіткої згоди суб'єкта або для надання медичної допомоги. GDPR також надає широкі права суб'єктам даних: зокрема право на доступ, виправлення та стирання своїх даних («право бути забутим»), право на перенесення даних, заперечення проти автоматизованого прийняття рішень тощо [17,18,19].

У США захист медичної інформації забезпечується передусім законом HIPAA (1996) і супутніми правилами. HIPAA націлена виключно на персональні медичні дані пацієнтів і застосовується до закладів охорони здоров'я та пов'язаних осіб і організацій. Основна вимога – використання даних дозволено лише у зв'язку з лікуванням, оплатою послуг та операціями системи охорони здоров'я, за згодою пацієнта або за іншим юридичним підґрунтям. При цьому ключовим є принцип «мінімально необхідного»: організація має використовувати/запитувати лише ті дані, які є мінімально необхідними для конкретної метиhhs.gov. HIPAA також передбачає технічні, фізичні та адміністративні заходи захисту: шифрування, контроль доступу, політики безпеки, навчання персоналу тощоhhs.gov. У пацієнтів є право отримувати копії своїх медичних даних і виправляти їх, але відповідних «прав на стирання» немає [20].

Висновок

У результаті проведеного аналізу бізнес-процесів у медичній сфері та врахування зростаючої цифровізації в охороні здоров’я було здійснено комплексне дослідження ключових аспектів інформаційної безпеки в медичних інформаційних системах.

У ході дослідження визначено функціональну роль МІС у сучасній архітектурі електронної системи охорони здоров’я. ІС було класифіковано за категоріями користувачів, масштабом охоплення, способом розгортання та архітектурним підходом. Для кожної групи розглянуто характерні ризики, специфічні вразливості та особливості застосування заходів безпеки.

Проведено порівняльний аналіз провідної української МІС Helsi та міжнародних платформ Epic та Dedalus. Виявлено, що, незважаючи на використання рольової моделі контролю доступу, зазначені системи мають низку суттєвих недоліків: обмежені механізми автентифікації, недостатню прозорість аудитів доступу для пацієнтів, більше зосередження на потребах лікарів а не пацієнтів тощо. Ці фактори створюють потенційні ризики порушення конфіденційності та знижують рівень довіри користувачів до систем.

Аналіз нормативно-правової документації показав, що Україна перебуває на етапі переходу від застарілої моделі комплексної системи захисту інформації до сучасної ризик-орієнтованої парадигми, яка ґрунтується на міжнародних стандартах, зокрема NIST та GDPR.

На основі можливих загроз ІС проведено їх відповідність до базових принципів інформаційної безпеки: конфіденційності, цілісності та доступності. Для кожної групи загроз описано потенційні сценарії реалізації атак та визначено механізми протидії.

Результати дослідження підтверджують, що подальше підвищення рівня безпеки МІС є складним у межах традиційних підходів. Визначено, що ключовим елементом забезпечення конфіденційності та цілісності даних є ефективне управління доступом. Тому надалі буде проведено аналіз принципів побудови,

методів реалізації та моделей контролю доступу, що дозволить сформувати персоналізовані політики безпеки для різних категорій користувачів, підвищити рівень довіри пацієнтів та забезпечити відповідність міжнародним стандартам інформаційної безпеки.

2. МЕТОДИ ТА МОДЕЛІ УПРАВЛІННЯ ДОСТУПОМ У МЕДИЧНИХ ІТ: АНАЛІЗ, ВИБІР ТА ОБҐРУНТУВАННЯ

2.1 Принципи та методи автентифікації, авторизації та контролю доступу

2.1.1 Основні складові управління доступом

Управління доступом у медичних інформаційних системах є ключовим елементом забезпечення безпеки пацієнтських даних і підтримки конфіденційності інформації. Воно включає комплекс процедур і механізмів, що визначають, хто і за яких умов може отримувати доступ до конкретних ресурсів системи. Основні складові управління доступом включають ідентифікацію, автентифікацію, авторизацію та аудит і контроль доступу. (Рис.2.1)

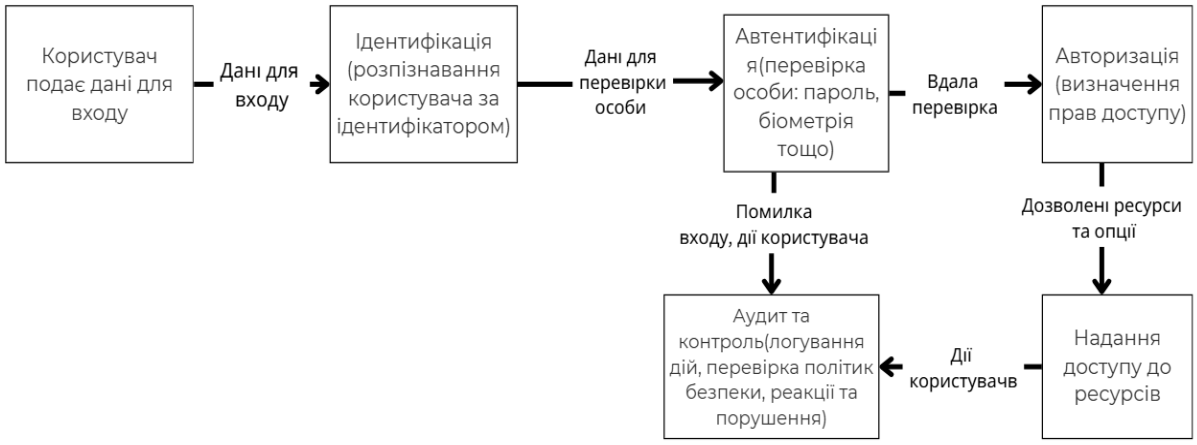


Рисунок.2.1 – Візуалізація процесу управління доступом

Алгоритм управління доступом в інформаційних системах що зображений на Рис.2.1 базується на послідовному виконанні кількох ключових процедур, що забезпечують цілісність і конфіденційність даних пацієнтів [21].

Першим етапом є ідентифікація – процедура, що дозволяє системі визначити користувача за унікальним ідентифікатором. У контексті МІС це може бути логін лікаря, штрих-код медичної карти пацієнта, службовий ідентифікаційний номер або цифровий сертифікат системного адміністратора. На цьому рівні відбувається

лише розпізнавання користувача за його ідентифікатором, без підтвердження його особи.

Наступним кроком є автентифікація – перевірка достовірності заявленого користувача. У медичних системах ця процедура набуває особливого значення, адже помилкова або ненадійна автентифікація може призвести до витоку конфіденційних медичних даних. Залежно від рівня захищеності застосовуються різні методи: введення пароля, використання одноразових кодів, апаратні токени або біометричні показники (наприклад, відбиток пальця чи скан обличчя). Сучасні МІС дедалі частіше комбінують кілька методів автентифікації для досягнення вищого рівня безпеки. Варто зазначити, що ідентифікаційні та автентифікаційні дані зазвичай передаються через одну форму. Однак їх обробка відбувається послідовно: спочатку система виконує ідентифікацію користувача, а вже потім – його автентифікацію.

Після підтвердження особи система переходить до етапу авторизації. Саме тут визначається, які ресурси та операції доступні конкретному користувачу. В МІС авторизація може мати рольовий характер: наприклад, лікар отримує доступ до історії хвороби своїх пацієнтів, тоді як медична сестра може переглядати лише необхідні для догляду пацієнта дані, а адміністратор – технічні параметри роботи системи. У деяких випадках застосовуються більш гнучкі підходи, наприклад атрибутивна авторизація, коли умови доступу залежать не лише від ролі, а й від контексту (часу доби, місця роботи, статусу пацієнта тощо).

Завершальним, але не менш важливим елементом є аудит і контроль доступу. У медичних ІС він полягає у постійному моніторингу дій користувачів, веденні журналів входів та операцій, перевірці відповідності встановленим політикам безпеки. Завдяки аудитам можна виявити підозрілу активність, спроби несанкціонованого доступу, а також забезпечити прозорість дій медичного персоналу. Наприклад, якщо зловмисник намагається отримати доступ до медичної картки пацієнта або до адміністративної частини системи – інцидент буде зафіксований та про нього сповіщено адміністратора безпеки. Таким чином, аудит

і контроль виступають механізмом зворотного зв’язку, що підвищує довіру до системи та гарантує дотримання нормативних вимог.

Окрім базових етапів управління доступом, у МІС також застосовуються додаткові складові, які забезпечують комплексний підхід до безпеки. До них належать:

- управління привілеями, що передбачає надання, зміну та позбавлення прав доступу, включно з реалізацією принципів найменших привілеїв, тимчасових прав та розподілу обов’язків;
- адміністрування облікових записів – створення, модифікацію та видалення облікових записів, а також управління групами користувачів і ролями;
- політики та правила доступу, які визначають, хто, коли і до яких ресурсів може отримати доступ, зокрема з урахуванням обмеження сеансів та контролю дій;
- реагування на інциденти доступу – виявлення та реагування на спроби несанкціонованого доступу або зловживань.

У сукупності ці елементи утворюють цілісний механізм управління доступом, який дозволяє медичним інформаційним системам поєднувати зручність роботи для медичного персоналу з високим рівнем захисту даних пацієнтів.

2.1.2 Основні принципи управління доступом

Отже, базові складові управління доступом – ідентифікація, автентифікація, авторизація та аудит – визначають логічну послідовність процесів, завдяки яким медична інформаційна система встановлює, хто є користувачем, підтверджує його особу, надає доступ до необхідних ресурсів та контролює виконані дії. Проте самі по собі ці механізми ще не гарантують достатнього рівня безпеки. Ефективність їх застосування залежить від того, за якими правилами і підходами вони реалізуються.

У міжнародних стандартах інформаційної безпеки, наприклад ISO/IEC 27002:2022, ці правила формуються у вигляді конкретних вимог (що потрібно

зробити), її мети та того як можна цю вимогу реалізувати. Наукова спільнота, узагальнюючи положення цих документів, виділяє низку ключових принципів управління доступом, які слугують універсальною основою для побудови безпечних систем у різних галузях, зокрема в медичних інформаційних системах. Під час дослідження було розглянуто лише деякі з них: принципи найменших привілеїв, необхідності знати, необхідності використовувати, розподілу обов'язків, контролю дій.

Принцип найменших привілеїв (Least Privilege) передбачає, що користувач отримує лише ті права та доступи, які мінімально необхідні для виконання його службових обов’язків та надання прав лише тим, хто компетентний і тільки на час, коли це потрібно;. Це означає, що навіть адміністратор або інший привілейований користувач не має постійного повного доступу до всіх ресурсів системи без потреби. Такий підхід зменшує ризик несанкціонованих дій, помилок та зловживань, підвищує безпеку даних пацієнтів [22, п. 8.2].

Принцип необхідності знати (Need-to-know) передбачає надання користувачу доступу лише до тієї інформації, яка необхідна йому для виконання конкретних завдань. Це означає, що різні ролі користувачів передбачають різний доступ: працівник клініки може бачити медичні дані пацієнта, необхідні для лікування, але не отримує доступ до фінансової інформації, яка йому не потрібна. Такий підхід зменшує ризик витоку конфіденційних даних, забезпечує захист приватності пацієнтів.

Принцип необхідності використовувати (Need-to-use) передбачає надання користувачу доступу лише до тих ресурсів і функцій системи, які безпосередньо потрібні для виконання його завдань. Наприклад, лікар може переглядати медичні записи пацієнта, але не має права редагувати фінансові або адміністративні налаштування системи; адміністратор може створювати облікові записи, але не має доступу до клінічних даних. Такий підхід знижує ймовірність випадкових або умисних порушень, підвищує контроль за використанням системи та відповідає передовим практикам інформаційної безпеки.[22, п 5.15]

Принцип розподілу обов’язків (Separation of Duties, SoD) передбачає розмежування критичних функцій між кількома користувачами чи ролями для зниження ризику зловживань, несанкціонованої модифікації даних, помилок і шахрайства. Його сутність полягає у тому, що жодна особа не повинна мати повний контроль над усіма етапами виконання важливої операції. У контексті MIC це означає, що, наприклад, лікар може внести призначення пацієнтові, але підтвердження видачі ліків виконує тільки фармацевт.[22, п.5.3]

Принцип обмеження сеансів (Session Limitation) передбачає, що користувач не повинен мати безконтрольного або необмеженого доступу до системи. Доступ до ресурсів завершується автоматично після встановленого часу неактивності, що зменшує ризик несанкціонованого використання облікового запису. Такий підхід підвищує безпеку даних пацієнтів, забезпечує контроль за роботою користувачів.[23, AC-12]

Принцип контролю дій (Accountability) передбачає, що всі дії користувача в системі повинні бути зафіксовані та однозначно пов’язані з його обліковою ідентичністю. У медичних інформаційних системах це означає, що кожна зміна даних пацієнта, призначення ліків чи адміністративна дія повинні реєструватися в журналах подій із зазначенням, хто, коли і яку операцію виконує. Такий підхід забезпечує прозорість дій користувачів, спрощує розслідування інцидентів, знижує ризик зловживань і помилок.[23, AU]

2.1.3 Технології реалізації процесу управління доступом

Для забезпечення безпечного доступу в медичних інформаційних системах використовуються різні методи автентифікації, авторизації та контролю дій користувачів. Вони реалізують описані принципи управління доступом і дозволяють ефективно захищати конфіденційні дані пацієнтів.(Рис.2.2)

Рисунок 2.2 – Загальні методи управлінням доступу

У МІС найпростішим методом автентифікації є вхід за допомогою пароля. Введений користувачем пароль хешується і порівнюється з базою паролів у системі. Додатковий рівень безпеки (двофакторна автентифікація) забезпечується одноразовим чи QR-кодом, що генеруються мобільними додатками чи токенами та діють обмежений час.

Більш надійним підходом є багатофакторна автентифікація (MFA), яка комбінує декілька факторів:

- Фактор знання (пароль, секретне слово, PIN-код) – це інформація, яку знає лише користувач і яка використовується для входу в систему.
- Фактор володіння (смартфон, карта, токен, SIM-карта) – це річ або пристрій, який є у користувача і підтверджує його особу.
- Фактор властивості, щось, що є частиною нас – біометрія. Біометричні методи – відбиток пальця, розпізнавання обличчя чи голосу – дозволяють точно підтвердити особу користувача, хоча потребують захищеного зберігання шаблонів.

Окрему категорію становлять методи на основі пристроїв, зокрема використання TLS-сертифікатів або мобільних ключів, які дозволяють перевірити не лише користувача, а й його пристрій за допомогою криптографії. Застосування таких методів у комплексі гарантує, що доступ до медичних даних отримує лише авторизований користувач, а ризик несанкціонованого входу значно знижується.

Узагальненою групою методів авторизації є моделі контролю доступу. Ключовими для МІС серед них є рольова, на основі атрибутів та на основі політик, мандатна. У рольовій моделі (RBAC) права користувача визначаються його роллю (лікар, молодший медичний персонал, адміністратор), що забезпечує простоту адміністрування, але має обмежену гнучкість. Модель на основі атрибутів (ABAC) враховує характеристики користувача, ресурсу та контексту (відділення, час доступу), дозволяючи створювати детальні правила, проте ускладнюючи управління.

Керування доступом на основі політик (PBAC) передбачає визначення доступу через централізовані правила, а не лише ролі чи атрибути. Часто PBAC реалізується у форматі Policy-as-Code, коли правила записуються як код і зберігаються в репозиторіях. При цьому адміністрування системи залишається необхідним: адміністратори формують структуру політик, перевіряють їхню відповідність вимогам і тестують зміни, переходячи від ручного призначення прав до управління життєвим циклом політик доступу.

Мандатний контроль доступу (MAC) передбачає доступ до ресурсів системи за рахунок визначення рівнів секретності, а не індивідуальних прав користувачів чи ролей. У цій моделі кожен об’єкт і користувач має певний рівень доступу, а всі операції контролюються системою відповідно до встановлених правил.

У сучасних медичних інформаційних системах авторизація виходить за межі класичних моделей і передбачає динамічний контроль умов доступу. Сегментація поділяє систему на ізольовані зони з окремими правилами, а мікросегментація деталізує цей підхід на рівні застосунків чи даних, реалізуючи принцип «необхідність знати» та ускладнюючи роботу зловмисникам. Важливим є також постійна перевірка користувача при кожному запиті до ресурсу відповідно до

концепції Zero Trust: система не довіряє навіть авторизованим користувачам без оцінки контексту (місце, час, пристрій, характер дій). Такий багаторівневий підхід критично важливий у сфері охорони здоров'я, де навіть незначний інцидент може спричинити витік чутливих даних.

Серед методів що забезпечують контроль дій користувачів і аудит є:

- Строге логування всіх дій, від перегляду медичних карток до внесення змін у рецепти та виписки, забезпечує прозору історію операцій, що дозволяє відстежувати підозрілі дії та проводити повноцінний аудит.
- Центральне зберігання журналів подій і інтеграція з системами моніторингу та аналізу активності дозволяє автоматизувати контроль та своєчасно виявляти аномалії. Також важливо передбачити незмінність журналів як гарантія того, що історія дій користувачів залишається достовірною і не піддається підробці. Саме дана технологія забезпечує цілісність, створюючи надійну доказову базу у разі інцидентів.
- Сповіщення та реагування на інциденти доповнюють логування, оскільки дозволяють оперативно виявляти небезпечні або нетипові дії. Наприклад, спроби доступу до ресурсів поза робочим часом або масові перегляди медичних карток можуть автоматично фіксуватися з відправкою повідомлення адміністратору для негайного реагування.
- Перевірка привілеїв користувачів є ще одним важливим аспектом контролю. Регулярне підтвердження того, що лікар, молодший медичний персонал або адміністратор має доступ лише до необхідних функцій та даних, дозволяє мінімізувати ризики неправомірного доступу і підтримувати ефективний аудит.

Також слід зазначити додаткові методи управління доступом, що підвищують безпеку і мінімізують ризики витоку даних:

- Just-In-Time передбачає надання користувачу прав лише на обмежений проміжок часу, достатній для виконання конкретного завдання, після чого

доступ автоматично відкликається. У МІС це може проявлятися, наприклад, у тимчасовому доступі лікаря до картки пацієнта під час консультації.

- Метод Just-Enough-Privilege забезпечує принцип мінімальних привілеїв, надаючи користувачу лише ті права, які необхідні для виконання конкретної дії. Так, молодший медичний персонал може вносити показники аналізів, але не має права змінювати діагноз пацієнта.
- Управління сесіями та обмеження спроб входу й часу автентифікації дозволяють контролювати, хто і коли працює із системою. Автоматичне завершення сесії після періоду бездіяльності або обмеження кількості невдалих спроб входу значно знижують ризик несанкціонованого доступу, особливо на робочих місцях з відкритими терміналами або планшетами.
- Політика паролів визначає вимоги до складності, довжини та періодичності зміни паролів, що унеможливлює використання простих або вгадуваних паролів і забезпечує базовий рівень захисту.

У сукупності ці методи дозволяють створити багаторівневу систему контролю доступу, яка поєднує динамічність, мінімізацію прав користувачів та постійний контроль за сесіями, що є критично важливим для захисту чутливих медичних даних.

2.2 Огляд моделей надання прав доступу

2.2.1 Рольова модель контролю доступу (RBAC)

При розгляді методів що застосовуються для управління доступом було згадано певну групу моделей контролю доступу (МКД). Їх розгляд є важливим, оскільки моделі визначають загальні принципи організації прав користувачів і дозволяють системно підходити до управління доступом, а не обмежуватися окремими методами чи точковими налаштуваннями.

МКД допомагають структурувати систему прав, забезпечувати цілісність і контроль доступу, спрощують адміністрування та дозволяють підтримувати

гнучкість у складних середовищах, таких як медичні інформаційні системи. Завдяки цьому адміністратори можуть одночасно забезпечувати безпеку, відповідність нормативним документам і можливість динамічного регулювання доступу в залежності від ролі, атрибутів користувача, політик систем тощо.

Таким чином МКД стають ключовим інструментом для організованого і контрольованого управління доступом, надаючи основу для впровадження як традиційних, так і сучасних методів захисту та аудиту.

Контроль доступу на основі ролей (Role-Based Access Control, RBAC) – це поширена модель авторизації, яка оптимізує керування правами користувачів шляхом об’єднання дозволів у ролі. Роль – це, по суті, набір дозволів, що визначають, які дії користувач може виконувати з певними ресурсами.[24]

Наприклад, користувач якому надали роль «Лікар» може мати право створювати та редагувати медичні записи, «Молодший медичний персонал» – оновлювати інформацію про стан пацієнтів, а «Реєстратор» – вносити дані про прийоми.

Робота моделі RBAC реалізується через послідовність логічних кроків. Спочатку адміністратор визначає перелік ролей, які відповідають функціональним обов’язкам користувачів у системі, та закріплює за ними відповідні дозволи. Кожна роль відображає набір операцій, які можна виконувати над конкретними ресурсами системи (найчастіше створення, читання, редагування чи видалення даних). Далі користувачам призначаються ролі відповідно до їхніх посадових функцій. У процесі надання ролі користувачу (кожного разу під час ідентифікації та автентифікації) система перевіряє, чи входить потрібний дозвіл до набору дозволів, закріплених за роллю користувача. Якщо дозвіл на виконання операції підтверджено, дія дозволяється, а якщо ні – доступ блокується.(Рис.2.3)

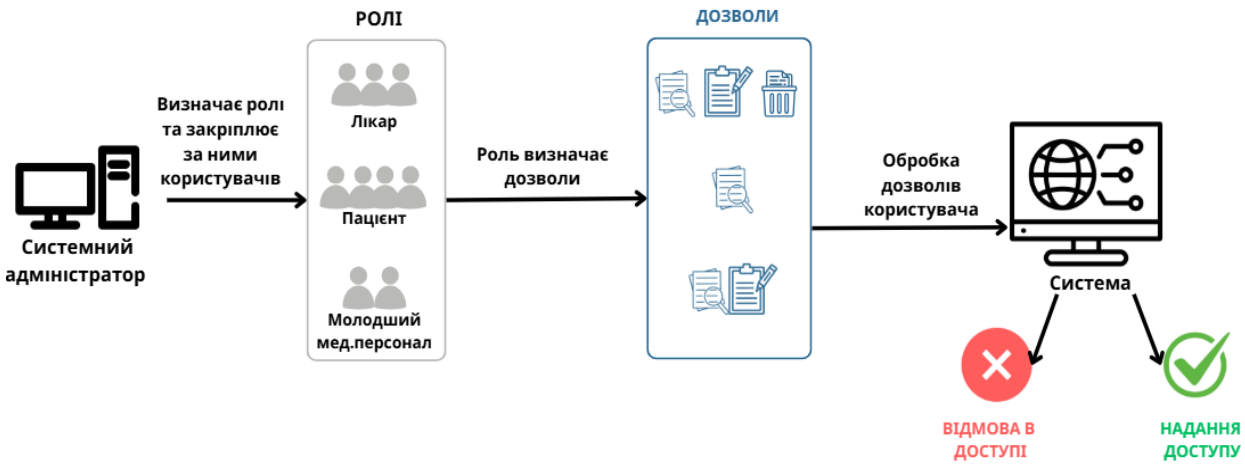


Рисунок 2.3 – Візуалізація роботи моделі RBAC

Для розпізнавання даної моделі можна виокремити декілька ключових характеристик:

- Централізованість управління – уся логіка контролю доступу зосереджена в руках системного адміністратора, який створює та налаштовує ролі відповідно до посадових обов’язків персоналу.
- Розмежування повноважень – кожна роль відповідає певному набору функцій, що знижує ризик зловживань і несанкціонованого доступу. Якщо зловмисник отримає облікові дані користувача, він все одно матиме доступ лише до обмеженого набору функцій, а не до всієї системи.
- Масштабованість – модель легко застосовується в організаціях із великою кількістю користувачів, що важливо для масштабних МІС. Це зумовлено тим, що дозволи призначаються не індивідуально, а групами.
- Успадкування дозволів – користувач автоматично отримує всі права, пов’язані з його ролями.
- Прозорість та передбачуваність – доступ завжди визначається набором ролей, що робить контроль більш зрозумілим та контрольованим.
- Відповідність організаційній структурі – ролі формуються відповідно до посадових обов’язків і робочих процесів. Такий підхід забезпечує узгодженість між політикою безпеки та реальною організаційною ієрархією,

спрощує управління доступом і робить систему інтуїтивно зрозумілою для персоналу.

Для того щоб комплексно оцінити сильні та слабкі сторони, а також визначити потенційні можливості й загрози моделі було прийняте рішення використовувати SWOT-аналіз.

SWOT-аналіз – це метод стратегічного аналізу, що дає змогу комплексно оцінити внутрішні й зовнішні фактори, які можуть впливати на ефективність функціонування системи чи організації. Його сутність полягає у дослідженні чотирьох напрямів: сильних сторін (Strengths), слабких сторін (Weaknesses), можливостей (Opportunities) та загроз (Threats):

- сильні сторони – те, що ви робите добре і що дає вам конкурентну перевагу;
- слабкі сторони – те, що ви можете покращити, щоб стати більш конкурентоспроможними;
- можливості – сприятливі фактори зовнішнього середовища, які ви можете використовувати для покращення ситуації;
- загрози – несприятливі фактори зовнішнього середовища, які можуть негативно вплинути на ваше становище. [25]

Проте даний інструмент для дослідження було вирішено модифікувати, розглядаючи не бізнес-проект, а моделі КД. Сильні сторони відображають переваги моделі, слабкі сторони – недоліки, можливості – де краще дана модель застосовується, а загрози – де модель не ефективна. Такий підхід дозволяє зрозуміти, у яких умовах модель контролю доступу на основі ролей є найбільш ефективною, а де її використання може бути обмеженим або нераціональним.

Результати SWOT-аналізу моделі RBAC відображено у вигляді схеми, представленої на рисунку 2.4.

Рисунок 2.4 – SWOT-аналіз моделі RBAC

Модель RBAC проста для розуміння та управління, що робить її зручною для адміністраторів медичних інформаційних систем. Вона добре масштабується у лікарнях та клініках із чіткою структурою персоналу. Групування дозволів у ролі знижує адміністративне навантаження і забезпечує чітке розмежування доступу, що підвищує безпеку медичних даних пацієнтів.

Недоліками даної моделі є обмежена гнучкість у динамічних середовищах, де часто змінюються обов'язки або тимчасові ролі. Керування великою кількістю ролей може ускладнюватися і стати ресурсозатратним, а для складних сценаріїв доступу може знадобитися додаткове налаштування політик.

RBAC ефективно застосовується у медичних організаціях із чіткою ієрархією персоналу, дозволяючи керувати доступом до медичних записів, результатів аналізів та модулів системи. Система спрощує автоматизацію надання та зміни прав при зміні посад чи обов'язків користувачів, що підвищує ефективність роботи персоналу. Натомість у великих або динамічних медичних організаціях ризиком є надмірна складність управління ролями, дублювання ролей та перевантаження

адміністративного персоналу. Також модель може бути неефективною при необхідності швидко реагувати на тимчасові або нестандартні сценарії доступу.

2.2.2 Обов’язкова модель контролю доступу (MAC)

Обов'язковий контроль доступу (Mandatory Access Control, MAC) – це модель, яка обмежує можливості звичайних користувачів самостійно надавати або забороняти доступ до об’єктів файлової системи. В даній моделі рішення про доступ ґрунтується на двох ключових факторах: рівні конфіденційності інформації, що міститься в об’єкті, та рівні авторизації користувача, який намагається отримати до нього доступ. Усі обмеження визначаються адміністратором і суворо контролюються операційною системою або ядром безпеки, що унеможливлює їх зміну кінцевими користувачами. [26]

Принцип дії MAC схематично відображено на рисунку 2.5

Рисунок 2.5 – Візуалізація роботи моделі MAC

Етап формування міток безпеки полягає у класифікації ресурсів (об’єктів) медичної інформаційної системи. Кожному об’єкту присвоюється рівень секретності та, за потреби, категорія безпеки (наприклад: «Обмежено», «Конфіденційно», «Секретно», «Цілком таємно»). Мітки мають ієрархічну структуру: більш високий рівень секретності включає доступ до всіх нижчих.

Далі, або паралельно з попереднім етапом, відбувається класифікація користувачів. Кожному користувачу МІС присвоюється рівень допуску, який визначає максимальний рівень секретності, з яким він може працювати. Наприклад, лікарю відділення може бути надано рівень «Конфіденційно» для даних свого

відділення, а медичному персоналу – «Обмежено» з категорією «Загальні дані пацієнтів». Варто зазначити, що MAC завжди централізовано управляється адміністратором: саме він встановлює рівні допуску для користувачів та мітки безпеки для ресурсів. Користувачі не можуть самостійно змінювати свої дозволи чи рівні доступу, навіть якщо вони є власниками ресурсу.

Перевірка політик MAC відбувається в момент, коли користувач намагається отримати доступ до об'єкта (наприклад, історії хвороби). Ядро MIC здійснює перевірку:

- чи рівень доступу користувача відповідає або перевищує рівень секретності об'єкта;
- чи об'єкт належить до категорій, дозволених цьому користувачу.

Рішення про доступ є завершальним етапом, на якому автоматично приймається рішення про дозвіл або заборону дії. Воно виконується системою без участі користувача чи адміністратора.

Після дослідження моделі MAC було створено схему SWOT-аналізу (Рис.2.6).

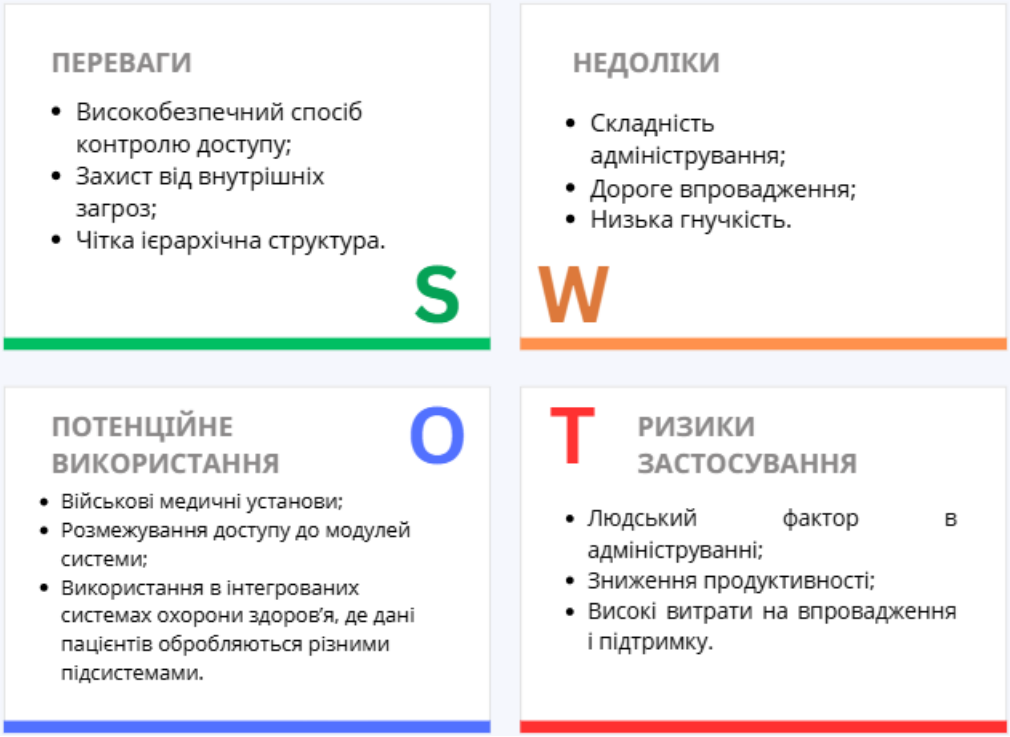


Рисунок 2.6 – SWOT-аналіз моделі MAC

Перевагами MAC відносно MIC є:

- Високобезпечний спосіб контролю доступу – система забезпечує чітке дотримання політик доступу, що знижує ризик несанкціонованого доступу до медичних даних.
- Захист від внутрішніх загроз – навіть користувачі з високими привілеями не можуть змінювати правила доступу самостійно, що запобігає зловживанням привілеями всередині організації.
- Чітка ієрархічна структура – доступ визначається ролями та рівнями секретності, що особливо важливо для медичних даних різного рівня конфіденційності.

Проте в даній моделі є і свої недоліки:

- Складність адміністрування – налаштування та підтримка політик доступу потребує спеціальних знань і часу, особливо в великих медичних установах.
- Дороге впровадження – потребує інвестицій у розробку правил доступу, інтеграцію та навчання персоналу.
- Низька гнучкість – користувачі не можуть самостійно змінювати доступ до даних, що іноді уповільнює робочі процеси.

Потенційне використання MAC необхідне там де потрібен максимальний рівень захисту даних. Наприклад, у військових медичних установах, де особлива увага приділяється секретності й збереженню конфіденційної інформації про пацієнтів. Треба зазначити що MAC ефективний для розмежування доступу до окремих модулів системи: лікар може працювати з історією хвороби, модоший медичний персонал – з планом процедур, а адміністратор – зі статистикою. Крім того, цей підхід забезпечує надійний контроль у інтегрованих системах охорони здоров’я, де дані пацієнтів обробляються різними підсистемами, запобігаючи несанкціонованому доступу між ними.

Ризики застосування MAC в MIC пов’язані насамперед із людським фактором: помилки в налаштуванні правил доступу можуть призвести до блокування правомірних дій медичного персоналу. Крім того, жорсткі політики доступу

знижують гнучкість і можуть уповільнити роботу, що негативно впливає на продуктивність. Важливим ризиком залишаються й високі витрати на впровадження та підтримку, адже система вимагає регулярного моніторингу й оновлення політик безпеки, що потребує додаткових ресурсів.

2.2.3 Модель контролю доступу на основі атрибутів (ABAC)

Контроль доступу на основі атрибутів (Attribute-Based Access Control, ABAC) – це сучасна модель безпеки, яка поступово набуває широкого поширення. Її особливість полягає у використанні атрибутів – характеристик користувачів, ресурсів та умов середовища – для прийняття рішень щодо доступу. На відміну від моделей, де права визначаються наперед (наприклад, через ролі), ABAC забезпечує динамічне керування доступом залежно від того, хто намагається отримати доступ, до чого та за яких умов. До прикладів атрибутів, які можуть враховуватися у ABAC, належать: ідентифікатор користувача, його роль або належність до відділення.[27]

Варто зазначити, що окремим різновидом ABAC деякі ресурси визначають контекстний контроль доступу (CBAC), у якому основний акцент робиться на врахуванні контексту доступу. До таких параметрів належать динамічні атрибути такі як час запиту, географічне розташування користувача, тип пристрою чи стан мережі. У медичних інформаційних системах CBAC може використовуватися, наприклад, для заборони доступу до даних пацієнта поза робочим часом або з незареєстрованого пристрою.[28] Проте оскільки контекстні атрибути це підвид то було прийнято рішення описувати загальну модель ABAC.

Кроки надання контролю доступу на основі моделі ABAC наведені на рисунку 2.7.

Рисунок 2.7 – Візуалізація роботи моделі ABAC

Алгоритм реалізується у три основні кроки. Перший крок полягає в запиті доступу до ресурсу користувачем. Користувач намагається виконати певну дію у системі, наприклад, лікар прагне переглянути електронну медичну картку пацієнта. Запит містить інформацію про користувача, ресурс, тип дії та контекст середовища, та передається на перевірку політик доступу системою.

Другий крок передбачає перевірку атрибутів користувача, ресурсу, дії та середовища:

- користувацькі атрибути включають роль, посаду, відділення та рівень доступу;
- атрибути ресурсу – тип документа, конфіденційність та стан пацієнта;
- атрибути дії – читання, редагування, видалення або створення;
- атрибути середовища – час доби, місцезнаходження користувача та стан мережі.

Система порівнює ці атрибути з політиками доступу та оцінює відповідність запиту встановленим правилам. Наприклад, перевіряється, чи належить лікар до відповідного відділення, чи дозволяє політика редагування карток певного типу та чи робиться запит у робочий час із корпоративного терміналу.

На третьому кроці система приймає рішення щодо надання або відмови у доступі. Якщо атрибути користувача та ресурсу відповідають політикам доступу, запит дозволяється, і дія виконується. У разі невідповідності запит відхиляється, а користувач отримує повідомлення про відмову.

На рисунку 2.8 зображена схема SWOT-аналізу.

Рисунок 2.8 – SWOT-аналіз моделі ABAC

Перевагою моделі ABAC перш за все є гнучкість та легке пристосування до потреб закладів охорони здоров’я. Доступ визначається на основі багатьох атрибутів (користувача, ресурсу, дії, середовища), що дозволяє точно налаштовувати правила доступу. Не менш важлива можливість масштабувати систему: легко додаються нові ролі, атрибути та політики без потреби змінювати всю модель доступу. До того ж система враховує умови запиту (час, місцезнаходження, відділення тощо), що підвищує захист конфіденційних медичних даних та контекстної безпеки.

Проте деякі переваги у великих системах стають недоліками: велика кількість атрибутів і політик ускладнює їхнє управління та підтримку, а також уповільнює роботу системи, особливо при великій кількості користувачів та ресурсів. До того ж неправильне або неповне визначення політик може призвести до помилкових рішень щодо доступу.

Потенційне використання у МІС:

- Обмеження доступу до медичних знімків, рецептів та історії лікування.
- Тимчасові або контекстні політики доступу, наприклад, доступ тільки у робочий час або з корпоративних пристроїв.
- Управління доступом для сторонніх консультантів та нових працівників на основі атрибутів.

Ризики застосування:

- Помилки у політиках доступу можуть надати доступ стороннім користувачам або заблокувати легітимних працівників.
- Велика кількість умов доступу може ускладнювати відстеження дій користувачів.
- Некоректні або застарілі атрибути призводять до помилкових рішень.
- Складні перевірки політик у реальному часі можуть створювати затримки у доступі до медичних даних.

2.2.4 Модель контролю доступу на основі політик (PBAC)

Контроль доступу на основі політик (Policy-Based Access Control, PBAC) – це модель керування доступом, у якій рішення приймаються на основі визначених політик безпеки. Політики описують умови, за яких користувач може отримати доступ до певних ресурсів, і часто поєднують бізнес-роль користувача з конкретними правилами та привілеями. Іншими словами, PBAC – це підхід, що дозволяє централізовано визначати, хто має доступ до яких ресурсів і на яких умовах, ґрунтуючись на наборах формалізованих правил (політик). [29]

Алгоритм роботи моделі PBAC зображено на рисунку 2.9

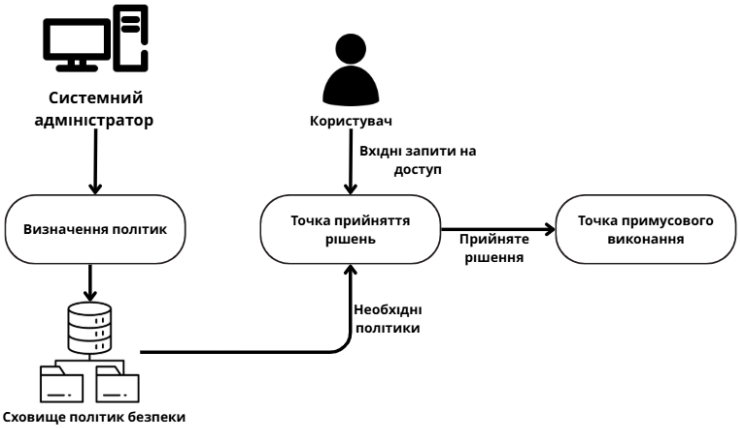


Рисунок 2.9 – Візуалізація роботи моделі PBAC

Системний адміністратор на етапі визначення політик формує правила у вигляді «якщо/тоді», які відображають логіку доступу. В подальшому саме адміністратор відповідає за управління цими політиками, а саме створення, оновлення й адміністрування правил доступу. Ці правила безпеки зберігаються не безпосередньо в коді програми, а в сховищі політик безпеки.

Точка прийняття рішень аналізує вхідні запити на доступ і порівнює їх з політиками. Далі точка примусового виконання виконує рішення точки прийняття рішень на рівні додатку чи ресурсу, дозволяючи або забороняючи доступ.[30]

Для моделі PBAC було створено схему SWOT-аналізу (Рис.2.10).

Рисунок 2.10 – SWOT-аналіз моделі PBAC

PBAC забезпечує детальний контроль доступу, оскільки дозволяє описувати умови взаємодії користувачів із ресурсами через чітко визначені політики. Це дає можливість налаштовувати доступ не лише на рівні ролей, а й врахувати контекст та бізнес-логіку, що особливо важливо у складних середовищах на кшталт МІС. Важливим аспектом є також відповідність принципу «нульової довіри», адже PBAC передбачає, що доступ надається лише за умови дотримання конкретних правил без винятків, незалежно від статусу користувача в організації. Такий підхід істотно знижує ризик внутрішніх загроз, оскільки мінімізує ймовірність несанкціонованого використання даних навіть з боку співробітників, які мають певні базові права. Крім того, PBAC створює умови для повноцінного аудиту дій, адже всі рішення щодо доступу базуються на централізованих політиках і підлягають протоколюванню. Це спрощує процес моніторингу, дозволяє відстежувати історію рішень системи та підтверджувати дотримання нормативних вимог.

Попри значні переваги, модель має певні недоліки. Найбільшою проблемою є складність розробки та підтримки політик: щоб забезпечити коректну роботу, необхідно ретельно формулювати умови доступу та згодом тестувати їх, а це потребує часу та високої кваліфікації адміністратора. Додатковим викликом є вплив на продуктивність, адже кожен запит повинен перевірятися у реальному часі. Також важливо враховувати, що перехід від традиційної RBAC до PBAC вимагає спеціального навчання для працівників та налагодження співпраці між технічним та медичним персоналом.

Використовувати дану МКД можна для реалізації динамічних політик – доступ можна надавати або обмежувати автоматично залежно від конкретних умов: часу, місцезнаходження, контексту дій користувача або стану пацієнта. Також варіантом застосування в режимі надзвичайних ситуацій – під час аварій, епідемій або критичних подій політики доступу можна швидко змінювати, щоб забезпечити оперативність роботи персоналу лікарні.

Ризики використання PBAC включають можливі помилки у політиках доступу – неправильно налаштовані правила можуть блокувати доступ або надавати його

несанкціоновано. Також система має високі вимоги до підтримки, оскільки потребує регулярного оновлення та контролю політик. До технічних ризиків належить можливе зниження продуктивності при великій кількості одночасних запитів, що критично для систем, які повинні працювати в реальному часі. Крім того, можливий організаційний опір – співробітники можуть не приймати нову систему через її складність або зміну звичних процесів роботи.

2.3 Порівняння моделей контролю доступу

Під час дослідження кожного з методів контролю доступу були виділені спільні характеристики. До них належать:

- Основний принцип – базова ідея, на якій побудована модель контролю доступу;
- Гнучкість – можливість моделі адаптуватися до змін у бізнес-процесах, контексті роботи чи вимогах безпеки;
- Управління – складність налаштування та адміністрування моделі, зокрема створення правил, ролей або політик;
- Підтримка – рівень ресурсів та навичок адміністраторів, необхідних для підтримання моделі в актуальному стані;
- Сфера застосування – тип організацій або сценаріїв, де дана модель застосовується найбільш ефективно.

За наведеними характеристиками було сформовано зведену таблицю 2.1, яка дозволяє наочно порівняти основні особливості моделей контролю доступу RBAC, ABAC, PBAC та MAC.

Таблиця 2.1 – Порівняння моделей контролю доступу

Модель	**Основний принцип**	**Гнучкість**	**Управління**	**Підтримка**	**Сфера застосування**
RBAC	Доступ за ролями користувачів	Низька - середня	Ролі розподіляються централізовано (проста)	Низькі-середні	Організації зі стабільними, добре визначеними ролями
ABAC	Доступ на основі атрибутів користувача, ресурсів та контексту	Висока	Потребує визначення атрибутів і правил (середня)	Високі вимоги	Складні системи з динамічними умовами доступу
PBAC	Доступ на основі політик, що описують правила доступу (“якщо…то”)	Дуже висока	Потребує створення та підтримки політик безпеки (висока)	Дуже високі вимоги	Де потрібна адаптація доступу під різні сценарії (наприклад, надзвичайні ситуації)
MAC	Поділ системи на рівні секретності, рівні доступу для користувачів контроль адміністратора	Низька	Адміністратор централізовано встановлює правила (висока)	Середні-високі вимоги	Високозахищені системи (військові, державні, критичні інфраструктури)

RBAC є найбільш зрозумілою моделлю, оскільки базується на ролях і дозволяє легко налаштовувати доступ у середовищах зі сталими бізнес-процесами. ABAC забезпечує більшу гнучкість, оскільки враховує атрибути користувачів і ресурсів, проте вимагає ретельного визначення правил і атрибутів. PBAC, у свою чергу, орієнтується на політики, які описують логіку доступу, що робить його надзвичайно гнучким, особливо у динамічних чи кризових ситуаціях, але створює додаткове навантаження на адміністраторів, які повинні підтримувати політики в актуальному стані. MAC використовується там, де потрібен найвищий рівень захисту, але через низьку гнучкість і жорсткі централізовані правила він рідко застосовується у комерційних системах.

Варто зазначити, що хоча PBAC і ABAC є різними моделями (PBAC керується політиками, ABAC – атрибутами), на практиці PBAC часто використовує атрибути користувачів, ресурсів або контексту, що наближає її до ABAC. Через це їх іноді плутають, але принципова відмінність залишається: у PBAC атрибути можуть виступати лише як умови для визначення політик б, тоді як в ABAC атрибути є основним механізмом прийняття рішень.

До того ж було помічено відмінність в постановці задач між цими двома моделями: ABAC визначає надання доступу, формулюючи прості правила у вигляді «якщо атрибут X відповідає умові Y, доступ дозволено», тоді як PBAC реалізує більш складні політики, включаючи винятки та додаткові умови. До прикладу, правило «Тільки авторизовані лікарі з відділення кардіології можуть переглядати записи пацієнтів цього відділення у робочий час (8:00-20:00)» відповідає моделі ABAC, а доповнення цього правила винятком «якщо пацієнт у критичному стані – доступ дозволено будь-якому лікарю незалежно від відділення та часу» демонструє реалізацію PBAC.

Гібридний підхід на основі політик, атрибутів і ролей вважається найбільш доцільним варіантом контролю доступу для МІС. У цьому випадку RBAC забезпечує сталу ієрархію ролей для доступу (лікар, молодший медичний персонал, реєстратура, адміністратор), ABAC уточнює доступ залежно від відділення, часу, статусу пацієнта чи конкретної дії користувача, а PBAC де це потрібно і не перевантажує систему централізовано керує правилами доступу, дозволяючи швидко адаптувати їх до змінних умов або надзвичайних ситуацій. Такий підхід дозволяє досягти максимальної гнучкості та прозорості, одночасно забезпечуючи швидкий і безпечний доступ персоналу лікарні до інформації.

У спрощених сценаріях для стандартної щоденної роботи МІС гібрид RBAC + ABAC залишається оптимальним варіантом: він простіший у впровадженні та підтримці, дозволяє ефективно розподіляти права доступу за ролями та уточнювати їх атрибутами, забезпечуючи баланс між безпекою та зручністю роботи.

Висновок

У цьому розділі було проведено комплексний аналіз методів та моделей управління доступом у медичних інформаційних системах з метою визначення методу, що забезпечує необхідний рівень безпеки медичних даних. Розглянуто базові принципи автентифікації, авторизації та контролю доступу, досліджено їх значення для побудови безпечної інфраструктури медичного IT-середовища.

Було проаналізовано чотири основні моделі управління доступом: RBAC, ABAC, PBAC, MAC). Для кожної моделі проведено SWOT-аналіз. Це дозволило оцінити їх сильні та слабкі сторони, можливості інтеграції в медичні системи та потенційні ризики застосування. У розділі наведено зведену порівняльну таблицю моделей за ключовими критеріями: основний принцип, гнучкість, складність управління, потреба в подальшій підтримці, сфера застосування.

Результати аналізу показали, що жодна з традиційних моделей у чистому вигляді не забезпечує достатнього рівня гнучкості та відповідності специфічним вимогам медичної галузі. Проведений аналіз підтвердив, що комбінована модель управління доступом (RBAC+ABAC+PBAC) є найбільш перспективною для медичних інформаційних систем, оскільки поєднує переваги традиційних підходів та нівелює їх недоліки.

Отримані результати сформували теоретичну основу для розробки концепту комбінованої моделі контролю доступу. На наступному етапі дослідження буде здійснено комплексне проєктування МІС шляхом візуалізації розподілу функціоналу системи за ролями та потоків даних, опису політик доступу, побудови логічної моделі бази даних та визначення ключових параметрів інформаційної безпеки.

3. МОДЕЛЮВАННЯ КОМПОНЕНТІВ УПРАВЛІННЯ ТА КОНТРОЛЮ ДОСТУПУ МІС

3.1 Методологічні основи та інструменти моделювання системи

Для подальшого розширення теми управління та контролю доступу в МІС варто зосередитись на моделюванні архітектури майбутньої розробки. Це дозволить комплексно оцінити взаємозв'язки між її компонентами, способи обробки та зберігання даних, а також механізми реалізації політик доступу. Таке моделювання покаже структурну логіку побудови системи, допоможе виявити потенційні вразливості на ранніх етапах проектування та забезпечить основу для подальшого впровадження механізмів безпеки керування доступом користувачів до чутливої медичної інформації.

Варто зазначити, що в попередній (бакалаврській) роботі було спроектовано один з модулів МІС – «Медична карта», де відображались основні функції роботи з медичною картою пацієнта: створення, редагування, збереження, видалення записів пошук і перегляд медичних карт тощо. Тоді частково була показана модель RBAC, але з точки зору функцій системи, а не її захисту.

Подальший аналіз показав необхідність удосконалення архітектури системи через комбінування ролей із політиками доступу, що враховують атрибути користувачів, контекст виконання дії та тип даних (PBAC+ABAC). Такий підхід забезпечить більш гнучке та динамічне керування доступом, дозволяючи не лише визначати роль користувача, а й оцінювати додаткові параметри, наприклад, місце виконання запиту, тип операції, час доби або критичність ситуації. Це, у свою чергу, підвищить рівень конфіденційності, цілісності та безпеки даних у системі.

Тож, продовжуючи удосконалення даного модулю, було прийнято рішення зосередитись саме на аспектах управління доступом у модулі «Медична карта». Розгляд одного модулю, а не всієї системи не обмежує дослідження, адже в подальшому буде враховано можливість масштабування архітектури, розширення

системи новими модулями та інтеграції з іншими компонентами (модулями страхування, аптечними підсистемами тощо).

Для цього в подальшому моделюванні буде використано діаграми UML, DFD та блок-схеми, які забезпечать наочне представлення:

- логіки функціонування програми;
- потоків даних між модулями;
- структури бази даних та зв’язків між сутностями.

Таким чином, архітектура системи буде описана багаторівнево – від логічного до структурного та фізичного рівнів. Архітектура системи на логічному рівні описує абстрактну структуру та функціональність, а на фізичному – конкретну реалізацію, що включає апаратне забезпечення та його зв'язки. Логічна архітектура фокусується на тому, що система робить і як її компоненти взаємодіють для досягнення мети, тоді як фізична архітектура пояснює, як система побудована та які реальні елементи забезпечують її функціонування.

3.2 Сценарії взаємодії користувачів із модулем «Медична карта»

3.2.1 Діаграма прецедентів та опис сценаріїв взаємодії

Діаграма прецедентів (варіантів використання) – це один із основних інструментів аналізу та проектування систем у методології Unified Modeling Language (UML). Вона описує, як зовнішні користувачі чи системи (актори) взаємодіють із системою через певні функції – прецеденти.

Прецедент (Use Case) – це функція або послідовність дій, яку система або її частина може виконати, взаємодіючи з зовнішніми користувачами або системами.

Актори – це групи користувачів з певними ролями. Акторами можуть бути люди, групи людей, частини інших системНаприклад, прецедент «Редагування електронної медичної карти (ЕМК)» прив’яне до актору Лікар, проте актор Лікар може представляти різні спеціалізації (педіатр, хірург, терапевт). Взаємодія акторів з прецедентами означає, що актори ініціюють або спричиняють виконання певних

прецедентів, визначаючи контекст, у якому система діє. Актори це узагальнене поняття для групи користувачів.

Не менш важливим компонентом цієї діаграми є відношення. В даному виді діаграм існують 4 типи відношень: [2, 31]

- асоціації (англ. association relationship) – загальне відношення між актором та прецедентом;
- включення (англ. include relationship) – відношення, яке показує на розширення загального прецедента, які обов'язково мають бути задіяні;
- розширення (англ. extend relationship) – відношення, яке показує розширені дії до загальної функції, які можуть ігноруватись акторами;
- узагальнення (англ. generalization relationship).

Під час розробки бакалаврської роботи вже було змодельовано діаграму варіантів використання (див. Додаток А), яка відображала взаємодію основних акторів системи: Гість, Пацієнт, Лікар та Сімейний лікар. Окремо була побудована діаграма для серверної частини. Такий підхід дозволяв продемонструвати навички побудови базової функціональної структури системи та її взаємодії з користувачами. Однак початкова діаграма мала певні обмеження – вона була перевантаженою і не охоплювала всіх ролей, що мають значення для повноцінного функціонування МІС.

У рамках магістерського дослідження акцент зміщено на аспекти управління безпекою системи, тому діаграму було доповнено актором Адміністратор. Цей актор відповідає за підтримку роботи системи, управління доступом користувачів, моніторинг безпеки та контроль коректності функціонування серверної частини (Рис. 3.1).

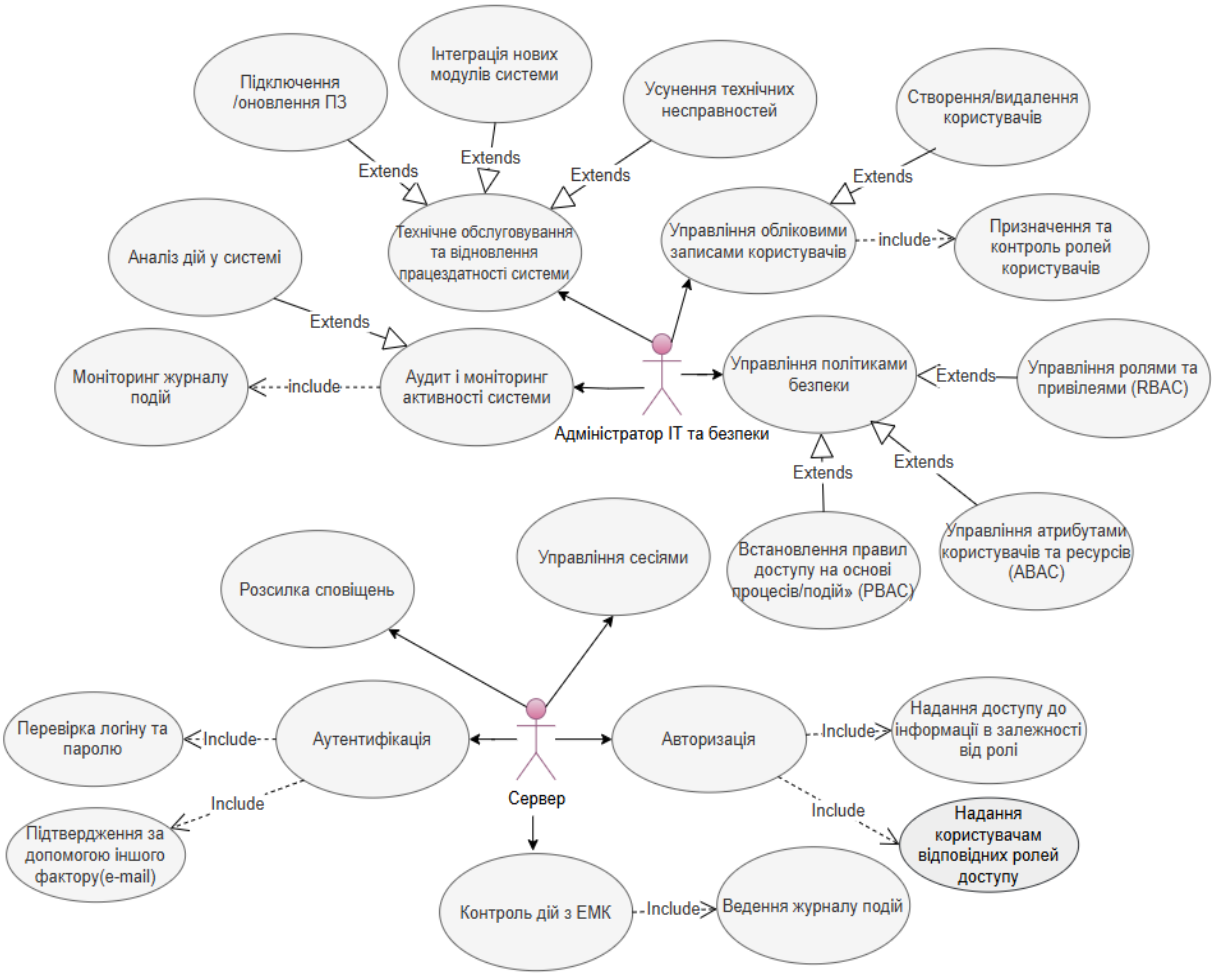


Рисунок 3.1 – Діаграма прецедентів модулю адміністрування системи

На цій діаграмі прецедентів показано двох основних акторів: Адміністратор IT та безпеки і Сервер. Обидва виконують технічні ролі, пов’язані з підтримкою стабільної роботи системи, контролем безпеки, автентифікацією та керуванням доступом користувачів. Тут акцент робиться не лише на функціональній взаємодії користувачів, а й на технічному рівні безпеки. Тобто, діаграма відображає, як система забезпечує свій захист зсередини – через функції адміністратора та сервера. До того ж показаний розподіл відповідальності між людиною (адміністратором) і машиною (сервером).

Адміністратор відповідає за підтримку, контроль та управління безпекою системи. Основні прецеденти, пов’язані з ним:

- Технічне обслуговування та відновлення працездатності системи охоплює підключення та оновлення ПЗ, інтеграцію нових модулів, усунення

несправностей техніки. Це забезпечує стабільну роботу системи без збоїв, що критично для швидкого реагування у випадку екстреної ситуації

- Аудит і моніторинг активності системи включає обов’язковий моніторинг журналу подій та аналіз дій користувачів за потреби. Моніторинг і періодичний аудит системи дозволяє виявляти аномалії: спроби несанкціонованого доступу, порушення політик безпеки, раптове зростання навантаження, зависання сервісів. Контроль над тим, хто і що може робити в системі, є базою для реалізації принципу "мінімально необхідних привілеїв".
- Управління політиками безпеки – визначення правил доступу до даних, ролей і атрибутів. Саме цей блок відображає реалізацію різних моделей контролю доступу (RBAC, ABAC, PBAC). Це показує, що система гнучко налаштовується під політику безпеки медичного закладу, а не лише працює за фіксованими ролями.

Сервер – актор, що ототожнює механізми та людей, які їх підтримують. Сервер загалом виконує важливі завдання для забезпечення працездатності та ефективності розробки. Сервер виступає як технічний актор, який автоматично реалізує певні процеси:

- Аутентифікація включає в себе перевірку логіну і паролю, підтвердження особи за допомогою іншого фактору (в даному випадку email).
- Авторизація – це прецедент, що відображає надання доступу до інформації в залежності від ролі. Варто зазначити, що в процесі авторизації відбувається надання користувачам відповідних ролей. Цей прецедент схожий на прецедент Адміністратора «Призначення та контроль ролей користувачів». Проте вони мають принципову відмінність:адміністратор формує політику доступу, а сервер забезпечує її дотримання під час роботи системи. Аутентифікація та авторизація це ядро безпеки ІС.

- Контроль дій з ЕМК – ведення журналу подій усіх операцій з електронними медичними картками. Ця автоматична дія забезпечує простежуваність змін даних, що необхідно для моніторингу та аудиту системи.
- Управління сесіями – це функція, що уособлює контроль активних підключень і тайм-аутів. Обмеження сесій – це додатковий рівень безпеки, який запобігає несанкціонованому використанню системи при залишеній відкритій сесії.
- Розсилка сповіщень – інформування користувачів або адміністратора про важливі події. Сюди можна віднести як сповіщення про вільні прийоми у лікаря для пацієнта чи неавторизованого користувача, так і про виявлення збоїв чи підозрілої активності для адміністратора системи. Це підвищує не тільки зручність користування системою, а й оперативність реагування на події в ній.

Після розгляду потреб користувачів було змодельовану відповідну діаграму (Рис.3.2).

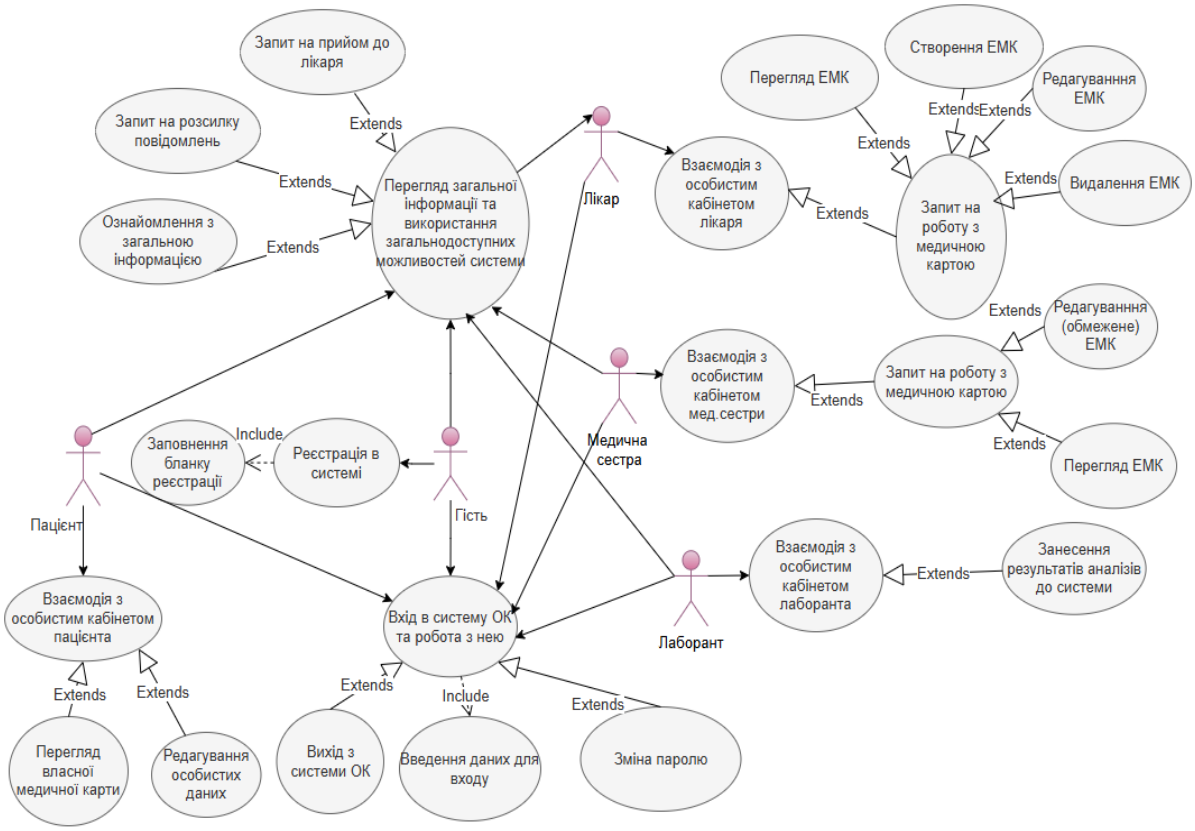


Рисунок 3.2 – Діаграма прецедентів користувацького рівня системи

На цій діаграмі прецедентів показано користувачів системи з різними ролями, що відповідають своїм акторам – Гість, Пацієнт, Лікар, Медична сестра та Лаборант – та доступний їм функціонал в МІС. Діаграма демонструє користувацький рівень системи, де основна увага приділена сценаріям роботи з особистими кабінетами, електронними медичними картами (ЕМК) та загальнодоступними можливостями. Вона показує перехід від неавторизованого користувача до авторизованого, а також демонсрує основу для подальшого впровадження моделі RBAC.

Актор Гість представляє користувача, який не має авторизованого доступу до системи. Його можливості обмежені загальнодоступними функціями, такими як:

- ознайомлення із загальною інформацією (перегляд головної сторінки сайту, сторінки контактів тощо);
- запит на розсилку повідомлень – вільне місце на запис до лікаря чи пропозиція підписання декларації;
- запит на прийом до лікаря відповідно його розкладу;
- реєстрація в системі, що включає заповнення відповідного бланку.

На практиці Гість також може ввести дані для входу до особистого кабінету (ОК). Але цінності в цій дії немає. Таке розділення дозволяє відокремити неавторизований доступ від захищених дій, що виконуються лише після реєстрації. Варто зазначити, що функціонал Гостя доступний абсолютно всім в системі.

Пацієнт – користувач системи, зареєстрований пацієнт лікарні. Окрім функцій гостя, цей актор отримує лише доступ до своїх даних – це реалізація принципу обмеження доступу за роллю, який є ключовим для безпеки персональної медичної інформації. З урахуванням цього можна виділити такі функції:

- вхід в систему через авторизацію (введення логіну та паролю, підтвердження через email);
- взаємодіяти зі своїм ОК, де доступні функції перегляду власної ЕМК, редагування особистих даних, зміна паролю та вихід із системи.

Лікар є центральним користувачем у системі, оскільки саме навколо нього зосереджені основні функції роботи з медичними картами. На етапі моделювання всі ключові функції, пов’язані з обробкою медичних даних, зібрані навколо актора Лікар. Він може:

- взаємодіяти зі своїм особистим кабінетом;
- здійснювати запити на роботу з медичною картою: створювати, переглядати, редагувати чи видаляти ЕМК.

Надалі ці функції будуть розподілені за допомогою політик доступу – ABAC або PBAC, щоб забезпечити розмежування доступу між сімейним лікарем, спеціалізованим лікарем, стажером тощо. Тобто, зараз лікар показаний як узагальнений актор, який у майбутньому стане основою для детального розгляду контролю доступу.

Актор Медична сестра має подібну структуру взаємодії, але з обмеженими правами – вона працює через власний кабінет, може переглядати ЕМК та здійснювати базові операції з даними пацієнтів: редагування та перегляд. При цьому редагування, здійснені молодшим медичним персоналом (ММП), мають бути позначені в системі як оновлення, що потребують підтвердження або перегляду головним лікарем. Це дозволяє відобразити принцип підлеглості ролей і демонструє, що різні користувачі мають різні рівні доступу до тієї ж інформації.

Лаборант працює з власним кабінетом і взаємодіє з медичними картами опосередковано – через занесення результатів аналізів до системи. Таке розмежування дозволяє контролювати цілісність даних: лаборант може вводити результати, але не редагувати чи видаляти інші записи в ЕМК.

3.2.2 Політики доступу для модулю «Медична карта»

Не всі правила доступу було доцільно відносити лише до RBAC, оскільки роль користувача не завжди визначає обсяг дозволів у системі. Наприклад, доступ лікаря до даних пацієнтів залежить від спеціалізації, відношення до пацієнта та контексту

роботи. Використання ABAC та PBAC дозволяє враховувати ці атрибути та сценарії, забезпечуючи гнучкіший і більш безпечний контроль доступу.

Тож доцільно визначити основні політики доступу та безпеки. Це дозволяє формалізувати правила взаємодії користувачів із системою, чітко розмежувати права між ролями та забезпечити захист медичних даних відповідно до вимог конфіденційності.

Після детального розгляду різних політик, що застосовуються в МІС, були обрані ті, які відповідають потребам системи, що розглядається та забезпечують баланс між доступністю функцій і безпекою даних.

В таблицях 3.1-3.5 описано ключові політики відповідно до ролей користувачів за такими параметрами:

- Умова / Ситуація – конкретний сценарій або стан, у якому користувач взаємодіє з системою. Тобто тут вказується коли і за яких обставин буде перевірятись право доступу.
- Модель доступу – тип моделі контролю доступу, яка застосовується для даного сценарію. Він показує яким механізмом система вирішує, кому і що дозволяти.
- Логіка (правило політики) – конкретне правило або алгоритм, який перевіряє, чи можна надати доступ у цій ситуації.

Таблиця 3.1 – Політики доступу для лікарів

Умова / Ситуація	**Модель доступу**	**Логіка (правило політики)**
Лікар має спеціалізацію “терапевт”	ABAC	Атрибут `specialization="therapist"` дозволяє ініціювати процес підписання декларацій із пацієнтами.
Терапевт підписує декларацію з пацієнтом	PBAC (на основі ABAC)	Після підписання лікар отримує атрибут `is_family_doctor=True`.
Сімейний лікар	RBAC + ABAC	Якщо `user.role="doctor"` і `user.is_family_doctor=True`, він має повні права на зміну та перегляд записів.
Лікар іншої спеціалізації (не сімейний)	PBAC	Може переглядати карти лише своїх пацієнтів або тих, хто надав дозвіл. Редагування можливе тільки після дозволу від сімейного лікаря або пацієнта.

Продовження таблиці 3.1

Надання доступу до ЕМК	PBAC	Сімейний лікар може надати доступ іншому лікарю лише до певного модуля карти (наприклад, “Лабораторні результати” або “Рентген”).
Редагування за дозволом	PBAC (із контекстом)	Після отримання дозволу лікар спеціаліст може вносити зміни лише в межах модуля, на який надано доступ.
Перегляд ЕМК іншими лікарями	RBAC + ABAC	Дозволений завжди, якщо `user.role="doctor"` і перед цим був отриманий доступ до ЕМК. Дозвіл на перегляд не обмежений у часі, а на редагування – так.
Журнал доступу	PBAC (з безпековим контролем)	Усі дії з ЕМК (перегляд, редагування, запит, підтвердження) фіксуються в системному журналі який доступний для перегляду сімейному лікарю.
Тимчасовий доступ у разі відсутності сімейного лікаря	PBAC + Context	У разі відпустки сімейного лікаря система може делегувати доступ іншому лікарю з тимчасовим токеном, що має обмеження за часом.
Аварійний режим (“Emergency mode”)	PBAC + ABAC	У надзвичайних випадках будь-який лікар може відкрити карту пацієнта, але система створює запис про екстрений доступ і сповіщає адміністратора.
Автоматичне відкликання дозволів	PBAC	Після завершення лікування або закінчення терміну дії дозволу, доступ анулюється автоматично, без участі лікаря.

Таблиця 3.2 – Політики доступу для пацієнтів

Умова / Ситуація	**Модель доступу**	**Логіка (правило політики)**
Пацієнт запитує копію даних	PBAC	Системою перевіряється, чи дані не містять обмеженої для пацієнта інформації.
Пацієнт надає доступ лікарю (друга думка)	PBAC	Пацієнт може тимчасово надати доступ іншому лікарю (на певний час або модуль карти).
Відкликання дозволу	PBAC	Після закінчення терміну дії доступу або вручну пацієнт може відкликати дозвіл.
Журнал дій	PBAC	Пацієнт має право переглядати історію доступів до своєї карти.

Таблиця 3.3 – Політики доступу для неавторизованих користувачів (Гість)

Умова / Ситуація	Модель доступу	Логіка (правило політики)
Гість проходить реєстрацію	PBAC	Після підтвердження email створюється новий користувач із базовим особистим кабінетом, не має жодної ролі.
Спроба доступу до захищених модулів	PBAC	Система блокує запит і перенаправляє на сторінку автентифікації.
Гість запитує консультацію онлайн	PBAC + ABAC	Дозволено лише перегляд розкладу та заповнення форми, без збереження медичних даних.

Таблиця 3.4 – Політики доступу для ММП

Умова / Ситуація	Модель доступу	Логіка (правило політики)
ММП належить до певного відділення	ABAC	Атрибут department визначає, з якими пацієнтами вона може працювати.
ММП оновлює параметри стану пацієнта (АТ, пульс, сигнальні позначки)	RBAC + ABAC	Дозволено лише для пацієнтів із того ж відділення (nurse.department == patient.department).
Редагування даних	PBAC	Внесені зміни маркуються як “потребують підтвердження” і відображаються головному лікарю або сімейному.
Перегляд історії пацієнтів	ABAC	Може бачити лише пацієнтів свого відділення.
Запит на доступ до інших модулів карти	PBAC	Може подати запит на доступ до лабораторних або діагностичних результатів, якщо це необхідно для догляду.
Робоча зміна	PBAC + ABAC	Усі права діють лише під час активної зміни (shift="active"). Після зміни доступ блокується.

Таблиця 3.5 – Політики доступу для лаборанта

Умова / Ситуація	Модель доступу	Логіка (правило політики)
Лаборант отримує запит на дослідження від лікаря	PBAC	Створюється новий запис у лабораторному модулі, закріплений за пацієнтом.
Внесення результатів аналізу	RBAC + ABAC	Дозволено лише внесення даних в спеціальну форму, що прикріплюється до запиту.
Редагування результатів після підтвердження	PBAC	Після затвердження результатів лікарем зміни заборонені.
Перегляд історії аналізів	ABAC	Може переглядати лише записи, створені власноруч або в межах своєї лабораторії.
Контроль достовірності	PBAC	Система автоматично перевіряє валідність даних лабораторних результатів при кожному перегляді.

3.3 Технологія функціонування процесів модулю контролю доступу

Після побудови UML-діаграм та опису ключових політик доступу до системи стало очевидним, що для такої кількості прецедентів взаємодії з системою доцільно розробити додаткові діаграми. Вони допоможуть відобразити не лише бізнес-процеси та функціональну логіку, а й технологічну сторону обробки даних. Зокрема, необхідні моделі, які показують, як саме інформаційні потоки проходять крізь функціональні модулі системи.

IDEF0-модель («Чорна скринька») – модель опису системи або окремих її блоків в графічному вигляді. Вона дозволяє досліджувати функції організації, не пов'язуючи їх конкретно з об'єктами, що забезпечують їх реалізацію.

Будується IDEF-модель за допомогою таких елементів:

- блоки, які зображують функції чи дії (в залежності від ступеню декомпозиції);
- стрілки, які показують механізми взаємодій, потоки інформації чи ресурси.[2, 32]

Стрілки в свою чергу мають значення:

1. Вхідні дані в систему або її функціональний блок (розташовані з лівої сторони від блоку функцій);
2. Вихідні дані – дані отримуються при завершені роботи функції (розташовані справа від блоку функцій);
3. Управління – правила яким підпорядковується система (розташовані зверху);
4. Механізм, за допомогою якого виконується функція (розташовані знизу).

З позиції управління безпекою та контролю доступу, IDEF-діаграми відіграють важливу роль, оскільки дозволяють чітко простежити, які функції використовують дані користувача і через які механізми вони проходять.

Відповідно до функціоналу системи було візуалізовано IDEF-0 модель потоків даних для ІС, що розглядається (Рис.3.3).

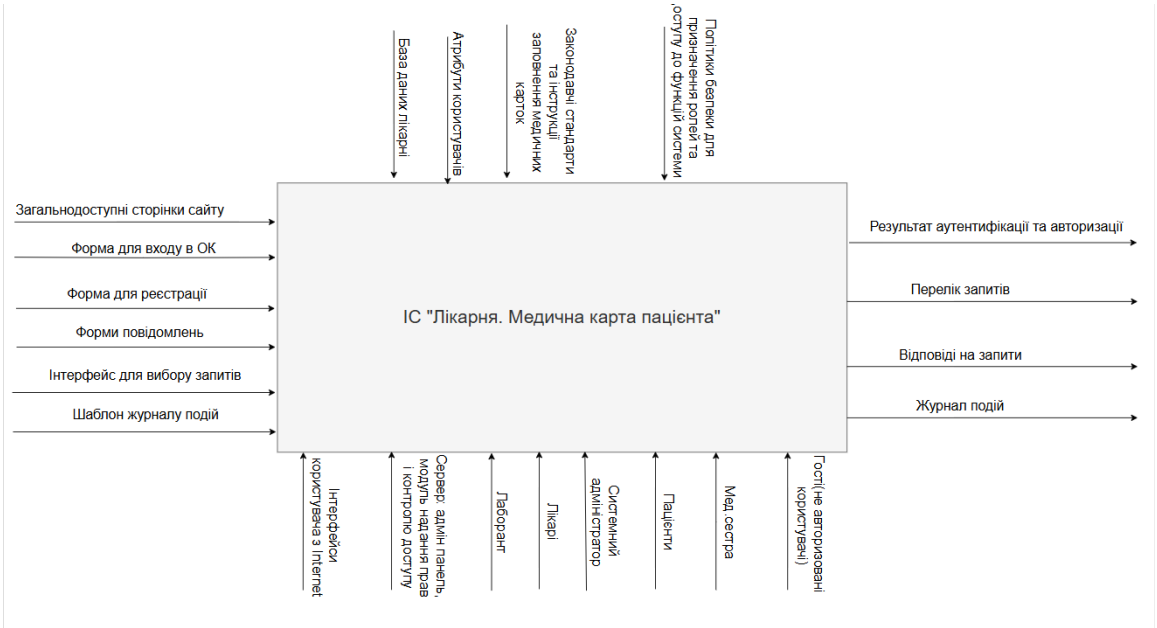


Рисунок 3.3 – IDEF-0 модель інформаційної системи лікарні

Вхідними даними мають бути форми для відображення або заповнення інформацією:

- для реєстрації в системі та для входу в ОК;
- для ведення журналу подій;

- для вибору запитів;
- шаблони загальнодоступних сторінок сайту та шаблони повідомлень для розсилки.

Механізмами, рушійною силою роботи системи, є пристрої інтерфейсу користувача забезпечені інтернетом, сервер та адміністратори що його обслуговують, користувачі (пацієнти, гості та медичний персонал лікарні), що відображений в діаграмі прецедентів).

Матеріалами, яким підпорядковується система, буде:

- законодавча база ведення медичних карток та інформація в них, що зберігається в базі даних;
- політики безпеки та атрибути користувачів, що використовуються в них (ролі користувачів, час входу в систему, відділення мед.працівників тощо).

Вихідними даними будуть інформація, яка є об’єктом запитів користувачів, результати приєднання до системи, журнал подій.

Після моделі «Чорна скринька» важливо провести декомпозицію за допомогою IDEF-3 моделі (Рис.3.4).

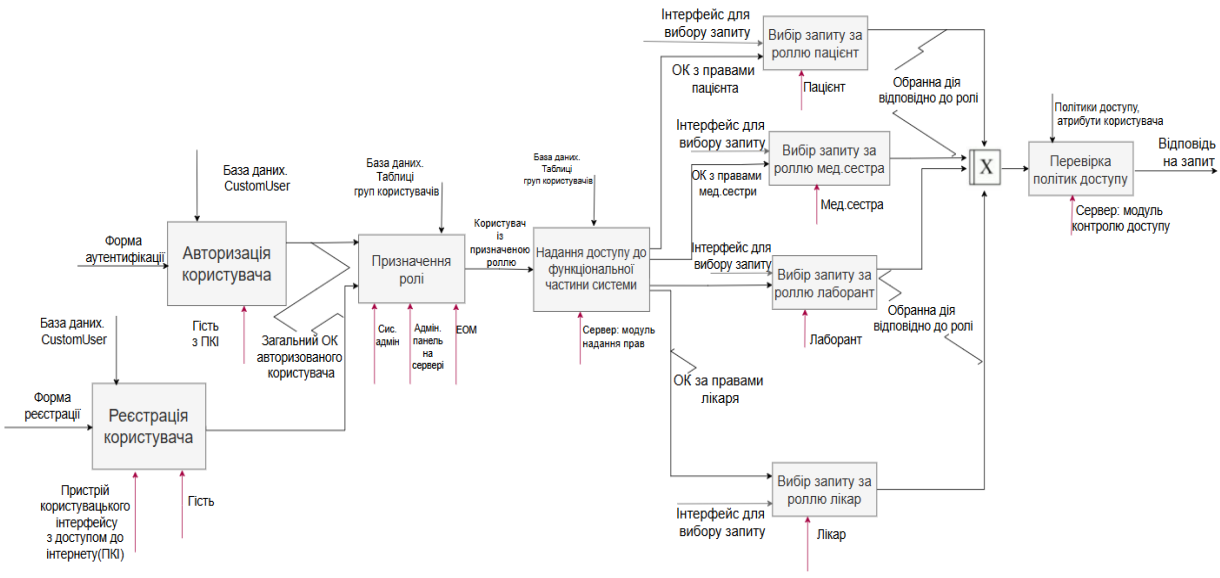


Рисунок 3.4 – IDEF-3 модель процесів автентифікації, авторизації та розподілу доступу в системі

IDEF-3 модель описує послідовність процесів керування доступом користувачів у медичній інформаційній системі. Кожен блок є критично важливим для створення безпечних умов оперування даними. Вони гарантують, що лише авторизовані користувачі з відповідними ролями та атрибутами, тільки за відповідних умов можуть отримати доступ до необхідних даних та функцій. На відміну від подібних діаграм змодельованих для бакалаврської роботи (Додаток Б), на рисунках 3.3-3.4 виділено нових користувачів (ММП, лаборант), що відображає розширення функціональних можливостей системи. Крім того, у новому варіанті діаграм акцент зроблено не лише на розподілі функцій між користувачами, а насамперед на процесах надання та перевірки доступу до функціональних частин системи відповідно до призначених ролей.

Реєстрація користувача з перевіркою ідентифікаційних даних та встановленням складного пароля, запобігає несанкціонованому створенню облікових записів (спам-реєстрація або підробка даних). Дані користувача мають зберігатись в захищеній БД та зашифрованому стані. ***Авторизація користувача*** виконується замість реєстрації, якщо його обліковий запис уже існує в системі. Успішна авторизація підтверджує особу, генеруючи "Загальний ОК" – створюючи безпечний токен або сесію, що використовуватиметься для подальших дій. Це захищає систему від неавторизованого доступу.

Блок Призначення ролі – це ядро моделі керування доступом на основі ролей .На цьому етапі призначаються користувачу одну або кілька ролей (Пацієнт, Лікар тощо). Це гарантує, що права доступу будуть надані за принципом найменших привілеїв, що запобігає доступу до функцій, які не відповідають професійним обов'язкам користувача.

Надання доступу до функціональної частини системи – блок, в якому створюються особисті кабінети для кожної ролі («ОК з правами пацієнта», «ОК з правами лікаря»). Вони за допомогою інтерфейсу визначають, що користувач може робити в системі. Це мінімальна вимога для запобігання ситуацій, коли користувач може отримати доступ до функцій іншої ролі.

Блок вибору запиту за роллю відповідає за визначення дії користувача відповідно до його ролі. Щоб уникнути перевантаження схеми, функції розподілено за ролями.

Перевірка політик доступу користувача – блок, в якому система отримує запит, порівнює його з актуальними політиками доступу та лише у разі позитивного результату перевірки надає дозвіл на виконання дії. Це забезпечує цілісність, конфіденційність та доступність даних, гарантуючи, що жоден запит, навіть сформований авторизованим користувачем, не буде виконано, якщо він порушує встановлені правила. Наприклад, лікар не зможе переглянути медичні дані пацієнта, який не є його поточним пацієнтом (розмежування за відношенням).

Така структура гарантує, що система не тільки знає, хто користувач, але й постійно перевіряє, що йому дозволено робити згідно з його роллю та актуальними політиками, забезпечуючи надійний захист даних.

3.4 Алгоритми функціонального блоку «Надання доступу»

Важливим аспектом функціонування МІС є надання доступу користувачів до даних. Коректна побудова цього процесу забезпечує захист конфіденційної інформації, достовірну ідентифікацію користувачів та цілісність медичних записів у базі даних.

Більшість процесів керування обліковими записами та правами доступу покладається на системного адміністратора. Під час реєстрації нового працівника (лікаря, медичної сестри чи лаборанта) адміністратор створює для нього обліковий запис, визначає роль у системі та призначає відповідний рівень доступу до ЕМК пацієнтів. Така централізована модель керування обліковими записами дозволяє:

- мінімізувати ризик несанкціонованого доступу до системи;
- забезпечити реалізацію принципу мінімальних привілеїв;
- спростити аудит і контроль змін у системі, оскільки всі дії з управління доступом фіксуються в журналі безпеки.

Проте надання доступу для пацієнтів медичного закладу відбувається за дещо іншою схемою. Це пояснюється тим, що користувачі цієї категорії не мають адміністративних прав і діють у межах власного облікового запису. Для візуалізації процесів, що передують та відбуваються після умови надання доступу пацієнту, розроблено діаграму діяльності (Рис. 3.5), яка відображає послідовність взаємодій між основними суб'єктами системи: пацієнтом, сервером та адміністратором.

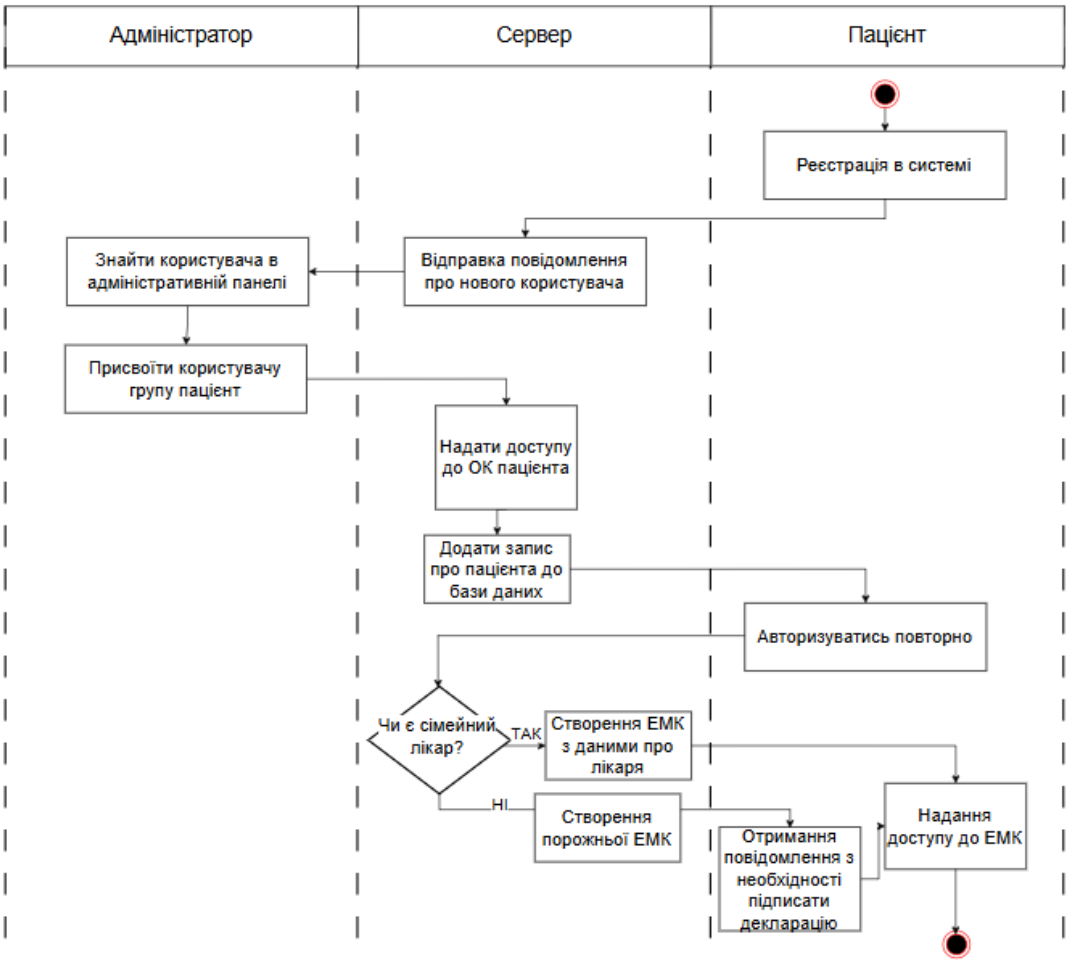


Рисунок 3.5 – Діаграма діяльності умов надання доступу пацієнту

Процес починається з реєстрації пацієнта через вебінтерфейс або мобільний застосунок, у результаті якої створюється попередній обліковий запис без підтвердженого статусу. Система автоматично формує запит до адміністратора з інформацією про нового користувача. Адміністратор у панелі керування перевіряє

дані, призначає групу «Пацієнт», після чого сервер активує обліковий запис і надає доступ до особистого кабінету.

Після активації особистого кабінету пацієнтом повторно, система створює запис у базі даних та перевіряє наявність підписаної декларації із сімейним лікарем. Пацієнту надається доступ до ЕМК в будь-якому випадку, але у разі відсутності сімейного лікаря користувач отримує повідомлення про необхідність підписання декларації.

Розглянутий алгоритм відображає етапи перед- та після-умови надання доступу, поєднуючи адміністративні дії та автоматизовані перевірки системи. З точки зору безпеки, він забезпечує контрольований доступ до персоніфікованих медичних даних, реалізуючи багаторівневий підхід до автентифікації, авторизації та розмежування прав користувачів відповідно до вимог стандартів ISO/IEC 27001 та GDPR.

Розглянутий алгоритм відіграє ключову роль у забезпеченні конфіденційності, цілісності та доступності даних у медичній інформаційній системі. Він поєднує адміністративні, організаційні та технічні механізми захисту, створюючи багаторівневу модель контролю доступу. Це не лише підвищує загальну безпеку системи, а й дозволяє відповідати вимогам міжнародних стандартів, таких як ISO/IEC 27001 та GDPR, щодо захисту персональних медичних даних.

Варто зазначити, що хоча безпосереднє надання дозволу на доступ до системи здійснює адміністратор, механізм доступу для кожної ролі має свої особливості, визначені логікою самої системи. Тому доцільно розглянути процес розподілу доступу до функціональних модулів відповідно до ролі користувача (Рис.3.6).

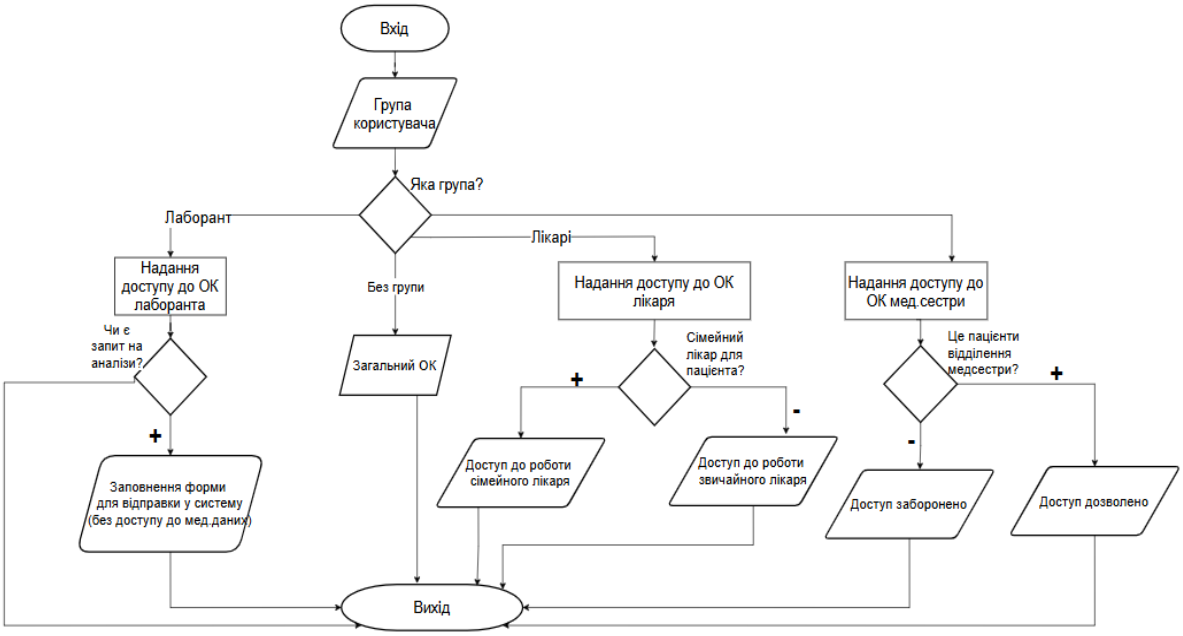


Рисунок 3.6 – Алгоритм надання доступу до функцій системи персоналу лікарні

Після успішного входу система визначає, до якої групи належить користувач – лікар, медична сестра або лаборант. Ця класифікація здійснюється автоматично на основі параметрів облікового запису в базі даних, що запобігає помилковому чи несанкціонованому призначенню ролей.

Для лікарів система розрізняє дві категорії: сімейних лікарів і звичайних фахівців. Сімейний лікар отримує розширений доступ до ЕМК своїх пацієнтів, з якими підписано декларацію. Звичайний лікар має обмежений доступ лише в межах своєї спеціалізації та призначень.

Якщо розглядати систему з точки зору ММП, то система перевіряє, чи працює вона у тому ж відділенні, де перебувають відповідні пацієнти: якщо так – надається дозвіл на доступ до ОК медичної сестри для ведення журналів процедур, якщо ні – доступ забороняється. Ця логіка дозволяє в майбутньому реалізувати контекстний контроль доступу, що є частиною ABAC.

Лаборанти отримують доступ лише до інтерфейсу для оформлення результатів аналізів без можливості переглядати повні медичні карти пацієнтів. Якщо запит на аналіз не знайдено – система взагалі не надає даних про конкретного пацієнта. Це

реалізація принципу сегментації доступу, який мінімізує ризики витоку даних через другорядних користувачів системи.

Якщо користувач не має призначеної групи (наприклад, у процесі первинного створення облікового запису), система надає лише загальний кабінет з мінімальним функціоналом і без доступу до будь-яких медичних записів.

Наведені діаграми не лише демонструють логіку функціонування системи доступу, а й підкреслюють її орієнтацію на багаторівневий захист персональних медичних даних. Це відбувається завдяки поєднанню автоматичних перевірок на сервері, контролю адміністратора та чіткому розмежуванню ролей користувачів. Крім того, обрана модель доступу є масштабованою – вона дозволяє легко додавати нові ролі, модулі чи правила безпеки без зміни основної логіки авторизації, що робить систему гнучкою та придатною до подальшого розвитку.

3.5 Діаграма класів

Останнім етапом проектування даної системи є побудова моделі бази даних. Існує значна кількість підходів до організації баз даних та способів їх візуального представлення. У межах цього проєкту архітектура бази даних розроблялася з урахуванням об’єктно-орієнтованої парадигми програмування, що забезпечує тісний зв’язок між програмним кодом і структурою даних.

Об’єктно-орієнтована база даних передбачає представлення інформації у вигляді об’єктів, які мають власні атрибути (властивості) та методи (поведінку). На відміну від традиційних реляційних баз, де дані подаються у вигляді таблиць із рядками та стовпцями, такий підхід дозволяє описувати сутності системи більш природно, відповідно до об’єктів предметної області.

Основні переваги цього підходу:[2]

- підтримка принципів об’єктно-орієнтованого програмування (наслідування, інкапсуляція, поліморфізм);
- узгодженість структури коду і бази даних, що спрощує розробку та супровід системи;

- зменшення витрат часу на обробку даних, оскільки відпадає потреба у складному перетворенні форматів між програмою і базою;
- підвищення ефективності запитів завдяки прямому відображенню логіки взаємодії об’єктів у структурі бази даних.

У даній концепції застосовуються зв’язки, що відображають тип взаємодії між сутностями:

- «один-до-одного» – коли кожному запису однієї сутності відповідає лише один запис іншої;
- «один-до-багатьох» – коли один запис може бути пов’язаний із кількома записами іншої сутності;
- «багато-до-багатьох» – коли зв’язок є взаємним для кількох об’єктів обох сутностей. Такий тип реалізується за допомогою проміжної таблиці, що містить зовнішні ключі обох зв’язаних таблиць.

На рисунку 3.7 показано архітектуру бази даних за допомогою діаграми класів.

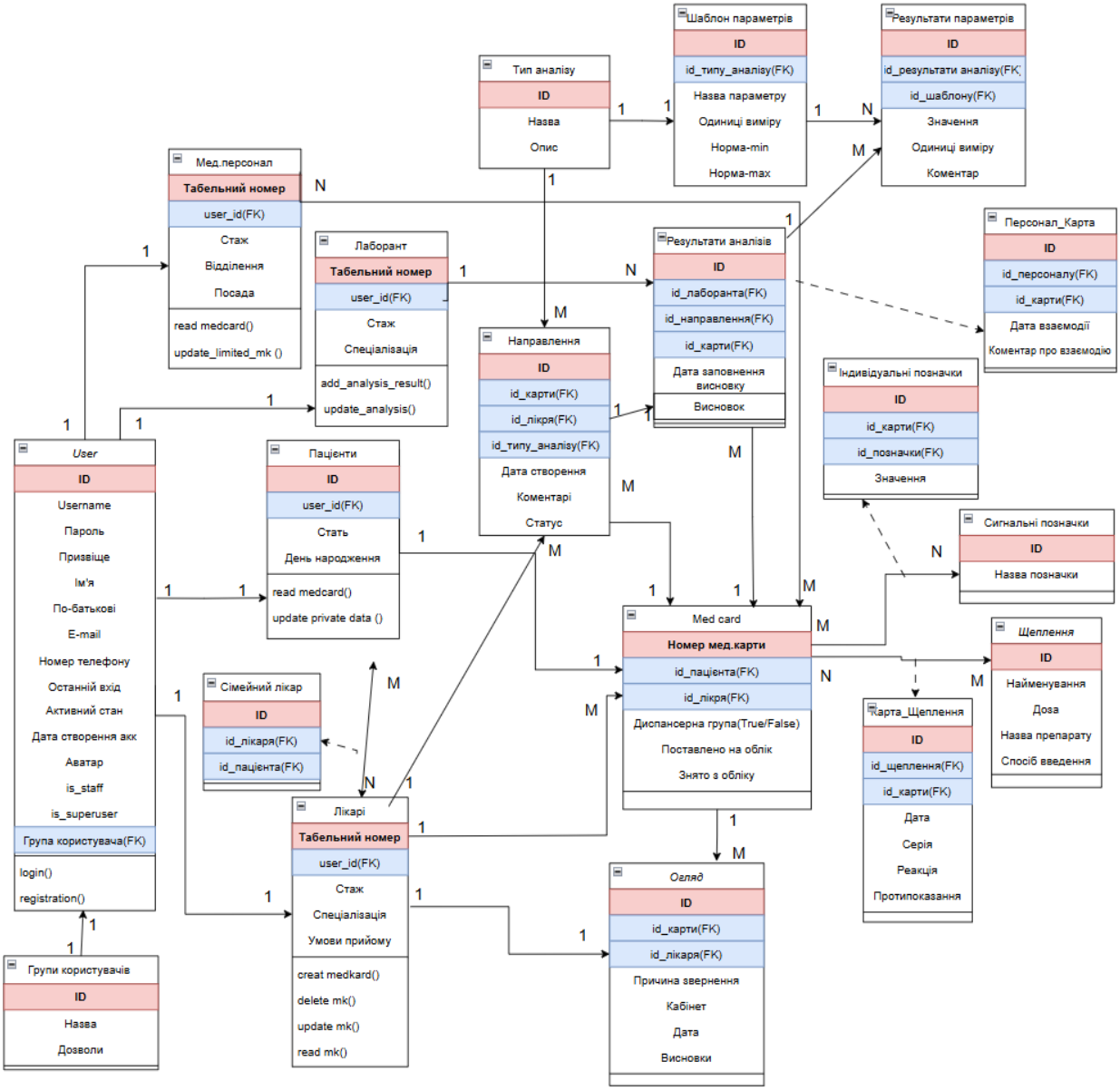


Рисунок 3.7 – Діаграма класів системи

Центральним елементом моделі є таблиця «User», що містить облікові дані користувачів і пов’язаний із таблицею груп користувачів, де визначаються права доступу. Зберігання персональних даних у централізованій таблиці користувачів дозволяє застосувати шифрування критичних полів (паролі, персональні дані) та забезпечити аудит усіх змін через механізми журналювання. Завдяки ж таблиці «Групи користувачів» реалізовано розмежування ролей. Кожна роль має власні атрибути та методи, які визначають її функціональні можливості у системі: наприклад, лікар може переглядати та оновлювати медичні карти, лаборант – додавати результати аналізів, а пацієнт – переглядати власні дані тощо.

В діаграмі можна чітко визначити основні складові ЕМК – запис про огляди, сигнальні позначки, щеплення, результати аналізів і направлення, що забезпечує повноту електронної історії пацієнта.

Модель, що розглядається, побудована з урахуванням принципів нормалізації баз даних. Це дає змогу уникнути дублювання даних і мінімізувати ризик виникнення аномалій під час оновлення або видалення записів. Використання зовнішніх ключів (FK) гарантує логічну узгодженість між таблицями, а чітке розділення логічних об’єктів підвищує стабільність і масштабованість бази даних. Варто зазначити, що використання проміжних сутностей, зокрема «Сімейний лікар», дозволяє ефективно реалізувати взаємозв’язки «багато-до-багатьох» без дублювання інформації, що є важливим для підтримання гнучкості та узгодженості даних у системі та відповідає вимогам нормалізації БД.

Загалом, діаграма класів демонструє добре продуману структуру, де поєднано логічну узгодженість, функціональну повноту та безпечну організацію даних. Така структурованість забезпечує зручну та ефективну взаємодію з даними на всіх рівнях системи: від користувацького інтерфейсу до серверної логіки та бази даних. Завдяки чітко визначеним зв’язкам між класами розробники можуть легко реалізовувати запити, оновлення та обробку інформації, не порушуючи цілісності структури. Це спрощує подальше розширення функціоналу, інтеграцію з іншими медичними чи аналітичними модулями та підтримку системи в довгостроковій перспективі.

Висновок

У цьому розділі більше уваги було зосереджено на розширенні компонентів системи та підвищенні рівня безпеки. Було уточнено й доповнено діаграми варіантів використання: додано акторів «Лаборант» і «Медична сестра», а також виокремлено сутність «Адміністратор». Це дало змогу охопити ширший спектр користувачів системи та відобразити додатковий функціонал, необхідний для коректного формування політик доступу.

Після вдосконалення попередньої моделі була розроблена оновлена архітектура системи, у якій визначено компоненти управління доступом у медичній інформаційній системі. Це забезпечило основу для створення безпечної, масштабованої та відповідної сучасним вимогам інфраструктури. Використання UML- та IDEF-моделей дозволило комплексно змоделювати процеси автентифікації, авторизації та контролю доступу до системи.

Розроблені діаграми прецедентів забезпечили структурне відображення бізнес-процесів як на рівні адміністрування, так і на рівні кінцевих користувачів, чітко визначивши межі доступу та функціональні взаємозв’язки.

Важливим результатом роботи стало застосування комбінованої моделі контролю доступу RBAC+PBAC+ABAC. Їх поєднання було відображено в основних політиках доступу, визначених на основі діаграм варіантів використання. Це забезпечує гнучкість системи, адаптивність до динамічних сценаріїв використання та можливість впровадження тимчасового, обмеженого або аварійного доступу без порушення принципу мінімізації привілеїв.

Моделювання на рівні потоків даних (IDEF-0 та IDEF-3) дозволило візуалізувати інформаційні потоки, зовнішні чинники та логіку послідовного виконання процесів надання доступу до системи.

Разом з тим, були розроблені алгоритми надання доступу до медичної інформації пацієнтів та персоналу лікарні. Ці алгоритми відобразили різний підхід надання доступу до даних та різних частин системи відносно ролей користувачів.

Завершальним етапом стало побудування діаграми класів, яка відобразила структуру бази даних з урахуванням принципів об’єктно-орієнтованого проєктування та нормалізації. Це дозволило закласти фундамент для реалізації інтегрованої підсистеми управління доступом, що підтримує аудит, шифрування та контроль життєвого циклу облікових записів.

4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОМПОНЕНТІВ СИСТЕМИ

4.1 Технологічний стек системи

Оскільки розробка MIC була розпочата в рамках бакалаврської роботи, технологічний стек залишився здебільшого незмінним. Проте, в процесі масштабування, вдосконалення компонентів безпеки та управління доступом було оновлено версії деяких інструментів. Це дозволило забезпечити стабільність, підтримку та покращити сумісність з сучасними стандартами.

Основою серверної частини системи залишається Python 3.12 у поєднанні з Django 5.2. Раніше у модулі застосовувалася версія Django 4.2, яка має довгострокову підтримку до квітня 2026 року. Для подальшого розвитку та вдосконалення системи було прийнято рішення перейти на Django 5.2, що дозволить скористатися останніми оновленнями функціоналу та безпеки фреймворку [33].

Django зберігає вбудовану ORM (Object-Relational Mapping), яка дозволяє працювати з базою даних на рівні об’єктів без прямого використання SQL-запитів, підвищуючи узгодженість та цілісність даних. Архітектурна модель MTV (Model–Template–View) забезпечує чітке розділення даних, бізнес-логіки та представлення, що спрощує супровід та подальше розширення системи. У новій версії також оптимізовано маршрутизацію, покращено підтримку асинхронних запитів і інтеграцію сучасних бібліотек. Варто зазначити, що в попередній версії фреймворку була можливість потрапити в адмін-панель без налаштування бази даних. В новій версії ця функція не доступна, що підвищує безпеку та додає більше логіки в функціонал.

Для реалізації інтерфейсу користувача використовуються HTML, CSS та Bootstrap, що забезпечує адаптивність і доступність інтерфейсу на різних пристроях. Також був використаний вбудований Django-шаблонізатор, що має синтаксис схожий на шаблонізатор Jinja.

База даних побудована на PostgreSQL, яка характеризується високим рівнем надійності, підтримкою транзакційної безпеки та розширеними механізмами контролю доступу. Вибір саме PostgreSQL зумовлений її стабільністю та відповідністю вимогам до обробки чутливих медичних даних.

Розробка здійснюється у середовищі Visual Studio Code, яке забезпечує інтеграцію з системою контролю версій Git та хмарним сховищем GitHub. Це дає можливість ефективно відстежувати зміни у проєкті, забезпечувати командну співпрацю та зберігати історію версій.

4.2 Реалізація функціоналу

4.2.1 Програмна базова структура ЕМК

Для забезпечення модульності та масштабованості медичної інформаційної системи проєкт було структуровано у вигляді окремих додатків (apps), що відповідає архітектурним принципам фреймворку Django. Такий підхід дозволяє розділити функціональність за логічними доменами, спростити тестування, підтримку та подальше розширення системи.

У проєкті реалізовано такі додатки:

- public_app – відповідає за обробку загальнодоступних сторінок сайту та формування базового інтерфейсу для неавтентифікованих користувачів.
- users_app – забезпечує функції автентифікації та авторизації, включно з реєстрацією, входом, виходом та керуванням профілем користувача.
- cards_app – реалізує управління електронними медичними картами пацієнтів, механізми прив’язки карток до медичного персоналу, а також логіку розмежування доступу до вкладок ЕМК для різних категорій користувачів.
- laboratory_app – відповідає за облік, збереження та обробку результатів лабораторних аналізів. Хоча додаток залишається максимально ізольованим

від основної структури ЕМК, дані синхронізуються з медичними картками для забезпечення цілісності медичної інформації.

Кожен з додатків містить типову структуру, що включає файли моделей, представлень, маршрутів, шаблонів та інші елементи, передбачені фреймворком Django (Рис. 4.1).

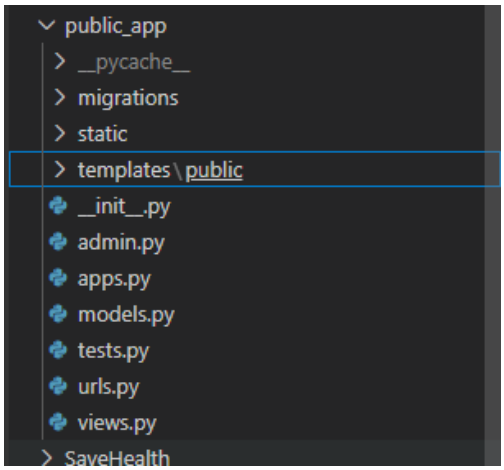


Рис.4.1 – Базова структура додатку на прикладі public_app

4.2.2 Рівень моделей у архітектурі Django

Основні операції з даними реалізуються на рівні логічного представлення бази даних, структура якого відповідає діаграмі класів (п. 3.5). На цьому рівні визначаються сутності, їх атрибути, зв’язки, обмеження цілісності та елементи бізнес-логіки.

Використання вбудованої ORM Django замість ручного написання SQL-запитів підвищує безпеку обробки даних. ORM автоматично екранує вхідні дані та формує параметризовані запити, що зменшує ризик SQL-ін’єкцій. Усі операції читання, фільтрації, створення та оновлення записів виконуються через написання класів та методів, які відповідають сучасним рекомендаціям OWASP. (Рис. 4.2).

```python
class IndividualMarks(models.Model):
    medcard = models.ForeignKey('MedCards', on_delete=models.CASCADE, verbose_name='Номер медичної карти')
    signal_mark = models.ForeignKey('SignalMarks', on_delete=models.CASCADE, verbose_name='Сигнальна позначка')
    value = models.CharField(max_length=100, blank=True, null=True, verbose_name='Значення')
    class Meta:
        db_table = 'IndividualMarks'
        verbose_name = 'Індивідуальну позначку'
        verbose_name_plural = 'Індивідуальні позначки'
        unique_together = ('medcard', 'signal_mark')
    def __str__(self):
        return f'{self.id}.{self.medcard} має позначку {self.signal_mark}'
```

Рис.4.2 – Фрагмент коду моделі IndividualMarks

Реалізація за допомогою ООП включає опис таблиць, їх зв'язків один з одною, індексів, обмежень та бізнес-правил, що регламентують роботу з медичними даними.

Крім того, визначення моделей через класи дозволяє централізовано задавати обмеження, правила валідації та поведінкову логіку об'єктів, що підвищує цілісність даних. Кожне поле моделі містить низку атрибутів, таких як null, blank, unique, choices, validators, які не лише описують структуру відповідної колонки в БД, але й встановлюють додаткові логічні та безпекові обмеження на рівні застосунку.

Оскільки значну частину логіки БД було описано в попередній роботі, було прийнято рішення описувати лише нові особливості та функціонал. Таким чином, до додатку Users_app додано два класи, що описують атрибути для ролей лаборанта та ММП (табл. 4.1-4.2).

Таблиця 4.1 – Атрибути класу LaboratoryAssistent

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
tab_nomer (PK)	Char	20	False	False
user	При видаленні користувача відомості про лаборанта будуть видалені			
stazh	Char	50	True	True
specialization	Char	100	False	False

Таблиця 4.2 – Атрибути класу Personnel

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
tab_nomer (PK)	Char	20	False	False
user	При видаленні користувача відомості про лаборанта будуть видалені			
stazh	Char	50	True	True
department	Char	100	False	False
position	Char	250	True	True

Клас, що уособлює проміжну таблицю FamilyDoctorмає записи сімейних лікарів та їх пацієнтів. До нього було додано поле, що показує дату та час зміни сімейного лікаря. (Таблиця 4.3)

Таблиця 4.3 – Атрибути для моделі FamilyDoctor

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
Patient	Якщо записи з таблиці Patient видалять, то і видаляться записи що стосуються того пацієнта			
Doctor	Якщо записи з таблиці Doctor видалять, то і видаляться записи що стосуються того лікаря			
date_of_appointment	Datatime	auto_now_add=True		

Cards_app зазнав змін більшою мірою через розширення функціоналу. До моделі з основними даними в медичній картці (табл 4.4) додалось поле, що поєднує карту з ММП. Також було додано клас, що є проміжним для зв'язку багато-до-багатьох між ММП та ЕМК (табл 4.5).

Таблиця 4.4 – Атрибути для моделі MedCards

ІМ'Я	**ТИП**	**NULL**	**BLANK**
Patient	При видаленні пацієнта запис з цієї таблиці видаляється		
Doctor	При видаленні лікаря поле залишиться порожнім		
dispensary_group	bool	default=False	
registration_date	DataTime	auto_now_add=True	
deregistration_date	DataTime	True	True
med_personnel	Поле зв'язку для проміжної таблиці Personnel_MedCard		

Таблиця 4.5 – Атрибути для моделі Personnel_MedCard

ІМ'Я	**ТИП**	**NULL**	**BLANK**
Medcard	При видаленні карти запис з цієї таблиці видаляється		
Personnel	При видаленні персоналу відомості для пацієнта будуть порожні		
interaction_date	DateTime	auto_now_add=True	
Comment	Text	True	True

В новому додатку **laboratory_app** було описано п'ять класів-моделей, які формують структуру таблиць для збереження результатів лабораторних аналізів та обліку направлень, що створюються лікарями для лабораторії.

Таблиці 4.6-4.7 описують загальний перелік типів аналізів (аналіз крові, аналіз на гормони тощо) та перелік параметрів відповідно до цих аналізів.

Таблиця 4.6 – Атрибути для моделі TypeAnalysis

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
name	Char	100	False	False
description	Text	500	True	True

Таблиця 4.7 – Атрибути для моделі TemplateParameter

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
name	Char	20	True	True
type_analys	При видаленні типу аналізу дані з таблиці будуть видалені			
unit_of_measurement	Char	20	False	False
normal_min	Decimal	3 цифри до коми та 4 після	True	True
normal_max	Decimal		True	True

Таблиці 4.8-4.10 відповідають за дані, що є індивідуальними для кожного пацієнта: їх результати аналізів та направлення до лабораторії.

Таблиця 4.8 – Атрибути для моделі MedReferral

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
doctor	Попередження про прив'язані записи			
medcard	При видаленні карти відомості про направлення будуть видалені			
type_analys	Попередження про прив'язані записи			
creation_date	DateTime	auto_now_add=True		
coment	Text	500	True	True
status	Char	1	False	False

Таблиця 4.9 – Атрибути для моделі ResultParameter

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
value	Decimal	3 цифри до коми та 4 після	False	False
unit_of_measurement	Char	20	True	True
coment	Char	100	False	False
template	При видаленні параметру відомості будуть видалені			
result_analys	При видаленні результату аналізу відомості будуть видалені			

Таблиця 4.10 – Атрибути для моделі ResultAnalysis

ІМ'Я	**ТИП**	**Макс.довжина**	**NULL**	**BLANK**
medcard	При видаленні мед.карти відомості будуть видалені			
referral	При видаленні направлення буде попередження про прив'язані записи			
lab_assistent	При видаленні лаборанта буде попередження про прив'язані записи			
providing_date	DateTime	auto_now_add=True		
report	Text	500	False	False

Важливими є auth_group, auth_permission та інші пов'язані з групами і дозволами вбудовані таблиці. В них внесені групи ЛІКАРІ, СІМЕЙНІ ЛІКАРІ, ПАЦІЄНТИ, МЕД.ПЕРСОНАЛ та ЛАБОРАНТИ, призначені відповідні дозволи. Завдяки цьому в подальшому будуть розмежовуватись функціональні блоки та інтерфейс користувачів, реалізована RBAC.

Записи в базі даних можна створювати не тільки через адмін.панель чи систему управління базами даних, а й через функції. В минулій роботі була реалізована функція ***create_or_update_medical_record*** яка створює запис в таблиці MedCards, якщо створюється відповідний запис в таблиці FamilyDoctor, що з'єднує пацієнта та лікаря. Проте ця функція загрожувала цілісності даних, оскільки при зміні в медичній карті даних про сімейного лікаря, в відповідній таблиці дані не змінювались. Було прийнято рішення автоматичного створення медичної карти з додаванням запису про пацієнта до бази даних. Окрім цього в адмін.панелі було заборонено редагування сімейних лікарів в медичній карті – призначати нових дозволено лише змінюючи таблицю FamilyDoctor (Рис.4.3)

```python
@receiver(post_save, sender=Patient)
def create_medcard_for_patient(sender, instance, created, **kwargs):
    """Створює медичну картку, як тільки створюється об'єкт пацієнта."""
    if created:
        MedCards.objects.create(patient=instance)


@receiver(post_save, sender=FamilyDoctor)
def sync_doctor_to_medcards(sender, instance, created, **kwargs):
    """Оновлює поле 'doctor' в MedCards при зміні FamilyDoctor."""

    patient = instance.patient
    new_doctor = instance.doctor
    try:
        medcard = MedCards.objects.get(patient=patient)
        if medcard.doctor != new_doctor:
            medcard.doctor = new_doctor
            medcard.save(update_fields=['doctor'])

    except MedCards.DoesNotExist:
        pass
```

Рисунок 4.3 – Функція створення та автоматичного редагування записів

Перехід до автоматичного створення медичної карти під час додавання пацієнта значно підвищили цілісність даних. Додаткове обмеження на редагування сімейного лікаря лише через таблицю FamilyDoctor гарантує коректність зв’язків і виключає несанкціоновані зміни даних у медичній карті.

У сукупності ці рішення формують надійну модель контролю доступу, яка відповідає принципам розмежування прав, прозорості змін та захисту від некоректних операцій, що є критично важливим для медичних інформаційних систем.

4.2.3 Заходи щодо забезпечення інформаційної безпеки

У межах розробки програмної частини модулю ЕМК, попередньо було вирішено застосовувати гібридний підхід комбінування моделей RBAC, ABAC та PBAC. Цей підхід добре інтегрується у логіку представлень (views), шаблонів (templates) і налаштувань фреймворку Django. Основою політики доступу слугує рольова модель RBAC, реалізована через вбудовані таблиці Django: auth_group, auth_permission. У систему внесено такі групи:

- Пацієнти – мають доступ виключно до власної ЕМК та не можуть переглядати дані інших користувачів. Для своїх даних є дозвіл лише перегляду.
- Медичний персонал – має розширені привілеї: перегляд усіх медичних карт, записів про щеплення та додавання нових щеплень.
- Лікарі – мають доступ до перегляду всіх ЕМК, внесення змін до позначок стану здоров'я та можливість створювати направлення на лабораторні дослідження.
- Сімейні лікарі – поєднують у собі можливості «Лікаря» та «Медичного персоналу».
- Лаборанти – не мають доступу до повної медичної карти, а бачать лише направлення, адресовані конкретній лабораторії та можуть вносити результати аналізів.

Саме за допомогою цього розподілу, а також встановлення відповідних груп та дозволів можливим розділити інтерфейс для різних груп, що є першою ланкою інформаційної безпеки. Різний інтерфейс мінімізує імовірність як випадкового, так і навмисного доступу до інформації, що не відповідає повноваженням користувача.

```
{% if "ЛІКАРІ" in user_groups or "МЕД.ПЕРСОНАЛ" in user_groups %}
  <li class="nav-item">
    <a class="nav-link text-center" href="{% url 'card:med_referral' %}">Мої направлення</a>
  </li>
  <hr class='hr_cards'>
  <li class="nav-item">
    <a class="nav-link text-center fw-bold" href="{% url 'card:all_cards' %}">Медичні картки</a>
  </li>


  {% if card %}
    <hr class='hr_cards'>
    <li class="nav-item">
      <a class="nav-link text-center" href="{% url 'card:index' card.display_id %}">Загальна інформація</a>
    </li>
    <hr class='hr_cards'>
    <li class="nav-item">
      <a class="nav-link text-center" href="{% url 'card:vaccine' card.display_id %}">Щеплення</a>
    </li>
    <hr class='hr_cards'>
    <li class="nav-item">
      <a class="nav-link text-center" href="{% url 'card:referral' card.display_id %}">Направлення пацієнта</a>
    </li>
    <hr class='hr_cards'>
    <li class="nav-item">
      <a class="nav-link text-center" href="{% url 'card:result_analysis' card.display_id %}">Результати аналізів</a>
    </li>
  {% endif %}

{% elif "ПАЦІЄНТИ" in user_groups %}
```

Рисунок 4.4 – Фрагмент коду макету сторінки

На рисунку 4.4 також є умова ***{% if card %}***. Ця конструкція є прикладом застосування ABAC в шаблонах. У цьому випадку атрибутом виступає фактична наявність доступної медичної карти, що дозволяє контролювати видимість пунктів меню. Щоб забезпечити більший рівень безпеки у файлах views.py було продубльовано рольову та атрибутну модель на рівні представлень. (Рис.4.5)

```
88
89  ### --Вакцинація-- ###
90  @login_required
91  def vaccine(request, card_id):
92      user_groups = request.user.groups.values_list('name', flat=True)
93      if "ЛАБОРАНТИ" in user_groups:
94          raise Http404("Сторінка не знайдена")
95       # Отримуємо медичну картку за її ID з вибіркою пов'язаних об'єктів (пацієнт, користувач пацієнта, лікар, користувач лікаря)
96      card = get_object_or_404(MedCards.objects.select_related('patient', 'patient__user', 'doctor', 'doctor__user'), id=card_id)
97      card_vaccinations = CardVaccine.objects.filter(medcard=card).order_by('id')
98      current_user = request.user
99      # Отримуємо користувача лікаря, якщо лікар прив'язаний до медичної картки, інакше None
100     doctor = card.doctor.user if card.doctor else None
101     is_medical_staff = "МЕД.ПЕРСОНАЛ" in user_groups
102     # Перевіряємо, чи поточний користувач є лікарем, прив'язаним до медичної картки
103     can_edit = doctor == current_user or is_medical_staff
104     context = {
105         'card': card,
106         'card_vaccinations': card_vaccinations,
107         'can_edit': can_edit}
108     return render(request, 'cards/vaccine_profile.html', context)
109
264 @login_required
265 def result_id_profile(request, card_analysis_id):
266     user = request.user
267     patient = get_object_or_404(Patient, user=user)
268     card_analysis = get_object_or_404(ResultAnalysis, id=card_analysis_id, medcard__patient=patient)
269
270     card = card_analysis.medcard
271
272     context = {
273         'user': user,
274         'patient': patient,
275         'card': card,
276         'card_analysis': card_analysis,
277     }
278     return render(request, 'cards/result_analysis_view.html', context)
```

Рисунок 4.5 – Фрагмент коду функції перегляду результату аналізів та вакцин

На рисунку 4.5 добре показана реалізація ABAC+RBAC+PBAC. На рівні представлень (views.py) ABAC проявляється у перевірці таких атрибутів:

- приналежність медичної карти пацієнту;
- наявність зв’язку між лікарем та картою;
- факт належності результатів аналізу конкретному користувачу.

RBAC в шаблонах застосовується задля зручності інтерфейсу та не є повноцінною гарантією безпеки. Згідно з OWASP, усі механізми контролю доступу мають виконуватися виключно на серверній стороні, оскільки клієнтська логіка є ненадійною [34]. Тому в представленнях важливо явно вказувати умови що

пов’язані з групами користувачів. На рисунку 4.5 присутній декоратор @login_required, що є методом реалізації PBAC.

Декоратор @login_required реалізує політику доступу на рівні представлень, обмежуючи виконання view лише автентифікованими користувачами. Це є прикладом механізму PBAC, оскільки доступ визначається не роллю або атрибутами, а встановленою політикою «дозвіл доступу лише за наявності чинної автентифікованої сесії». Таким чином, цей декоратор виступає базовим елементом захисту системи, запобігаючи виконанню будь-яких операцій анонімними користувачами. Його слід зазначати для всіх представлень окрім загальнодоступних сторінок, функцій входу та реєстрації в системі. (Рис.4.6)

```
26
27
28  def registration(request):
29      if request.method == 'POST':
30          form = UserRegForm(data=request.POST)
31          if form.is_valid():
32              form.save()
33              return HttpResponseRedirect(reverse('user:login'))
34      else:
35          form = UserRegForm()
36      context = {'form': form}
37      return render(request, 'users/reg.html', context)
38
39
40  @login_required
41  def logout(request):
42      auth.logout(request)
43      return redirect(reverse('public:index'))
44
45
46  @login_required
47  def profile(request):
48      user = request.user # поточний користувач
49      user_groups = user.groups.values_list('name', flat=True) # список груп
50
51      form = ProfileForm(instance=user) # отримання даних з форми профілю
52      patient_form = None ## змінні з незаповненими формами
53      doctor_form = None
```

Рисунок 4.6 – Функції реєстрації, виходу та частина коду для функції відображення профілю

Окрім моделей контролю доступу важливим аспектом безпеки медичної інформаційної системи є коректна реалізація механізмів автентифікації, зокрема політик управління паролями. Фреймворк Django має вбудовану систему перевірки складності паролів та реалізує хешування паролів. Це унеможливлює зберігання відкритих паролів у базі даних. (Рис.4.7)

12	12	pbkdf2_sha256$1000000$QRLmS2s34o...	2025-11-02 ...	false	test_p5	false	true
13	13	pbkdf2_sha256$1000000$h89NmCnGEO...	2025-11-16 ...	false	test_p3	false	true
14	14	pbkdf2_sha256$1000000$kLUK4MJeyV...	2025-11-07 ...	false	test_p4	false	true
15	15	pbkdf2_sha256$1000000$qrHx4eXlWIzo...	2025-11-14 ...	false	test_p6	false	true
16	16	pbkdf2_sha256$1000000$9PQkkjehJTlx...	[null]	false	test_l3	false	true

Рисунок 4.7 – База даних з хешованими паролями

У попередній реалізації реєстрація всіх користувачів здійснювалася через веб-інтерфейс системи. У цьому випадку паролі автоматично оброблялися засобами Django та зберігалися у вигляді криптографічного хешу. Однак після оновлення політики безпеки було прийнято рішення, що профілі працівників медичного закладу (лікарів, медичний персонал, лаборантів) має створювати адміністратор в адмін-панелі.

У процесі тестування було виявлено, що при використанні стандартної реєстрації моделі користувача через ModelAdmin у панелі адміністратора паролі зберігалися у відкритому вигляді. Це пов’язано з тим, що базовий ModelAdmin не застосовує метод set_password() і не виконує хешування введеного паролю.

Для вирішення цієї проблеми було розширено стандартний клас UserAdmin (Рис.4.8), який містить коректні форми та механізми автоматичного хешування паролів.

```
class CustomUserAdmin(UserAdmin):
    # Додаємо до стандартних полів нові в форму додавання користувача
    add_fieldsets = UserAdmin.add_fieldsets + (
        ('Персональна інформація', {'fields': ('email','last_name', 'first_name',
                                               'patronymic', 'groups', 'phone_number', 'img')}),
    )

    # Тут ми додаємо ТІЛЬКИ нові поля для кастом юзер, щоб уникнути дублювання email, first_name, last_name.
    fieldsets = UserAdmin.fieldsets + (
        ('Додаткова інформація профілю', {'fields': ('patronymic', 'phone_number', 'img')}),
    )
```

Рисунок 4.8 – Реєстрація кастомної моделі користувача з використанням UserAdmin

Це забезпечило застосування однакових криптографічних механізмів незалежно від способу створення облікових записів: через сайт або через адмінпанель.

Окрім цього було вдосконалено валідатори, що відповідають за складність паролів, а саме збільшена мінімально допустима кількість символів (Рис.4.9).

```
AUTH_PASSWORD_VALIDATORS = [
    {'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',},

    {'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {"min_length": 14}},

    {'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',},

    {'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',},
]
```

Рисунок 4.9 – Налаштування вбудованих валідаторів

Захист від CSRF у Django реалізовано через CsrfViewMiddleware, який генерує унікальний токен для кожної сесії. Токен обов’язково передається у формах або заголовках AJAX-запитів і перевіряється сервером у запитах. Без наявності цього токену буде повідомлення про помилку. Це запобігає виконанню шкідливих запитів від імені користувача з інших сайтів. (Рис.4.10)

```
<form action="{% url 'user:login' %}" method='POST'>
    {% csrf_token %}
    {% if request.GET.next %}
        <input type="hidden" name="next" value={{request.GET.next}}>
    {% endif %}


    <div class="mb-3">
        <label for="id_username" class="form-label">Логін</label>
        <input type="text" class="form-control"
        name="username" id="id_username"
        value="{% if form.username.value %}{{form.username.value}}{% endif %}"
        placeholder="Введіть нік" required>
    </div>
    <div class="mb-3">
        <label for="id_password" class="form-label">Пароль</label>
        <input type="password" class="form-control"
        name="password" id="id_password"
        placeholder="Введіть пароль" required>
    </div>
    <button type="submit" class="custom-button">Ввійти</button>
</form>
```

Рисунок 4.10 – CSRF-токен в формі входу в систему

Не менш вагомою проблемою є відображення даних в URL-маршрутах. В попередній версії проекту були застосовані конвертори URL-адрес [2]. (Рис.4.11)

```
path('<int:card_id>/',views.index, name='index'), # med-card/00005/
path('vaccine/<int:card_id>/',views.vaccine, name='vaccine'), # med-card/vaccine/00005/
path('examination/<int:card_id>/',views.examination, name='examination'), # med-card/examination/00005/
```

Рисунок 4.11 – URL-маршрути з конверторами для медичних карт

Конвертори URL – це частина маршрутизації в Django, яка дозволяє визначати шаблони для URL-адрес. Вони використовуються для визначення структури URL-адрес, які відповідають конкретним шаблонам, даним або форматам. Такі маршрути мають ідентифікатор – частину URL, яка або генерується, або як на Рис. 4.11 передається через змінну card_id[2]. Проте показ не зашифрованого ідентифікатора є вразливістю з точки зору безпеки. Зловмисник може змінити число та потрапити на чужу сторінку, що є порушенням конфіденційності.

Для підвищення рівня надійності та зменшення ризику несанкціонованого доступу до вразливої інформації було застосовано обфускацію ідентифікаторів. Метод полягає у приховуванні реального числового первинного ключа (PK) від користувача шляхом перетворення його на нечитаємий фрагмент URL-адреси, що забезпечує одночасну можливість однозначної ідентифікації об’єкта на сервері. Це зменшує ризики несанкціонованого доступу до даних, при цьому зберігає можливість однозначної ідентифікації об’єкта на сервері через декодування hashid[35].

Реалізація включала чотири основні кроки: зміна URL для передачі hashid замість числового ID (Рис.4.12), модифікація view для декодування hashid у PK та отримання об’єкта з бази даних (Рис.4.13), створення централізованого файлу utils.py з функціями encode_id і decode_id (Рис.4.14), а також додавання custom template filter для автоматичного кодування ID у шаблонах (Рис.4.15). Функції encode_id і decode_id варто помістити в новий додаток core.

```
path('view/<str:hid>/', views.index, name='index'),
path('edit-mark/<str:hid>/',views.edit_mark, name='edit_mark'),
path('vaccine/<str:hid>/',views.vaccine, name='vaccine'),
path('add-vaccine/<str:hid>/', views.add_vaccine, name='add_vaccine'),
```

Рисунок 4.12 – Частина змінених URL-маршрутів

```
@login_required
def edit_mark(request, hid):
    card_id = decode_id(hid)  # розшифровуємо в PK
    if card_id is None:
        raise Http404("Сторінка не знайдена")
    user_groups = request.user.groups.values_list('name', flat=True)
    if not "ЛІКАРІ" in user_groups:
        raise Http404("Сторінка не знайдена")
    # Отримуємо конкретну медкарту за id
    card = get_object_or_404(
        MedCards.objects.prefetch_related('individualmarks_set'), id=card_id)
    # Обробка POST-запиту (збереження форми)
    if request.method == 'POST':
        # Створюємо FormSet, прив'язуючи дані з POST та instance
        formset = IndividualMarksFormSet(request.POST, instance=card)
        if formset.is_valid():
            formset.save()
            return redirect('card:index', hid=hid)
    # Обробка GET-запиту (відображення форми)
    else:
        formset = IndividualMarksFormSet(instance=card)
    context = {
        'card': card,
        'formset': formset}
    return render(request, 'cards/edit_signal_marks.html', context)
```

Рисунок 4.13 – Приклад модифікованої функції (view)

```
1  from django.conf import settings
2  from hashids import Hashids
3
4  hashids = Hashids(salt=settings.HASHIDS_SALT, min_length=10)
5
6  def encode_id(num: int):
7      ### Перетворює числовий PK в hashid.
8      return hashids.encode(num)
9
10 def decode_id(hashid: str):
11     ### Перетворює hashid назад у PK.
12     ### Повертає None, якщо хеш некоректний.
13     decoded = hashids.decode(hashid)
14     return decoded[0] if decoded else None
```

Рисунок 4.14 – Файл utils.py

Варто зазначити, що в змінну HASHIDS_SALT поміщено значення, згенероване випадковим чином, яке використовується як криптографічна «сіль» для генерації hashid. Це значення додає додаткову непередбачуваність до закодованих ідентифікаторів, щоб навіть при відомих PK hashids залишалися унікальними і невгадуваними.

```
1   from django import template
2   from core.utils import encode_id
3
4   register = template.Library()
5
6   @register.filter
7   def encodeid(value):
8       """Перетворює числовий id в hashid"""
9       return encode_id(value)
0
```

Рисунок 4.15 – Кастомний фільтр для кодування ID

4.3 Результати розробки

Після написання коду та розробки додаткового функціоналу, з урахуванням проведеного аналізу методів управління доступом і загроз інформаційній безпеці, систему було розширено та вдосконалено. Першочергово за сучасними стандартами Держспецзв’язку було змінено вимогу до довжини пароля. На рисунку 4.16 зображена помилки валідації, серед яких помилка про недостатню довжину пароля.

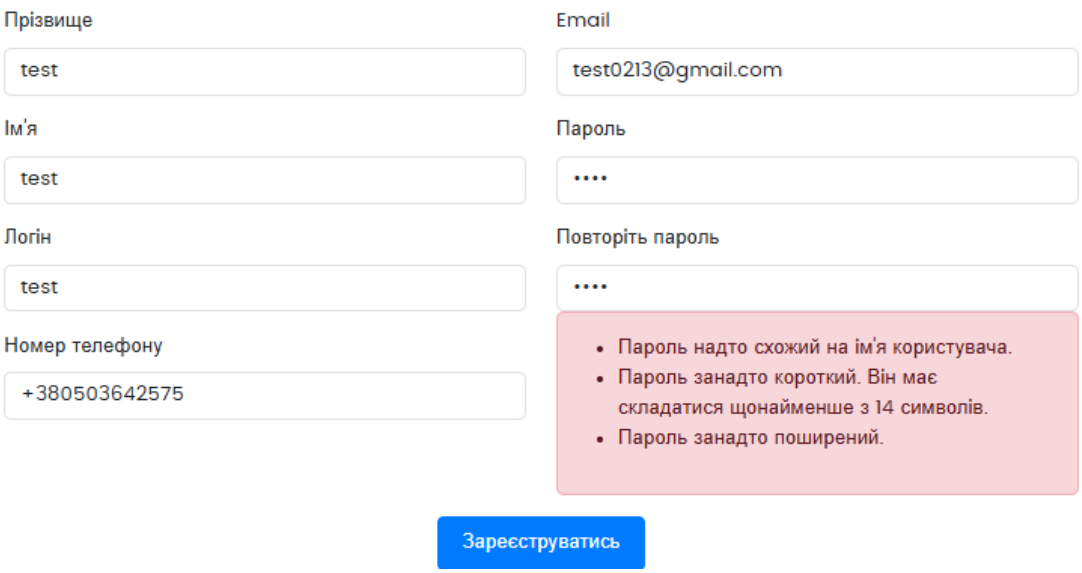


Рисунок 4.16 – Форма реєстрації

Як і в минулій версії форма входу в систему має попередження про неправильне введення інформації. При цьому не зазначено, яке поле було хибним, що є правильним з точки зору безпеки. (Рис.4.17)

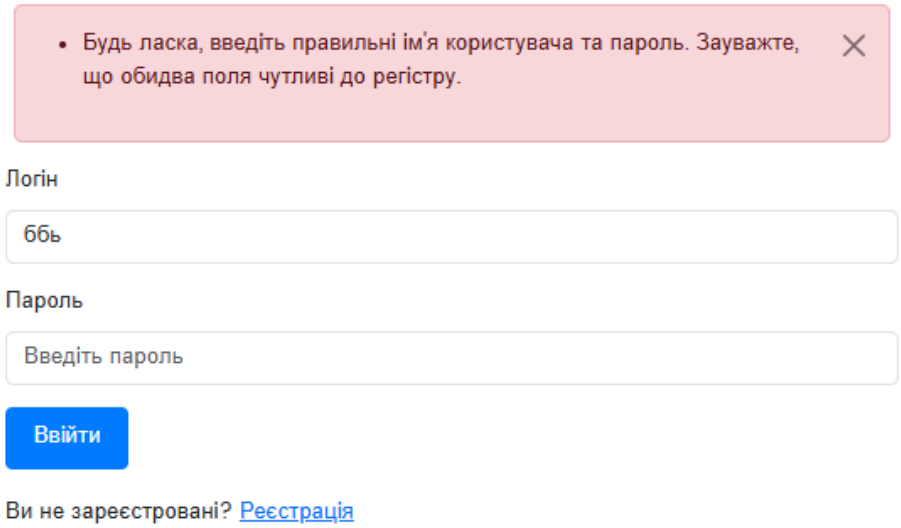


Рисунок 4.17 – Форма входу в систему

На основі сформульованих вимог і спроєктованої моделі контролю доступу функціонал системи був структуровано за ролями користувачів. У результаті розробки виділено різний функціонал для лаборантів, молодшого медичного

персоналу, лікарів, сімейних лікарів і пацієнтів, що дозволило забезпечити керування правами, прозорість взаємодії та відповідність політикам інформаційної безпеки.

Додано персоналізований ОК для лаборантів та ММП. (Рис.4.18-4.19)

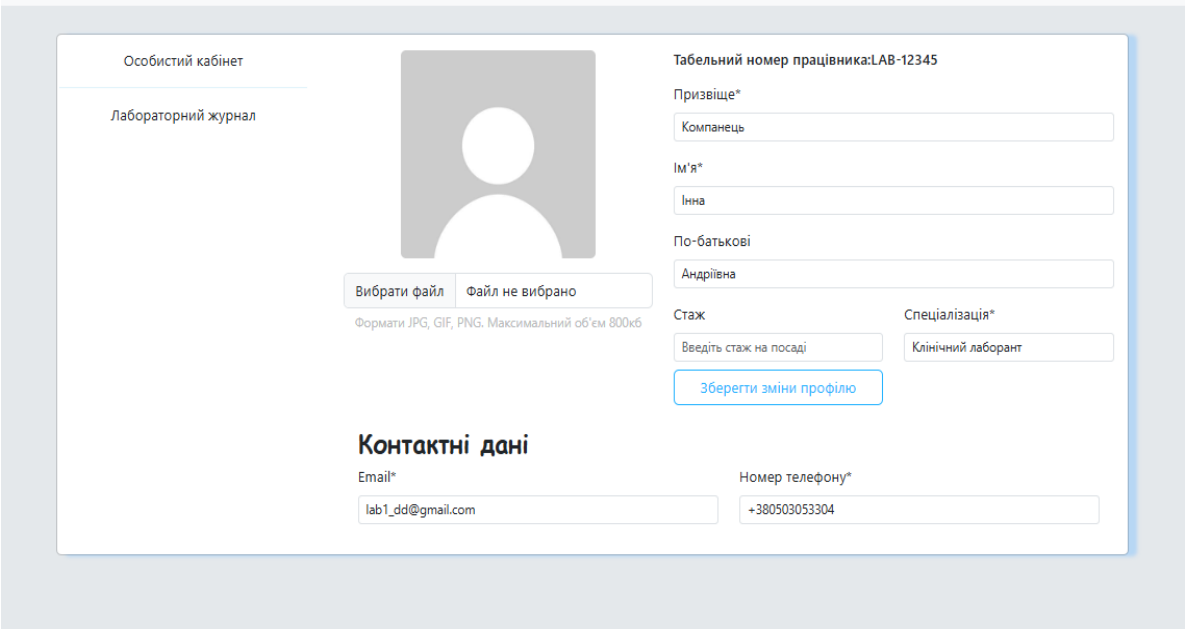


Рисунок 4.18 – ОК лаборантів

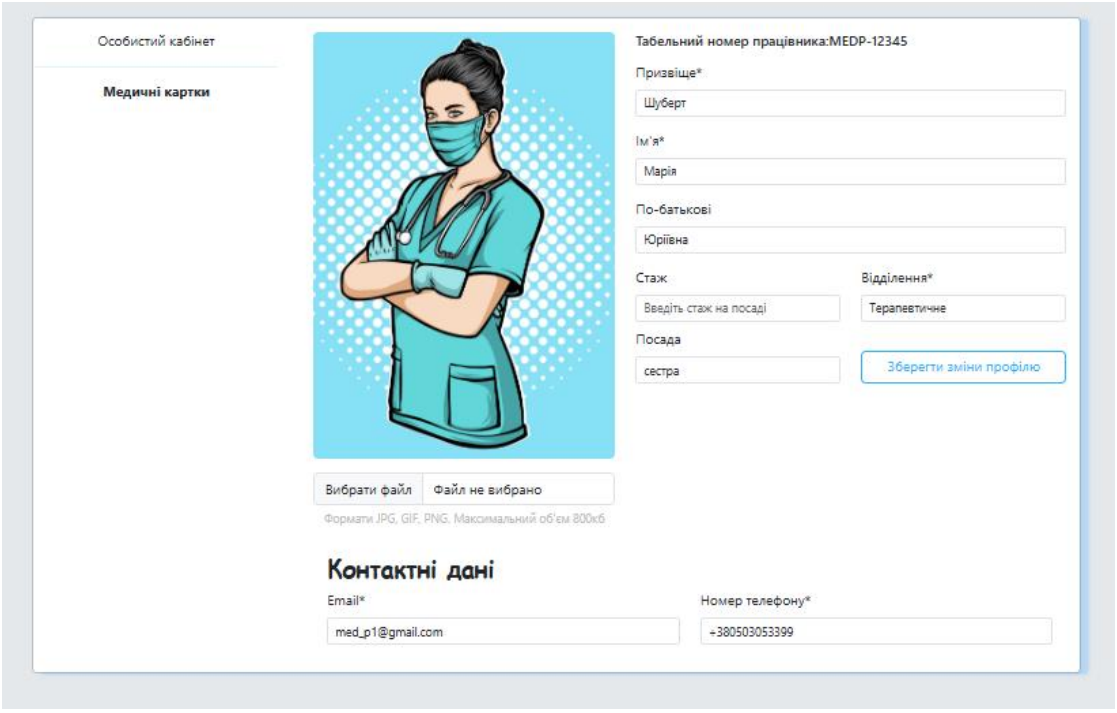


Рисунок 4.19 – ОК молодшого медичного персоналу

В ОК іншого персоналу лікарні немає зайвих пунктів допоки не буде обрана відповідна медична карта. Окремо варто зазначити, що лаборанти не мають доступу до розділів, пов’язаних із медичними картками пацієнтів. Такий підхід забезпечує дотримання принципу мінімальних привілеїв, відповідно до якого користувач отримує лише ті права, які необхідні для виконання його службових обов’язків.

Натомість, лаборанти мають доступ до лабораторного журналу, де розміщені всі направлення, що надійшли в лабораторію (без анульваних). А також форму заповнення результатів та сторінку для перегляду. (Рис.4.20-4.23)

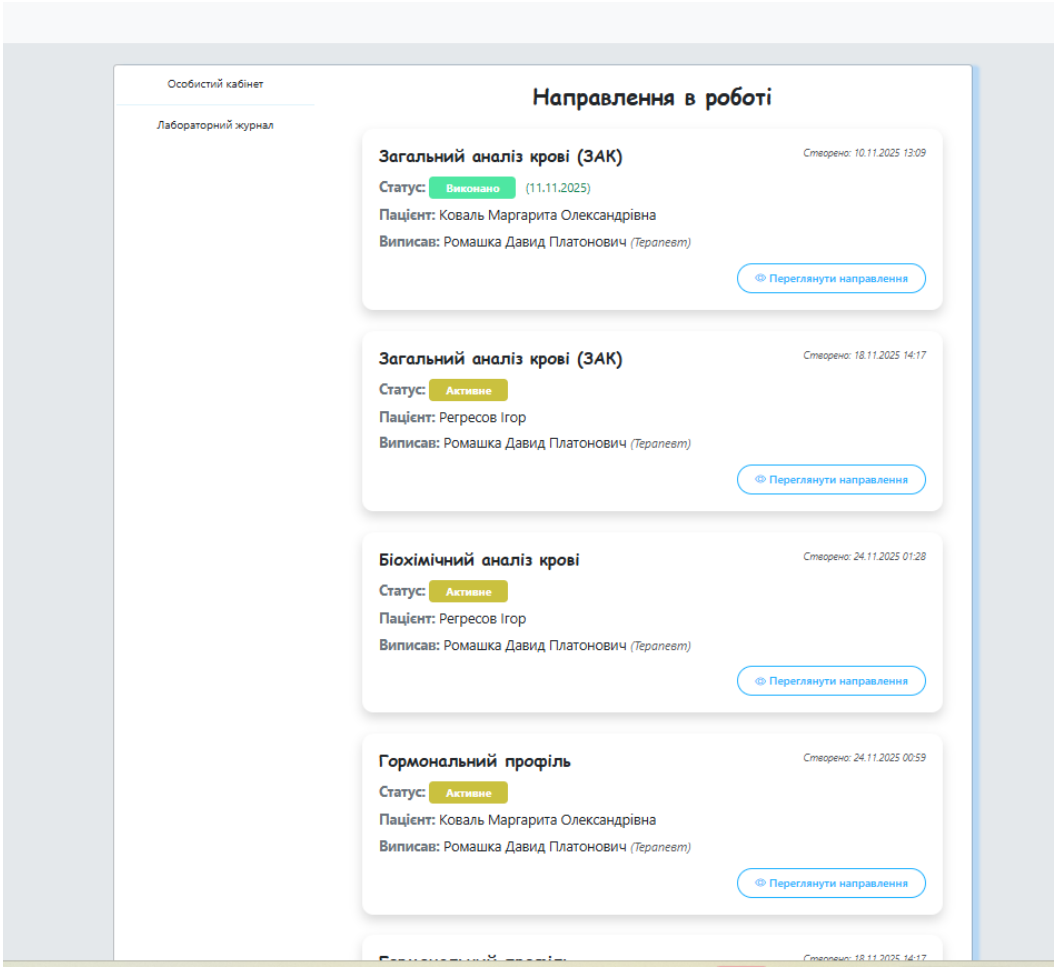


Рисунок 4.20 – Вкладка «Лабораторний журнал» для лаборантів

Вкладка «Направлення» для ММП і пацієнтів призначена лише для перегляду та схожа на лабораторний журнал, що зображений вище.

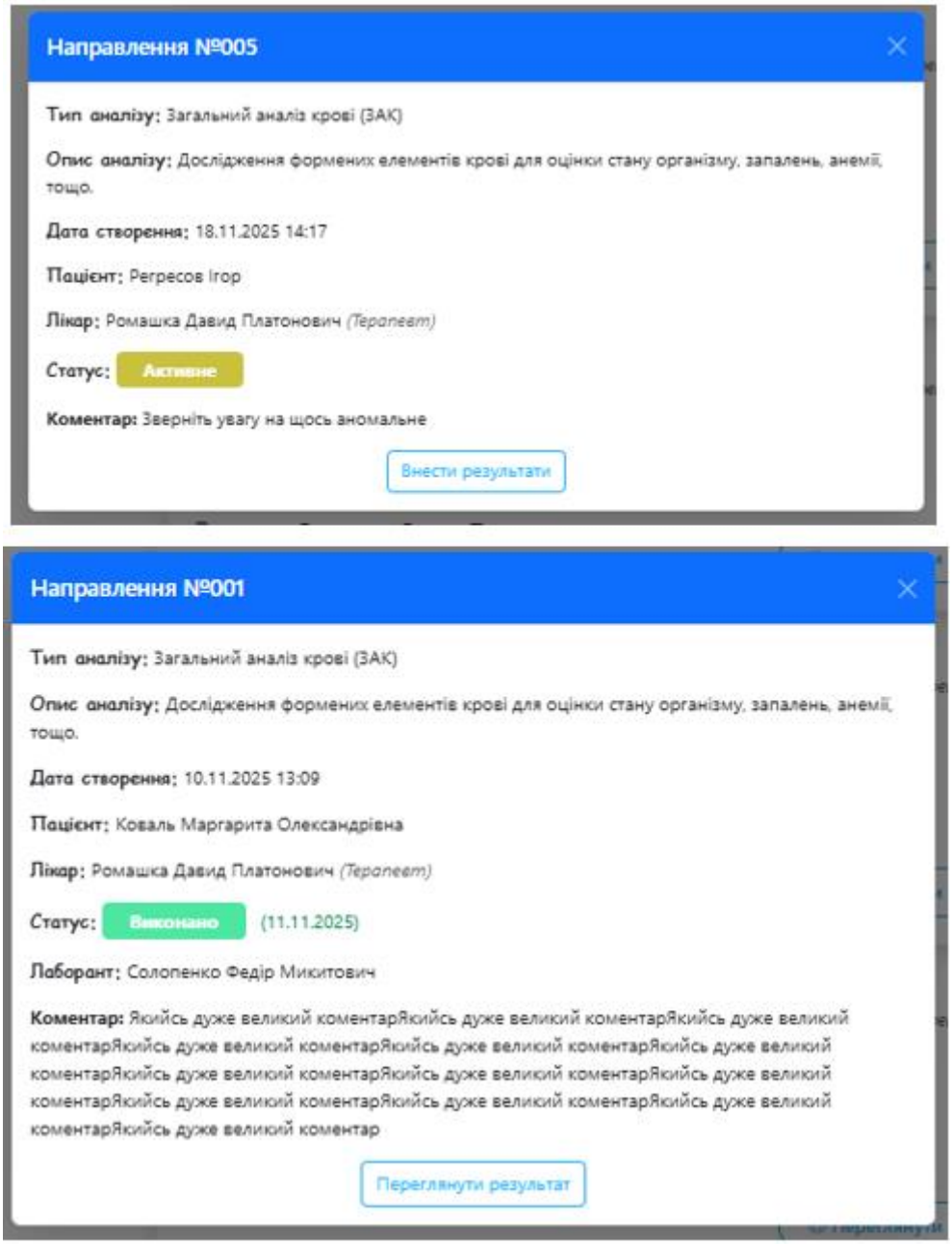


Рисунок 4.21 – Модальні вікна з розширеною інформацією по направленню для лаборанта

Пацієнти не бачать коментарі під час перегляду направлення в модульному вікні. Це зроблено для того, щоб запобігти розкриттю службової інформації медичного персоналу та уникнути можливих непорозумінь у трактуванні професійних приміток. Кнопка «Внести результати» доступна лише лаборантам.

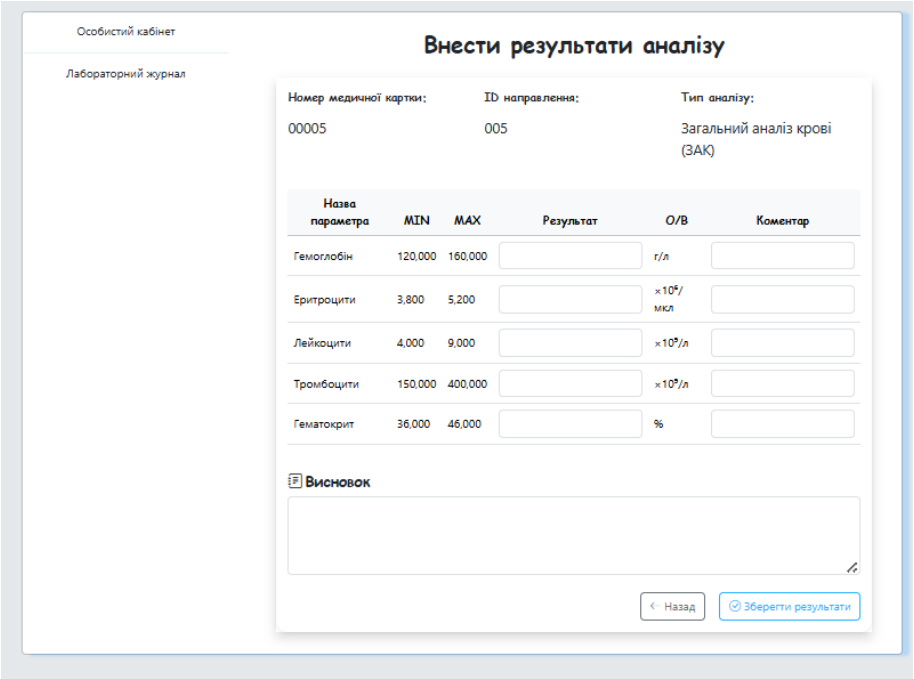


Рисунок 4.22 – Форма вводу інформації про результати аналізів

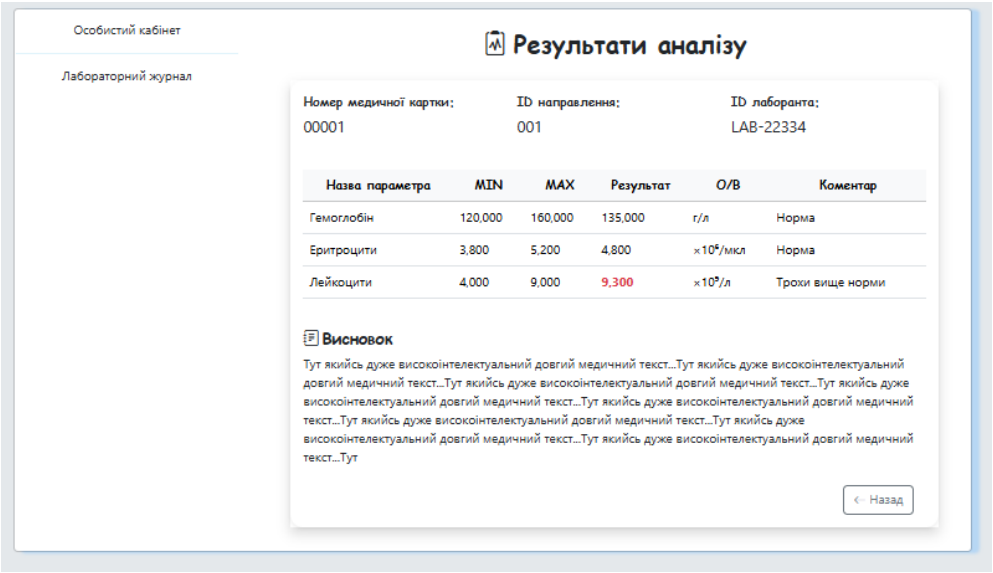


Рисунок 4.23 – Дані лабораторного дослідження

В подальшому можна організувати фільтрацію направлень за статусом, за лікарем тощо. Також можна розробити функцію-запит на редагування вже виконаних направлень з можливим погодженням лише Сімейного лікаря.

Було змінено функціонал, який належить до вакцинації. Оскільки видалення медичних записів не дозволяється законодавством (впродовж якогось часу), то

видалення було прибрано. ММП та сімейні лікарі можуть не тільки додавати нові записи а й редагувати вже створені. (Рис.4.24)

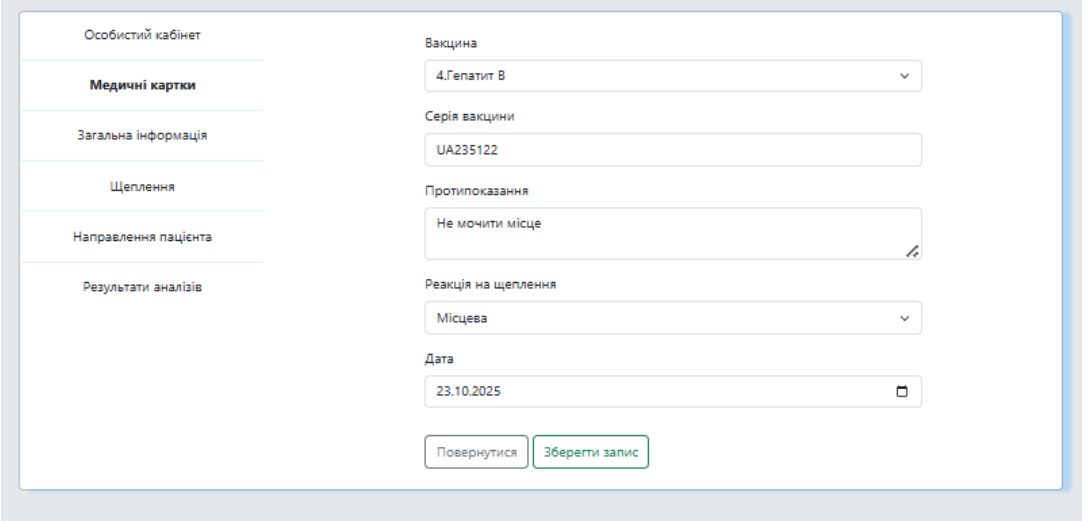


Рисунок 4.24 – Форма редагування щеплень

Також у лікарів є вкладка «Мої направлення», що містить ті записи, що зробив цей лікар. Лікарям, як сімейним так і лікарі-спеціалістам, доступна функція додати направлення. (Рис.4.25-4.26) Також сімейні лікарі та лікарі, яким належить конкретне направлення можуть його редагувати. (Рис.4.27)

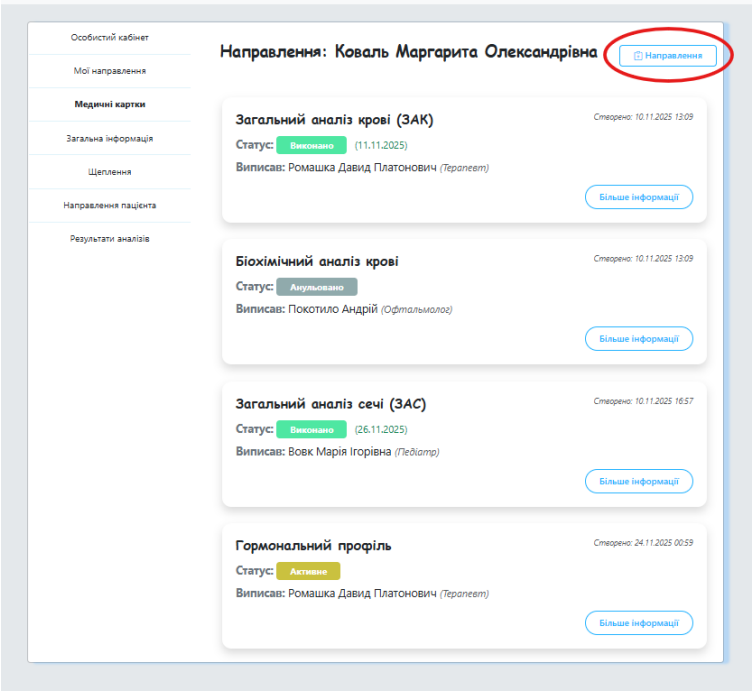


Рисунок 4.25 – Вкладка «Направлення» для лікарів

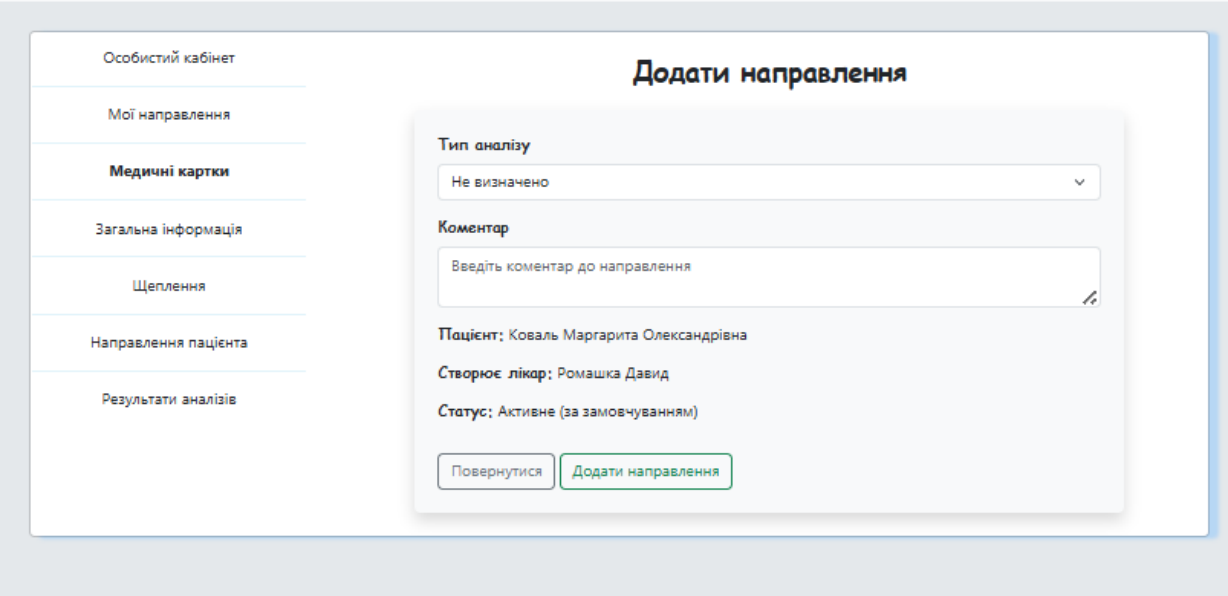


Рисунок 4.26 – Форма додавання направлення

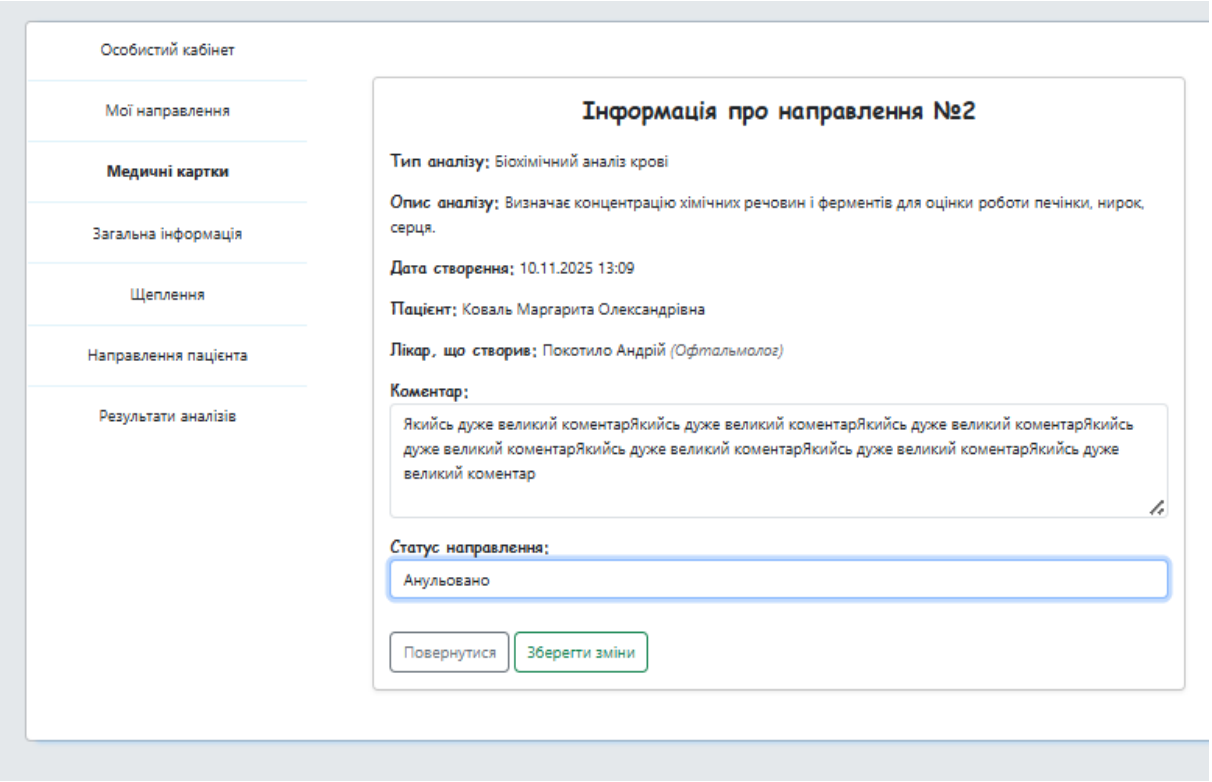


Рисунок 4.27 – Форма редагування направлення

Вкладка «Результати аналізів» (Рис.4.28) містить перелік проведених лабораторних досліджень.

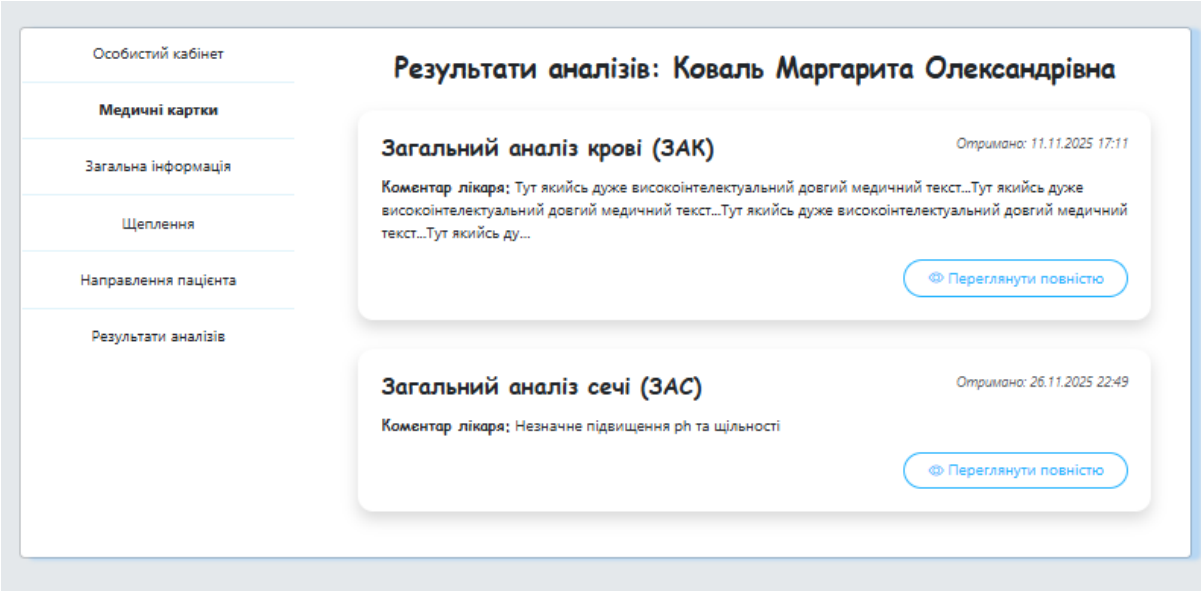


Рисунок 4.28 – Вкдадка «Результати аналізів»

Кнопка «Переглянути повністю» надає доступ до даних лабораторних досліджень, що можна побачити вище. (Рис. 4.23)

Окрім цього, до вкладки, що була в попередній версії модулю ЕМК було додано функцію редагування сигнальних позначок, яка доступна для лікарів. (Рис. 4.29-4.30)

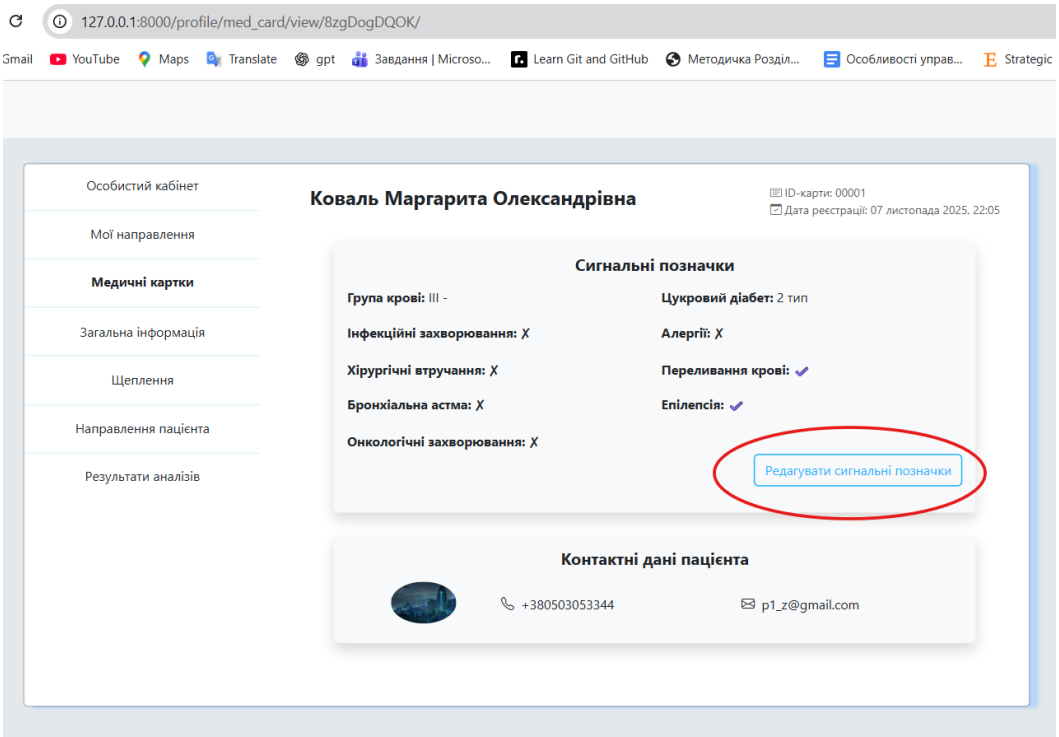


Рисунок 4.29 – Вкладка «Загальна інформація»

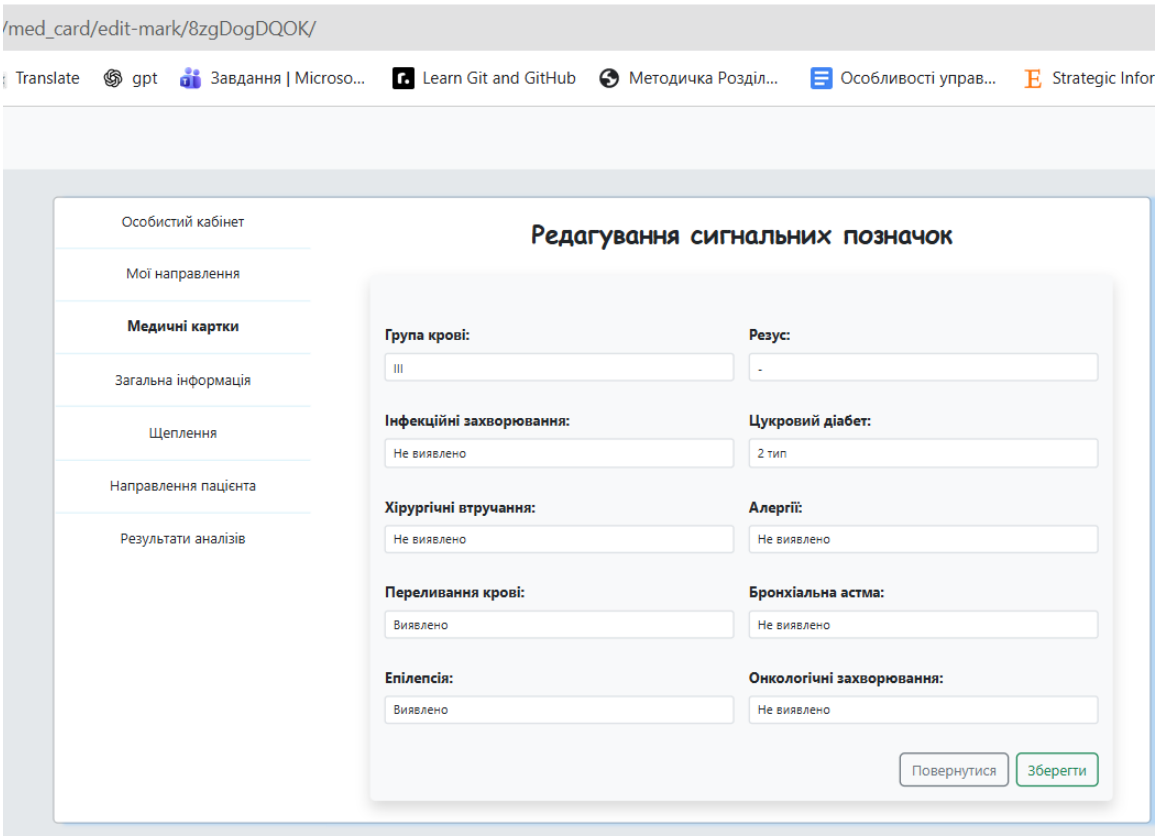


Рисунок 4.30 – Форма редагування сигнальних позначок

На рисунках 4.29-4.30 видно фрагменти URL-адрес з зашифрованими ID медичних карток. Такий захист наразі реалізовано для вклалок «Загальна інформація» та «Щеплення». В подальшому можна захистити і інші URL-адреси за допомогою обфускації ідентифікаторів.

Передбачиний випадок коли в медичній картці немає даних, що відносяться до конкретного розділу. (Рис.4.31) Та відповідно до рекомендації

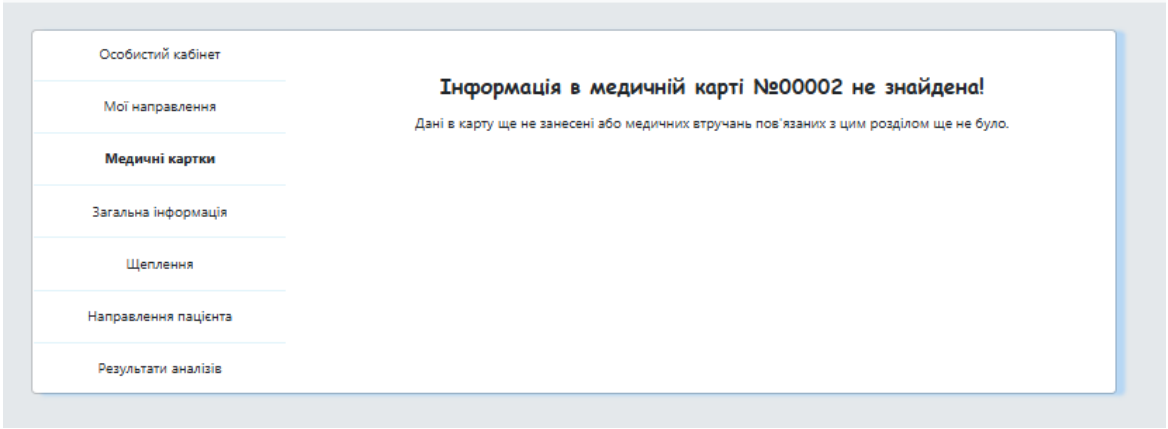


Рисунок 4.31 – Інформація не знайдена

Відповідно до рекомендацій Міністерства охорони здоров’я України щодо укладання декларації з сімейним лікарем, пацієнтам, які ще не підписали декларацію, може відображатися попередження із проханням укласти декларацію. Це нагадування не є обов’язковим нормативом, але допомагає пацієнту отримати доступ до планових послуг первинної медичної допомоги. [36,37]. (Рис.4.32)

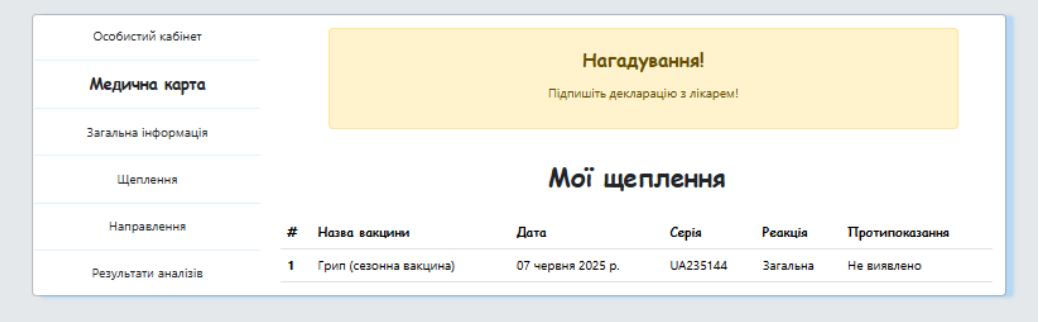


Рисунок 4.32 – Нагадування про підписання декларації

Як видно на рисунку, дані пацієнта зберігаються в системі незалежно від того, чи підписана ним декларація з сімейним лікарем. Такий підхід реалізовано з метою підвищення зручності роботи з системою та забезпечення доступу до власних медичних даних навіть до моменту вибору лікаря.

Для підвищення рівня безпеки системи необхідно передбачити логування дій користувачів у вигляді журналу подій. Хоча розширення функціоналу, що дозволяє змінювати або додавати дані, підвищує зручність і гнучкість роботи системи, воно водночас створює ризик несанкціонованої модифікації даних. Тому всі операції повинні контролюватися через механізми аудиту, що забезпечують відстеження змін, фіксацію часу, користувача та типу дії, а також можливість відновлення попереднього стану даних у разі потреби.

Висновок

У цьому розділі було зосереджено увагу на підвищенні рівня безпеки та розширенні функціональних можливостей ЕМК. Для забезпечення актуальності та

підтримки сучасних стандартів було обрано новіші версії програмних платформ, зокрема Django 5.2.

Моделі баз даних розширено: окрім пацієнтів, лікарів та сімейних лікарів, додано моделі лаборантів та молодшого медичного персоналу. Також впроваджено нові функціональні можливості – редагування відомостей про щеплення для медичного персоналу, керування сигнальними позначками для всіх лікарів, а також окремі розділи ЕМК, що відповідають направленням та результатам лабораторних досліджень.

Опрацьовано питання захисту даних і контролю доступу. Оскільки базова рольова модель Django (RBAC), що базується на групах, виявилася недостатньо гнучкою, її було вирішено використовувати передусім у шаблонах сторінок для відображення різних елементів інтерфейсу користувачів відповідно до їх груп.

Більш детальний і точний розподіл прав доступу реалізовано за допомогою декораторів та у представленнях, де використовуються команди з відповідними атрибутами. Такий підхід дозволив сформувати наближену до раніше запланованої гібридної концепції RBAC+ABAC+PBAC модель, яка враховує не лише роль користувача, а й контекст виконання дії та властивості об’єкта доступу. Окрему увагу приділено захисту даних пацієнтів. Були усунені випадки, коли чутливі ідентифікатори «просвічувалися» в URL. Також додатково розглянуті та застосовані інструменти, що передбачені фреймворком Django, для запобігання SQL-ін’єкціям, підвищення стійкості до CSRF-атак та інших типових загроз веб-додатків. Реалізовано принцип мінімальних привілеїв, відповідно до якого кожен користувач отримує доступ лише до тих даних, що дійсно потрібні для виконання його професійних функцій. Наприклад, лаборантам відкрито лише модуль роботи з направленнями та можливість додавання результатів аналізів, без доступу до повної медичної картки пацієнта.

Таким чином, у ході роботи було створено більш захищений, структурований та функціонально розширений модуль ЕМК, який є вдосконаленою версією попереднього прототипу як у плані безпеки, так і з погляду відповідності практичним потребам медичної системи.

ВИСНОВОК

У кваліфікаційній магістерській роботі було здійснено комплексне дослідження питань інформаційної безпеки в МІС, проаналізовано підходи до управління доступом та вдосконалено прототип електронної медичної картки, який був розроблений для бакалаврської роботи. Отримані результати охоплюють як теоретичні аспекти, так і практичну реалізацію концепцій безпеки у програмному середовищі.

На основі аналізу бізнес-процесів медичної сфери та впровадження інформаційних технологій до сфери охорони здоров'я визначено місце й функціональну роль МІС у сучасній архітектурі електронної системи охорони здоров'я України. Проведено класифікацію МІС за типами користувачів, рівнями охоплення, способами розгортання та архітектурними особливостями. Для кожної категорії визначено характерні ризики та специфічні вразливості, що є критично важливими для формування ефективних політик безпеки.

Аналіз систем-аналогів Helsi, Epic та Dedalus засвідчив, що навіть провідні МІС, які використовують рольову модель доступу, мають низку обмежень – від недостатньої гнучкості авторизації до проблем із прозорістю аудиту доступу та орієнтацією переважно на потреби лікарів. Це підтверджує актуальність пошуку моделей та методів, які краще відповідають принципам конфіденційності, цілісності та доступності медичної інформації.

Важливою складовою роботи став аналіз нормативно-правового поля. Він показав, що Україна перебуває на етапі переходу систем до більш орієнтованих на міжнародні стандарти типу NIST та GDPR.

Дослідження моделей RBAC, ABAC, PBAC та MAC показало, що жодна з них у «чистому вигляді» не задовольняє специфічні вимоги медичної галузі. Проведений SWOT-аналіз продемонстрував їхні сильні та слабкі сторони, можливості інтеграції та ризики застосування. На основі цього сформульовано концепцію комбінованої моделі RBAC+ABAC+PBAC, здатної забезпечити

потрібний рівень гнучкості, підтримку динамічних сценаріїв доступу та відповідність сучасним вимогам кіберзахисту.

Подальше моделювання системи – діаграм варіантів використання, потоків даних (IDEF-0, IDEF-3), класів і логічної структури бази даних – створило комплексну архітектурну основу для майбутньої реалізації. Відображено механізми автентифікації, авторизації, аудитів доступу, процесів делегування прав та визначені політики доступу до інформації або модулю.

Практична частина роботи полягала у вдосконаленні модулю ЕМК, розробленого у бакалаврській роботі. Для оновленої системи обрано новішу версію фреймворку – Django 5.2. В модулі ЕМК було розширено набір ролей (лаборант, ММП) і додано модулі обробки направлень та результатів аналізів, удосконалено моделі бази даних. Особливу увагу приділено питанням захисту: впроваджено принцип мінімальних привілеїв, розмежовано інтерфейси в шаблонах, додано декоратор, що контролює вхід на сторінку неавторизованих користувачів та атрибутивні перевірки. Ці заходи формують практичний аналог комбінованої моделі RBAC+ABAC+PBAC. Усунуто ризики витоку ідентифікаторів, застосовано механізми Django для запобігання загрозам типу SQL-ін’єкцій, CSRF, шифрування паролів.

Результати наукових досліджень були представлені на конференції «Build-Master-Class-2024». Зокрема, у тезі «Безпека web-додатків: SQL-ін’єкції – один з найпопулярніших методів кібератак» наголошено, що SQL-ін’єкції залишаються однією з найпоширеніших і найнебезпечніших загроз для веб-систем, особливо тих, що працюють з конфіденційними даними. Дана теза є підтвердженням застосування у модулі ЕМК механізмів ORM, як ключового засобу підвищення рівня кіберзахисту від SQL-ін’єкцій.[38]

Варто зазначити, що дослідження представлені в тезі «Застосування фреймворку Django для побудови безпечної інформаційної системи управління доступом на прикладі медичної сфери» («Build-Master-Class-2025»), додатково підсилюють практичні результати роботи. У ній підкреслено ефективність використання архітектури MVT, механізмів автентифікації та авторизації, а також

вбудованих засобів захисту Django для створення безпечних систем, що працюють із чутливими медичними даними. Це науково обґрунтовує вибір Django 5.2 як технологічної основи модулю ЕМК та підтверджує правильність впроваджених механізмів контролю доступу, аудитів і технічних засобів захисту, реалізованих у практичній частині магістерської роботи.[39]

Дана робота показала, що традиційні, базові моделі контролю доступу залишаються дієвими й актуальними, однак лише у поєднанні з сучасними підходами вони здатні забезпечити високий рівень захисту. Їх інтеграція у гібридну модель дозволяє охопити широкий спектр сценаріїв, усунути обмеження кожної окремої моделі та відповідати нинішнім міжнародним стандартам безпеки.

Таким чином, у роботі отримано як теоретичні, так і програмні результати, що підтверджують можливість побудови сучасної, безпечної та гнучкої медичної інформаційної системи на основі комбінованих моделей доступу. Запропоновані методи та архітектурні рішення можуть бути використані для подальшого розвитку МІС, удосконалення модулю ЕМК, впровадження додаткових механізмів моніторингу, багатофакторної автентифікації та підтримки інших компонентів електронної системи охорони здоров’я.

Список використаних джерел

1. Бурківський П.О. ІТ технології в медицині / П.О. Бурківський., В.О. Скачков - Тези доповідей І Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення» (19–20 жовтня 2018 р.) – Житомир: Вид. О.О. Євенок, 2018 – С. 104–105.

2. Балобольченкова М. І. Модуль "Медична карта" в інформаційно-комунікаційній системі медичного закладу: бакалаврська робота / М. І. Балобольченкова. – Київ: КНУБА, 2024. – 113 с.

3. Malakhov K. S. Insight into the Digital Health System of Ukraine (eHealth): Trends, Definitions, Standards, and Legislative Revisions [Електронний ресурс] / K. S. Malakhov // International Journal of Telerehabilitation. – 2023. – Vol. 15, No. 2. – С. 6–8. – Режим доступу: https://telerehab.hpu.edu/index.php/Telerehab/article/view/6599/7094

4. Модулі МІС ЕМСІМЕД [Електронний ресурс]. – Режим доступу: https://emci.ua/products/emcimed/moduli/

5. МОЗ України. Як медичному закладу підійти до вибору медичної інформаційної системи (МІС)? [Електронний ресурс]. – Режим доступу: https://moz.gov.ua/uploads/6/33130-yak_obrati_mis.pdf

6. 5 Types of Information Systems [Електронний ресурс]. – 2024. – Режим доступу:https://online.fit.edu/degrees/undergraduate/cis/bachelor-science-computer-information-systems/5-types-of-information-systems/

7. Multi-cloud vs. hybrid cloud: What's the difference? [Електронний ресурс]. – Режим доступу:https://www.cloudflare.com/ru-ru/learning/cloud/multicloud-vs-hybrid-cloud

8. Lazarenko S. Exploring Common Software Architecture Patterns: Monolithic, SOA, Microservices, and Layered [Електронний ресурс]. / S. Lazarenko. – 2023. – Режим доступу: https://staskoltsov.medium.com/exploring-common-software-architecture-patterns-monolithic-soa-microservices-and-layered-804ed88e6bef

9. Красномовець П. Угода з багатьма невідомими. «Київстар» купив сервіс Helsi … [Електронний ресурс] / П. Красномовець. – 20.12.2022. – Режим доступу: https://forbes.ua/company/ugoda-iz-bagatma-nevidomimi-14122022-10093

10. ПРО HELSI [Електронний ресурс]. – Режим доступу: https://helsi.me/about

11. Smith T. Who are the largest EHR vendors? [Електронний ресурс]. / T. Smith. – 2025. – Режим доступу: https://www.ehrinpractice.com/largest-ehr-vendors.html

12. Epic EHR software profile [Електронний ресурс]. – Режим доступу: https://www.ehrinpractice.com/epic-ehr-software-profile-119.html

13. ORBIS U: The Foundation for Modern Digital Healthcare [Електронний ресурс]. – Режим доступу:https://www.dedalus.com/uki/our-offer/products/orbis-u

14. Security [Електронний ресурс]. – Режим доступу: https://www.dedalus.com/uki/security/

15. Unauthorized access [Електронний ресурс]. – Режим доступу: https://www.vpnunlimited.com/help/cybersecurity/unauthorized-access

16. Базовий профіль безпеки системи, де обробляється відкрита або конфіденційна інформація: дод. до наказу Адміністрації Держспецзв'язку від 30.06.2025 № 409 [Електронний ресурс]. – Режим доступу: https://cip.gov.ua/ua/docs/nakaz-pro-zatverdzhennya-bazovogo-profilyu-bezpeki-sistemi-de-obroblyayetsya-vidkrita-abo-konfidenciina-informaciya

17. Article 5 GDPR. Principles relating to processing of personal data [Електронний ресурс]. – Режим доступу: https://gdpr-info.eu/art-5-gdpr/

18. Article 9 GDPR. Processing of special categories of personal data [Електронний ресурс]. – Режим доступу: https://gdpr-info.eu/art-9-gdpr/

19. Article 17 GDPR. Right to erasure (‘right to be forgotten’) [Електронний ресурс]. – Режим доступу: https://gdpr-info.eu/art-17-gdpr/

20. Summary of the HIPAA Security Rule [Електронний ресурс]. – Режим доступу:https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html

21. Аутентифікація, авторизація та ідентифікація: як не сплутати [Електронний ресурс]. – 2023. – Режим доступу:

https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/

22. ISO/IEC 27002:2022. Information security, cybersecurity and privacy protection – Information security controls [Електронний ресурс]. – 2022. – Режим доступу:le arn.ztu.edu.ua/pluginfile.php/271472/mod_resource/content/2/ISO%2027002%202022.pdf

23. NIST Special Publication 800-53, Revision 5. Security and Privacy Controls for Information Systems and Organizations [Електронний ресурс]. – 2020. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

24. Mestci H. RBAC vs ABAC: main differences and which one you should use [Електронний ресурс] / H. Mestci. – Режим доступу: https://www.osohq.com/learn/rbac-vs-abac

25. Васильченко С. Як провести SWOT-аналіз: інструкція, поради та приклади [Електронний ресурс]. / С. Васильченко. – 2024. – Режим доступу: https://happymonday.ua/yak-provesty-swot-analiz

26. Awati R.: mandatory access control (MAC) [Електронний ресурс]. / R. Awati. – 2023. – Режим доступу: https://www.techtarget.com/searchsecurity/definition/mandatory-access-control-MAC

27. Din A. RBAC vs ABAC vs PBAC: How to Choose Between Access Control Models [Електронний ресурс]. / A. Din. – 2025. – Режим доступу: https://heimdalsecurity.com/blog/rbac-vs-abac-vs-pbac/

28. Context-Based Access Control [Електронний ресурс]. – Режим доступу: https://www.apono.io/wiki/context-based-access-control-cbac/

29. Popa M. Policy-Based Access Control (PBAC) – The Complete Know How for Organizations [Електронний ресурс]. / M. Popa. – 2023. – Режим доступу: https://heimdalsecurity.com/blog/policy-based-access-control/

30. Sata M. What is PBAC – Policy Based Access Control? [Електронний ресурс]. / M. Sata. – 2025. – Режим доступу: https://www.authx.com/blog/policy-based-access-control-pbac

31. Курсова робота «Інформаційна система для роботи з медичними картами в лікарні. UML-проектування» /М.І. Балобольченкова. – 2023. – 21 с.

32. Ізмайлова О.В. Організаційно-технологічні моделі в інформаційних технологіях управління будівництвом: Навчальний посібник. – К.: КНУБА, 2005. – 140 с.

33. Django documentation [Електронний ресурс]. – Режим доступу: https://django.fun/docs/django/4.2/

34. Access control/OWASP [Електронний ресурс]. – Режим доступу: https://andrewwhite.gitbooks.io/owasp/content/03-Build/0x05-AccessControl.html

35. hashids [Електронний ресурс]. – Режим доступу: https://pypi.org/project/hashids/

36. МОЗ України. Наказ від 19.03.2018 № 503 «Про затвердження Порядку вибору лікаря, який надає первинну медичну допомогу, та форми декларації про вибір лікаря, який надає первинну медичну допомогу» [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/main/z0347-18#Text

37. Капацін Ю. Декларація про вибір лікаря: PMD-бланк і правила заповнювання [Електронний ресурс] / Ю. Капацін. – 2025. – Режим доступу: https://medplatforma.com.ua/article/1086-deklaratsiya-pro-vibir-likarya-pmd-blank-i-pravila-zapovnyuvannya

38. Балобольченкова М., Шабала Є. Безпека web-додатків: SQL-ін'єкції – один з найпопулярніших методів кібератак / М. Балобольченкова, Є. Шабала // «БУД-МАЙСТЕР-КЛАС-2024». – 2024. – С. 433–434.

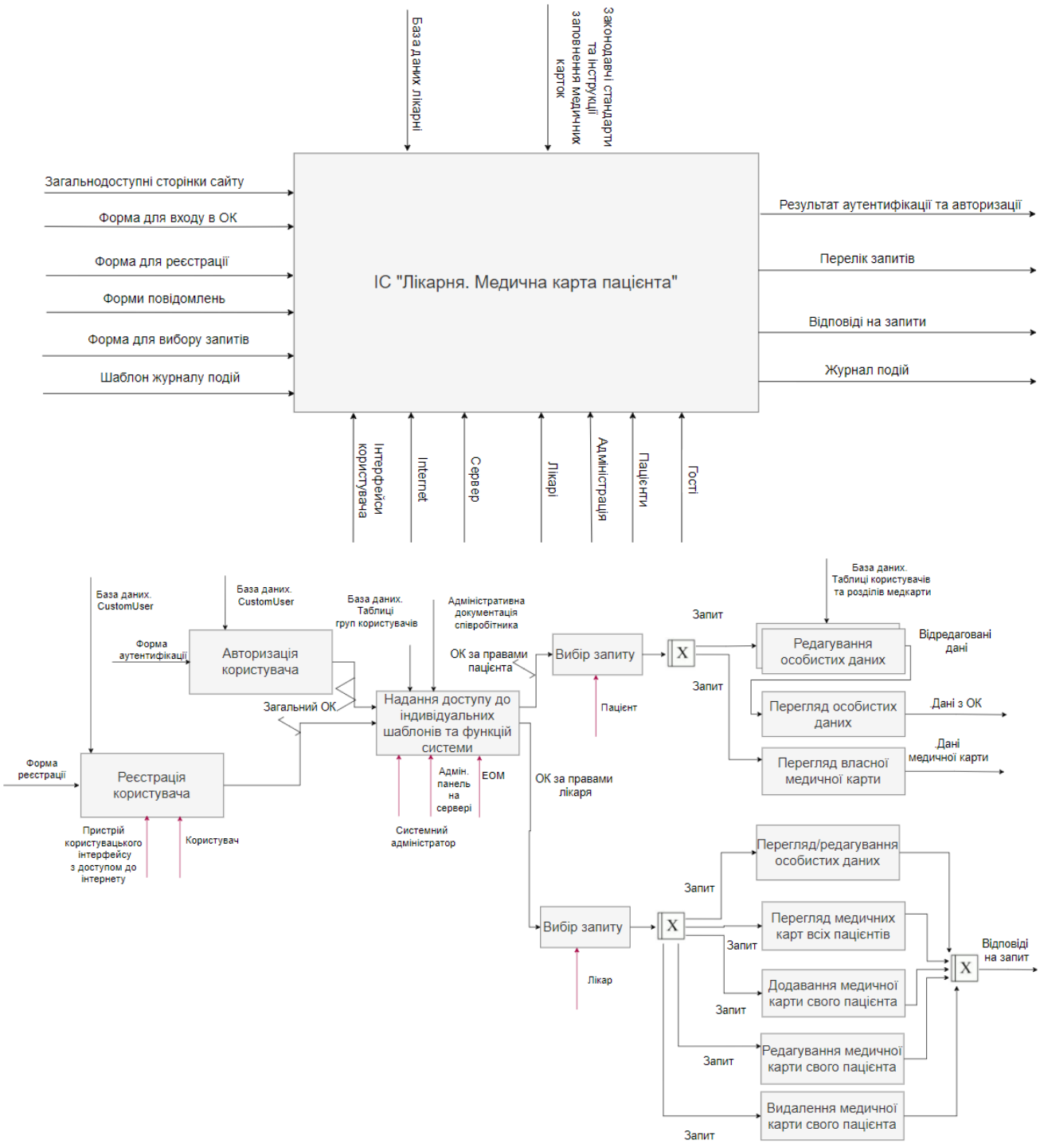
39. Балобольченкова М., Ізмайлова О. Застосування фрейворку Django для побудови безпечної інформаційної системи управління доступом на прикладі медичної сфери [Теза доповіді] / М. Балобольченкова, О. Ізмайлова – Конференція «БУД-МАЙСТЕР-КЛАС-2025». – 2025.

40. Положення про кваліфікаційну роботу здобувачів вищої освіти Київського національного університету будівництва і архітектури [Нормативний документ]. – Київ: КНУБА, 2024.

ДОДАТКИ

Додаток А – Use-case діаграми з диплому бакалавра

Додаток Б – DFD-діаграми з диплому бакалавра



Додаток В – Програмний код функціональної частини сайту

core

utils.py

```
from django.conf import settings
from hashids import Hashids
hashids = Hashids(salt=settings.HASHIDS_SALT, min_length=10)
def encode_id(num: int):
    ### Перетворює числовий PK в hashid.
    return hashids.encode(num)
def decode_id(hashid: str):
    ### Перетворює hashid назад у PK.
    ### Повертає None, якщо хеш некоректний.
    decoded = hashids.decode(hashid)
    return decoded[0] if decoded else None
```

SaveHealth

settings.py

```
from pathlib import Path
import os
from dotenv import load_dotenv

# Ключ для url
load_dotenv()
HASHIDS_SALT = os.getenv("HASHINDS_SALT")
..............
AUTH_PASSWORD_VALIDATORS = [
    {'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
},
    {'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {"min_length":14}},
    {'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',},
    {'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',}]

..............
```

urls.py

```
from django.contrib import admin
from django.urls import include, path
from SaveHealth import settings
from django.conf.urls.static import static
urlpatterns = [
    path('root/', admin.site.urls), #mysite.com/root
```

```
    path('', include('public_app.urls', namespace='public')),
    path('user/', include('users_app.urls', namespace='user')),
    path('profile/med_card/', include('cards_app.urls', namespace='card')),
    path('profile/laboratory/', include('laboratory_app.urls',
namespace='laboratory'))]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

users_app

admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import Doctor, Personnel, Patient, LaboratoryAssistent,
CustomUser, FamilyDoctor

class CustomUserAdmin(UserAdmin):
    # Додаємо до стандартних полів нові в форму додавання користувача
  add_fieldsets = UserAdmin.add_fieldsets + (
        ('Персональна інформація', {'fields': ('email','last_name',
'first_name', 'patronymic', 'groups', 'phone_number', 'img')}),)
    # Тут ми додаємо ТІЛЬКИ нові поля для кастом юзер, щоб уникнути
дублювання email, first_name, last_name..
  fieldsets = UserAdmin.fieldsets + (('Додаткова інформація профілю',
{'fields': ('patronymic', 'phone_number', 'img')}),)

@admin.register(FamilyDoctor)
class FamilyDoctorAdmin(admin.ModelAdmin):
    list_display = ('id', 'patient', 'doctor')
    def formfield_for_foreignkey(self, db_field, request, **kwargs):
        if db_field.name == "doctor":
            kwargs["queryset"] =
Doctor.objects.filter(specialization="Терапевт")
        return super().formfield_for_foreignkey(db_field, request,
**kwargs)

admin.site.register(CustomUser, CustomUserAdmin)
admin.site.register(Patient)

class BasePersonnelAdmin(admin.ModelAdmin):
    """Базовий клас для Doctor, Personnel, LabAssistant:
    додає префікс до tab_nomer, якщо користувач ввів лише цифри."""
    list_display = ('tab_nomer',)
    def save_model(self, request, obj, form, change):
        # Визначаємо префікс для поточної моделі
        model_class = obj.__class__
```

```
        prefix = ''
        if model_class == Doctor:
            prefix = 'DOC-'
        elif model_class == Personnel:
            prefix = 'MEDP-'
        elif model_class == LaboratoryAssistent:
            prefix = 'LAB-'
        tab_nomer_input = obj.tab_nomer
        # Перевіряємо, чи існує ввід і чи не починається він вже з нашого
префікса
        if tab_nomer_input and prefix and not
tab_nomer_input.startswith(prefix):
            # Якщо введений текст складається лише з цифр
            if tab_nomer_input.isdigit():
                obj.tab_nomer = f"{prefix}{tab_nomer_input}"
            # Якщо введений текст не є лише цифрами або вже має префікса,
        super().save_model(request, obj, form, change)

@admin.register(Doctor)
class DoctorAdmin(BasePersonnelAdmin):
    list_display = ('tab_nomer', 'user', 'stazh', 'specialization',
'Umovy_pryyomu')
@admin.register(Personnel)
class PersonnelAdmin(BasePersonnelAdmin):
    list_display = ('tab_nomer', 'user', 'stazh', 'department', 'position')
@admin.register(LaboratoryAssistent)
class LaboratoryAssistentAdmin(BasePersonnelAdmin):
    list_display = ('tab_nomer', 'user', 'stazh', 'specialization')
```

context_processors.py

```
def user_groups(request):
  if request.user.is_authenticated:
    return {'user_groups': request.user.groups.values_list('name',
flat=True)}
  return {}
```

forms.py

```
from django import forms
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm,
UserChangeForm
from .models import CustomUser, Patient, Doctor, FamilyDoctor,
LaboratoryAssistent, Personnel
from datetime import date, timedelta

class UserLoginForm(AuthenticationForm):
```

```
  username = forms.CharField()
  password = forms.CharField()
  class Meta:
    model = CustomUser
    fields = ['username', 'password']

class UserRegForm(UserCreationForm):
  password1 = forms.CharField(label='Пароль', widget=forms.PasswordInput())
  password2 = forms.CharField(label='Повторіть пароль',
widget=forms.PasswordInput())
  class Meta:
    model = CustomUser
    fields = ("first_name", "last_name", "username",
                  "email", "phone_number", "password1", "password2",)
class ProfileForm(UserChangeForm):
    class Meta:
        model = CustomUser
        fields = ("img", "first_name", "last_name",
                  "patronymic", "email", "phone_number",)
class DoctorForm(forms.ModelForm):
  class Meta:
    model = Doctor
    fields = ("specialization", "stazh", "Umovy_pryyomu")
class LabAssistantForm(forms.ModelForm):
  class Meta:
    model = LaboratoryAssistent
    fields = ("specialization", "stazh")
class PersonnelForm(forms.ModelForm):
  class Meta:
    model = Personnel
    fields = ("stazh", "department", "position")

class PatientForm(forms.ModelForm):
    class Meta:
        model = Patient
        fields = ("date_of_birth", "sex")
        date_of_birth = forms.DateField(
        widget=forms.DateInput(attrs={'type': 'date'}))
    def clean_date_of_birth(self):
        value = self.cleaned_data.get("date_of_birth")
        today = date.today()
        eighty_years_ago = today - timedelta(days=80*365.25)
        if value and value < eighty_years_ago:
            raise forms.ValidationError(f'Дане поле не приймає дати, менші
за {eighty_years_ago}')
        elif value and value> date.today():
```

```
            raise forms.ValidationError(f'Дане поле не приймає дати, більші
за {date.today()}' )
        return value

class FamilyDoctorForm(forms.ModelForm):
  class Meta:
    model = FamilyDoctor
    fields = '__all__'
  def clean(self):
    cleaned_data = super().clean()
    patient = cleaned_data.get("patient")
    if self.instance.pk:
      # Якщо запис вже існує, виключити його з перевірки
      if
FamilyDoctor.objects.filter(patient=patient).exclude(pk=self.instance.pk).exists():
        raise forms.ValidationError("Цей пацієнт вже має сімейного лікаря")
    else:
      # Якщо запис новий, виконати стандартну перевірку
      if FamilyDoctor.objects.filter(patient=patient).exists():
        raise forms.ValidationError("Цей пацієнт вже має сімейного лікаря")
    return cleaned_data
```

models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from phonenumber_field.modelfields import PhoneNumberField

class CustomUser(AbstractUser):
  img = models.ImageField(upload_to='users_img', blank=True, null=True,
verbose_name='Фото')
  email = models.EmailField(unique=True, verbose_name= 'Email')
  first_name = models.CharField(max_length=100, verbose_name= 'Імя')
  last_name = models.CharField(max_length=100, verbose_name= 'Призвіще')
  patronymic = models.CharField(max_length=100, blank=True, null=True,
verbose_name= 'По-батькові')
  phone_number = PhoneNumberField(unique=True, verbose_name= 'Номер
телефону')
  class Meta:
    db_table = 'Users'
    verbose_name = 'Користувача'
    verbose_name_plural = 'Користувачі'
  def __str__(self):
    return f"{self.id}. Логін: {self.username}"
class Doctor(models.Model):
  tab_nomer = models.CharField(max_length=20,
primary_key=True,verbose_name="Табельний номер")
```

```
  user = models.OneToOneField("users_app.CustomUser", verbose_name="ID
користувача", on_delete=models.CASCADE)
  stazh = models.CharField(max_length=50, blank=True, null=True,
verbose_name='Стаж')
  specialization =
models.CharField(max_length=100,verbose_name='Спеціалізація')
  Umovy_pryyomu = models.CharField(max_length=250, blank=True, null=True,
verbose_name='Умови прийому')
  class Meta:
    db_table = 'Doctor'
    verbose_name = 'Лікаря'
    verbose_name_plural = 'Лікарі'
  def __str__(self):
    return self.tab_nomer
class Personnel(models.Model):
  tab_nomer = models.CharField(max_length=20,
primary_key=True,verbose_name="Табельний номер")
  user = models.OneToOneField("users_app.CustomUser", verbose_name="ID
користувача", on_delete=models.CASCADE)
  stazh = models.CharField(max_length=50, blank=True, null=True,
verbose_name='Стаж')
  department = models.CharField(max_length=100,verbose_name='Відділення')
  position = models.CharField(max_length=250, blank=True, null=True,
verbose_name='Посада')
  class Meta:
    db_table = 'MedPersonnel'
    verbose_name = 'Мед.персонал'
    verbose_name_plural = 'Мед.персонал'
  def __str__(self):
    return self.tab_nomer
class LaboratoryAssistent(models.Model):
  tab_nomer = models.CharField(max_length=20,
primary_key=True,verbose_name="Табельний номер")
  user = models.OneToOneField("users_app.CustomUser", verbose_name="ID
користувача", on_delete=models.CASCADE)
  stazh = models.CharField(max_length=50, blank=True, null=True,
verbose_name='Стаж')
  specialization =
models.CharField(max_length=100,verbose_name='Спеціалізація')
  class Meta:
    db_table = 'LabAssistent'
    verbose_name = 'Асистента'
    verbose_name_plural = 'Лаб.Асистенти'
  def __str__(self):
    return self.tab_nomer
class Patient(models.Model):
```

```
  user = models.OneToOneField("users_app.CustomUser", verbose_name="ID
користувача", on_delete=models.CASCADE)
  SEX = [('Ч','Чоловік'), ('Ж','Жінка'),]
  sex = models.CharField(blank=True, null=True, max_length=1, choices=SEX,
verbose_name='Стать')
  date_of_birth = models.DateField(blank=True, null=True,
verbose_name='Дата народження')
  id_doctor = models.ManyToManyField("users_app.Doctor",
                                     through='FamilyDoctor')
  class Meta:
    db_table = 'Patient'
    verbose_name = 'Пацієнта'
    verbose_name_plural = 'Пацієнти'
  def __str__(self):
    return f"{self.user}"
class FamilyDoctor(models.Model):
  patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
verbose_name='Пацієнт')
  doctor = models.ForeignKey(Doctor,
on_delete=models.CASCADE,verbose_name='Лікар')
  date_of_appointment = models.DateTimeField(auto_now_add=True, blank=True,
null=True, verbose_name='Дата призначення')
  class Meta:
    db_table = 'FamilyDoctor'
    verbose_name = 'Сімейного лікаря'
    verbose_name_plural = 'Сімейні лікарі'
    unique_together = ('patient', 'doctor')
  def __str__(self):
    return f"{self.doctor} - {self.patient}"
```

urls.py

```
from django.urls import path
from users_app import views

app_name = 'users_app'
urlpatterns = [
    path('login/',views.login, name='login'), #mysite.com/user/login/
    path('registration/',views.registration, name='registration'),
    path('profile/',views.profile, name='profile'),
    path('logout/',views.logout, name='logout'),]
```

views.py

```
from django.contrib.auth.decorators import login_required
from django.http import HttpResponseRedirect
from django.shortcuts import redirect, render
from django.contrib import auth
```

```
from django.urls import reverse
from users_app.forms import ProfileForm, UserLoginForm, UserRegForm,
PatientForm, DoctorForm, LabAssistantForm, PersonnelForm
from users_app.models import Doctor, Patient, LaboratoryAssistent,
Personnel

def login(request):
  if request.method == 'POST':
    form = UserLoginForm(data=request.POST)  # екземпляр форми логіну, що
передає дані, які користувач ввів
    if form.is_valid():
      username = request.POST['username'] ## зі словника добуваєм username
& password
      password = request.POST['password']
      user = auth.authenticate(username=username, password=password)
      if user: ## чи є такий користувач??
        auth.login(request, user)
      if request.POST.get('next', None):
          return HttpResponseRedirect(request.POST.get('next'))
      return HttpResponseRedirect(reverse('user:profile'))
  else:
    form = UserLoginForm()
  context = {'form': form}
  return render(request, 'users/login.html', context)
def registration(request):
  if request.method == 'POST':
    form = UserRegForm(data=request.POST)  # екземпляр форми логіну, що
передає дані, які користувач ввів
    if form.is_valid():
      form.save()
      return HttpResponseRedirect(reverse('user:login'))
  else:
    form = UserRegForm()
  context = {'form': form}
  return render(request, 'users/reg.html', context)
@login_required
def logout(request):
  auth.logout(request)
  return redirect(reverse('public:index'))
@login_required
def profile(request):
    user = request.user # поточний користувач
    user_groups = user.groups.values_list('name', flat=True)
    form = ProfileForm(instance=user) # отримання даних з форми профілю
    patient_form = None ## змінні з незаповненими формами
    doctor_form = None
    assistant_form = None
```

```
    personnel_form = None
    if request.method == 'POST':
      form = ProfileForm(data=request.POST, instance=user,
files=request.FILES) # Створення форми з даними з POST-запиту
      if 'ПАЦІЄНТИ' in user_groups:
        patient_instance, created =
Patient.objects.get_or_create(user=user) # Отримання або створення об'єкту
пацієнта для користувача
        patient_form = PatientForm(request.POST, instance=patient_instance,
prefix='p') # Створення форми пацієнта з даними з POST-запиту
      if 'ЛІКАРІ' in user_groups:
        try:
          doctor_instance = Doctor.objects.get(user=user) # Спроба отримати
об'єкт лікаря для користувача
          doctor_form = DoctorForm(request.POST, instance=doctor_instance,
prefix='d') # Створення форми лікаря з даними з POST-запиту
        except Doctor.DoesNotExist: # Якщо об'єкт лікаря не знайдено
          doctor_form = None
      if 'МЕД.ПЕРСОНАЛ' in user_groups:
        try:
          personnel_instance = Personnel.objects.get(user=user) # Спроба
отримати об'єкт лікаря для користувача
          personnel_form = PersonnelForm(request.POST,
instance=personnel_instance, prefix='med-p') # Створення форми лікаря з
даними з POST-запиту
        except Personnel.DoesNotExist:
          personnel_form = None

      if 'ЛАБОРАНТИ' in user_groups:
        try:
          assistant_instance =  LaboratoryAssistent.objects.get(user=user)
          assistant_form = LabAssistantForm(request.POST,
instance=assistant_instance, prefix='lab')
        except LaboratoryAssistent.DoesNotExist:
          assistant_form = None
      if form.is_valid() and (patient_form is None or
patient_form.is_valid()) and \
        (doctor_form is None or doctor_form.is_valid()) and \
        (personnel_form is None or personnel_form.is_valid()) and \
        (assistant_form is None or assistant_form.is_valid()):
          form.save()
          if patient_form:
            patient_form.save()
          if doctor_form:
            doctor_form.save()
          if personnel_form:
            personnel_form.save()
```

```
            if assistant_form:
              assistant_form.save()
            return HttpResponseRedirect(reverse('user:profile'))

    else:
      if 'ПАЦІЄНТИ' in user_groups:
        patient_instance, created =
Patient.objects.get_or_create(user=user) # Отримання або створення об'єкту
пацієнта для користувача
        patient_form = PatientForm(instance=patient_instance, prefix='p') #
Створення форми пацієнта з даними користувача

      if 'ЛІКАРІ' in user_groups:
        try:
          doctor_instance = Doctor.objects.get(user=user) # Спроба отримати
об'єкт лікаря для користувача
          doctor_form = DoctorForm(instance=doctor_instance, prefix='d')  #
Створення форми лікаря з даними користувача
        except Doctor.DoesNotExist:
          doctor_form = None

      if 'МЕД.ПЕРСОНАЛ' in user_groups:
        try:
          personnel_instance =  Personnel.objects.get(user=user)
          personnel_form = PersonnelForm(instance=personnel_instance,
prefix='med-p')
        except Personnel.DoesNotExist:
          personnel_form = None
      if 'ЛАБОРАНТИ' in user_groups:
        try:
          assistant_instance =  LaboratoryAssistent.objects.get(user=user)
# Спроба отримати об'єкт лікаря для користувача
          assistant_form = LabAssistantForm(instance=assistant_instance,
prefix='lab') # Створення форми лікаря з даними з POST-запиту
        except LaboratoryAssistent.DoesNotExist:
          assistant_form = None
    context = {'form': form, 'user_groups': list(user_groups),
        'patient_form': patient_form,'doctor_form': doctor_form,
        'assistant_form': assistant_form,
        'personnel_form': personnel_form,}
    return render(request, 'users/profile.html', context)
```

laboratory_app

admin.py

```
from django.contrib import admin
from .models import ResultAnalysis, TemplateParameter,
```

```
TypeAnalysis, MedReferral, ResultParameter

class ResultParameterInline(admin.TabularInline):
    """Дозволяє редагувати ResultParameter безпосередньо
    на сторінці ResultAnalysis."""
    model = ResultParameter
    fields = ('template', 'value', 'unit_of_measurement', 'coment')
    readonly_fields = ()
    extra = 1

@admin.register(ResultAnalysis)
class ResultAnalysisAdmin(admin.ModelAdmin):
    inlines = [ResultParameterInline]
    def save_formset(self, request, form, formset, change):
        """Перезаписуємо для автоматичного заповнення одиниці вимірювання
на основі вибраного шаблону."""
        if formset.model == ResultParameter:
            instances = formset.save(commit=False)
            for instance in instances:
                if instance.template:
                    try:
                        unit = instance.template.unit_of_measurement
                        instance.unit_of_measurement = unit

                    except AttributeError:
                        instance.unit_of_measurement = 'ПОМИЛКА'
                instance.save()
            formset.save_m2m()
            for obj in formset.deleted_objects:
                obj.delete()
        else:
            super().save_formset(request, form, formset, change)

admin.site.register(TemplateParameter)
admin.site.register(TypeAnalysis)
admin.site.register(MedReferral)
```

models.py

```
from django.db import models
from django.forms import ValidationError

STATUS_CHOICES = [('N', 'Активне'), ('C', 'Анульовано'),]

class MedReferral(models.Model):
  doctor = models.ForeignKey('users_app.Doctor', on_delete=models.PROTECT,
verbose_name='ID лікаря')
```

```
  medcard = models.ForeignKey('cards_app.MedCards',
on_delete=models.CASCADE, verbose_name='Номер медичної карти')
  type_analys = models.ForeignKey('TypeAnalysis', on_delete=models.PROTECT,
verbose_name='ID типу аналізу')
  creation_date = models.DateTimeField(auto_now_add=True,
verbose_name='Дата створення направлення')
  coment = models.TextField(max_length=500, verbose_name='Коментар')
  status = models.CharField(max_length=1, choices=STATUS_CHOICES,
default='N', verbose_name='Статус направлення')
  class Meta:
    db_table = 'MedReferral'
    verbose_name = 'Направлення'
    verbose_name_plural = 'Направлення'
  def __str__(self):
    return f'{self.id}'
  def display_id(self):
    return f'{self.id:03}'
class TypeAnalysis(models.Model):
  name = models.CharField(max_length=100, verbose_name='Назва аналізу')
  description = models.TextField(max_length=500, blank=True, null=True,
verbose_name='Опис аналізу')
  class Meta:
    db_table = 'TypeAnalys'
    verbose_name = 'Тип аналізу'
    verbose_name_plural = 'Типи аналізів'
  def __str__(self):
    return f'{self.id} - {self.name}'
class TemplateParameter(models.Model):
  name = models.CharField(max_length=100, verbose_name='Назва параметру')
  unit_of_measurement = models.CharField(max_length=20,
verbose_name='Одиниці вимірювання')
  normal_min = models.DecimalField(max_digits=7,decimal_places=3,
blank=True, null=True, verbose_name='Мінімально допустиме значення')
  normal_max = models.DecimalField(max_digits=7,decimal_places=3,
blank=True, null=True, verbose_name='Максимально допустиме значення')
  type_analys = models.ForeignKey('TypeAnalysis', on_delete=models.CASCADE,
verbose_name='ID типу аналізу')
  class Meta:
    db_table = 'TemplateParameter'
    verbose_name = 'Шаблон параметру'
    verbose_name_plural = 'Шаблони параметрів'
    unique_together = ('name', 'type_analys')
  def clean(self):
    super().clean()
    if self.normal_max <= self.normal_min:
      raise ValidationError({
```

```
        'normal_max': 'Максимальне значення має бути більшим за
мінімальне.'})
  def __str__(self):
    return f'{self.id} - {self.name}'
class ResultParameter(models.Model):
  value = models.DecimalField(max_digits=6,decimal_places=3,
verbose_name='Результат')
  unit_of_measurement = models.CharField(max_length=20, blank=True,
null=True, verbose_name='Одиниці вимірювання')
  coment = models.CharField(max_length=100, verbose_name='Коментар')
  template = models.ForeignKey('TemplateParameter',
on_delete=models.CASCADE, verbose_name='ID шаблону аналізу')
  result_analys = models.ForeignKey('ResultAnalysis',
on_delete=models.CASCADE, verbose_name='ID результатів аналізів')
  class Meta:
    db_table = 'ResultParameter'
    verbose_name = 'Результат параметру'
    verbose_name_plural = 'Результати параметрів'
  def __str__(self):
    return f'{self.id}'
class ResultAnalysis(models.Model):
  medcard = models.ForeignKey('cards_app.MedCards',
on_delete=models.CASCADE, verbose_name='Номер медичної карти')
  referral = models.OneToOneField('MedReferral', on_delete=models.PROTECT,
verbose_name='ID направлення')
  lab_assistent = models.ForeignKey('users_app.LaboratoryAssistent',
on_delete=models.PROTECT, verbose_name='ID лаборанта')
  providing_date = models.DateTimeField(auto_now_add=True,
verbose_name='Дата заповнення висновку')
  report = models.TextField(max_length=500, verbose_name='Висновок')
  class Meta:
    db_table = 'ResultAnalys'
    verbose_name = 'Результат аналізу'
    verbose_name_plural = 'Результати аналізів'
  def __str__(self):
    return f'{self.id}'
  def display_id(self):
    return f'{self.id:03}'
```

urls.py

```
from django.urls import path
from laboratory_app import views
app_name = 'laboratory_app'
urlpatterns = [
  path('lab-journal/', views.lab_journal, name='lab_journal'),
  path('lab-norms/', views.lab_norms, name='lab_norms'),
  path('add-res/<int:referral_id>', views.add_result, name='add_result'),]
```

views.py

```
from django.shortcuts import get_object_or_404, redirect, render
from django.contrib.auth.decorators import login_required
from laboratory_app.models import MedReferral, ResultAnalysis,
ResultParameter, TemplateParameter

@login_required
def lab_journal(request):
  return render(request, 'laboratory/lab_journal.html')
@login_required
def lab_norms(request):
  return render(request, 'laboratory/lab_norms.html')
@login_required
def add_result(request, referral_id):
    referral = get_object_or_404(MedReferral, id=referral_id)
    card = referral.medcard
    ref_type = referral.type_analys
    parameters = TemplateParameter.objects.filter(type_analys=ref_type)

    if request.method == "POST":
        # Створюємо результат аналізу
        result_analysis = ResultAnalysis.objects.create(
            medcard=card, referral=referral,
            lab_assistent=request.user.laboratoryassistent,
            report=request.POST.get("report", ""))
        # Створюємо результати параметрів
        for param in parameters:
            value = request.POST.get(f"value_{param.id}")
            unit = request.POST.get(f"unit_{param.id}") or
param.unit_of_measurement
            comment = request.POST.get(f"comment_{param.id}", "")
            if value:
                ResultParameter.objects.create(value=value,
                    unit_of_measurement=unit,
                    coment=comment, template=param,
                    result_analys=result_analysis)
        referral.save()
        return redirect('card:journal_referral')

    context = {'referral': referral, 'card': card,
'ref_type': ref_type, 'parameters': parameters,}
    return render(request, 'laboratory/add_result.html', context)
```

cards_app

admin.py

```
from django.contrib import admin
from .models import MedCards, SignalMarks, IndividualMarks, Vaccination,
CardVaccine, DoctorExamination, Personnel_MedCard

class MedCardsAdmin(admin.ModelAdmin):
  list_display = ('id', 'patient', 'doctor', 'dispensary_group',
'registration_date')
  readonly_fields = ('registration_date','patient', 'doctor',)
admin.site.register(MedCards, MedCardsAdmin)
admin.site.register(IndividualMarks)
admin.site.register(SignalMarks)
admin.site.register(Vaccination)
admin.site.register(CardVaccine)
admin.site.register(DoctorExamination)
admin.site.register(Personnel_MedCard)
```

forms.py

```
from datetime import date, timedelta
from django.core.validators import MinLengthValidator
from django import forms
from django.utils import timezone
from laboratory_app.models import MedReferral, TypeAnalysis
from .models import CardVaccine, IndividualMarks, Vaccination
from django.forms import inlineformset_factory
from .models import MedCards

class IndividualMarkForm(forms.ModelForm):
    class Meta:
        model = IndividualMarks
        fields = ['value']
IndividualMarksFormSet = inlineformset_factory(
    parent_model=MedCards, model=IndividualMarks, form=IndividualMarkForm,
fields=['value'], extra=0, can_delete=False)

class CardVaccineForm(forms.ModelForm):
    class Meta:
        model = CardVaccine
        fields = ['vaccination', 'product_series', 'reaction',
'contraindication','date_vaccine']
    vaccination = forms.ModelChoiceField(
    queryset=Vaccination.objects.all(),
    widget=forms.Select(attrs={
        'class': 'form-select', 'id': 'id_vaccination',}))
    contraindication = forms.CharField(widget=forms.Textarea())
    product_series = forms.CharField(validators=[MinLengthValidator(5)])
    date_vaccine = forms.DateField(
```

```
        widget=forms.DateInput(attrs={'type': 'date'}))

        # ---- Перевірка дати створення (редагування) ----
    def clean(self):
        cleaned_data = super().clean()
        if self.instance.pk:
            created_date = self.instance.date_vaccine
            now = timezone.now().date()
            days_passed = (now - created_date).days
            hours_passed = days_passed * 24
            if hours_passed > 48:
                raise forms.ValidationError("Редагування цього запису
заборонено — минуло більше 48 годин від моменту створення.")
        return cleaned_data
    def clean_date_vaccine(self):
        value = self.cleaned_data.get("date_vaccine")
        today = date.today()
        week_ago = today - timedelta(days=7)
        if value and value < week_ago:
            raise forms.ValidationError(f'Дане поле не приймає дати, менші
за {week_ago}')
        elif value and value > date.today():
            raise forms.ValidationError(f'Дане поле не приймає дати, більші
за {date.today()}' )
        return value

class MedReferralForm(forms.ModelForm):
    class Meta:
        model = MedReferral
        fields = [ 'coment', 'status']
class ReferralAddForm(forms.ModelForm):
    type_analys = forms.ModelChoiceField(
        queryset=TypeAnalysis.objects.all(),
        empty_label="Виберіть тип аналізу", widget=forms.Select())
    class Meta:
        model = MedReferral
        fields = ['type_analys', 'coment']
```

models.py

```
from django.db import models

class MedCards(models.Model):
  patient = models.OneToOneField('users_app.Patient',
on_delete=models.CASCADE, verbose_name='ID пацієнта')
  doctor = models.ForeignKey('users_app.Doctor', on_delete=models.SET_NULL,
blank=True, null=True, verbose_name='ID лікаря')
```

```
  dispensary_group = models.BooleanField(verbose_name='Диспансерна група',
default=False)
  registration_date = models.DateTimeField(auto_now_add=True,
verbose_name='Поставлено на облік')
  deregistration_date = models.DateTimeField(blank=True, null=True,
verbose_name='Знято з обліку')
  id_med_personnel = models.ManyToManyField('users_app.Personnel',
through='Personnel_MedCard')
  class Meta:
    db_table = 'MedCard'
    verbose_name = 'Медичну карту'
    verbose_name_plural = 'Медичні картки'
  def __str__(self):
    return f'{self.id}'
  def display_id(self):
    return f'{self.id:05}'

class SignalMarks(models.Model):
  name = models.CharField(max_length=50, unique=True,
verbose_name='Сигнальна позначка')
  id_medcard = models.ManyToManyField('MedCards',
through='IndividualMarks')
  class Meta:
    db_table = 'SignalMark'
    verbose_name = 'Сигнальну позначку'
    verbose_name_plural = 'Сигнальні позначки'
  def __str__(self):
    return f'{self.id}.{self.name}'

class IndividualMarks(models.Model):
  medcard = models.ForeignKey('MedCards', on_delete=models.CASCADE,
verbose_name='Номер медичної карти')
  signal_mark = models.ForeignKey('SignalMarks', on_delete=models.CASCADE,
verbose_name='Сигнальна позначка')
  value = models.CharField(max_length=100, blank=True, null=True,
verbose_name='Значення')
  class Meta:
    db_table = 'IndividualMarks'
    verbose_name = 'Індивідуальну позначку'
    verbose_name_plural = 'Індивідуальні позначки'
    unique_together = ('medcard', 'signal_mark')
  def __str__(self):
    return f'{self.id}.{self.medcard} має позначку{self.signal_mark}'

class Vaccination(models.Model):
  name = models.CharField (max_length=100, unique=True,
verbose_name='Найменування щеплення')
```

```
  dose_number = models.PositiveSmallIntegerField(verbose_name='Кількість
доз')
  age = models.CharField (max_length=100, verbose_name='Вік для щеплення')
  active_substance = models.CharField(max_length=100, blank=True,
null=True, verbose_name='Активна речовина')
  METHOD = [("Внутрішньом'язова", "Внутрішньом'язова ін'єкція"),
("Підшкірна", "Підшкірна ін'єкція"), ("Внутрішньошкірна",
"Внутрішньошкірна ін'єкція"), ("Перорально", "Перорально")]
  input_method = models.CharField(choices=METHOD,
max_length=30,  blank=True, null=True, verbose_name='Спосіб введення')
  id_medcard = models.ManyToManyField('MedCards', through='CardVaccine')
  class Meta:
    db_table = 'Vaccination'
    verbose_name = 'Щеплень'
    verbose_name_plural = 'Щеплення'
  def __str__(self):
    return f'{self.id}.{self.name}'

class CardVaccine(models.Model):
  date_vaccine = models.DateField(verbose_name='Дата вакцинації')
  REACTION = [('Місцева', 'Місцева'), ('Загальна', 'Загальна')]
  reaction = models.CharField(choices=REACTION, max_length=20,
verbose_name='Реакція на щеплення')
  contraindication  = models.TextField(max_length=300,
verbose_name='Протипоказання')
  product_series = models.CharField(max_length=20, verbose_name= 'Серія
вакцини')
  medcard = models.ForeignKey('MedCards', on_delete=models.CASCADE,
verbose_name='Номер медичної карти')
  vaccination = models.ForeignKey('Vaccination', on_delete=models.PROTECT,
verbose_name='Найменування щеплення')
  class Meta:
    db_table = 'CardVaccine'
    verbose_name = 'Щеплень в медичній карті'
    verbose_name_plural = 'Щеплення в картці'
  def __str__(self):
    return f'{self.id}.{self.medcard} проведено {self.vaccination}
{self.date_vaccine}'

class DoctorExamination(models.Model):
  reason_for_contacting = models.CharField(max_length=100,
verbose_name='Причина звернення')
  cabinet = models.CharField(max_length=10, blank=True, null=True,
verbose_name='Кабінет')
  date_visit = models.DateField(verbose_name='Дата прийому')
  report = models.TextField(max_length=1500, verbose_name='Висновки')
```

```
  doctor = models.ForeignKey('users_app.Doctor', on_delete=models.PROTECT,
verbose_name='Табельний номер лікаря')
  medcard = models.ForeignKey('MedCards', on_delete=models.CASCADE,
verbose_name='Номер медичної карти')
  class Meta:
    db_table = 'DoctorExamination'
    verbose_name = 'Запис про огляд'
    verbose_name_plural = 'Записи про огляд'
  def __str__(self):
    return f'{self.id}.{self.medcard} - {self.reason_for_contacting}'

class Personnel_MedCard(models.Model):
  medcard = models.ForeignKey('MedCards', on_delete=models.CASCADE,
verbose_name='Номер медичної карти')
  personnel = models.ForeignKey('users_app.Personnel',
on_delete=models.SET_NULL, blank=True, null=True, verbose_name="Прив'язаний
персонал")
  interaction_date = models.DateTimeField(auto_now_add=True,
verbose_name='Дата взаємодії')
  comment = models.TextField(blank=True, null=True, verbose_name='Коментар
про взаємодію/прив’язку')
  class Meta:
    db_table = 'PersonnelCard'
    verbose_name = 'Персонал до карти'
    verbose_name_plural = 'Персонал до карти'
  def __str__(self):
    return f"{self.id}.{self.medcard} прив'язаний до{self.personnel}"
```

signals.py

```
from django.dispatch import receiver
from cards_app.models import IndividualMarks, MedCards, SignalMarks
from users_app.models import FamilyDoctor, Patient
from django.db.models.signals import post_save

@receiver(post_save, sender=Patient)
def create_medcard_for_patient(sender, instance, created, **kwargs):
"""Створює медичну картку, як тільки створюється об'єкт пацієнта."""
    if created:
        MedCards.objects.create(patient=instance)

@receiver(post_save, sender=FamilyDoctor)
def sync_doctor_to_medcards(sender, instance, created, **kwargs):
    """Оновлює поле 'doctor' в MedCards при зміні FamilyDoctor."""
    patient = instance.patient
    new_doctor = instance.doctor
```

```
    try:
        medcard = MedCards.objects.get(patient=patient)
        if medcard.doctor != new_doctor:
            medcard.doctor = new_doctor
            medcard.save(update_fields=['doctor'])
    except MedCards.DoesNotExist:
        pass

@receiver(post_save, sender=MedCards)
def create_individual_marks(sender, instance, created, **kwargs):
    """Автоматичне створення пустих індивідуальних позначок при створенні
мед.карти"""
    if created:
        signal_marks = SignalMarks.objects.all()
        IndividualMarks.objects.bulk_create([
            IndividualMarks(medcard=instance, signal_mark=mark, value='')
            for mark in signal_marks])
```

urls.py

```
from django.urls import path
from cards_app import views

app_name = 'cards_app'
urlpatterns = [
  ###### ЛІКАРІ #######
    path('all_cards/',views.all_cards, name='all_cards'),
    path('view/<str:hid>/', views.index, name='index'),
    path('edit-mark/<str:hid>/',views.edit_mark, name='edit_mark'),
    path('vaccine/<str:hid>/',views.vaccine, name='vaccine'),
    path('add-vaccine/<str:hid>/', views.add_vaccine, name='add_vaccine'),
    path('edit-vaccine/<int:vaccine_id>/', views.edit_vaccine,
name='edit_vaccine'),
    path("med-referral_staff/", views.lab_journal,
name="journal_referral"),
    path('med-referral/<int:card_id>/',views.med_referral,
name='referral_card'),
    path('edit-referral/<int:referral_id>/',views.edit_referral,
name='edit_referral'),
    path('add-referral/<int:card_id>/',views.add_referral,
name='add_referral'),
    path('result-analysis/<int:card_id>/',views.result_analysis,
name='result_analysis'),
  ###### ПАЦІЄНТИ #######
    path('', views.card_profile, name='card_profile'),
    path('vaccine/', views.vaccine_profile, name='vaccine_profile'),
    path('referral/', views.referral_profile, name='referral_profile'),
```

```
    path('result-analysis/', views.result_analysis_profile,
name='result_analysis_profile'),
    path('result-card/<int:card_analysis_id>/', views.result_id_profile,
name='result_id_profile'),]
```

views.py

```
from django.http import Http404, HttpResponseForbidden,
HttpResponseRedirect
from django.core.paginator import Paginator
from django.shortcuts import get_object_or_404, redirect, render
from django.contrib.auth.decorators import login_required
from django.urls import reverse
from cards_app.forms import CardVaccineForm, IndividualMarksFormSet,
MedReferralForm, ReferralAddForm
from cards_app.models import CardVaccine, IndividualMarks, MedCards
from core.utils import decode_id, encode_id
from laboratory_app.models import MedReferral, ResultAnalysis
from users_app.models import Doctor, Patient

################ Мед персонал ##############
@login_required
def all_cards(request):
    user = request.user
    user_groups = request.user.groups.values_list('name', flat=True)
    doctor = Doctor.objects.filter(user=user).first()
    patients_with_cards =
Patient.objects.select_related('user').prefetch_related('medcards').filter(
medcards__isnull=False).distinct()
    if not any(group in ["ЛІКАРІ", "МЕД.ПЕРСОНАЛ"] for group in
user_groups):
        raise Http404("Сторінка не знайдена")
    if doctor:
        card_active_redact = patients_with_cards.filter(id_doctor=doctor)
    else:
        card_active_redact = Patient.objects.none()
    paginator = Paginator(patients_with_cards, 5)
    page_number = request.GET.get("page")  # ← читаємо номер сторінки
    page_obj = paginator.get_page(page_number)

    context = {'user_groups': list(user_groups),
    'patients_with_cards': page_obj, 'page_obj': page_obj,
    'paginator': paginator, 'user': request.user, 'doctor': doctor,
    'card_active_redact': card_active_redact,}
    return render(request, 'cards/all_cards.html', context)

@login_required
def index(request, hid):
```

```
    card_id = decode_id(hid)
    if card_id is None:
        raise Http404("Сторінка не знайдена")
    user_groups = request.user.groups.values_list('name', flat=True)
    if "ЛАБОРАНТИ" in user_groups:
        raise Http404("Сторінка не знайдена")
    card = get_object_or_404(MedCards.objects.select_related('patient',
       'patient__user', 'doctor',
'doctor__user').prefetch_related('individualmarks_set'),  id=card_id)
    individual_marks = IndividualMarks.objects.filter(medcard=card)
    current_user = request.user
    doctor = card.doctor.user if card.doctor else None
    can_edit = doctor == current_user
    context = {'card': card, 'can_edit': can_edit,
        'individual_marks': individual_marks}
    return render(request, 'cards/card_profile.html', context)

@login_required
def edit_mark(request, hid):
    card_id = decode_id(hid)
    if card_id is None:
        raise Http404("Сторінка не знайдена")
    user_groups = request.user.groups.values_list('name', flat=True)
    if not "ЛІКАРІ" in user_groups:
        raise Http404("Сторінка не знайдена")
    card = get_object_or_404(
        MedCards.objects.prefetch_related('individualmarks_set'),
id=card_id)
    if request.method == 'POST':
        formset = IndividualMarksFormSet(request.POST, instance=card)
        if formset.is_valid():
            formset.save()
            return redirect('card:index', hid=hid)
    else:
        formset = IndividualMarksFormSet(instance=card)
    context = {'card': card, 'formset': formset}
    return render(request, 'cards/edit_signal_marks.html', context)

@login_required
def vaccine(request, hid):
    card_id = decode_id(hid)
    if card_id is None:
        raise Http404("Сторінка не знайдена")
    user_groups = request.user.groups.values_list('name', flat=True)
    if "ЛАБОРАНТИ" in user_groups:
        raise Http404("Сторінка не знайдена")
```

```
    card = get_object_or_404(MedCards.objects.select_related('patient',
'patient__user', 'doctor', 'doctor__user'), id=card_id)
    card_vaccinations =
CardVaccine.objects.filter(medcard=card).order_by('id')
    current_user = request.user
    doctor = card.doctor.user if card.doctor else None
    is_medical_staff = "МЕД.ПЕРСОНАЛ" in user_groups
    can_edit = doctor == current_user or is_medical_staff
    context = {'card': card,
        'card_vaccinations': card_vaccinations, 'can_edit': can_edit}
    return render(request, 'cards/vaccine_profile.html', context)

@login_required
def add_vaccine(request, hid):
    card_id = decode_id(hid)
    if card_id is None:
        raise Http404("Сторінка не знайдена")
    card = get_object_or_404(MedCards, id=card_id)
    user = request.user
    user_groups = user.groups.values_list('name', flat=True)
    is_medical_staff = "МЕД.ПЕРСОНАЛ" in user_groups
    is_assigned_doctor = "ЛІКАРІ" in user_groups and card.doctor and
card.doctor.user == user
    has_access = is_medical_staff or is_assigned_doctor
    if not has_access:
        raise Http404("Сторінка не знайдена")
    if request.method == 'POST':
        form = CardVaccineForm(request.POST)
        if form.is_valid():
            card_vaccine = form.save(commit=False)
            card_vaccine.medcard = card
            card_vaccine.save()
            return HttpResponseRedirect(reverse('card:vaccine',
kwargs={'hid': hid}))
    else:
        form = CardVaccineForm()
    context = {'form': form, 'card': card, 'has_access': has_access}
    return render(request, 'cards/add_vaccine.html', context)

@login_required
def edit_vaccine(request, vaccine_id):
    vaccine = get_object_or_404(CardVaccine, id=vaccine_id)
    card = vaccine.medcard
    hid = encode_id(card.id)
    if request.method == 'POST':
        form = CardVaccineForm(request.POST, instance=vaccine)
        if form.is_valid():
```

```
            form.save()
            return redirect('card:vaccine', hid=hid)
    else:
        form = CardVaccineForm(instance=vaccine)
    context = {'form': form, 'card': card, 'hid': hid}
    return render(request, 'cards/edit_vaccine.html', context)

# Перегляд направлень без прив'язки до карти
@login_required
def lab_journal(request):
    user = request.user
    user_groups = request.user.groups.values_list('name', flat=True)
    if "ЛАБОРАНТИ" in user_groups:
        card_referral = MedReferral.objects.exclude(status='C')
        paginator = Paginator(card_referral, 5)
        page_number = request.GET.get("page")
        page_obj = paginator.get_page(page_number)
        context = {'user': user, 'card_referral': page_obj,
            'page_obj': page_obj, 'paginator': paginator,}
        return render(request, 'laboratory/lab_journal.html', context)
    elif "ЛІКАРІ" in user_groups:
        doctor = get_object_or_404(Doctor, user=request.user)
        card_referral = MedReferral.objects.filter(doctor=doctor)
        context = {'user': user, 'doctor':doctor, 'card_referral':
card_referral,}
        return render(request, 'laboratory/lab_journal.html', context)
    return HttpResponseForbidden("Немає доступу до цієї сторінки")

# Перегляд з прив'язкою до карти
@login_required
def med_referral(request, card_id):
    user = request.user
    user_groups = request.user.groups.values_list('name', flat=True)
    if "ЛІКАРІ" in user_groups or "МЕД.ПЕРСОНАЛ" in user_groups:
        card = get_object_or_404(MedCards, id=card_id)
        card_referral = MedReferral.objects.filter(medcard=card)

        context = {'user': user, 'card_referral': card_referral,
            'card': card,}
        return render(request, 'cards/referral_profile.html', context)
    return HttpResponseForbidden("Немає доступу до цієї сторінки")

@login_required
def add_referral(request, card_id):
    card = get_object_or_404(MedCards, id=card_id)
    user = request.user
```

```
    if "ЛІКАРІ" not in user.groups.values_list('name', flat=True):
        return HttpResponseForbidden("У вас немає доступу до цієї
сторінки")
    if request.method == 'POST':
        form = ReferralAddForm(request.POST)
        if form.is_valid():
            referral = form.save(commit=False)
            referral.medcard = card
            referral.doctor = get_object_or_404(Doctor, user=user)
            referral.status = 'N'  # Активне при створенні
            referral.save()
            return redirect('card:referral_card',
card_id=card.display_id())
    else:
        form = ReferralAddForm()
    return render(request, 'cards/add_referral.html', {'form': form,
'card': card})

@login_required
def edit_referral(request, referral_id):
    referral = get_object_or_404(MedReferral, id=referral_id)
    card = referral.medcard
    if not (referral.doctor.user == request.user or
        card.doctor.user == request.user):
        return HttpResponseForbidden("У вас немає доступу до редагування
цього направлення")
    if request.method == 'POST':
        form = MedReferralForm(request.POST, instance=referral)
        if form.is_valid():
            form.save()
            return redirect('card:referral_card',
card_id=card.display_id())
    else:
        form = MedReferralForm(instance=referral)
    return render(request, 'cards/edit_referral.html', {
        'form': form,'referral': referral, 'card': card,})

@login_required
def result_analysis(request, card_id):
    user = request.user
    if not user.groups.filter(name__in=["ЛІКАРІ", "МЕД.ПЕРСОНАЛ",
"ЛАБОРАНТИ"]).exists():
        return redirect('card:result_analysis_profile')
    card = get_object_or_404(
        MedCards.objects.select_related('patient', 'patient__user',
            'doctor', 'doctor__user'),id=card_id)
    card_analysis = ResultAnalysis.objects.filter(medcard=card)
```

```
    context = {'user': user, 'card': card, 'card_analysis': card_analysis,}
    return render(request, 'cards/result_analysis.html', context)

################ Пацієнти ##############

@login_required
def card_profile(request):
  user = request.user
  patient = get_object_or_404(Patient, user=user)
  card = get_object_or_404(MedCards.objects.select_related('patient',
                                                'patient__user', 'doctor',
'doctor__user').prefetch_related('individualmarks_set'),patient=patient)
  individual_marks = IndividualMarks.objects.filter(medcard=card)
  context = {'user': user, 'patient': patient, 'card': card,
               'individual_marks' : individual_marks}
  return render(request, 'cards/card_profile.html', context)

@login_required
def vaccine_profile(request):
    user = request.user
    patient = get_object_or_404(Patient, user=user)
    card = MedCards.objects.select_related('patient',
'patient__user','doctor').filter(patient=patient).first()
    card_vaccinations = []
    if card:
        card_vaccinations = CardVaccine.objects.filter(medcard=card)
    context = {'user': user, 'patient': patient,'card': card,
        'card_vaccinations': card_vaccinations,}
    return render(request, 'cards/vaccine_profile.html', context)

@login_required
def referral_profile(request):
  user = request.user
  patient = get_object_or_404(Patient, user=user)
  card = MedCards.objects.select_related('patient',
'patient__user','doctor').filter(patient=patient).first()
  card_referral = []
  if card:
    card_referral = MedReferral.objects.filter(medcard=card)
  context = {'user': user, 'patient': patient, 'card': card,
        'card_referral': card_referral,}
  return render(request, 'cards/referral_profile.html', context)

@login_required
def result_analysis_profile(request):
  user = request.user
```

```
  patient = get_object_or_404(Patient, user=user)
  card = MedCards.objects.select_related('patient',
'patient__user','doctor').filter(patient=patient).first()
  card_analysis = []
  if card:
    card_analysis = ResultAnalysis.objects.filter(medcard=card)

  context = {'user': user,'patient': patient,'card': card,
        'card_analysis': card_analysis,}
  return render(request, 'cards/result_analysis_profile.html', context)

@login_required
def result_id_profile(request, card_analysis_id):
    user = request.user
    user_groups = request.user.groups.values_list('name', flat=True)
    if "ПАЦІЄНТИ" in user_groups:
        card_analysis = get_object_or_404(
            ResultAnalysis,
            id=card_analysis_id,
            medcard__patient__user=user
        )
    elif "ЛІКАРІ" in user_groups or "МЕД.ПЕРСОНАЛ" in user_groups or
"ЛАБОРАНТИ" in user_groups:
        card_analysis = get_object_or_404(ResultAnalysis,
id=card_analysis_id)
    else:
        return HttpResponseForbidden("Немає доступу до ресурсу.")
    card = card_analysis.medcard
    context = {'user': user, 'card': card, 'card_analysis':card_analysis,}
    return render(request, 'cards/result_analysis_view.html', context)
```

Додаток Г – Слайди презентації

Слайд 1-3 Вступ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА ТА АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Кафедра кібербезпеки та комп'ютерної інженерії

Система управління доступом в медичних інформаційних технологіях

Виконала студентка 2-ого курсу, група БІКСм-24:
Балобольченкова Марія Ігорівна
Керівник:
к.т.н., доцент Ізмайлова О.В.

Вступ

Мета дипломної роботи	Дослідження моделей та методів управління доступом для медичної інформаційної системи з урахуванням сучасних вимог до інформаційної безпеки.
Об'єкт дослідження	Медичні інформаційні системи як комплексні програмні засоби, що забезпечують збір, зберігання, обробку та захист медичних даних.
Предмет дослідження	Моделі та методи управління доступом в медичній інформаційній системі, включно з процесами автентифікації, авторизації та реалізації політик доступу.

Актуальність

Залучення інформаційних технологій в сферу охорони здоров'я та їх швидкий розвиток зумовлюють потребу у впровадженні надійних механізмів захисту даних.

Медичні інформаційні системи опрацьовують великі обсяги конфіденційної інформації, що робить їх вразливими до кіберзагроз. Неналежне управління доступом до цих систем може призвести до витоку персональних даних, несанкціонованої зміни медичних записів, збоїв у роботі медичних установ і значних репутаційних та фінансових втрат.

Тому дослідження механізмів управління доступом за для подальшого їх моделювання, впровадження чивдосконалення є важливим етапом захисту медичної інформації та надійного функціонування сучасних медичних систем.

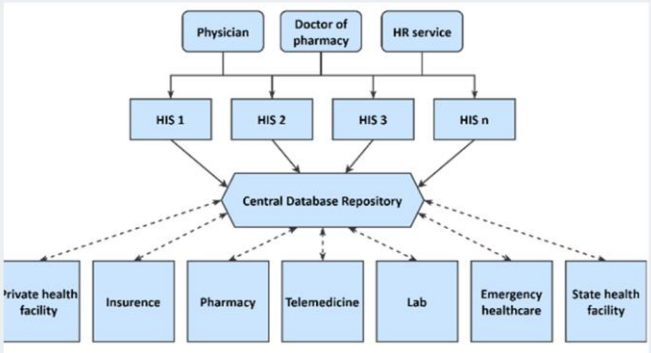
Слайд 4-7 Аналіз предметної області

Аналіз предметної області

Заклади охорони здоров'я використовують комплексні рішення: телемедичні платформи, мобільні додатки, лабораторні системи, сховища медичних зображень.

Такі комплексні підходи зокрема в Україні(ЕСОЗ/eHealth), включають в себе:

медичні інформаційні системи та їх підсистеми;
центральну базу даних;
програмне/апаратне забезпечення;
мережеве з'єднання...



Медичні інформаційні системи. Класифікація

Медична інформаційна система – це програмне забезпечення(ПЗ), за допомогою якого є можливим збір, зберігання, обробка та передача медичних даних.

Класифікація медичних інформаційних систем (МІС):

За типом користувачів:
- медичний персонал, адміністратори, пацієнти, державні структури.

За рівнем охоплення:
- локальні, регіональні, національні.

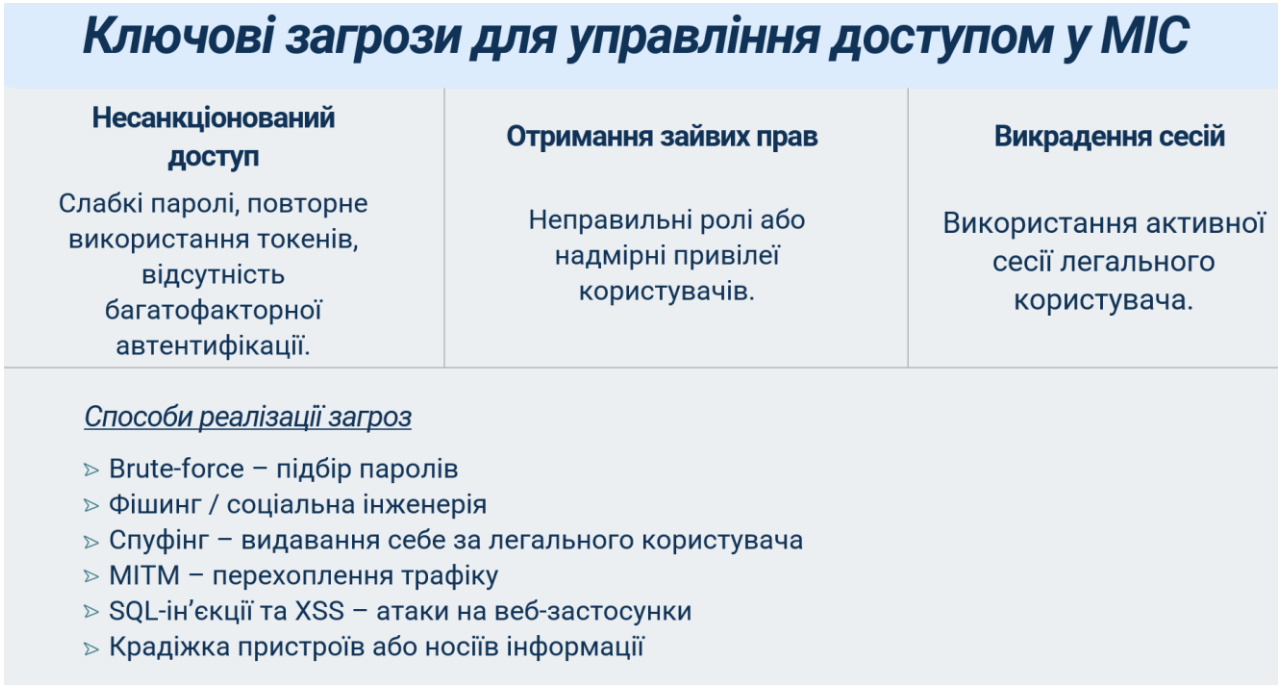
За способом розгортання:
- локальні, хмарні, гібридні.

За архітектурою:
- монолітні, SOA, мікросервісні, багатошарові, подієві, CQRS.

Аналіз існуючих рішень та нормативно-правової бази

Головні недоліки існуючих МІС
- Обмежені механізми автентифікації
- Централізоване зберігання даних
- Низька прозорість доступу до персональних медичних даних
- Проблеми з надмірним навантаженням на систему та не зручного інтефейсу
- Ризики, пов'язані з приватним статусом сервісів
- Відсутність механізмів, які вимагає міжнародна практика (HIPAA, GDPR).

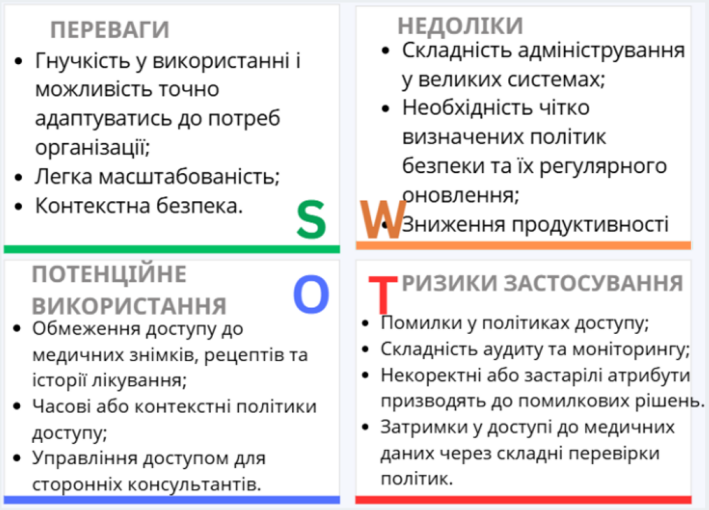
В українських МІС доступ до даних формується під впливом:
- закону України «Про захист персональних даних» та стандартів ЕСОЗ;
- закону України «Про інформацію»;
- постанови КМУ «Деякі питання захисту інформаційних, електронних комунікаційних, інформаційно-комунікаційних, технологічних систем».



Слайд 8-12 Аналіз моделей контролю доступу

ABAC. SWOT-аналіз.

Контроль доступу на основі атрибутів (Attribute-Based Access Control, ABAC) – це модель безпеки, якій притаманно використовування атрибутів для прийняття рішень щодо доступу.

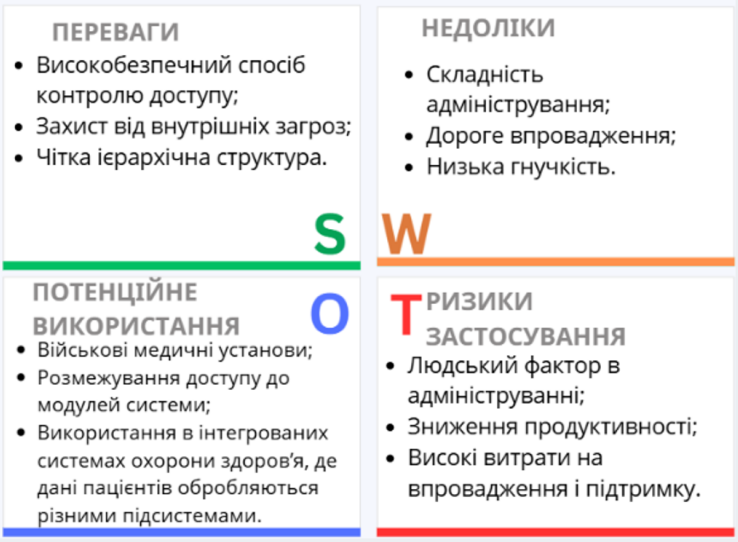


PBAC. SWOT-аналіз.

Контроль доступу на основі політик (Policy-Based Access Control, PBAC) – це модель керування доступом, у якій рішення приймаються на основі визначених політик безпеки.

MAC. SWOT-аналіз.

Обов'язковий контроль доступу (Mandatory Access Control, MAC) – це модель, яка обмежує можливості звичайних користувачів самостійно надавати або забороняти доступ до об’єктів файлової системи.



Порівняння

Модель	Основний принцип	Гнучкість	Управління	Підтримка	Сфера застосування
RBAC	Доступ за ролями користувачів	Низька - середня	Ролі розподіляються централізовано (проста)	Низькі-середні	Організації зі стабільними, добре визначеними ролями
ABAC	Доступ на основі атрибутів користувача, ресурсів та контексту	Висока	Потребує визначення атрибутів і правил (середня)	Високі вимоги	Складні системи з динамічними умовами доступу
PBAC	Доступ на основі політик, що описують правила доступу (“якщо…то”)	Дуже висока	Потребує створення та підтримки політик безпеки (висока)	Дуже високі вимоги	Де потрібна адаптація доступу під різні сценарії (наприклад, надзвичайні ситуації)
MAC	Поділ системи на рівні секретності, рівні доступу для користувачів контроль адміністратора	Низька	Адміністратор централізовано встановлює правила (висока)	Середні-високі вимоги	Високозахищені системи (військові, державні, критичні інфраструктури)

Слайд 13-20 Архітектура системи

Політики доступу

Політики доступу для модулю «Медична карта» реалізовано через поєднання RBAC+ABAC+PBAC, що дозволяє враховувати не лише роль користувача, а й низку додаткових параметрів.

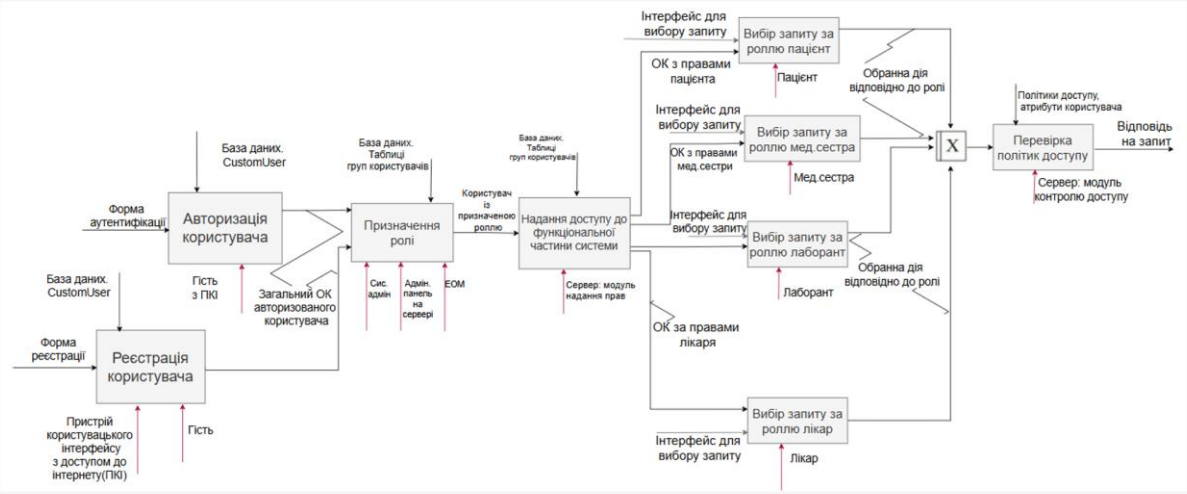
Доступ лікаря визначається його спеціалізацією, рівень залученості лікаря у ведення пацієнта, робочою зміною або режимом екстреного доступу. Такий підхід забезпечує гнучке, безпечне та ситуаційно адаптивне управління доступом до електронної медичної карти.

Таблиця 3.1 – Політики доступу для лікарів

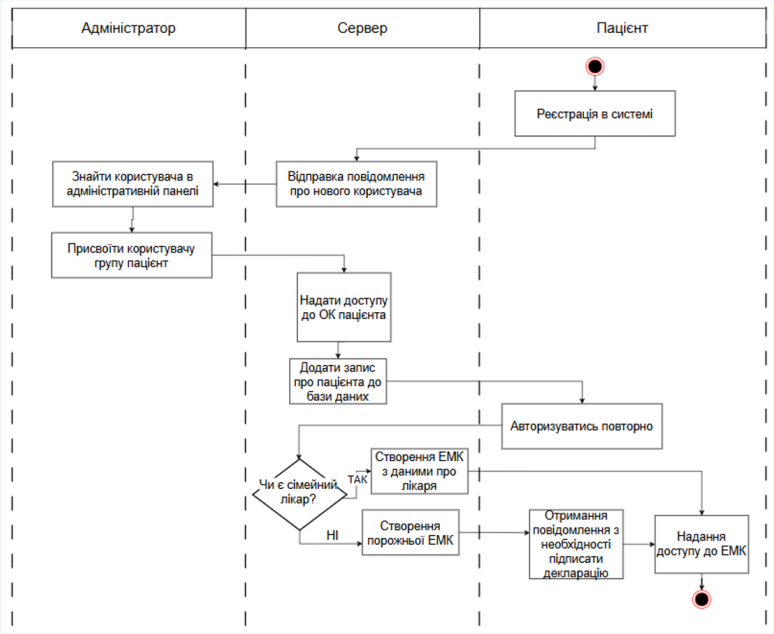
Умова / Ситуація	Модель доступу	Логіка (правило політики)
Лікар має спеціалізацію "терапевт"	ABAC	Атрибут `specialization="therapist"` дозволяє ініціювати процес підписання декларацій із пацієнтами.
Терапевт підписує декларацію з пацієнтом	PBAC (на основі ABAC)	Після підписання лікар отримує атрибут `is_family_doctor=True`.
Сімейний лікар	RBAC + ABAC	Якщо `user.role="doctor"` і `user.is_family_doctor=True`, він має повні права на зміну та перегляд записів.
Лікар іншої спеціалізації (не сімейний)	PBAC	Може переглядати карти лише своїх пацієнтів або тих, хто надав дозвіл. Редагування можливе тільки після дозволу від сімейного лікаря або пацієнта.

IDEF: модель чорної скриньки

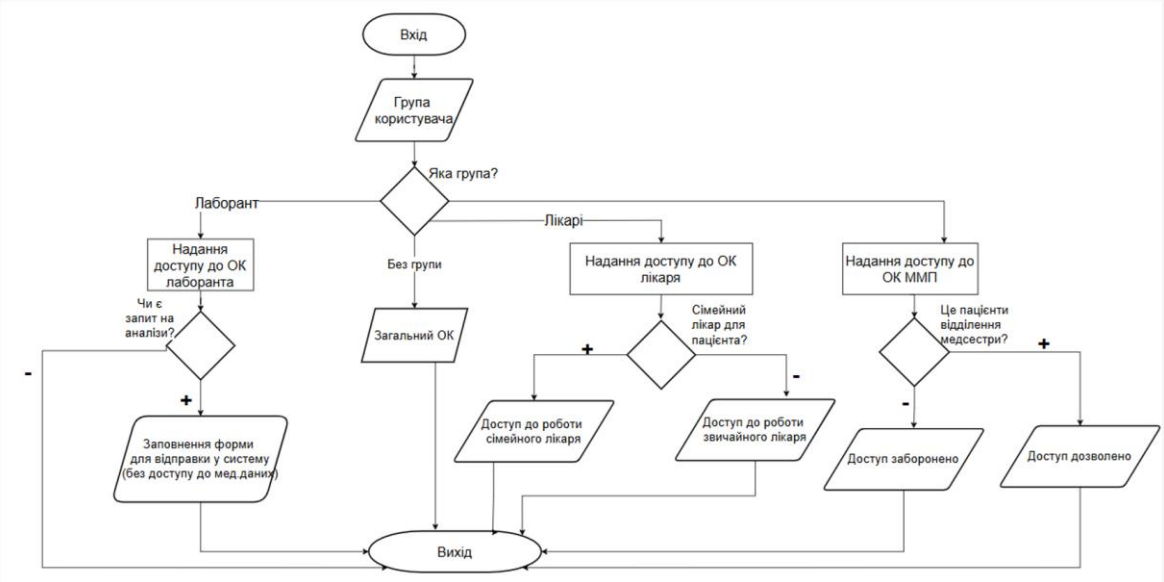
IDEF: декомпозиція моделі чорної скриньки (послідовність процесів керування доступом в МІС)



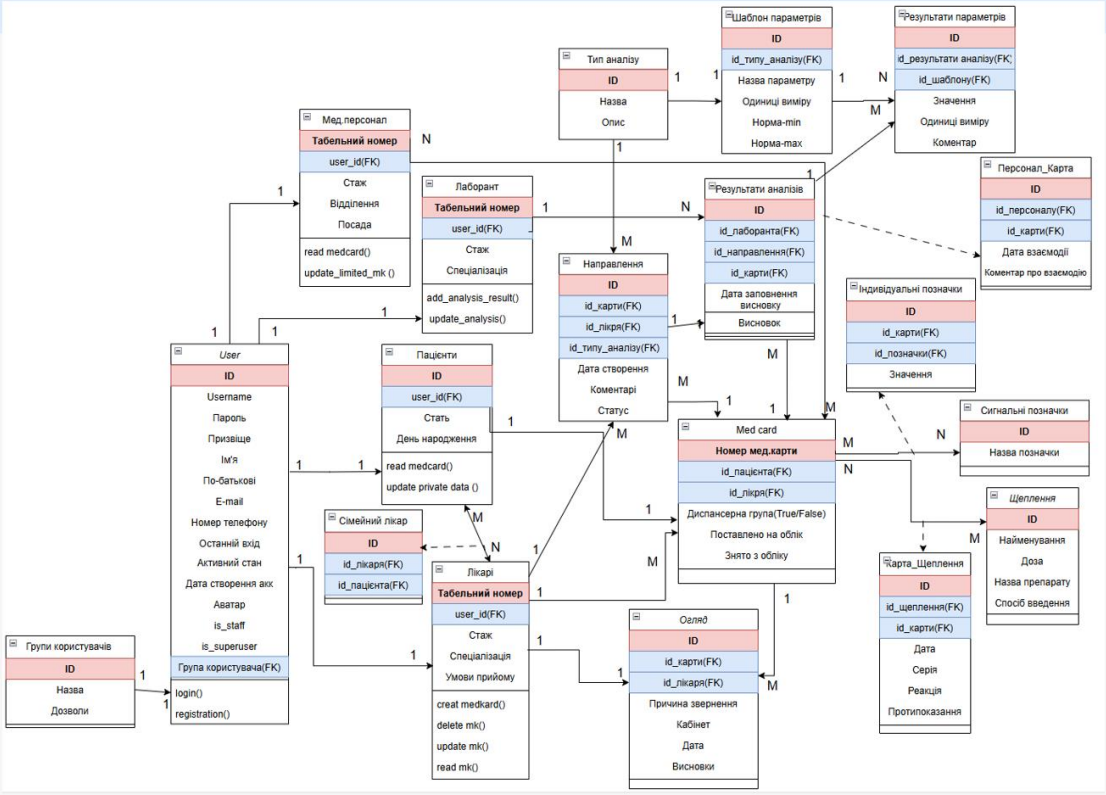
Діаграма діяльності надання доступу до ЕМК пацієнту



Алгоритм надання доступу до системи мед.персоналу



UML: діаграма класів



Слайд 21-24 Реалізація системи

Засоби реалізації та структура проєкту

Засоби реалізації

Серверна частина: Python 3.12, Django 5.2

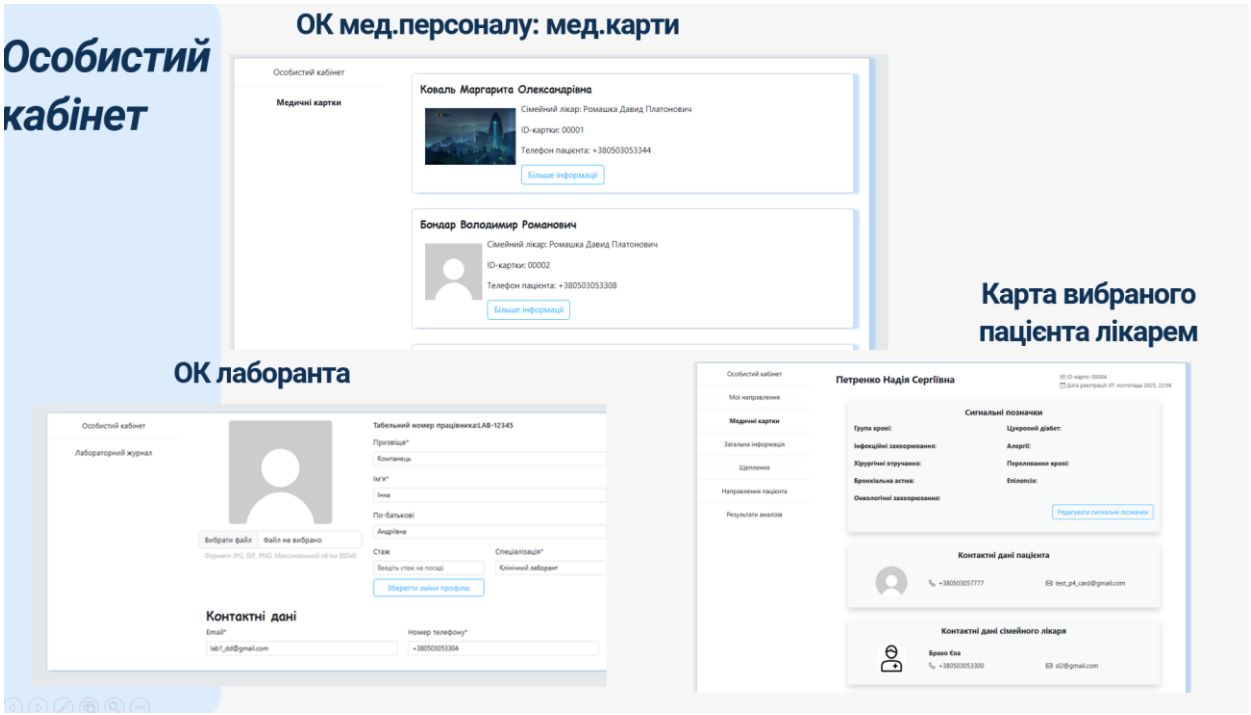
База даних: PostgreSQL

Інтерфейс: HTML, CSS, Bootstrap, Django Templates

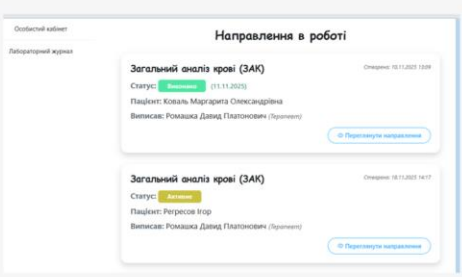
Інструменти: VS Code, Git, GitHub

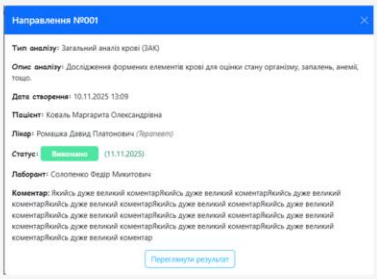
Структурно проєкт поділений на 4 додатки, відповідно до вимог фреймворку:

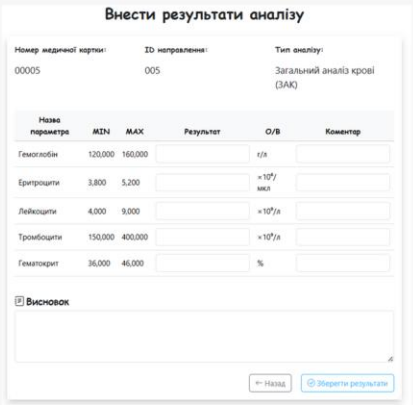
- public_app – оброблює загальнодоступні сторінки сайту.
- users_app – відповідає за роботу з користувачем.
- cards_app – реалізує управління ЕМК пацієнтів.
- laboratory_app – відповідає за облік, збереження та оброблення результатів лабораторних аналізів.

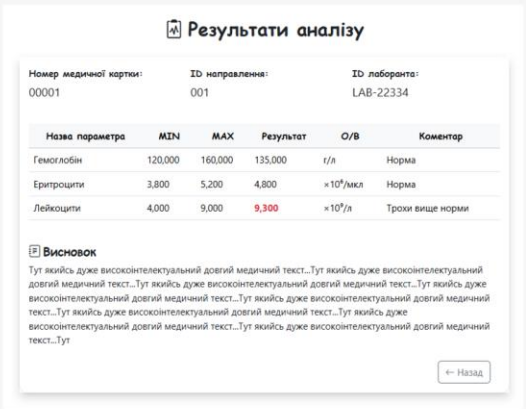


Лабораторія



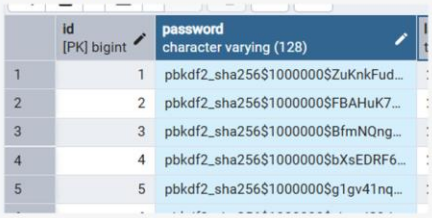






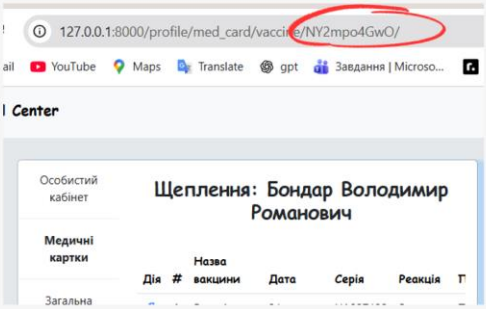
Засоби захисту

Шифрування паролів

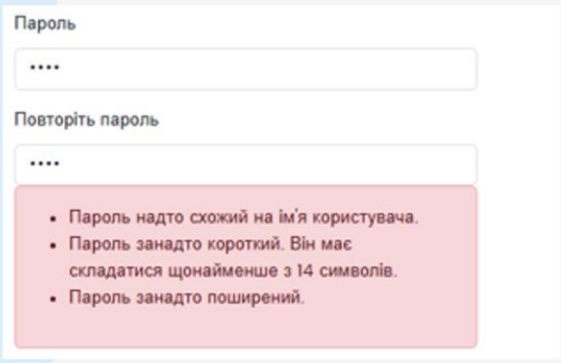


	id [PK] bigint	password character varying (128)
1	1	pbkdf2_sha256$1000000$ZuKnkFud...
2	2	pbkdf2_sha256$1000000$FBAHuK7...
3	3	pbkdf2_sha256$1000000$BfmNQng...
4	4	pbkdf2_sha256$1000000$bXsEDRF6...
5	5	pbkdf2_sha256$1000000$g1gv41nq...

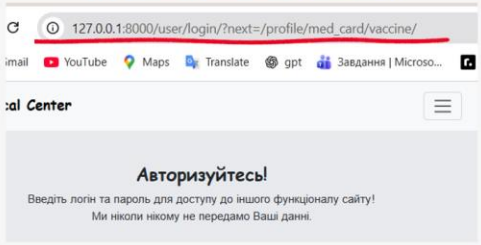
Обфускація даних в url-адресах



Стандартизовані паролі



Перенаправлення на вхід



Слайд 25-26 Висновки

Висновки

У роботі проведено дослідження питань інформаційної безпеки в МІС, класифіковано системи, визначено їхні ризики та проаналізовано сучасні моделі управління доступом.

Порівняння моделей RBAC, ABAC, PBAC і MAC та виконаний SWOT-аналіз показали, що жодна модель окремо не покриває повний спектр вимог до захисту МІС, тому запропоновано комбіновану модель RBAC+ABAC+PBAC.

У практичній частині удосконалено модуль електронної медичної картки на основі фреймворку Django 5.2: розширено систему ролей, додано нові функції та посилено механізми автентифікації, авторизації й захисту від веб-загроз.

Отримані теоретичні та практичні результати підтверджують можливість створення сучасної, безпечної та масштабованої МІС, придатної для подальшого розвитку та інтеграції з компонентами електронної системи охорони здоров'я.

Висновки. Публікації

В рамках міжнародної науково-практичних конференцій молодих вчених «БУД-МАЙСТЕР-КЛАС-2024» та «БУД-МАЙСТЕР-КЛАС-2025» мною було представлено дві наукові тези:

- «Безпека web-додатків: SQL-ін'єкції – один з найпопулярніших методів кібератак» (2024), де висвітлено актуальні загрози та необхідність застосування безпечних підходів до розробки;
- «Застосування фреймворку Django для побудови безпечної інформаційної системи управління доступом на прикладі медичної сфери» (2025), де обґрунтовано ефективність використання Django та сучасних моделей контролю доступу у МІС.