ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для ведення персонального архіву медичних даних»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Геннадій СОКАЛЬСЬКИЙ
(підпис)

Виконав: Здобувач вищої освіти групи ПД-42

Геннадій СОКАЛЬСЬКИЙ

Керівник: Тимур ДОВЖЕНКО
доц. каф. к.т.н.

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сокальський Геннадій Костянтинович

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для ведення персонального архіву медичних даних»
керівник кваліфікаційної роботи доц. каф. к.т.н. Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи зберігання та захисту персональних медичних даних, аналіз існуючих медичних інформаційних систем, вимоги до безпечного збереження електронної медичної інформації, документація щодо технологій ASP.NET Core, Entity Framework Core, React та JWT, а також рекомендації з побудови багатошарових веб-додатків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз сучасних систем і технологій зберігання та управління персональними медичними даними.
2. Проєктування веб-додатку для ведення персонального архіву медичн даних з можливістю надання контрольованого доступу лікарям.
3. Програмна реалізація та опис функціональних можливостей системи ведення персонального архіву медичних даних.

4. Тестування основних сценаріїв роботи та аналіз результатів функціонування програмного забезпечення.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд сучасних рішень та технологій для зберігання та обробки персональних медичних даних	14.03–20.03.2025	
4	Проєктування веб-додатку для ведення персонального архіву медичних даних	21.03–01.04.2025	
5	Програмна реалізація веб-додатку	02.04–20.04.2025	
6	Тестування системи та аналіз результатів	21.04–28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Геннадій СОКАЛЬСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 стор., 1 табл., 33 рис., 23 джерел.

Мета роботи – розробка програмного забезпечення для централізованого зберігання, впорядкування та контролю доступу до особистих медичних даних користувача.

Об'єкт дослідження – процес зберігання та управління персональною медичною інформацією.

Предмет дослідження – програмне забезпечення для ведення персонального архіву медичних даних із можливістю надання доступу лікарям.

Короткий зміст роботи: у роботі проаналізовано сучасні підходи до організації персональних медичних архівів, вивчено функціональні можливості існуючих рішень та виявлено їхні обмеження. Здійснено проєктування системи з урахуванням вимог до безпеки, масштабованості та зручності використання. Реалізовано ключову функціональність: реєстрацію та аутентифікацію користувачів, створення та категоризацію медичних записів (прийоми у лікаря, рецепти, направлення, довідки, щеплення), завантаження та зберігання сканів документів, фільтрацію та пошук записів, надання лікарям контрольованого доступу до даних.

У розробці використано ASP.NET Core Web API для серверної частини, Entity Framework Core для взаємодії з базою даних, Microsoft SQL Server для збереження структурованої інформації, JWT для реалізації аутентифікації та авторизації, AutoMapper для мапінгу об'єктів, React з TypeScript для клієнтської частини. Система протестована вручну, зокрема перевірено основні сценарії роботи користувачів і механізми захисту даних.

Сфера застосування системи – особисте використання пацієнтами для впорядкованого зберігання медичної інформації та безпечного її надання лікарям за потреби.

Ключові слова: ПЕРСОНАЛЬНИЙ МЕДИЧНИЙ АРХІВ, ЗБЕРІГАННЯ ДАНИХ, JWT, АВТОРИЗАЦІЯ, ДОСТУП ЛІКАРІВ, ASP.NET CORE, REACT, SQL SERVER

ЗМІСТ

ВСТУП

Активний розвиток цифрових технологій відкриває нові можливості для зберігання та використання персональних медичних даних. В умовах зростання обсягів медичної інформації особливо важливим є її впорядковане зберігання, безпечний доступ і доступність для пацієнтів та лікарів. Часто така інформація розпорошена або зберігається у паперовому вигляді, що ускладнює її використання у важливих ситуаціях.

Існуючі інформаційні системи охорони здоров'я, зокрема ті, що інтегровані з державною електронною системою охорони здоров'я, часто не передбачають повноцінної підтримки для самостійного ведення та контролю особистих медичних архівів. Більшість таких рішень зосереджені на автоматизації процесів у межах конкретного медичного закладу або мережі, і не дозволяють пацієнтам централізовано зберігати всю свою медичну історію в одному безпечному середовищі.

У зв'язку з цим виникає потреба у створенні веб-застосунку, який дозволить користувачам самостійно вести електронний архів медичних даних: зберігати інформацію про прийоми у лікаря, рецепти, направлення, щеплення, медичні довідки, прикріплювати документи, а також контролювати доступ до них з боку лікарів. Таке рішення сприятиме підвищенню поінформованості пацієнтів про стан власного здоров'я, покращенню комунікації з лікарями, а також забезпеченню безпечного та конфіденційного обміну медичними даними.

У рамках цієї роботи було розроблено програмне забезпечення для ведення персонального архіву медичних даних з використанням сучасних веб-технологій. Робота охоплює аналіз існуючих аналогів, визначення функціональних і нефункціональних вимог, проєктування архітектури системи, програмну реалізацію та тестування. Особливу увагу приділено безпеці зберігання і передачі медичних даних, а також реалізації механізмів контрольованого доступу.

Результатом роботи є прототип веб-додатку, який забезпечує користувачу простий та безпечний інструмент для зберігання особистої медичної інформації, з можливістю подальшого масштабування, розвитку і впровадження в практику.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості зберігання та обробки медичних даних

Медичні дані є одним із найчутливіших видів персональної інформації, оскільки містять відомості про фізичний та психічний стан здоров’я людини, діагнози, результати обстежень, рецепти, історію лікування, а також особисту інформацію пацієнтів. Їх зберігання та обробка потребують особливої уваги як з боку безпеки, так і з боку правового регулювання.

Однією з ключових особливостей є необхідність довготривалого зберігання даних. Медичні записи можуть знадобитися через багато років після їх створення, наприклад, для аналізу хронічних захворювань або призначення нового лікування. Тому система збереження повинна забезпечувати стійкість до втрати даних, підтримку архівації, а також універсальність форматів (наприклад, зберігання як структурованих даних, так і файлів-документів, сканів, зображень тощо).

Ще одна важлива особливість — необхідність постійного оновлення медичних записів. Пацієнти регулярно звертаються до лікарів, отримують нові призначення, проходять процедури й обстеження. Система повинна підтримувати динамічне оновлення інформації, зберігаючи при цьому повну історію змін для медичної аналітики та аудиту.

Крім того, медичні дані часто мають ієрархічну або взаємопов’язану структуру: прийоми у лікаря можуть бути пов’язані з направленнями, результатами аналізів, рецептами на ліки. Тому ефективне зберігання такої інформації вимагає правильного проєктування структури даних та реляційної моделі в базі даних.

Не менш важливою є потреба у контрольованому доступі. Доступ до медичних записів повинні мати лише ті особи, яким пацієнт свідомо надав відповідні права, — переважно лікарі. При цьому кожен випадок доступу має бути зафіксований для підзвітності та безпеки.

У контексті цифрових технологій все більш актуальним стає питання сумісності з іншими медичними системами. Це може включати обмін даними з

електронними медичними картками, лабораторіями, страхуванням тощо, що потребує підтримки стандартів, таких як HL7 або FHIR.

Таким чином, зберігання та обробка медичних даних — це складний процес, який вимагає врахування високих вимог до безпеки, конфіденційності, надійності, доступності та масштабованості. Успішна реалізація такого рішення вимагає комплексного підходу до проєктування програмного забезпечення.

1.2 Проблеми управління особистими медичними записами

На сьогоднішній день управління особистими медичними записами стикається з рядом суттєвих проблем, які ускладнюють ефективне ведення та використання медичної інформації пацієнтів. Більшість із цих проблем пов'язані з відсутністю централізованого підходу до зберігання даних, низьким рівнем цифровізації у медичних установах та складністю обміну інформацією між пацієнтом і лікарем.

Однією з головних проблем є розпорошеність даних. Медична інформація зазвичай зберігається у паперовому вигляді або в локальних базах даних окремих закладів охорони здоров’я. У результаті пацієнт змушений самостійно зберігати результати аналізів, рецепти, довідки та інші документи. Це не лише незручно, але й підвищує ризик втрати важливої інформації.

Ще одна проблема — відсутність постійного доступу до медичної історії. У разі зміни лікаря або закладу пацієнт часто втрачає можливість швидко надати новому лікарю повну інформацію про попереднє лікування. Це може призвести до повторного проходження обстежень, некоректних діагнозів або помилок у призначенні терапії.

Крім того, пацієнти зазвичай не мають інструментів для самостійного аналізу або впорядкування своїх медичних записів. Усі документи зберігаються у довільному вигляді, без категоризації, міток чи можливості пошуку. Це значно ускладнює орієнтацію в медичній інформації, особливо у випадках тривалого лікування або наявності хронічних хвороб.

Також варто відзначити недостатню захищеність медичних документів, коли вони зберігаються в месенджерах, на пошті чи в хмарних сховищах без належного шифрування. Це створює ризики витоку конфіденційної інформації та порушення прав пацієнтів.

Ще однією актуальною проблемою є ускладнений процес передачі даних лікарям. Часто обмін інформацією відбувається в ручному режимі — через фотографії, скани або паперові копії. Це не лише неефективно, але й може призводити до втрати якості інформації.

Таким чином, існує нагальна потреба у створенні зручного, безпечного та централізованого інструменту для управління медичними записами, який дозволить пацієнтам контролювати свою медичну інформацію, а лікарям — швидко отримувати доступ до повної історії хвороби пацієнта.

1.3 Аналіз існуючих рішень та систем

У сучасному цифровому середовищі з’являється дедалі більше сервісів, орієнтованих на покращення взаємодії між пацієнтами та медичною системою. Проте більшість з них розроблялись не як універсальні архіви медичних даних, а як допоміжні платформи для онлайн-запису, перегляду результатів аналізів чи отримання консультацій. Через це функціональність таких систем часто обмежена, і вони не повністю відповідають потребам користувача, який прагне мати централізоване сховище для всієї своєї медичної історії.

Буде розглянуто три приклади наявних рішень — MyChart, Helsi.me та MyHealthApp, які частково реалізують концепцію електронного медичного архіву. Для кожного з них буде проведено аналіз функціональних можливостей, а також виявлено обмеження, що заважають використовувати ці сервіси як повноцінні засоби управління медичною інформацією.

Однією з ключових проблем, характерних для більшості таких рішень, є відсутність можливості самостійного завантаження медичних документів користувачем. Наприклад, у сервісі Helsi.me користувач має доступ до результатів аналізів, призначень і даних про прийоми у державних медичних установах, але не

може додавати власні довідки, рецепти, скани щеплень чи інші документи, отримані в приватних клініках або за кордоном. Таким чином, важлива частина медичної історії залишається поза системою.

Ще одним обмеженням є відсутність повноцінного контролю за доступом до інформації. У деяких системах пацієнт не має гнучкого механізму надання й відкликання дозволів на перегляд окремих категорій записів. Це створює ризики неконтрольованого доступу та знижує рівень приватності.

Крім того, існуючі рішення часто не підтримують категоризацію та систематизацію записів. Дані представлені у вигляді хронологічного списку без можливості зручно фільтрувати записи за типом (рецепти, направлення, довідки, щеплення тощо) або призначенням. Це ускладнює пошук потрібної інформації, особливо при великій кількості записів.

Також виявлено недостатню гнучкість та масштабованість платформ: більшість з них орієнтовані на конкретні функції (наприклад, запис до лікаря чи перегляд обстежень) та не дозволяють легко додавати нові типи медичних даних або налаштовувати логіку збереження під потреби користувача.

Буде подано порівняльну таблицю, яка систематизує результати аналізу за ключовими критеріями. На її основі буде обґрунтовано переваги розроблюваного програмного забезпечення, яке має на меті усунути наявні недоліки та забезпечити користувача зручним і безпечним інструментом для повноцінного управління особистими медичними записами.

1.3.1 MyChart

MyChart — це популярна пацієнтська платформа, розроблена компанією Epic Systems, яка використовується у США, Канаді, країнах Європи та інших регіонах світу. Вона дозволяє пацієнтам отримувати доступ до своєї медичної інформації, записуватись на прийом до лікаря, переглядати результати лабораторних досліджень, отримувати рекомендації, а також спілкуватися з медичними працівниками через захищене повідомлення.

На рисунку 1.1 зображений інтерфейс MyChart.

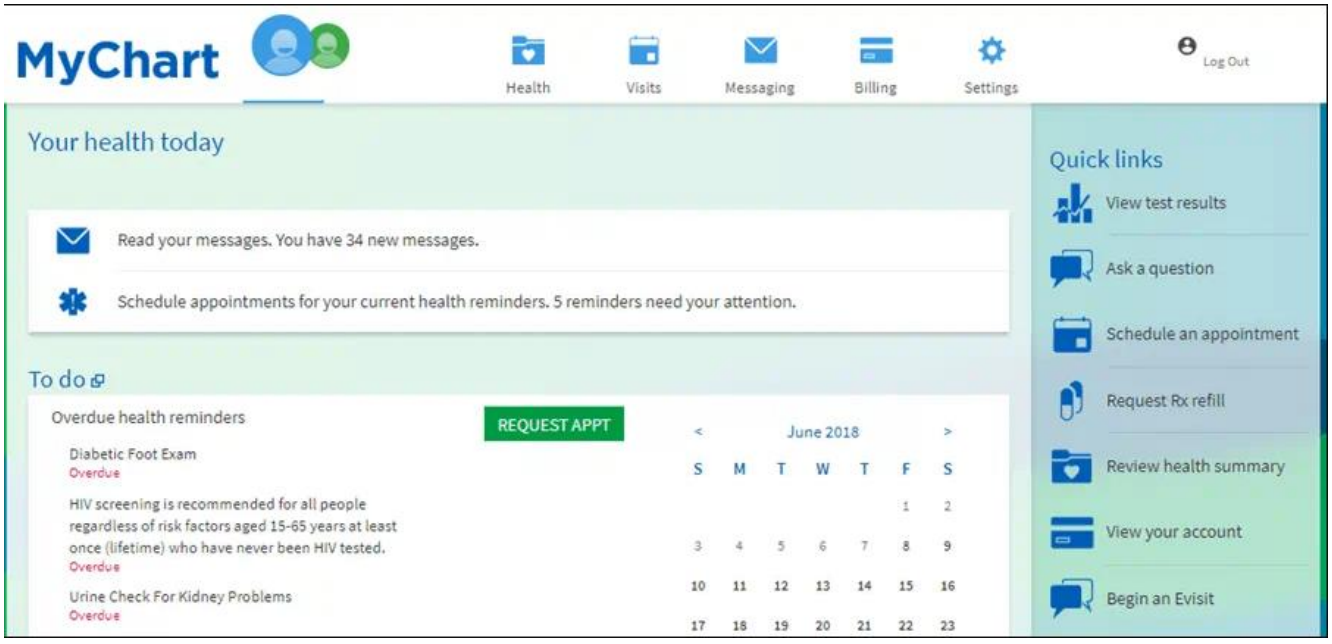


Рис. 1.1 Інтерфейс MyChart

Серед функцій, що наближають MyChart до концепції персонального архіву медичних даних, можна виділити:

- доступ до результатів аналізів та візитів;
- перегляд графіка щеплень і призначених медикаментів;
- можливість надсилати медичну інформацію лікарю;
- завантаження деяких типів медичних документів.

Разом із тим, MyChart не є універсальним сховищем медичних даних, оскільки:

- усі записи формуються лише на основі інформації, наданої клініками, що використовують Epic. Користувач не може вільно додавати записи з інших джерел або створювати власні категорії.
- самостійне завантаження медичних файлів (наприклад, довідок, сканів обстежень, PDF-документів) або їх редагування часто недоступне або обмежене, залежно від конфігурації конкретного закладу.
- немає повного контролю над інформацією — пацієнт не може гнучко керувати тим, які саме дані будуть доступні лікарю, немає вибіркового надання доступу за категоріями або документами.

- інтерфейс переважно англомовний, і платформа прив’язана до американської медичної системи, що ускладнює її застосування в українських реаліях.

Попри сучасний підхід до цифровізації медичної інформації, MyChart залишається сервісом, орієнтованим на обмін даними між пацієнтом і лікарем у межах однієї медичної мережі, а не на самостійне, централізоване ведення особистої медичної історії користувачем.

На відміну від цього, розроблюване програмне забезпечення націлене на повну автономію пацієнта:

- можливість додавати будь-який тип запису (довідки, щеплення, направлення, рецепти);
- підтримка завантаження файлів будь-якого формату;
- чіткий контроль над доступом лікарів до конкретних даних;
- універсальність — система не залежить від певної мережі клінік або державного реєстру.

Таким чином, MyChart є прогресивним і зручним рішенням у межах закритих медичних екосистем, але не покриває потреб користувача у створенні повноцінного, незалежного персонального архіву медичних даних.

1.3.2 Helsi

Helsi.me — одна з найвідоміших медичних інформаційних систем в Україні, яка надає онлайн-сервіси для пацієнтів і лікарів. Платформа є частиною електронної системи охорони здоров’я (eHealth) і дозволяє користувачам записуватись до лікарів, переглядати історію візитів, рецепти та результати деяких аналізів, отриманих у державних медичних закладах. Сервіс також підтримує створення електронних декларацій з сімейним лікарем.

На рисунку 1.2 зображена система Helsi.

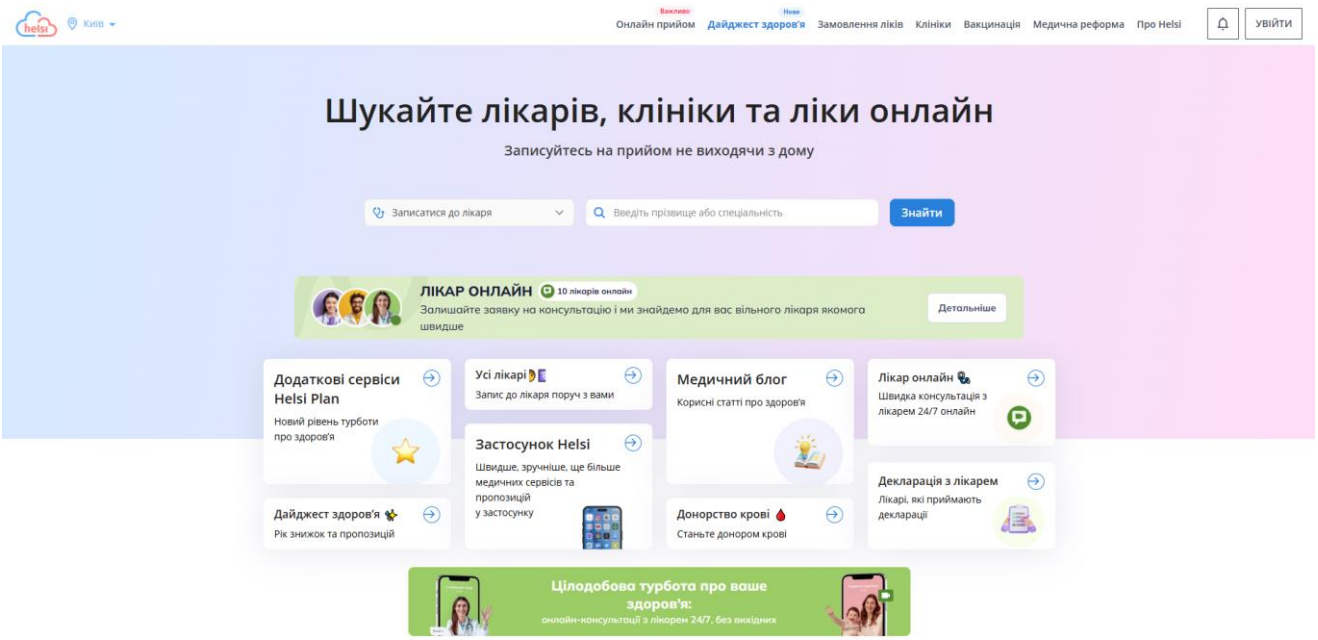


Рис. 1.2 Інтерфейс Helsi

Основна цінність Helsi.me — це зручність взаємодії з державними закладами охорони здоров’я. Пацієнт може переглянути, до якого лікаря він приписаний, які рецепти було виписано, які прийоми відбулися, а також ознайомитися з деякими медичними висновками, внесеними лікарями у систему.

Водночас, попри свій масштаб і офіційний статус, Helsi.me не є повноцінним персональним архівом медичних даних, оскільки має ряд суттєвих обмежень, серед яких:

- Відсутність можливості самостійного додавання записів. Користувач не може створити власні записи, додати прийом у приватній клініці, записати результати обстежень, вести власні нотатки чи історію хвороби.
- Неможливість завантаження медичних документів. Helsi.me не підтримує функцію зберігання сканів або фотографій довідок, результатів аналізів, медичних заключень, щеплень чи направлень, що отримані поза системою.
- Обмежений перелік типів даних. Система орієнтована переважно на дані з державного сектору — рецепти, декларації, прийоми. Інші важливі категорії, такі як щеплення, довідки чи направлення з приватних клінік, не відображаються.

- Немає механізму управління доступом лікарів до власної інформації. Пацієнт не може вибірково відкривати або закривати дані, а також надавати тимчасовий доступ до певної частини медичної історії конкретному лікарю.

Ці обмеження свідчать про те, що Helsi.me виконує вузькоспеціалізовану функцію — обслуговування медичних послуг, наданих державними закладами, але не може виступати єдиною системою для збереження повної, персоналізованої, гнучко контрольованої медичної історії користувача.

В межах розроблюваного проєкту передбачається усунення цих недоліків шляхом реалізації наступних можливостей:

- створення, редагування та категоризація будь-яких типів медичних записів;
- зберігання файлів будь-якого формату (PDF, JPG, PNG);
- гнучке управління доступом лікарів до інформації;
- підтримка приватних, іноземних, або особистих записів, незалежно від джерела.

Таким чином, аналіз Helsi.me підтверджує актуальність потреби в окремому, незалежному інструменті для персонального ведення медичних даних, який дає повний контроль над інформацією без прив’язки до державної системи.

1.3.3 MyHealthApp

MyHealthApp — це мобільний додаток, створений компанією Inhealthcare (Велика Британія), який орієнтований на покращення комунікації між пацієнтами та медичними установами. Додаток дозволяє пацієнтам отримувати медичні рекомендації, проходити опитування, переглядати інформацію про стан здоров’я та виконувати базове відстеження хронічних захворювань.

На рисунку 1.3 зображений інтерфейс MyHealthApp.

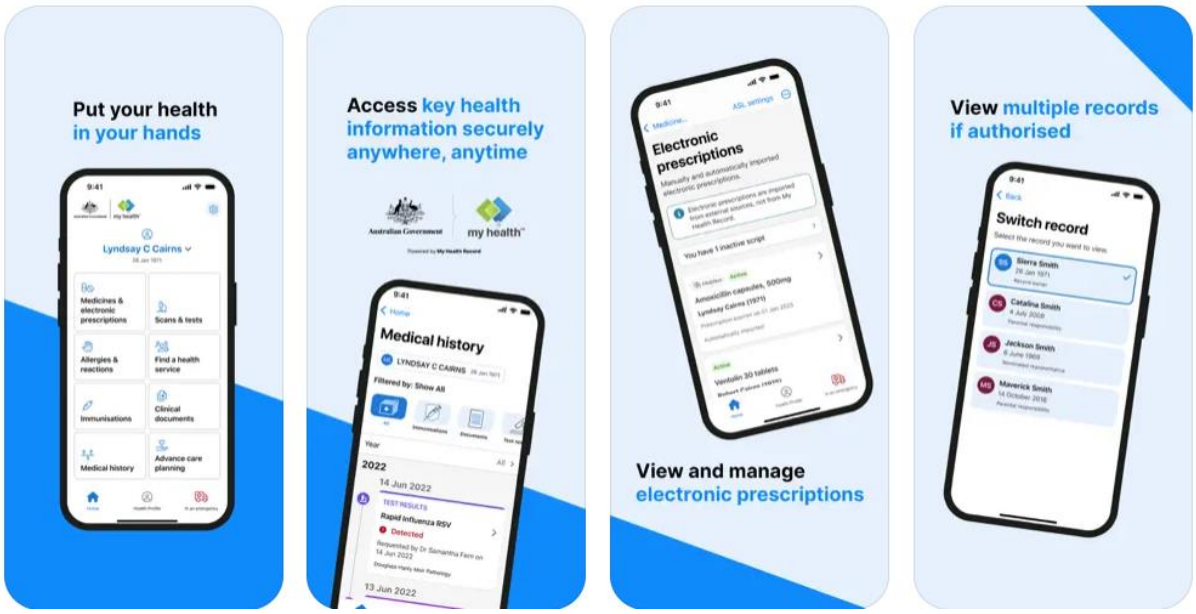


Рис. 1.3 Інтерфейс MyHealthApp

Особливістю MyHealthApp є підтримка індивідуальних планів лікування, які створюються лікарями, а також можливість введення базових показників (температура, тиск, рівень глюкози тощо), що дозволяє лікарям контролювати динаміку стану пацієнта в режимі онлайн. Система також може надсилати пацієнту нагадування, опитування щодо самопочуття або рекомендації на основі даних.

Попри це, MyHealthApp не є повноцінним сховищем особистої медичної інформації. Серед ключових обмежень варто виокремити:

- Відсутність структури для збереження довідок, результатів обстежень або сканів документів. Користувач не може завантажити PDF або зображення медичних заключень, довідок чи щеплень.
- Уся взаємодія відбувається в рамках наданого шаблону або плану від лікаря. Самостійне створення записів не передбачене — пацієнт лише відповідає на запитання або вводить показники, заздалегідь визначені системою.
- Немає централізованої історії лікування — дані фрагментовані за епізодами лікування або за кампаніями моніторингу.
- Обмежені можливості для персонального управління доступом. Користувач не має контролю над тим, яка саме інформація потрапляє до медичного персоналу, а також не може відкликати доступ до окремих даних.

Таким чином, хоча MyHealthApp є корисним інструментом для цільового дистанційного моніторингу стану здоров’я, він не відповідає потребам користувача, який прагне зберігати повну, довготривалу та структуровану історію медичних даних.

У межах розробки власної системи ці обмеження будуть усунені шляхом реалізації таких функцій:

- вільне створення будь-яких медичних записів із категоризацією;
- можливість завантаження та зберігання сканів, фото та документів;
- централізована історія всіх медичних подій користувача;
- гнучке управління доступом для лікарів до окремих категорій даних.

Таким чином, MyHealthApp демонструє користь мобільних технологій у моніторингу пацієнтів, але не покриває потреби у веденні повноцінного персонального архіву медичних записів, що ще раз підкреслює доцільність створення нової, незалежної та гнучкої системи.

1.4 Порівняння аналогів

На основі проведеного аналізу трьох існуючих систем — MyChart, Helsi.me та MyHealthApp — було складено порівняльну таблицю за ключовими критеріями, що відображають функціональні можливості та обмеження кожного з рішень.

Таблиця 1.1

Порівняльна характеристика аналогів

Критерій	MyChart	Helsi	MyHealthApp	MedicalArchive
Самостійне додавання записів	-	-	-	+
Категоризація записів	-	-	-	+
Завантаження сканів, фото документів	+	-	-	+

Продовження таблиці 1.1

Порівняльна характеристика аналогів

Критерій	MyChart	Helsi	MyHealthApp	MedicalArchive
Керування доступом лікарів	-	-	-	+
Незалежність від конкретної медичної мережі	-	-	+	+
Централізована історія медичних даних	+	+	-	+
Фільтрація та пошук по записах	-	+	-	+
Орієнтованість на повноцінний архів	-	-	-	+

Як видно з таблиці, жоден із аналізованих аналогів не забезпечує повної реалізації концепції персонального медичного архіву, де користувач має:

- повний контроль над медичними записами;
- можливість структурувати дані за типами;
- безпечно зберігати медичні документи в цифровому вигляді;
- гнучко надавати доступ до даних лікарям.

MyChart — сильна система в межах певної медичної мережі, але користувач обмежений у додаванні або зміні інформації. Helsi.me інтегрована з державною системою, однак теж не дозволяє додавати власні записи чи документи. MyHealthApp має переваги для моніторингу стану здоров’я, але не є інструментом для ведення повного архіву.

Усі ці обмеження враховані в розроблюваному програмному забезпеченні. Воно спроектоване як незалежна, масштабована, безпечна платформа, що дозволяє кожному користувачу вести власну медичну історію в єдиному цифровому середовищі, із гнучким керуванням і доступом.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Визначення вимог

Ефективне проєктування системи починається з чіткого визначення вимог, які вона повинна задовольняти. Для досягнення поставленої мети необхідно сформулювати як функціональні, так і нефункціональні вимоги. Функціональні вимоги описують, що саме повинна робити система, тоді як нефункціональні — якою має бути якість реалізації цих функцій.

2.1.1 Функціональні вимоги

Перед початком проєктування системи важливо чітко визначити вимоги, які вона повинна задовольняти. Це дозволяє сформувати цілісне уявлення про функціональність програмного забезпечення, забезпечити узгодженість між очікуваннями користувача та технічною реалізацією, а також уникнути архітектурних помилок у подальшому. Усі вимоги до системи умовно поділяються на функціональні та нефункціональні.

Функціональні вимоги визначають основну поведінку системи з погляду її призначення. Одним із ключових аспектів є реалізація механізмів реєстрації та аутентифікації користувачів. Система повинна забезпечити безпечний вхід до облікового запису за допомогою токенів авторизації та підтримувати розмежування доступу за ролями (пацієнт і лікар). Після входу користувач отримує можливість створювати, редагувати, переглядати та видаляти власні медичні записи. Кожен запис має бути прив’язаний до певної категорії, наприклад: прийоми у лікаря, рецепти, направлення, довідки або щеплення.

Окрему увагу приділено роботі з медичними документами. Користувач повинен мати змогу завантажувати до системи скани або фотографії відповідних документів у зручному форматі (PDF, JPG, PNG) та переглядати їх у своєму особистому архіві. Для спрощення навігації система повинна підтримувати фільтрацію та пошук записів за назвою, датою або категорією. Це дозволить легко знаходити потрібну інформацію навіть за наявності великого обсягу даних.

Крім цього, система повинна забезпечити можливість надання контрольованого доступу лікарям. Пацієнт має мати змогу надати або відкликати доступ до своїх записів, використовуючи електронну пошту лікаря, а також обмежити доступ лише до окремих типів медичних записів. У свою чергу, лікарі, отримавши доступ, повинні мати змогу переглядати інформацію про пацієнтів, що його надали, з можливістю фільтрувати й аналізувати їхню медичну історію.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги стосуються якості роботи системи та визначають її технічні характеристики. Перш за все, особлива увага приділяється безпеці: усі запити повинні передаватись через захищене з'єднання (HTTPS), а паролі зберігатись у хешованому вигляді з використанням сучасних криптографічних алгоритмів. Авторизація користувачів здійснюється за допомогою JWT-токенів, що дозволяє ефективно захищати REST API. Також система повинна забезпечити високий рівень продуктивності: інтерфейс повинен залишатися швидким та чуйним навіть за наявності великої кількості записів або запитів до бази даних.

Масштабованість системи — ще одна критично важлива вимога. Архітектура має дозволяти розширення функціоналу без кардинальної перебудови проєкту. Це забезпечується шляхом модульного підходу, дотриманням принципів SOLID і використанням багатошарової структури. Також важливою є зручність користування. Інтерфейс повинен бути інтуїтивно зрозумілим, адаптивним до різних розмірів екранів і доступним як з комп'ютера, так і з мобільного пристрою.

Надійність і відмовостійкість також мають бути на високому рівні. Усі дії користувача повинні зберігатися без ризику втрати даних, а в разі виникнення помилки система має правильно її обробити й повідомити користувача про проблему. Логування помилок необхідне для подальшої діагностики та вдосконалення роботи програми.

Таким чином, визначені вимоги дозволяють сформувати повноцінну технічну основу для реалізації системи, яка буде не лише функціональною, але й зручною, безпечною, надійною та готовою до розширення.

2.2 Загальний опис архітектури

Архітектура системи "Персональний архів медичних даних" побудована на основі багатошарової архітектури (N-layer architecture), яка забезпечує чітке розділення відповідальностей між різними компонентами системи. Такий підхід дозволяє досягти високої модульності, полегшує тестування та подальший розвиток системи.

Система складається з чотирьох основних архітектурних шарів, кожен з яких виконує специфічні функції та має чітко визначені інтерфейси взаємодії з іншими шарами. На рисунку 2.1 зображена діаграма загальної архітектури системи.

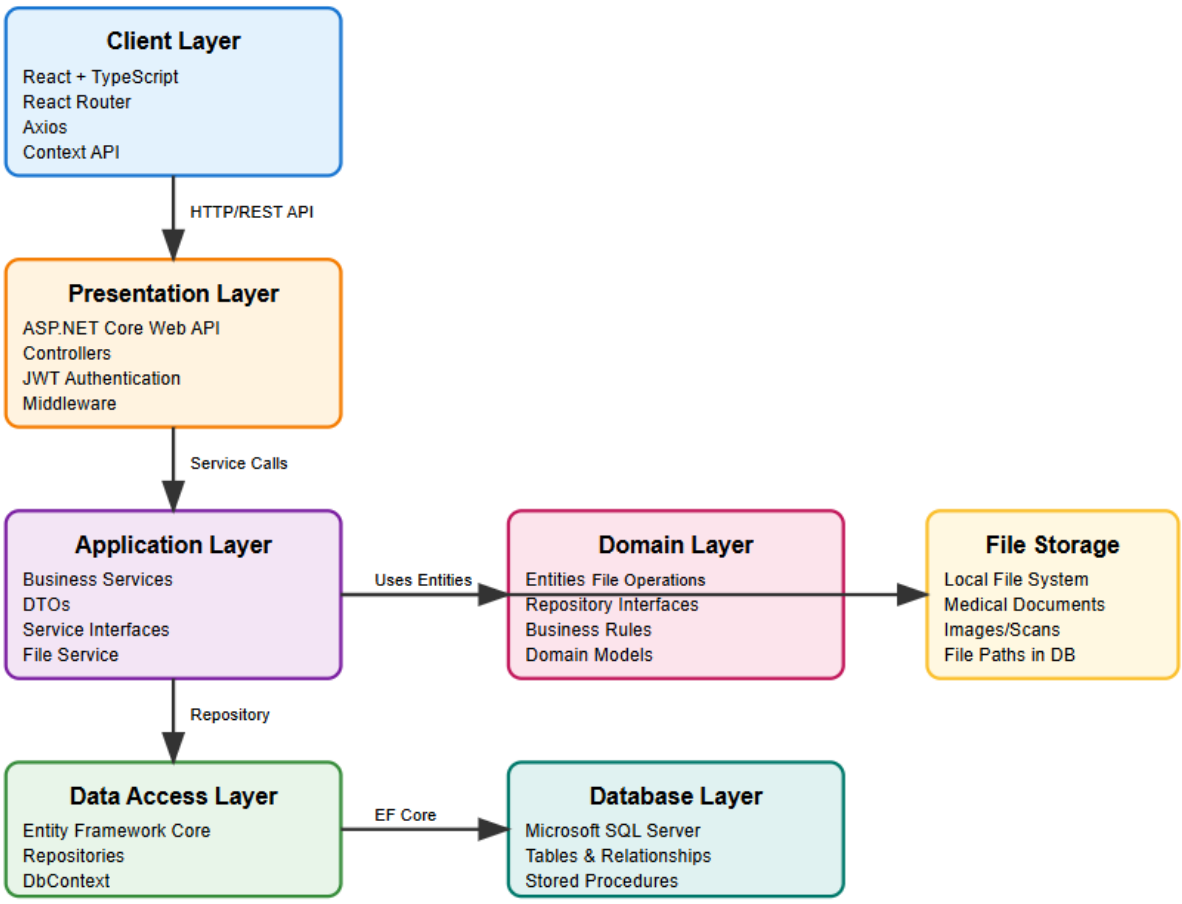


Рис. 2.1 Діаграма загальної архітектури системи

Domain Layer (Доменний шар) є основою всієї системи і містить бізнес-логіку та доменні моделі. Цей шар включає в себе сутності (Entity), які представляють основні об'єкти предметної області: користувачі, прийоми у лікаря, рецепти, направлення, щеплення та медичні довідки. Також в цьому шарі визначені

інтерфейси репозиторіїв, які описують контракти для доступу до даних, не залежачи від конкретної реалізації.

Data Access Layer (Шар доступу до даних) відповідає за взаємодію з базою даних і реалізацію персистентності даних. Цей шар використовує Entity Framework Core як ORM-рішення для спрощення роботи з Microsoft SQL Server. Включає реалізації репозиторіїв, DbContext для налаштування з'єднання з базою даних, а також міграції для управління схемою бази даних.

Application Layer (Шар додатку) містить бізнес-логіку додатку та координує взаємодію між різними компонентами системи. В цьому шарі реалізовані сервіси, які інкапсулюють бізнес-правила та забезпечують виконання конкретних операцій. Також тут знаходяться DTO (Data Transfer Objects), які використовуються для передачі даних між шарами та забезпечують розв'язку між внутрішніми моделями і зовнішніми інтерфейсами.

Presentation Layer (Шар представлення) забезпечує взаємодію з зовнішнім світом і включає ASP.NET Core Web API для серверної частини. Цей шар містить контролери, які обробляють HTTP-запити, middleware для обробки різних аспектів запитів (аутентифікація, обробка помилок), а також конфігурацію системи безпеки з JWT-токенами.

Система також включає клієнтську частину, побудовану на React з TypeScript, яка взаємодіє з серверною частиною через REST API. Клієнт організований за компонентним підходом і використовує Context API для управління глобальним станом додатку.

Міжшарова взаємодія здійснюється через чітко визначені інтерфейси, що дозволяє зберігати принцип інверсії залежностей (Dependency Inversion Principle) з SOLID. Кожен шар залежить тільки від абстракцій (інтерфейсів), а не від конкретних реалізацій нижчих шарів.

Така архітектура забезпечує низьку зв'язність між компонентами системи, що полегшує тестування окремих модулів, дозволяє легко змінювати реалізацію окремих шарів без впливу на інші частини системи та сприяє подальшому масштабуванню та розвитку додатку.

2.3 Вибір технологій для реалізації

Правильний вибір технологій є одним з ключових етапів у процесі розробки програмного забезпечення. Він впливає на продуктивність, масштабованість, безпеку, зручність підтримки та майбутній розвиток системи. У процесі створення веб-додатку для ведення персонального архіву медичних даних було обрано сучасний та перевірений стек технологій, що поєднує у собі надійність серверної частини, гнучкість клієнтського інтерфейсу та ефективну роботу з базою даних.

Для розробки бекенд-частини системи використано ASP.NET Core Web API, що є сучасною платформою від компанії Microsoft. Вона забезпечує високу продуктивність, підтримку REST-архітектури, вбудовані механізми безпеки та зручну інтеграцію з іншими компонентами екосистеми .NET. ASP.NET Core дозволяє реалізовувати багатошарову архітектуру, що є важливою умовою для масштабованості та підтримки принципів SOLID.

Для зберігання та управління даними обрано Microsoft SQL Server як основну реляційну базу даних. Ця система керування базами даних забезпечує високу стабільність, потужні можливості запитів, підтримку транзакцій, індексування, а також тісну інтеграцію з Entity Framework Core — ORM, яка використовується в проєкті. Entity Framework Core дозволяє працювати з базою даних через об’єктно-орієнтовану модель, що спрощує доступ до даних і знижує кількість ручного коду SQL.

У ролі фреймворку для автоматичного мапінгу об'єктів застосовується AutoMapper — популярний інструмент, який дозволяє легко перетворювати моделі домену у DTO-об'єкти та навпаки. Це підвищує чистоту архітектури та зменшує дублювання коду.

Для документування API використовується Swagger/OpenAPI, який автоматично генерує інтерактивну документацію до всіх доступних кінцевих точок. Це суттєво полегшує тестування та взаємодію з API для інших розробників або зовнішніх систем.

Механізм аутентифікації реалізується за допомогою JWT (JSON Web Tokens) — сучасного та широко поширеного способу авторизації, який дозволяє реалізувати безпечний безстанний доступ користувачів до ресурсів системи.

Клієнтську частину веб-додатку реалізовано з використанням React — однієї з найпопулярніших бібліотек для створення сучасних інтерфейсів. Завдяки використанню TypeScript, клієнтський код стає більш структурованим, безпечним до помилок та зручним для масштабування. Для маршрутизації всередині додатку використовується React Router DOM, а для роботи з API — Axios як HTTP-клієнт.

Для управління станом додатку на фронтенді використовується Context API, що дозволяє централізовано зберігати інформацію про поточного користувача, його роль, стан авторизації та інші глобальні дані. Стилизація інтерфейсу реалізується з використанням CSS/SCSS, що забезпечує гнучкість в оформленні зовнішнього вигляду системи.

Таким чином, обрані технології дозволяють створити надійну, масштабовану та зручну у використанні систему з розподіленою архітектурою. Комбінація ASP.NET Core, SQL Server та React забезпечує ефективну реалізацію як серверної, так і клієнтської частини додатку, відповідаючи всім функціональним та нефункціональним вимогам до проєкту.

2.4 Багатошарова архітектура (N-Layer Architecture)

Багатошарова архітектура (N-layer architecture) є основоюструктурної організації системи "Персональний архів медичних даних". Цей архітектурний підхід забезпечує четке розділення відповідальностей, високу модульність та легкість супроводу системи. Кожен шар виконує специфічні функції і має чітко визначені інтерфейси для взаємодії з іншими шарами.

Система організована у вигляді ієрархічної структури, де кожен верхній шар може залежати від нижчих шарів, але не навпаки. Така організація дозволяє досягти принципу інверсії залежностей та забезпечує можливість незалежного тестування кожного компонента.

2.4.1 Domain Layer

Domain Layer є основою системи і містить усю бізнес-логіку та доменні моделі. Цей шар не залежить від інших шарів і представляє чисту бізнес-логіку без прив'язки до конкретних технологій.

На рисунку 2.2 зображена діаграма структури Domain Layer.

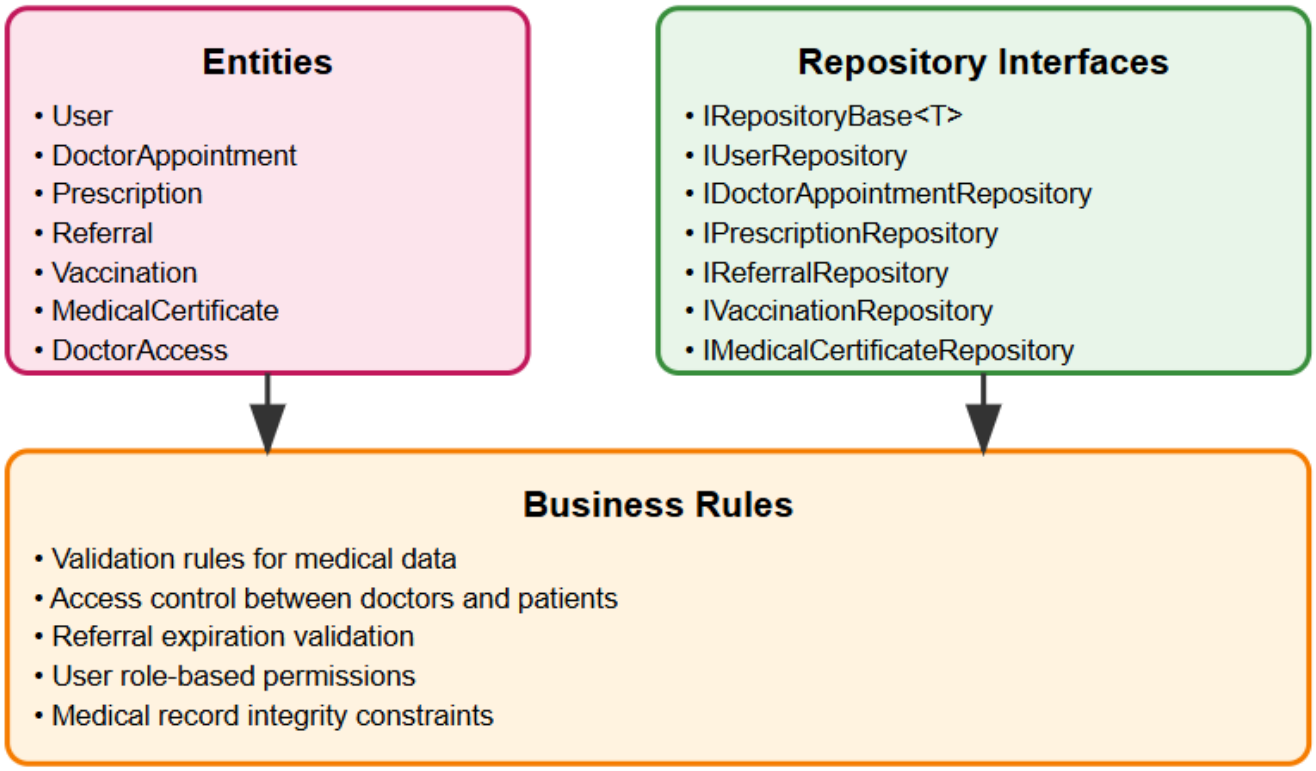


Рис. 2.2 Діаграма структури Domain Layer

Доменний шар включає наступні компоненти:

1. Entities (Сутності) - це основні об'єкти предметної області, які представляють реальні бізнес-концепції. В системі визначені такі сутності:

- User - представляє користувача системи (пацієнта або лікаря)
- DoctorAppointment - прийом у лікаря з усіма деталями
- Prescription - рецепт на ліки
- Referral - медичне направлення
- Vaccination - інформація про щеплення
- MedicalCertificate - медичні довідки
- DoctorAccess - доступ лікаря до даних пацієнта

2. Repository Interfaces - інтерфейси, які визначають контракти для доступу до даних. Кожна сутність має відповідний інтерфейс репозиторію, який описує операції CRUD та специфічні методи пошуку. Це забезпечує принцип інверсії залежностей, оскільки доменний шар визначає що потрібно, але не як це реалізовано.

3. Business Rules - бізнес-правила та валідація, які інкапсульовані в доменних об'єктах. Наприклад, правила перевірки термінів дії направлень, валідація медичних даних, контроль доступу між лікарями та пацієнтами.

2.4.2 Data Access Layer

Data Access Layer відповідає за всі операції, пов'язані з персистентністю даних та взаємодією з базою даних. Цей шар реалізує інтерфейси, визначені в доменному шарі. На рисунку 2.3 зображена діаграма структури Data Access Layer.

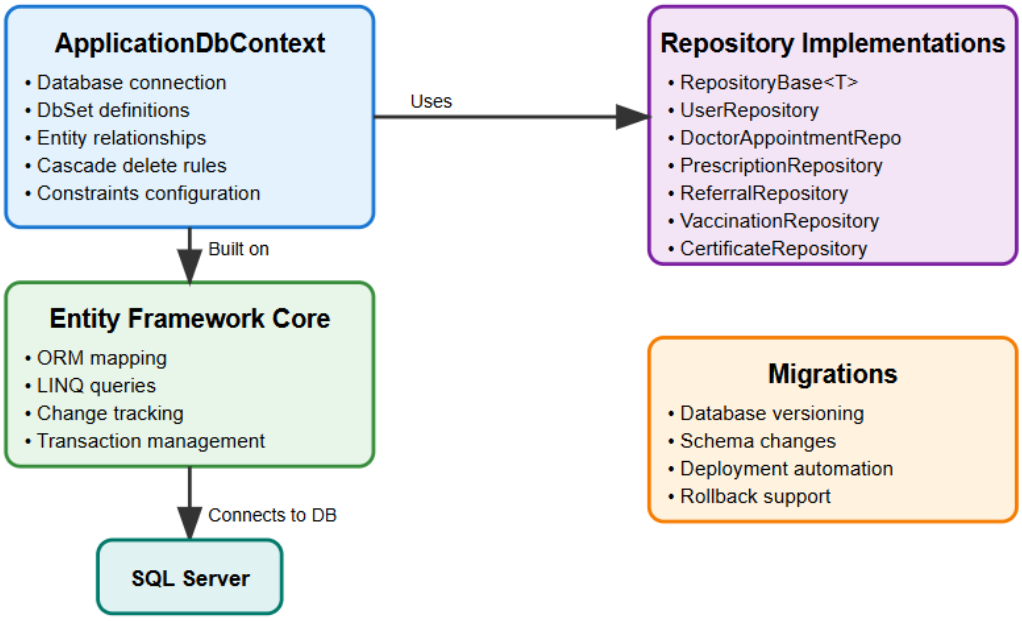


Рис. 2.3 Структура Data Access Layer

Основні компоненти цього шару:

1. ApplicationDbContext - основний контекст Entity Framework Core, який налаштовує з'єднання з базою даних, визначає DbSet для кожної сутності та налаштовує зв'язки між таблицями. Також включає конфігурацію каскадного видалення та обмежень цілісності.

2. Repository Implementations - конкретні реалізації інтерфейсів репозиторіїв з доменного шару. Кожен репозиторій наслідується від базового класу RepositoryBase та реалізує специфічні методи для своєї сутності. Всі репозиторії використовують Entity Framework Core для виконання операцій з базою даних.

3. Migrations - файли міграцій Entity Framework, які описують зміни схеми бази даних та дозволяють версіонувати структуру бази. Міграції забезпечують можливість оновлення схеми бази при розгортанні нових версій системи.

2.4.3 Application Layer

Application Layer містить бізнес-логіку додатку та координує взаємодію між різними компонентами системи. Цей шар оріеструє виконання бізнес-процесів та забезпечує трансформацію даних.

На рисунку 2.4 зображена діаграма структури Application Layer.

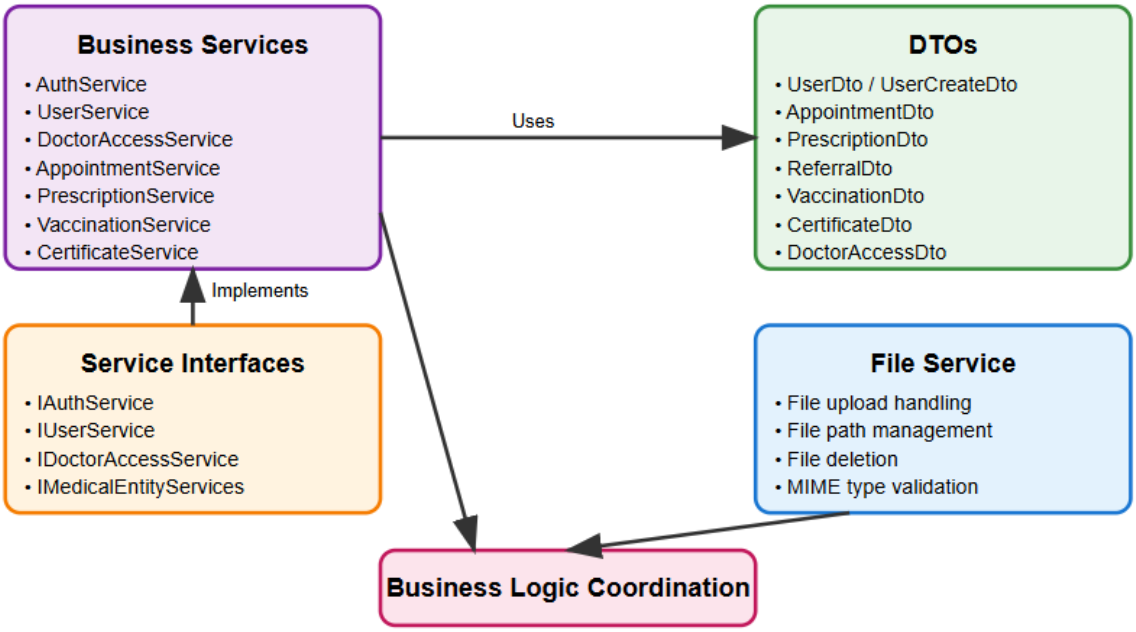


Рис. 2.4 Структура Application Layer

Ключові елементи цього шару:

1. Business Services - сервіси, які реалізують бізнес-логіку додатку. Кожна доменна сутність має відповідний сервіс, який інкапсулює операції над цією сутністю. Сервіси використовують репозиторії для доступу до даних та виконують валідацію, трансформацію та бізнес-правила.

2. DTOs (Data Transfer Objects) - об'єкти для передачі даних між шарами. DTOs забезпечують розв'язку між внутрішніми доменними моделями та зовнішніми інтерфейсами. Вони включають CreateDto для створення нових записів, UpdateDto для оновлення та звичайні Dto для читання.

3. Service Interfaces - інтерфейси, які визначають контракти для бізнес-сервісів. Це дозволяє легко підміняти реалізації сервісів для тестування або зміни бізнес-логіки.

4. File Service - спеціальний сервіс для роботи з файлами (медичними документами). Забезпечує збереження, видалення та отримання файлів з файлової системи.

2.4.4 Presentation Layer

Presentation Layer забезпечує взаємодію з зовнішніми клієнтами через REST API. Цей шар отримує HTTP-запити, обробляє їх та повертає відповіді у форматі JSON.

Компоненти шару представлення:

1. Controllers - класи, які обробляють HTTP-запити та маршрутизують їх до відповідних сервісів. Кожен контролер відповідає за конкретну доменну область та забезпечує всі CRUD операції. Контролери також відповідають за валідацію вхідних даних та трансформацію результатів.

2. Authentication & Authorization - система аутентифікації на базі JWT-токенів та role-based авторизація. Забезпечує безпечний доступ до API endpoints та контролює права доступу користувачів.

3. Middleware - компоненти, які обробляють запити та відповіді на різних етапах їх життєвого циклу. Включає middleware для обробки помилок, логування, CORS та аутентифікації.

4. API Documentation - автоматична генерація документації API за допомогою Swagger/OpenAPI, що спрощує інтеграцію та тестування.

На рисунку 2.5 зображена діаграма структури Presentation Layer.

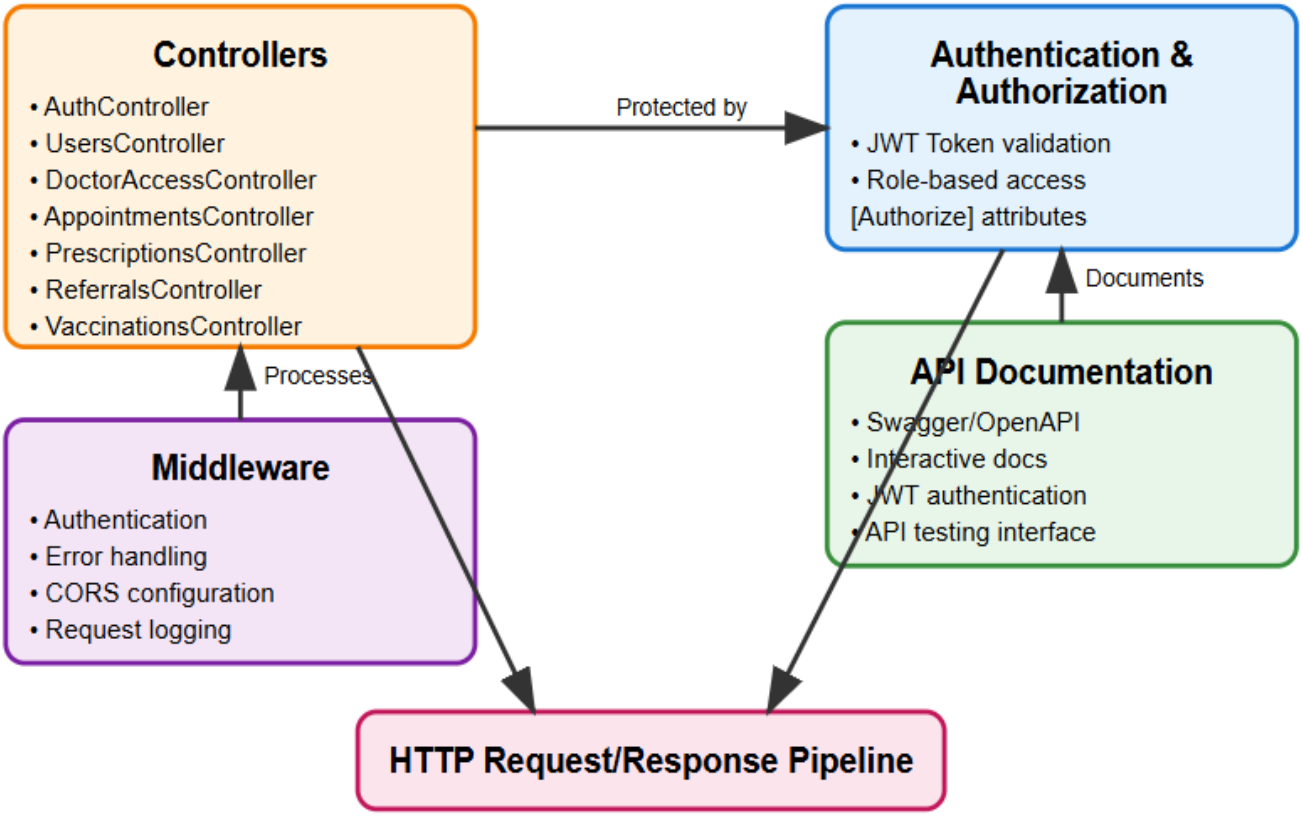


Рис. 2.5 Структура Presentation Layer

2.4.5 Client Layer

Client Layer представляє фронтенд додатку, побудований на React з TypeScript. Цей шар забезпечує користувацький інтерфейс та взаємодіє з серверною частиною через REST API.

Організація клієнтського шару:

1. Components - React компоненти, організовані за функціональними областями. Кожна медична сутність має свій набір компонентів для списків, деталей, форм створення/редагування.

2. Pages - сторінки додатку, які об'єднують кілька компонентів для створення повноцінних екранів інтерфейсу.

3. API Services - модулі для взаємодії з backend API, організовані за доменними областями. Використовують Axios для виконання HTTP-запитів.

4. State Management - управління станом додатку за допомогою React Context API. Включає контекст аутентифікації та локальний стан компонентів.

5. Routing - клієнтська маршрутизація за допомогою React Router, включаючи захищені роути та авторизацію за ролями.

Така організація багатошарової архітектури забезпечує чітке розділення відповідальностей, легкість тестування, можливість незалежного розвитку різних частин системи та високу гнучкість при внесенні змін.

На рисунку 2.6 зображена діаграма структури Client Layer.

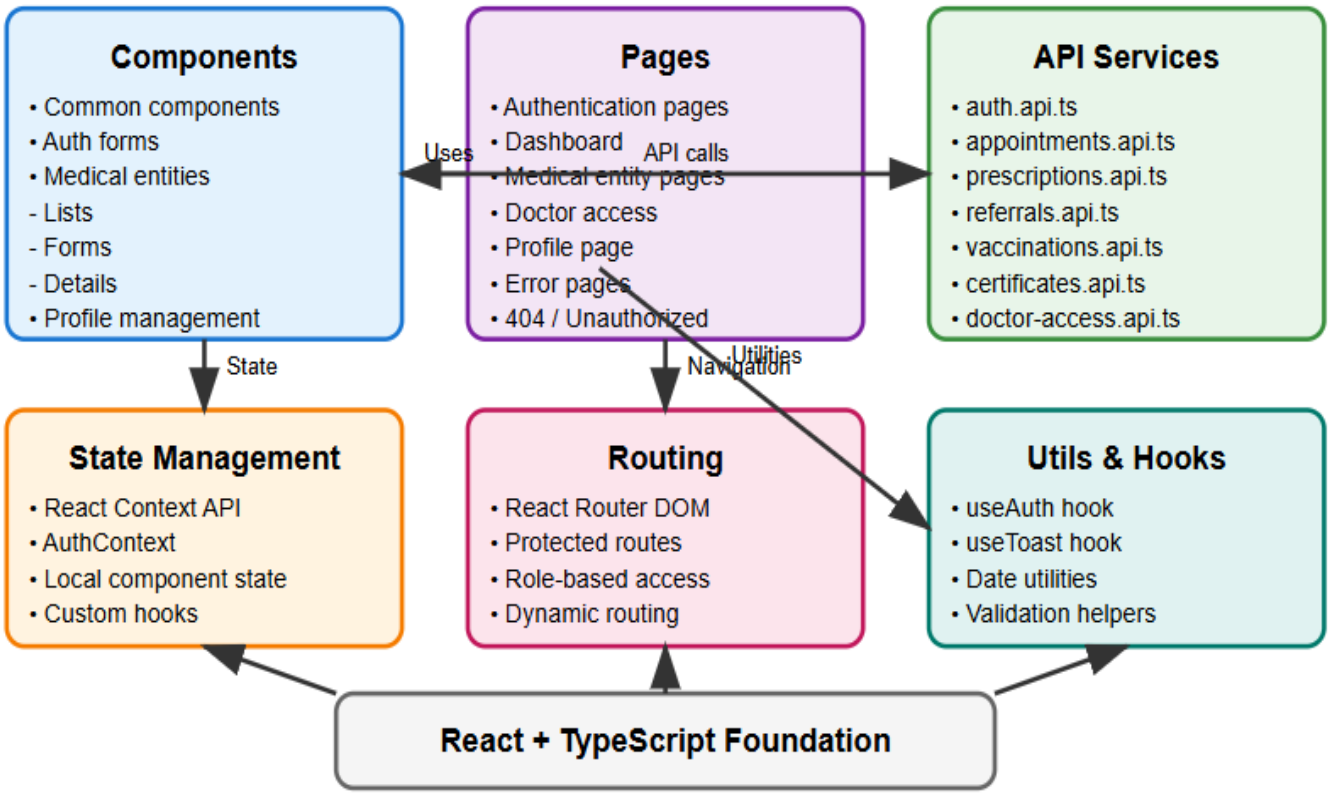


Рис. 2.6 структури Client Layer

2.5 Проєктування бази даних

Проєктування бази даних для системи "Персональний архів медичних даних" базується на реляційній моделі з використанням Microsoft SQL Server. Структура бази даних спроєктована з урахуванням специфіки медичних даних, забезпечення цілісності інформації та ефективної роботи з великими обсягами даних.

На рисунку 2.7 зображена діаграма схеми бази даних системи.

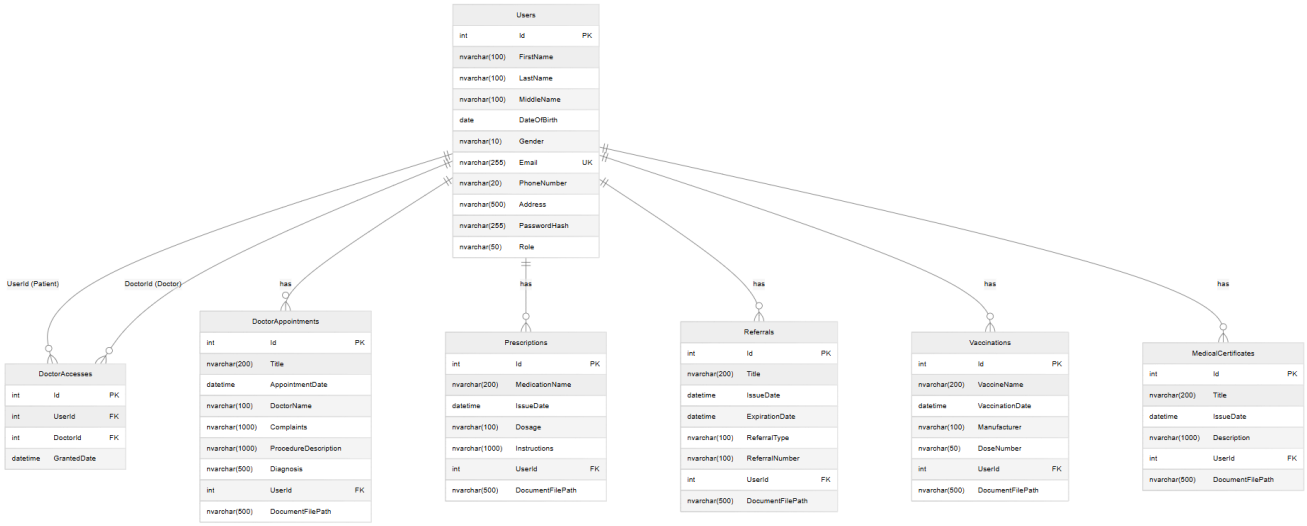


Рис. 2.7 Схеми бази даних системи

Основу структури бази даних складають вісім основних таблиць, які забезпечують зберігання всіх необхідних медичних та користувацьких даних. Центральною таблицею є Users, яка містить інформацію про всіх користувачів системи - як пацієнтів, так і лікарів. Таблиця включає поля для зберігання персональних даних (ПІБ, дата народження, стать, контакти), облікових даних (email, хеш паролю) та роль користувача.

Таблиця DoctorAccesses реалізує механізм контролю доступу лікарів до даних пацієнтів. Вона створює зв'язок many-to-many між користувачами в ролі пацієнта та лікаря, зберігаючи дату надання доступу. Це дозволяє пацієнту контролювати, які лікарі мають доступ до його медичних даних, та при необхідності відкликати цей доступ.

П'ять таблиць медичних сутностей - DoctorAppointments, Prescriptions, Referrals, Vaccinations та MedicalCertificates - зберігають різні типи медичних

записів. Кожна з цих таблиць має зв'язок one-to-many з таблицею Users через поле UserId, що забезпечує належність кожного запису конкретному користувачу.

Таблиця DoctorAppointments містить інформацію про прийоми у лікаря, включаючи назву прийому, дату, ім'я лікаря, скарги пацієнта, опис процедур, діагноз та шлях до прикріпленого документа. Структура таблиці дозволяє зберігати повну інформацію про кожен візит до лікаря.

Prescriptions зберігає дані про рецепти, включаючи назву препарату, дату виписки, дозування, інструкції по застосуванню та посилання на скановані документи. Така структура забезпечує повноту інформації про призначені ліки.

Таблиця Referrals призначена для зберігання медичних направлень. Вона включає назву направлення, дати видачі та закінчення терміну дії, тип направлення, унікальний номер та шлях до документа. Поле з терміном дії дозволяє відстежувати актуальність направлень.

Vaccinations містить інформацію про щеплення: назву вакцини, дату вакцинації, виробника, номер дози та прикріплені документи. Це забезпечує повний облік історії вакцинації користувача.

Таблиця MedicalCertificates зберігає різні медичні довідки з полями для назви, дати видачі, опису та прикріпленого документа. Гнучка структура дозволяє зберігати різноманітні типи довідок.

Всі таблиці медичних записів мають схожу структуру з обов'язковими полями UserId для зв'язку з користувачем та DocumentFilePath для зберігання шляху до прикріпленого файла. Це забезпечує консистентність структури та спрощує роботу з даними на рівні додатку.

Індексація бази даних оптимізована для частих запитів: створені індекси на UserId у всіх таблицях медичних записів, на Email у таблиці Users для швидкого пошуку користувачів, та композитні індекси на UserId та DoctorId у таблиці DoctorAccesses для ефективної перевірки доступу.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація серверної частини

Серверна частина системи реалізована з використанням ASP.NET Core 8.0 та архітектурного підходу "чиста архітектура" (Clean Architecture) з чітким розділенням на шари. Реалізація забезпечує високу продуктивність, масштабованість та підтримуваність коду за рахунок дотримання принципів SOLID та використання сучасних технологій.

3.1.1 Реалізація Domain Layer

Domain Layer містить основні доменні моделі та бізнес-логіку системи. На рисунку 3.1 та 3.2 зображені діаграми класів Domain Layer.

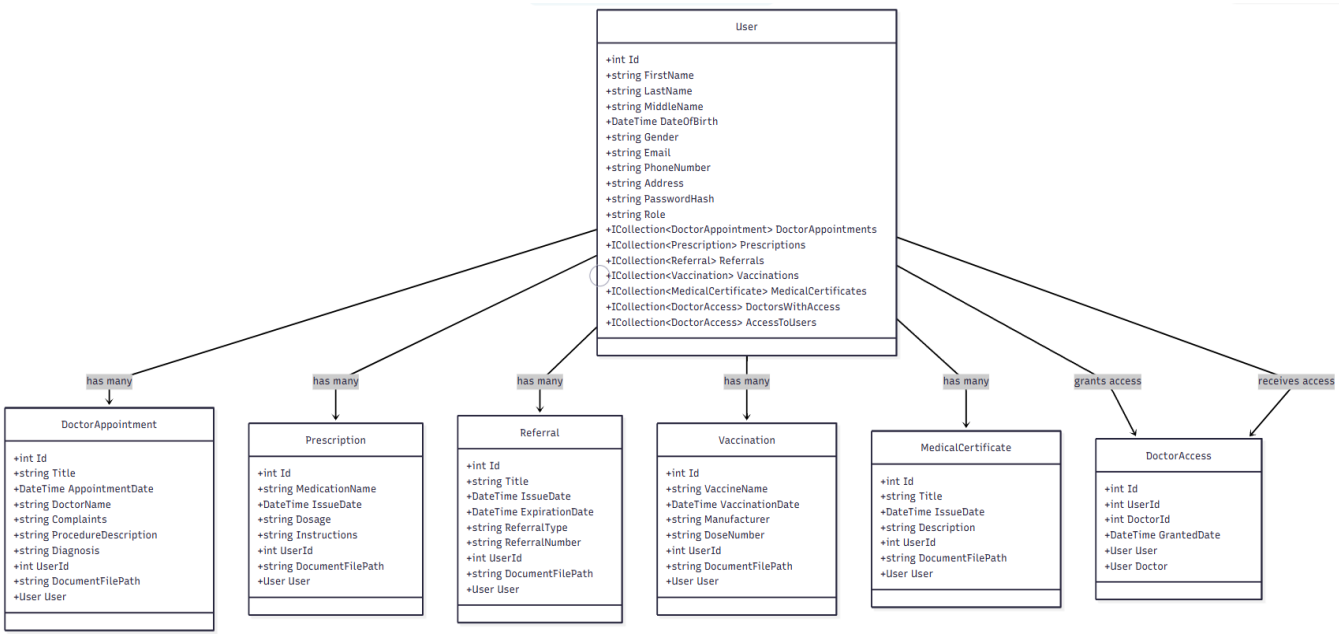


Рис. 3.1 Діаграма класів Domain Layer

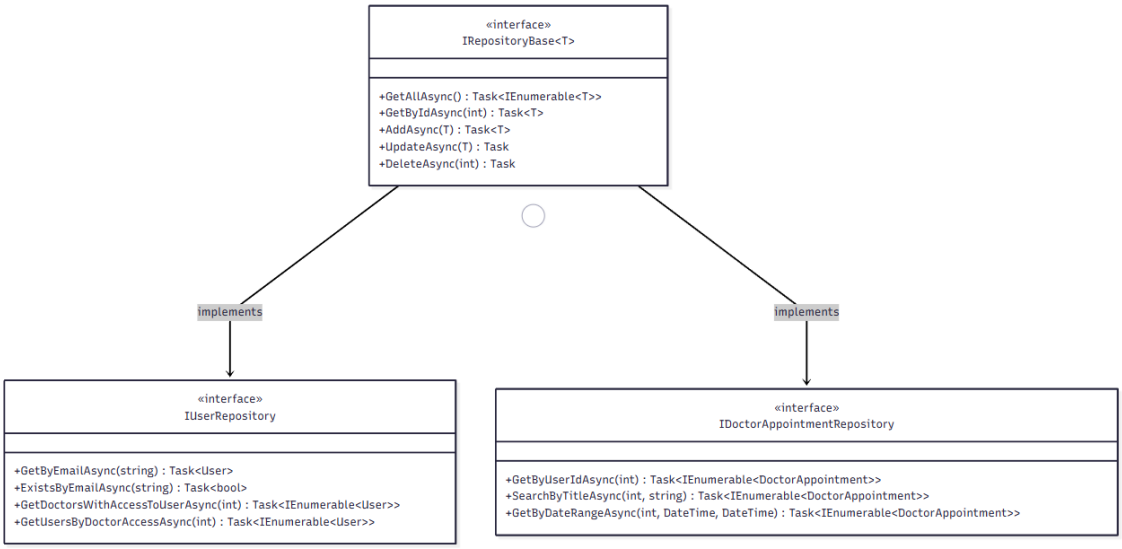


Рис. 3.2 Діаграма класів Domain Layer

Центральним елементом доменного шару є клас User, який представляє користувача системи. Клас містить властивості для зберігання персональних даних, облікової інформації та навігаційні властивості для зв'язку з медичними записами. Поле Role визначає тип користувача (User або Doctor), що дозволяє реалізувати role-based авторизацію.

Медичні сутності - DoctorAppointment, Prescription, Referral, Vaccination та MedicalCertificate - наслідують спільну структуру з обов'язковими полями UserId та DocumentFilePath. Кожна сутність містує специфічні поля, що відповідають її призначенню.

Клас DoctorAccess реалізує механізм контролю доступу між лікарями та пацієнтами. Він містить зовнішні ключі на двох користувачів та дату надання доступу, забезпечуючи гнучке управління правами доступу.

Процес автентифікації починається з отримання облікових даних (email та пароль) від клієнта. Система виконує валідацію формату email, перевіряє існування користувача в базі даних та порівнює хеш пароля. У разі успішної автентифікації генерується JWT-токен з необхідними claims, який повертається клієнту для подальшого використання в запитах до API.

На рисунку 3.3 зображена блок-схема процесу автентифікації користувача.

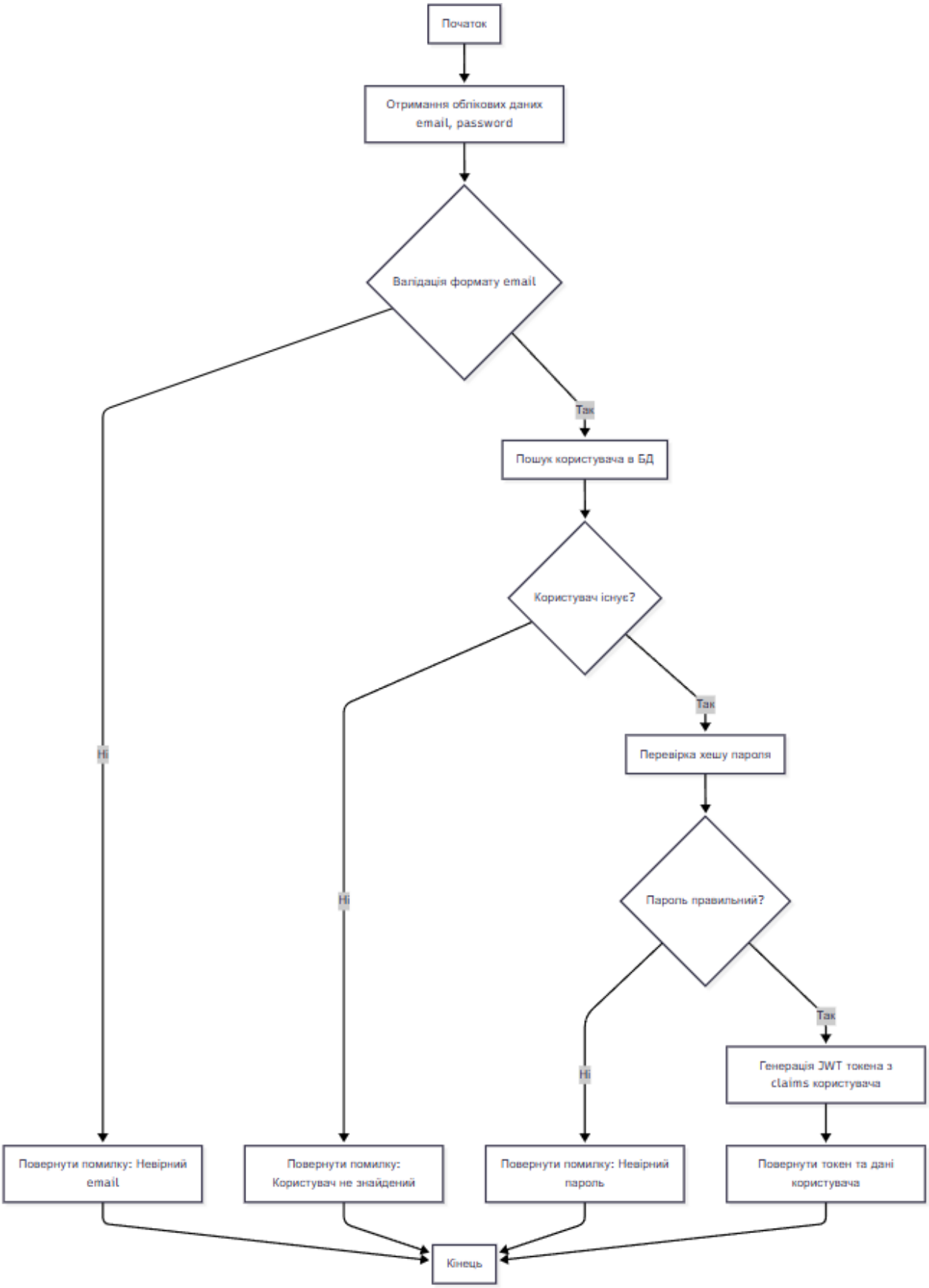


Рис. 3.3 Блок-схема процесу автентифікації користувача

3.1.2 Реалізація Data Access Layer

Data Access Layer забезпечує абстракцію доступу до даних та реалізує патерн Repository. На рисунку 3.4 зображена діаграма класів Data Access Layer.

ApplicationDbContext є основним контекстом Entity Framework Core, який налаштовує підключення до бази даних та визначає DbSet для кожної сутності. В методі OnModelCreating налаштовуються зв'язки між таблицями, каскадне видалення та обмеження цілісності.

Базовий клас RepositoryBase<T> реалізує інтерфейс IRepositoryBase<T> та надає загальні CRUD операції для всіх сутностей. Спеціалізовані репозиторії наслідують від RepositoryBase та додають специфічні методи для роботи з конкретними сутностями.

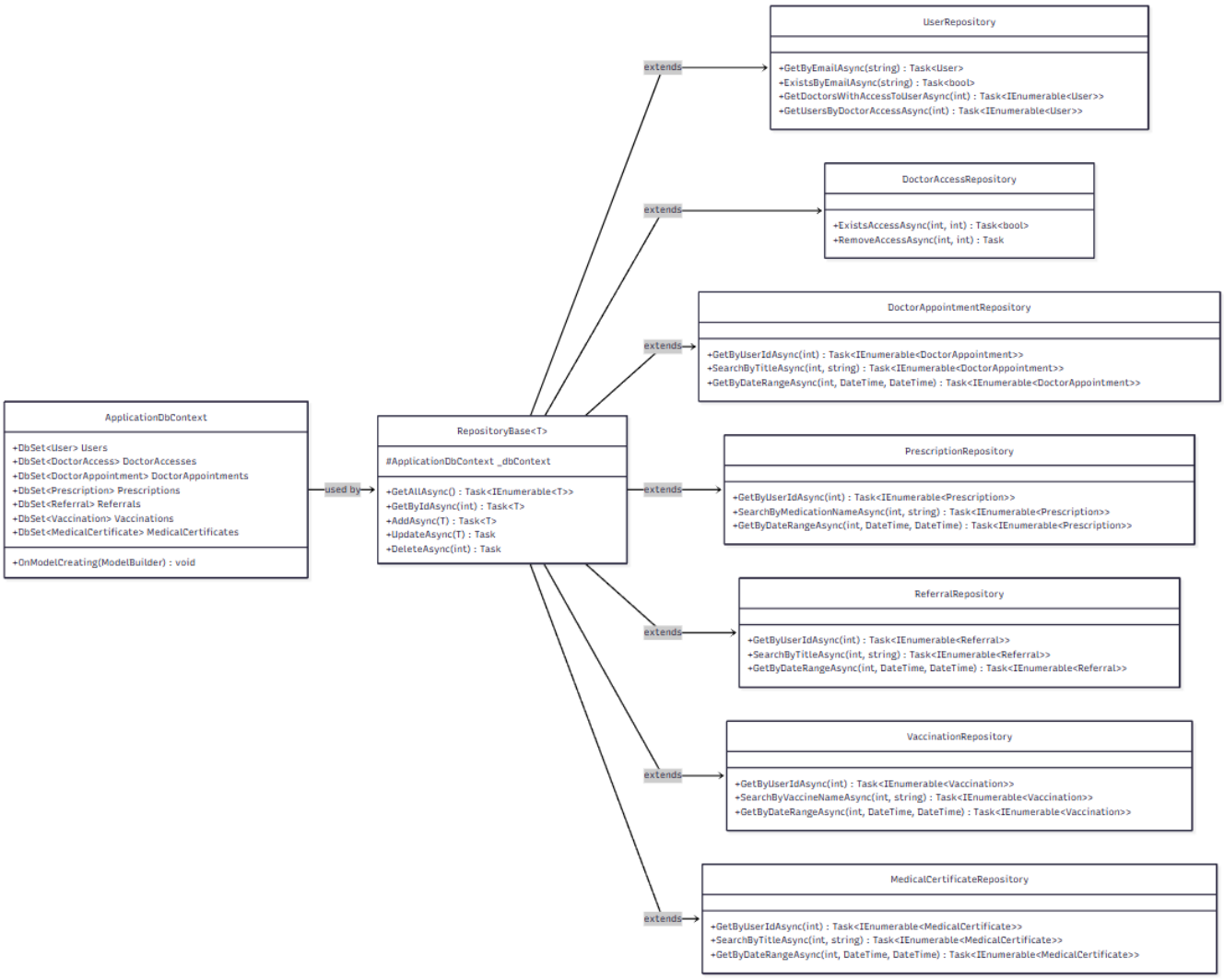


Рис. 3.4 Діаграма класів Data Access Layer

Процес створення починається з валідації вхідних даних DTO. Якщо до запису прикріплено файл, він зберігається в файловій системі з унікальним ім'ям. Створюється об'єкт доменної моделі, який зберігається в базі даних через репозиторій. Результат повертається у вигляді DTO.

На рисунку 3.5 зображена блок-схема процесу створення медичного запису.

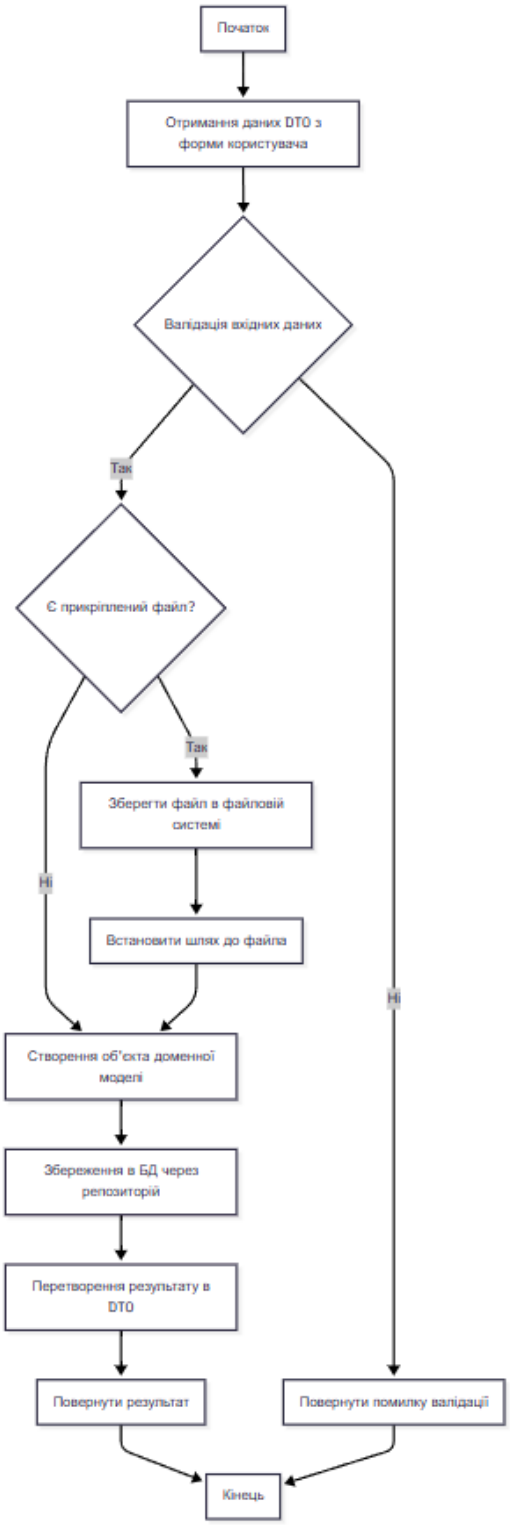


Рис. 3.5 Блок-схема процесу створення медичного запису

3.1.3 Реалізація Application Layer

Application Layer містить бізнес-логіку та координує взаємодію між різними компонентами системи. На рисунку 3.6 зображена діаграма класів Application Layer.

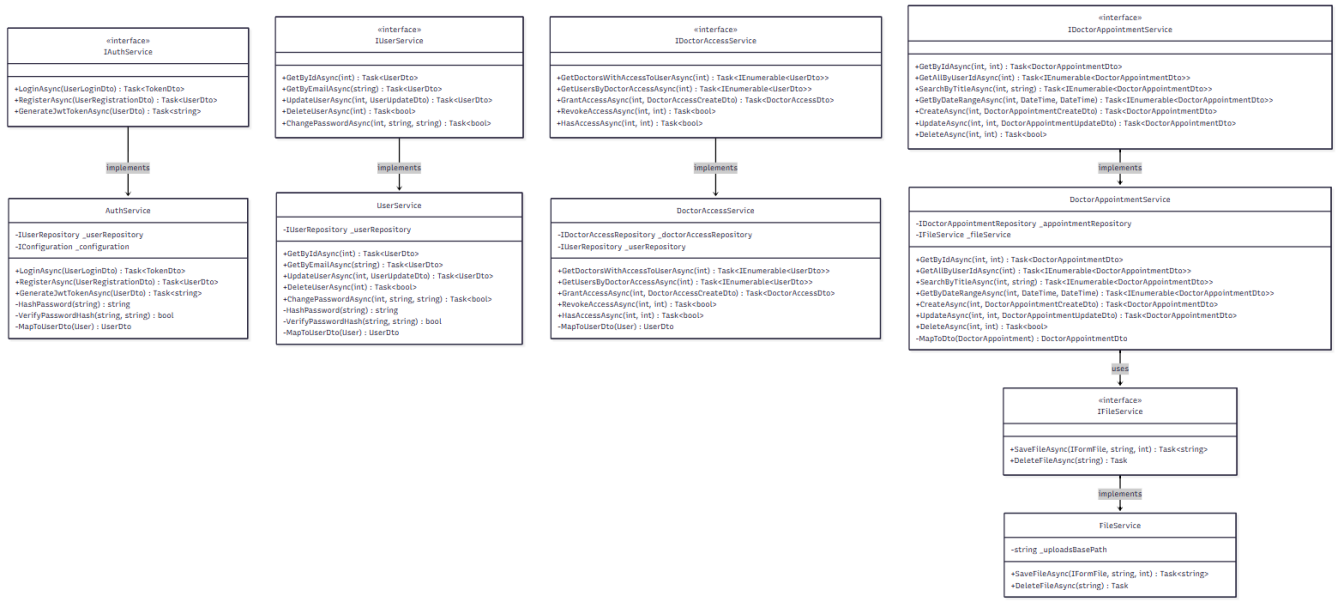


Рис. 3.6 Діаграма класів Application Layer

AuthService реалізує логіку автентифікації та авторизації, включаючи генерацію JWT-токенів, хешування паролів та валідацію облікових даних. Сервіс використовує IUserRepository для роботи з даними користувачів.

Сервіси для медичних сутностей (DoctorAppointmentService, PrescriptionService тощо) реалізують однотипну логіку CRUD операцій із специфічною валідацією для кожного типу даних. Всі сервіси використовують AutoMapper для перетворення між доменними моделями та DTO.

FileService забезпечує роботу з файлами медичних документів, включаючи збереження, видалення та генерацію унікальних шляхів. Файли організовуються у структуру директорій за userId та типом сутності.

На рисунку 3.7 зображена блок-схема процесу надання доступу лікарю.

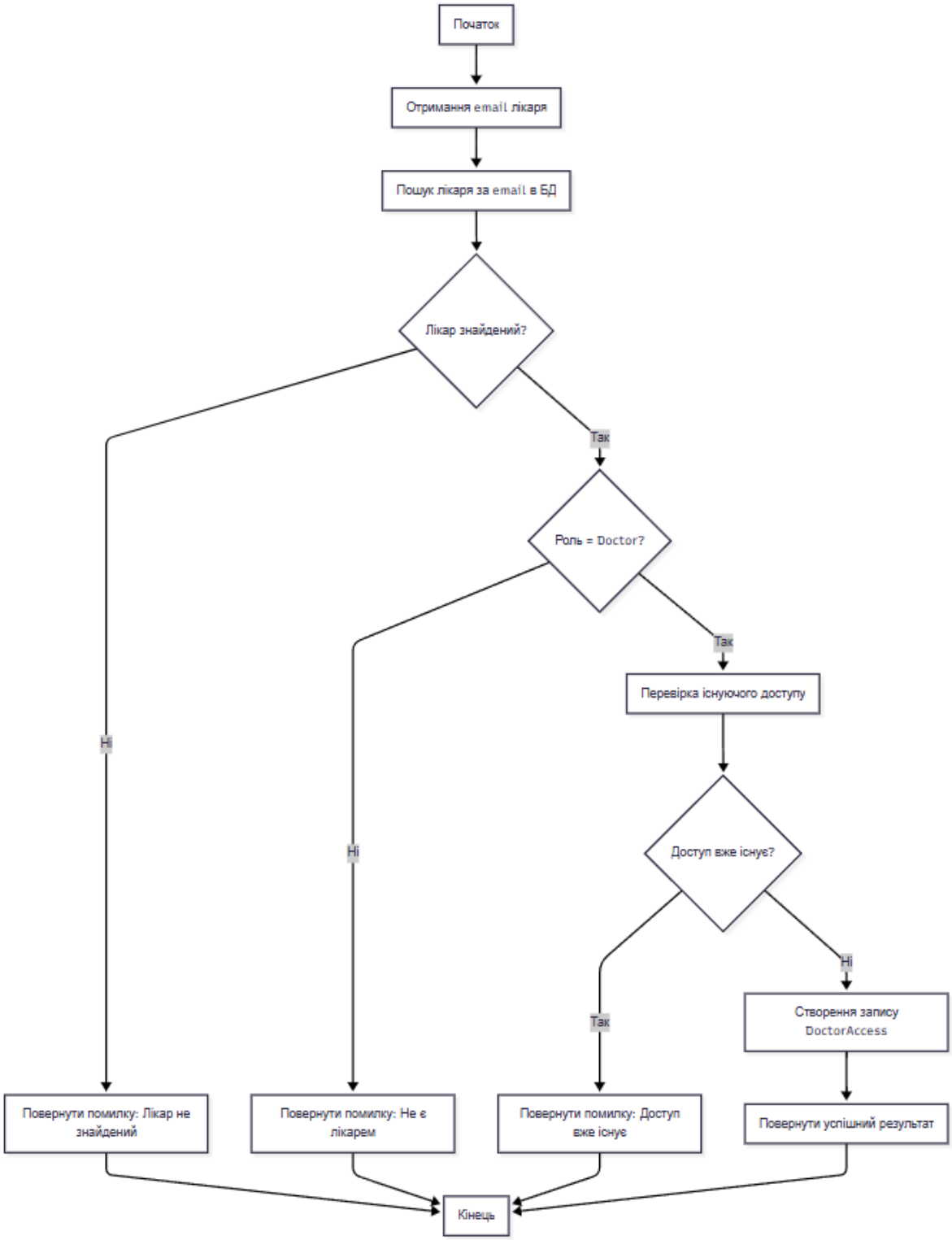


Рис. 3.7 Блок-схема процесу надання доступу лікарю

Процес надання доступу включає пошук лікаря за email, валідацію його ролі, перевірку наявності існуючого доступу та створення нового запису DoctorAccess. Система забезпечує, що доступ може бути наданий тільки користувачам з роллю Doctor.

3.1.4 Реалізація Presentation Layer

Presentation Layer реалізує REST API для взаємодії з клієнтськими додатками.

Контролери наслідують від ControllerBase та використовують атрибути для маршрутизації та авторизації. AuthController не вимагає автентифікації для endpoints реєстрації та входу, тоді як інші контролери захищені атрибутом [Authorize].

Контролери медичних сутностей реалізують стандартний набір CRUD операцій та додаткові методи для пошуку та фільтрації. Для лікарів додаються спеціальні endpoints для доступу до даних пацієнтів з перевіркою прав доступу.

На рисунку 3.8 зображена блок-схема обробки HTTP-запиту.

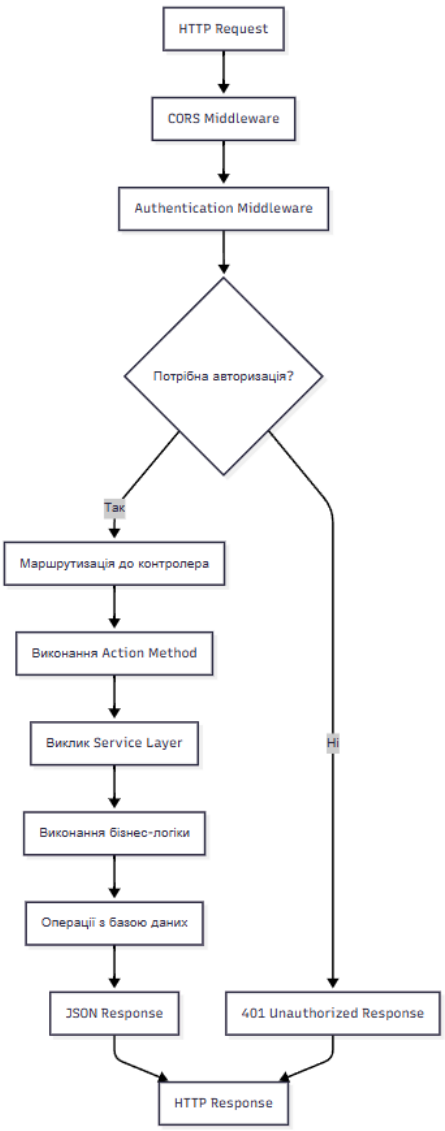


Рис.3.8 Блок-схема обробки HTTP-запиту

Кожен HTTP-запит проходить через middleware pipeline, включаючи CORS, автентифікацію та обробку помилок. Після успішної автентифікації запит маршрутизується до відповідного контролера, який викликає необхідний сервіс та повертає результат у форматі JSON.

3.1.5 Налаштування залежностей та middleware

Конфігурація сервісів та middleware здійснюється в файлі Program.cs. Система використовує вбудований DI-контейнер ASP.NET Core для реєстрації залежностей.

На рисунку 3.9 зображена діаграма реєстрації залежностей.

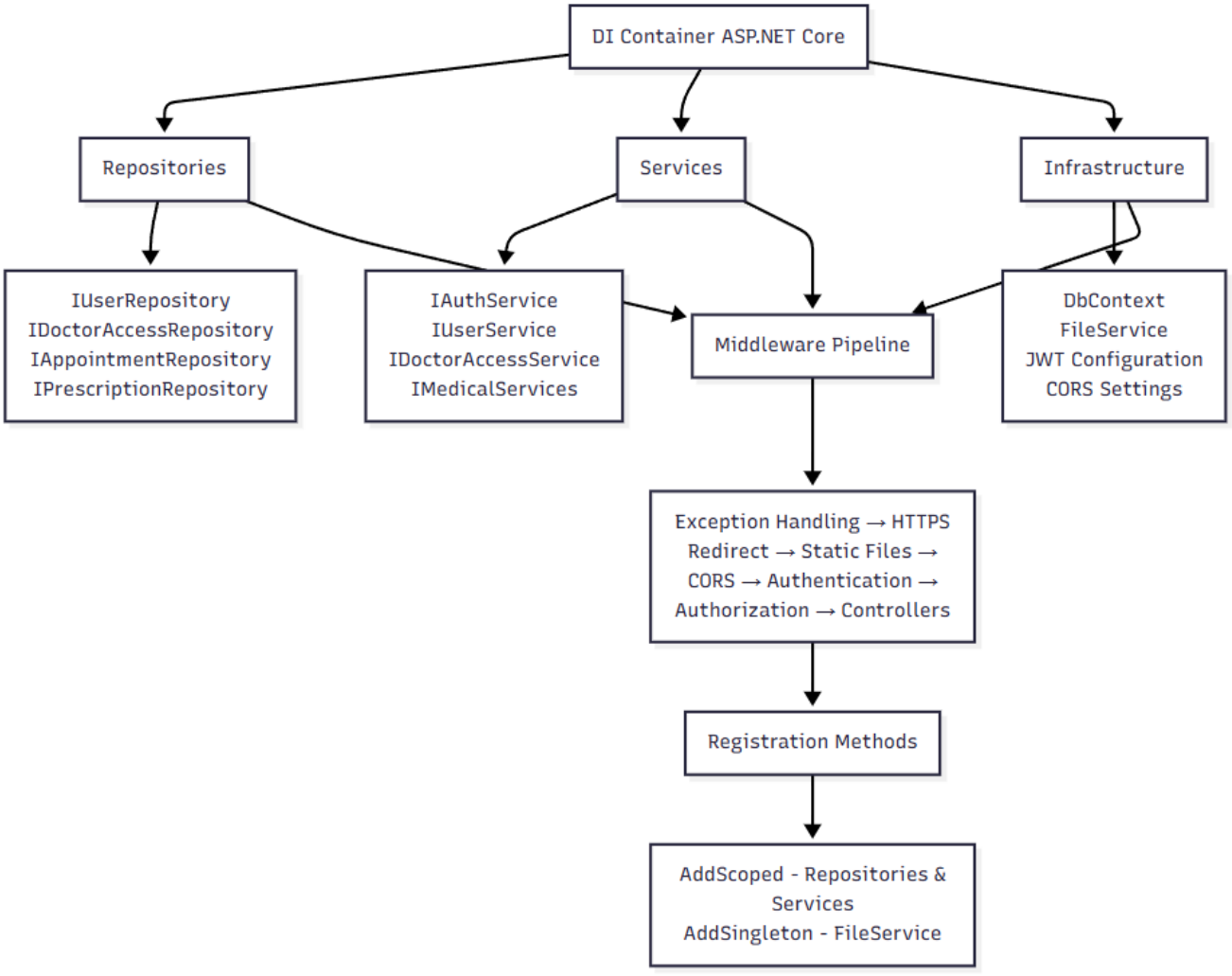


Рис. 3.9 Діаграма реєстрації залежностей

Реєструються всі репозиторії, сервіси, DbContext та конфігурація JWT-автентифікації. CORS налаштовується для дозволу запитів з frontend-додатку. Swagger додається для автоматичної генерації документації API.

Middleware pipeline включає обробку помилок, HTTPS-редирект, статичні файли, CORS, автентифікацію та авторизацію. Така конфігурація забезпечує безпеку та коректну роботу API.

Реалізація серверної частини забезпечує надійну, масштабовану та підтримувану архітектуру, яка може ефективно обробляти запити клієнтів та керувати медичними даними користувачів з дотриманням принципів безпеки та цілісності даних.

3.2 Реалізація клієнтської частини

Клієнтська частина системи реалізована з використанням React 18 та TypeScript, що забезпечує типобезпеку та сучасний підхід до розробки користувацьких інтерфейсів. Додаток організований за компонентним підходом з використанням React Router для навігації та Context API для управління станом.

Система автентифікації є першою точкою взаємодії користувача з додатком. Реалізовано дві основні сторінки для автентифікації: вхід та реєстрація.

На рисунку 3.10 зображена сторінка входу в систему.

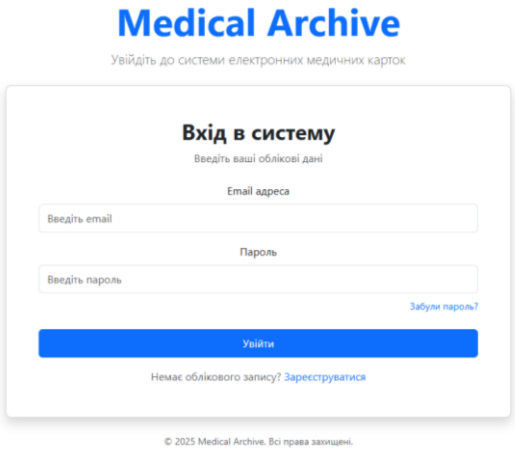


Рис. 3.10 Сторінка входу в систему

Сторінка входу містить форму з полями для введення email та пароля. Інтерфейс має мінімалістичний дизайн з акцентом на зручність використання. Форма включає валідацію полів на клієнтській стороні, яка перевіряє формат email та заповненість полів. При невдалій спробі входу користувач отримує чіткі повідомлення про помилки.

На рисунку 3.11 зображена сторінка реєстрації в системі.

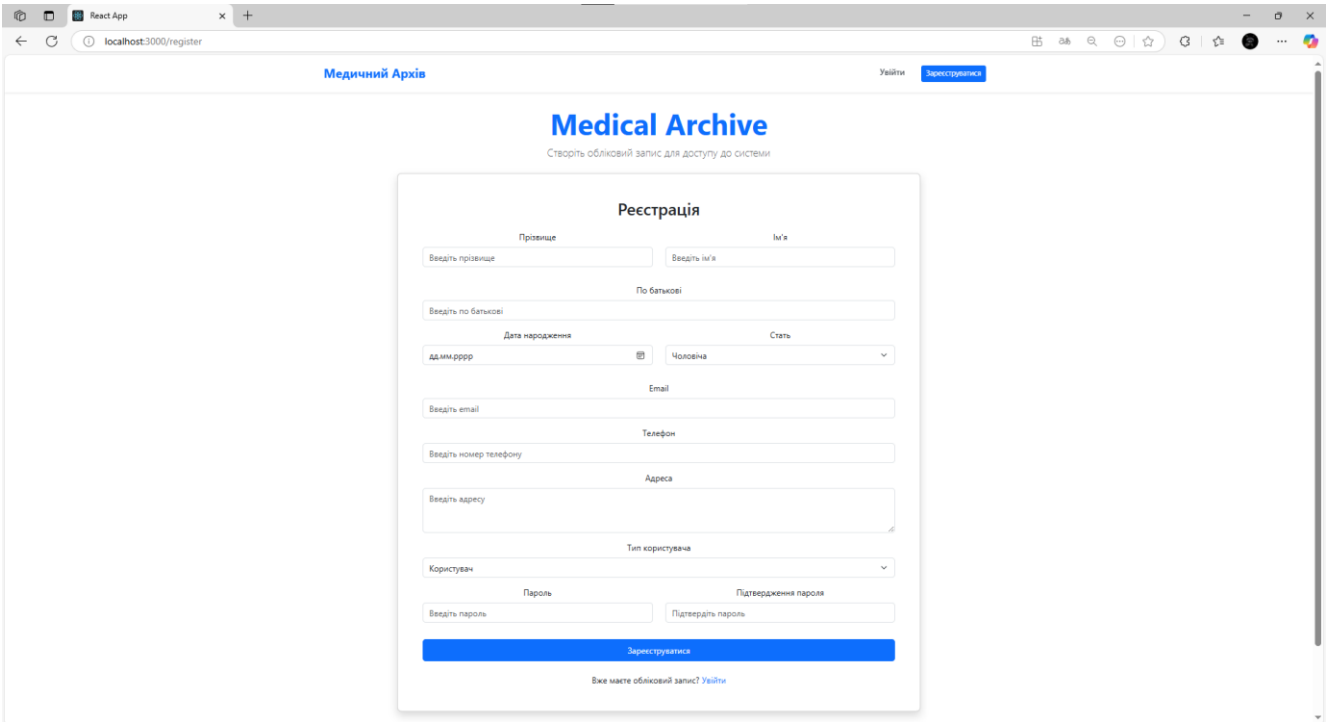


Рис. 3.11 Сторінка реєстрації в системі

Сторінка реєстрації містить розширену форму з усіма необхідними персональними даними: ім'я, прізвище, по батькові, дата народження, стать, контактна інформація та роль користувача (пацієнт або лікар). Форма забезпечує валідацію всіх полів, включаючи перевірку сили пароля та відповідності підтвердження пароля.

Після успішної автентифікації користувач потрапляє на головну сторінку, яка є центральним хабом для навігації по функціям системи.

На рисунку 3.12 зображена головна сторінка системи.

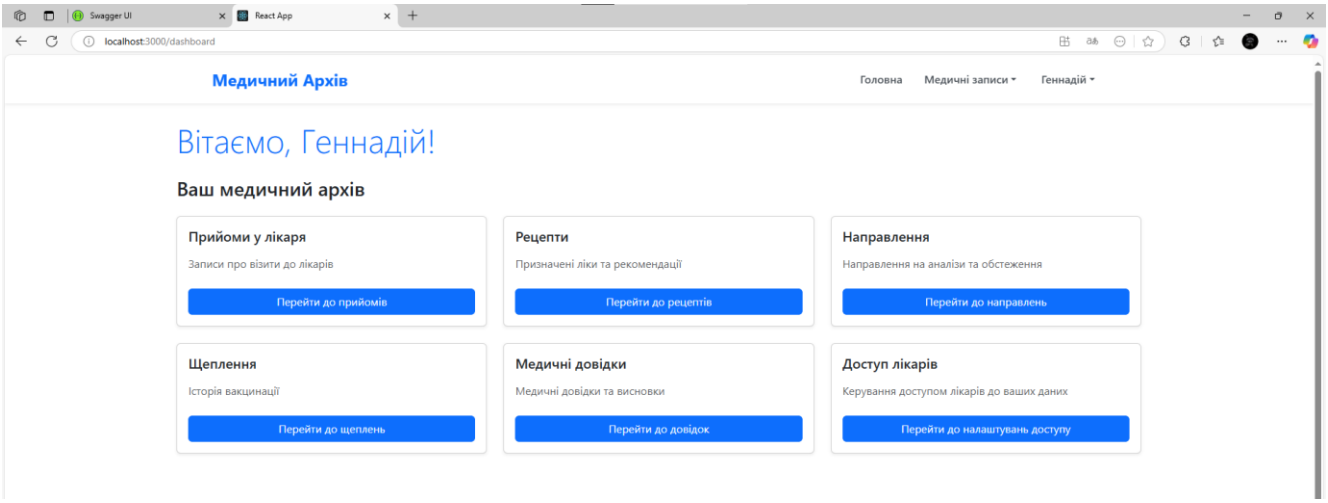


Рис. 3.12 Головна сторінка системи

Dashboard містить огляд основних функцій системи у вигляді карток-посилань, що ведуть до відповідних розділів. Кожна картка має іконку, назву розділу та коротку інформацію про кількість записів. Сторінка також включає швидкий доступ до останніх доданих записів та статистику використання.

Розділ для управління прийомами у лікаря включає список записів, детальний перегляд та форми для створення/редагування.

На рисунку 3.13 зображена сторінка зі списком прийомів у лікаря.

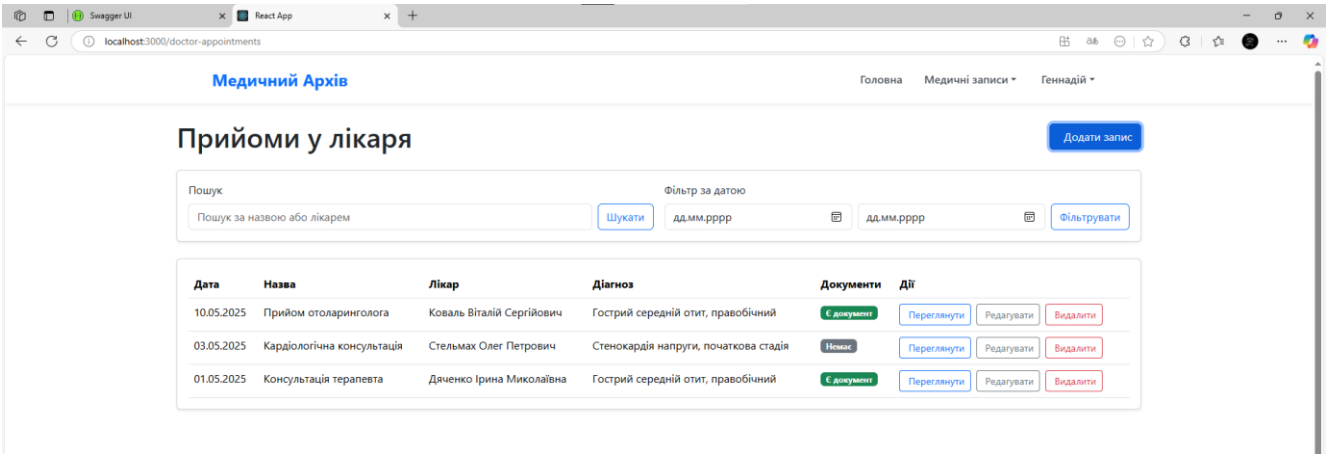


Рис. 3.13 Сторінка зі списком прийомів у лікаря

Сторінка містить таблицю з усіма прийомами користувача, відсортованими за датою. Кожен запис показує основну інформацію: назву прийому, дату, лікаря та діагноз. Доступні функції пошуку за назвою та фільтрації за датою. Користувач може додати новий запис, редагувати існуючі або переглянути деталі.

На рисунку 3.14 зображена форма створення нового прийому у лікаря.

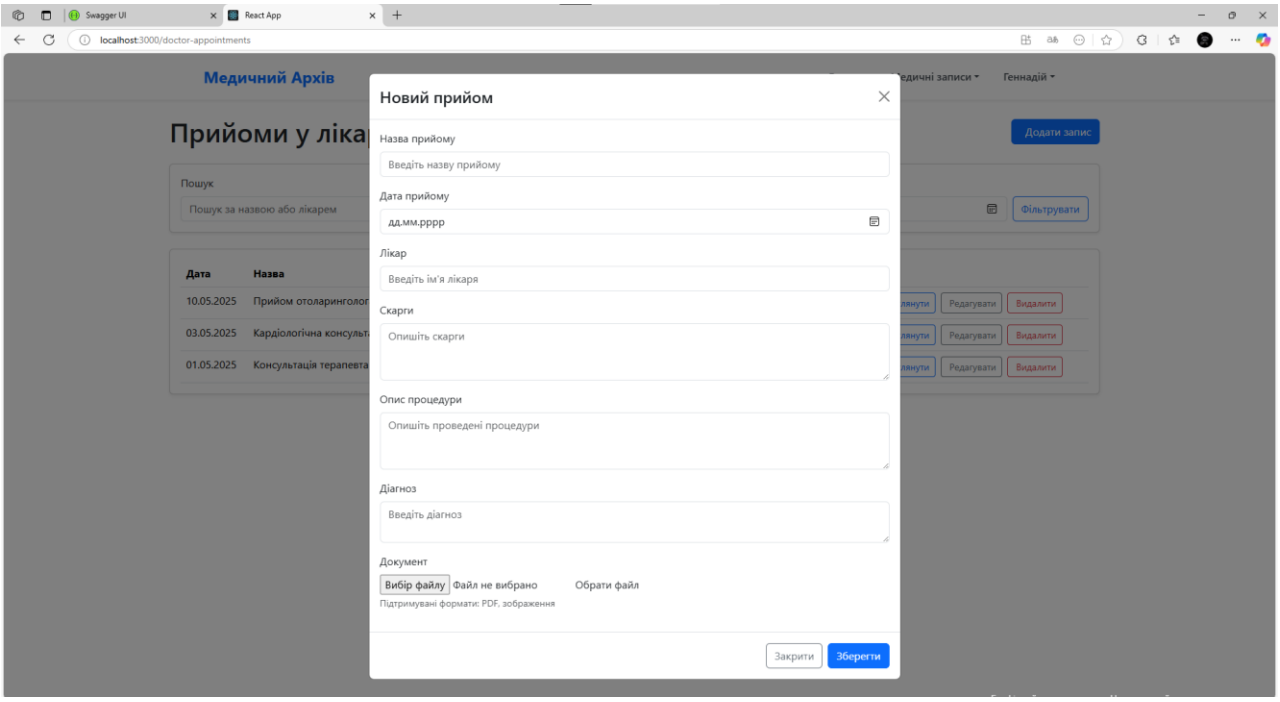


Рис. 3.14 Форма створення нового прийому у лікаря

Форма містить всі необхідні поля для створення запису про прийом: назва, дата, ім'я лікаря, скарги, опис процедур, діагноз та можливість завантажити скан документа. Реалізована валідація всіх полів та попередній перегляд завантаженого файла.

На рисунку 3.15 зображена детальна сторінка прийому у лікаря.

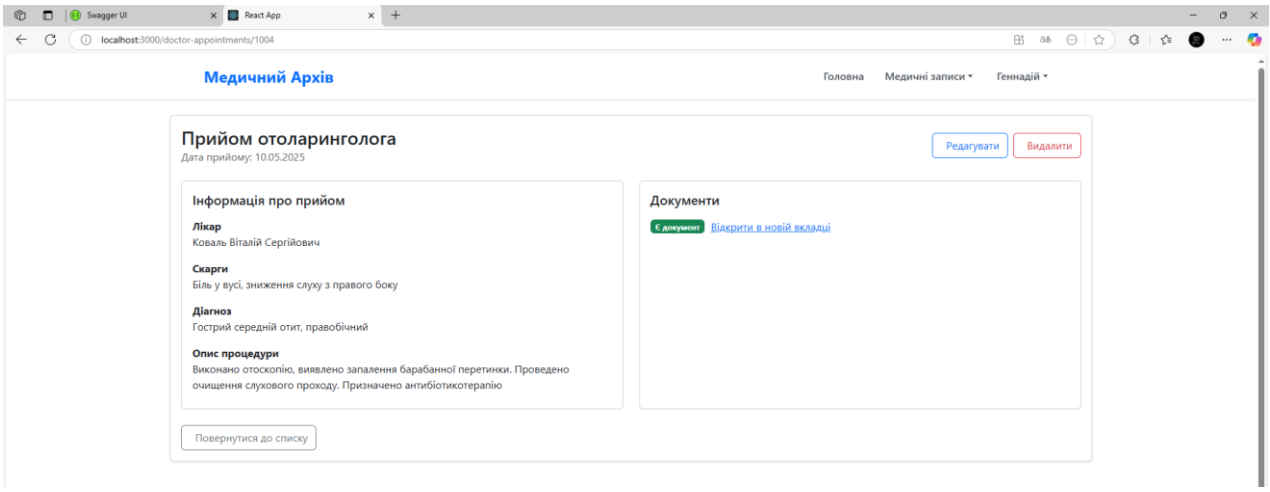


Рис. 3.15 Детальна сторінка прийому у лікаря

Детальна сторінка показує всю інформацію про прийом у зручному для читання форматі. Якщо до запису прикріплено документ, користувач може його переглянути або завантажити. Доступні кнопки для редагування та видалення запису.

Система забезпечує повний цикл управління рецептами з аналогічним інтерфейсом до прийомів.

На рисунку 3.16 зображена сторінка зі списком рецептів.

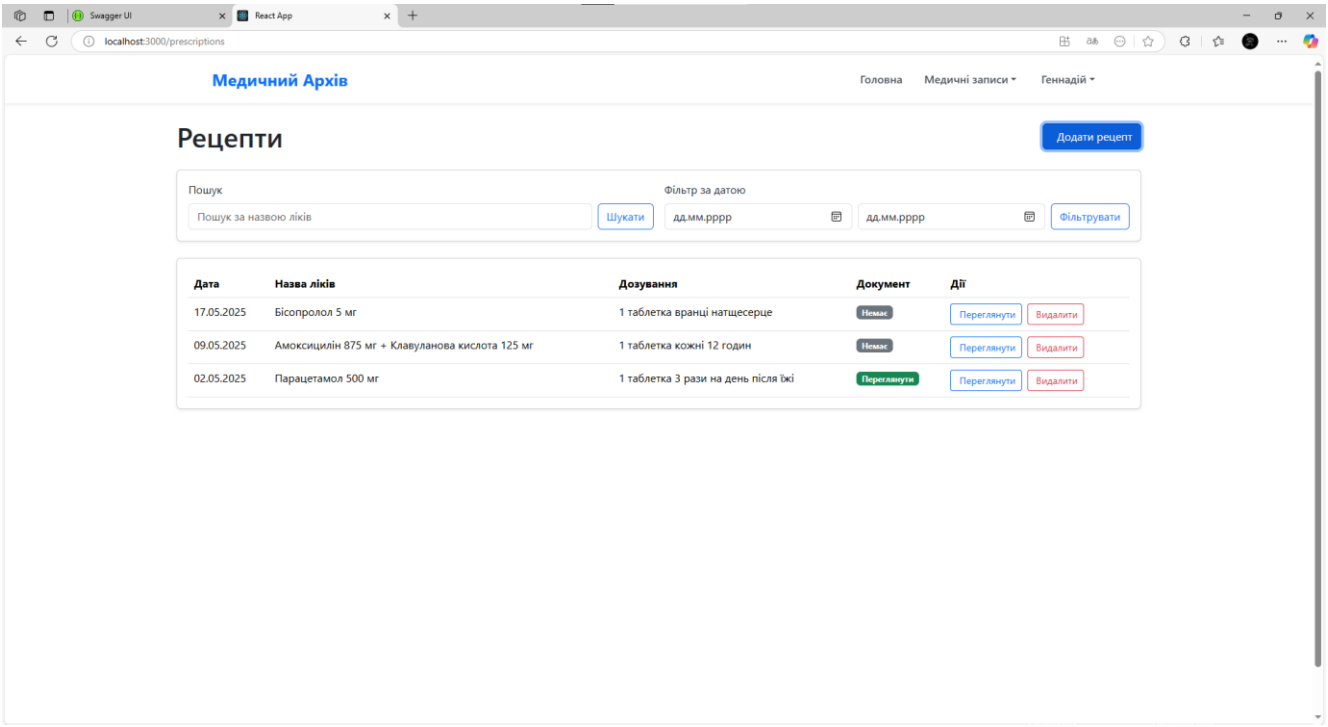


Рис. 3.16 Сторінка зі списком рецептів

Список рецептів показує назву препарату, дату виписки, дозування та інструкції. Реалізовано пошук за назвою препарату та фільтрацію за датами.

Форма включає поля для назви препарату, дати виписки, дозування, інструкцій по застосуванню та завантаження скану рецепту.

Розділ направлень дозволяє зберігати всі медичні направлення з можливістю відстеження термінів дії.

На рисунку 3.17 зображена сторінка зі списком направлень.

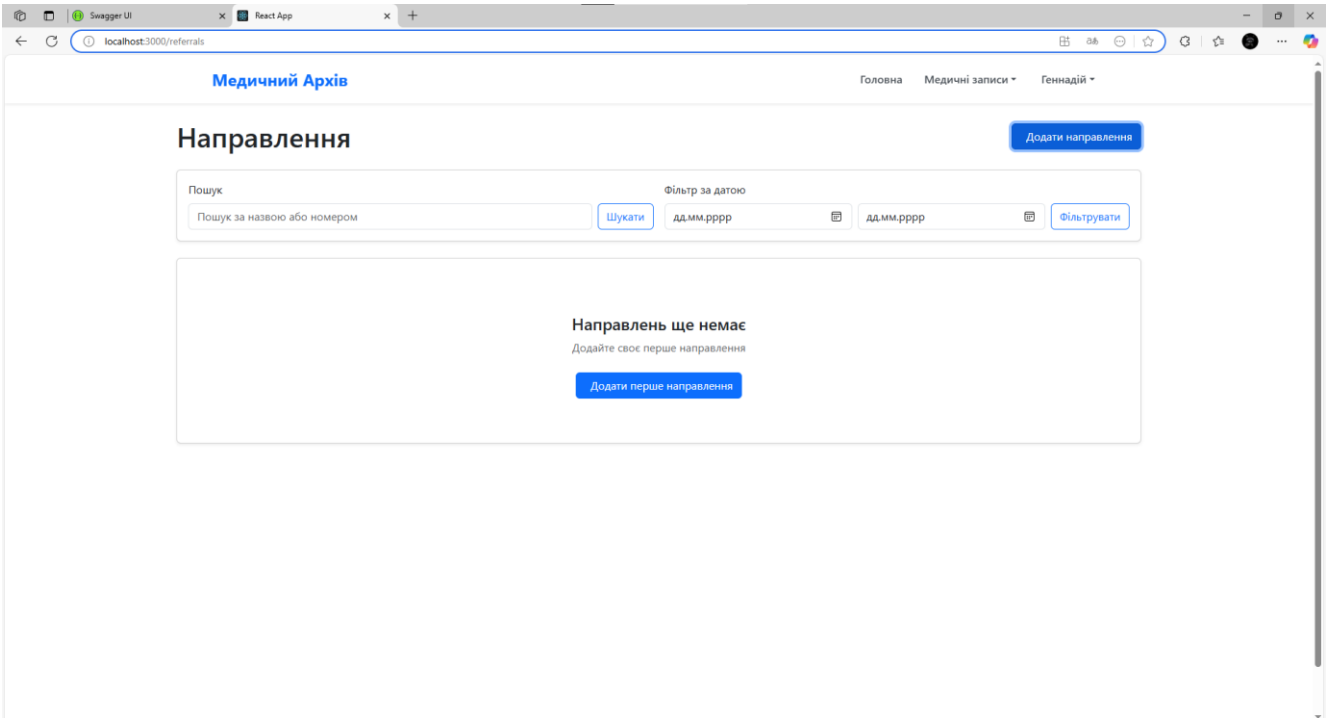


Рис. 3.17 Сторінка направлень

Список показує назву направлення, тип, дати видачі та закінчення терміну дії, номер направлення. Система автоматично виділяє прострочені направлення іншим кольором.

Система ведення щеплень включає повну історію вакцинації користувача.

На рисунку 3.18 зображена сторінка зі списком щеплень.

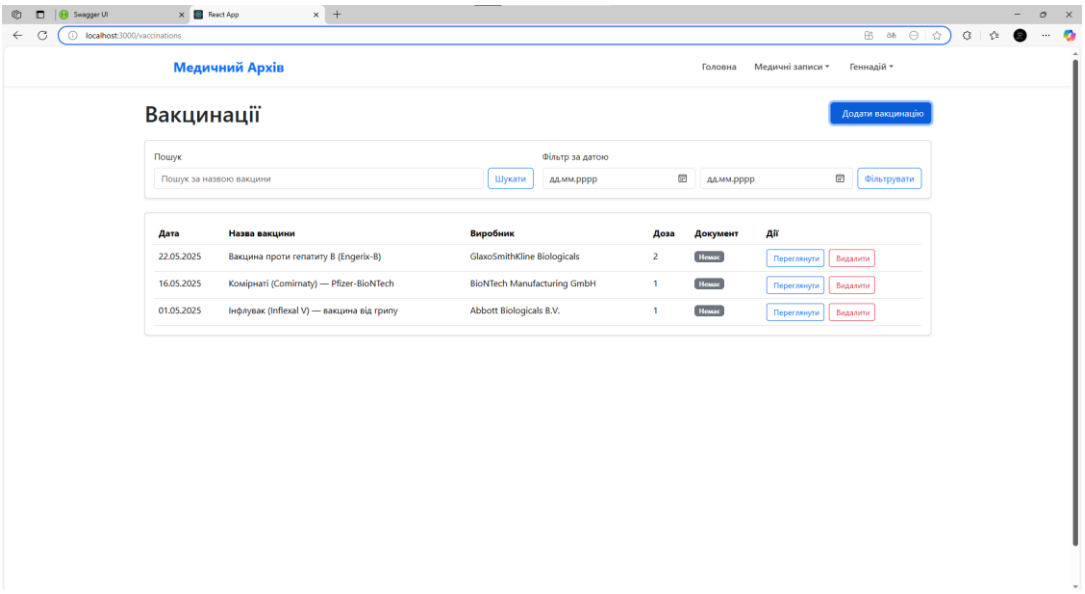


Рис. 3.18 Сторінка зі списком щеплень

Відображається назва вакцини, дата вакцинації, виробник та номер дози. Реалізовано календарний вигляд для зручного перегляду графіку вакцинації.

Розділ довідок дозволяє зберігати різноманітні медичні документи та довідки.

На рисунку 3.19 зображена сторінка зі списком медичних довідок.

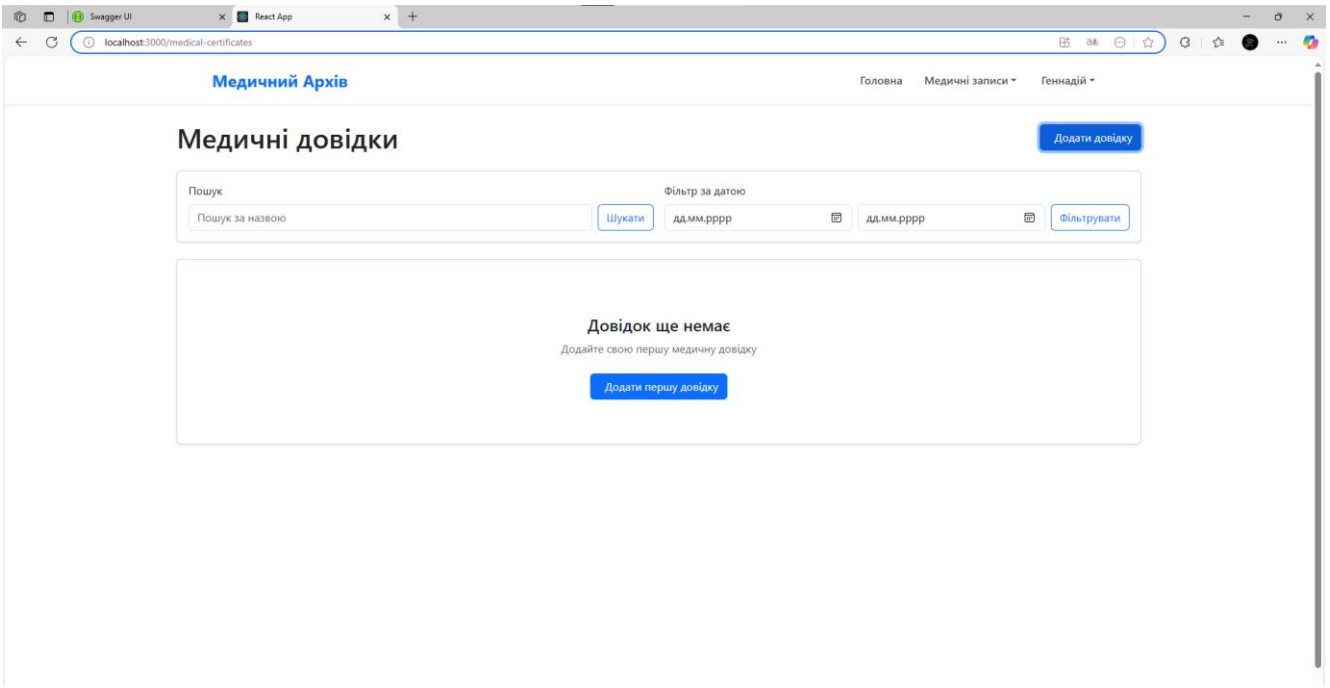


Рис. 3.19 Сторінка медичних довідок

Список показує назву довідки, дату видачі, опис та статус документа. Користувач може організовувати довідки за категоріями.

Система дозволяє пацієнтам надавати лікарям доступ до своїх медичних даних.

На рисунку 3.20 зображена сторінка управління доступом лікарів.

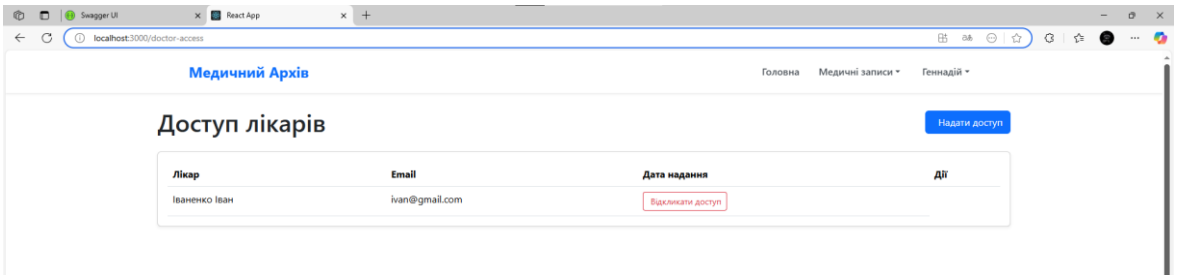


Рис. 3.20 Сторінка управління доступом лікарів

Сторінка показує список лікарів, які мають доступ до даних користувача, з можливістю видалення доступу. Також присутня форма для надання доступу новому лікарю за email-адресою.

Лікарі мають спеціальний інтерфейс для перегляду даних пацієнтів, які надали їм доступ.

На рисунку 3.21 зображена сторінка списку пацієнтів для лікаря.

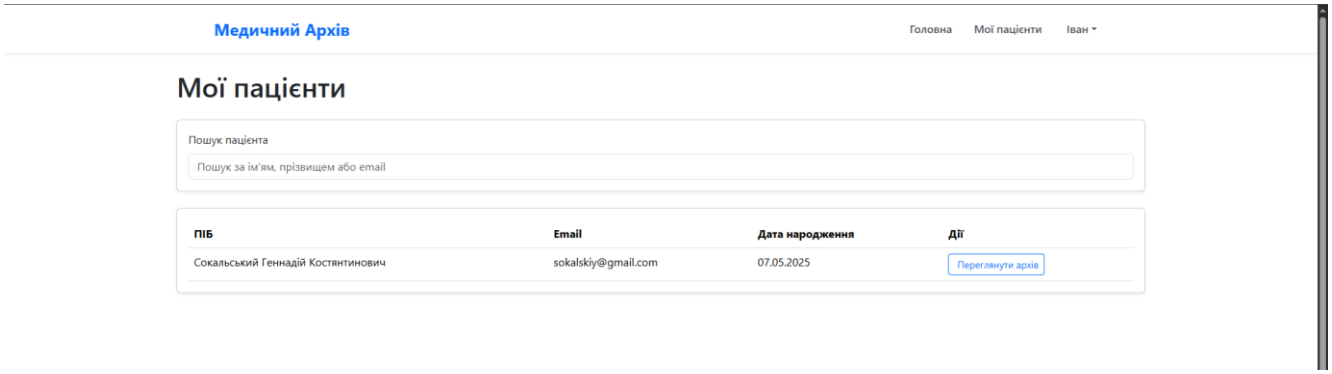


Рис. 3.21 Сторінка списку пацієнтів для лікаря

Лікар бачить список пацієнтів, які надали йому доступ, з можливістю переглянути медичні записи кожного пацієнта у режимі "тільки для читання".

Система включає розділ для управління особистими даними користувача.

На рисунку 3.22 зображена сторінка профілю користувача.

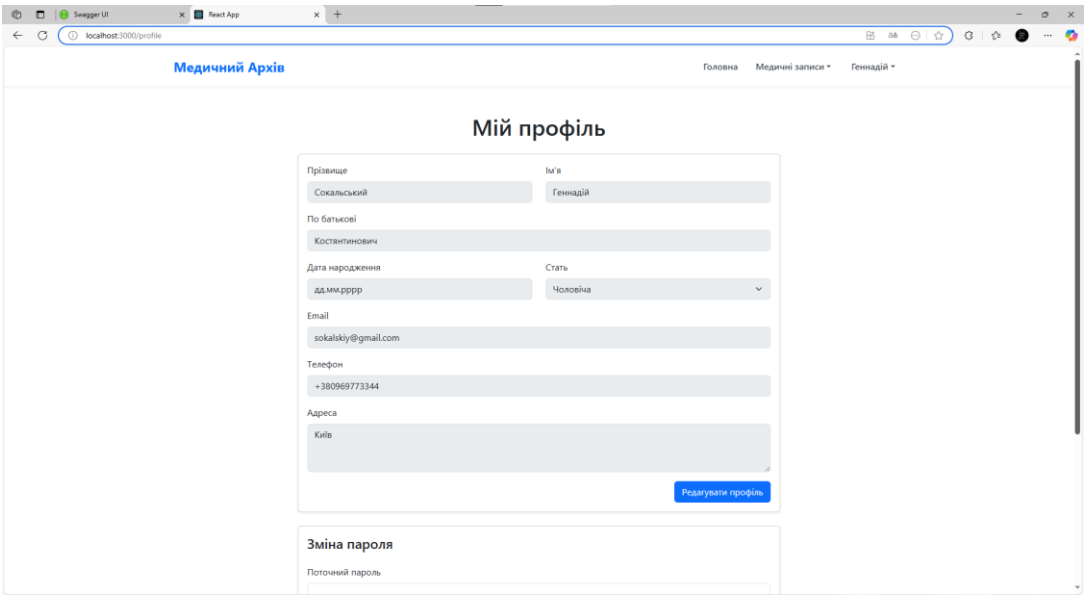


Рис. 3.22 Сторінка профілю користувача

4 ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ

4.1 Методи тестування

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, особливо у випадках, коли йдеться про роботу з чутливими даними, такими як медична інформація. Його мета — виявити помилки, перевірити відповідність функціоналу очікуваній поведінці, а також забезпечити надійність, зручність і безпеку роботи системи. Для перевірки працездатності та якості розробленого програмного забезпечення було застосовано декілька підходів до тестування, які охоплюють як функціональні, так і нефункціональні аспекти.

Основним методом, що використовувався в процесі перевірки системи, є функціональне тестування. Воно полягає у перевірці того, чи система виконує всі функції, визначені у специфікації вимог. Зокрема, тестувалися можливості реєстрації та аутентифікації користувачів, створення та редагування медичних записів, завантаження документів, фільтрація інформації, а також надання доступу лікарям. Було перевірено роботу кожного з модулів системи на рівні окремих сценаріїв — від створення облікового запису до перегляду історії записів лікарем, якому надано дозвіл.

Окрему увагу було приділено тестуванню API. Для цього використовувався інструмент Postman, який дозволяє формувати запити до REST API та перевіряти коректність відповідей. Таким чином, проводилася перевірка відповідності статус-кодів, правильності обробки помилок, відповідності структури відповідей заданим форматам, а також перевірка обмежень доступу відповідно до ролі користувача. Завдяки цьому вдалося забезпечити стабільну роботу API при взаємодії як з фронтендом, так і з іншими клієнтськими додатками.

У рамках перевірки захищеності даних було проведено перевірку механізмів авторизації та аутентифікації. Зокрема, тестувалося правильне використання JWT-токенів, обробка запитів без авторизації, обмеження доступу до адміністративних

функцій для неавторизованих користувачів, а також перевірка реакції системи на спроби несанкціонованого доступу до медичних записів інших користувачів.

Для перевірки коректності взаємодії з базою даних використовувалося інтеграційне тестування. Його метою було переконатися, що створені записи зберігаються у базі даних, дані не дублюються, правильно відображаються у відповідних категоріях, а також що система коректно обробляє оновлення та видалення даних. Особлива увага приділялася зв’язкам між сутностями та дотриманню обмежень цілісності даних.

Також застосовувалося ручне тестування інтерфейсу користувача, під час якого перевірялась логіка переходів між сторінками, правильність відображення даних, зручність використання форм, інформативність повідомлень про помилки та загальна інтуїтивність інтерфейсу як для пацієнта, так і для лікаря. Було враховано адаптивність дизайну для різних пристроїв та розмірів екранів.

Хоча в межах цієї роботи не було реалізовано автоматизованого тестування, архітектура системи спроєктована так, щоб у майбутньому можна було легко додати юніт-тести та інтеграційні тести для ключових модулів системи.

Загалом, застосовані методи тестування дозволили виявити та усунути низку технічних та логічних помилок, підтвердити відповідність реалізованої системи початковим вимогам та забезпечити її стабільну й безпечну роботу в межах заявленого функціоналу.

4.2 Тестування основних сценаріїв роботи

Після реалізації основної функціональності системи було проведено тестування найбільш критичних і типових сценаріїв роботи користувачів. Такий підхід дозволяє не лише виявити технічні помилки, а й оцінити зручність, логічність і стабільність системи в реальних умовах використання. Тестування охоплювало як дії звичайного користувача (пацієнта), так і користувача з роллю лікаря.

Перший протестований сценарій — реєстрація нового користувача та вхід до системи. Було перевірено, що при заповненні реєстраційної форми з коректними

даними обліковий запис створюється успішно, пароль хешується, а система автоматично видає токен доступу. Також перевірено обробку некоректних введень: дублювання електронної адреси, слабкий пароль, відсутність обов’язкових полів — у всіх випадках користувач отримував інформативне повідомлення про помилку.

Наступний сценарій — створення медичного запису пацієнтом. Користувач заходив у систему, обирав категорію (наприклад, «Прийом у лікаря»), заповнював відповідну форму, додавав опис, дату, а також прикріплював скан документа у форматі PDF. Після збереження запис був коректно відображений у списку, і його можна було переглянути або редагувати. При спробі додати файл у недопустимому форматі система відображала попередження.

Окремо перевірявся сценарій фільтрації та пошуку записів. У тестах моделювались ситуації, коли у пацієнта велика кількість записів різних типів. Була перевірена можливість відфільтрувати їх за категорією, датою або ключовими словами. Усі пошукові запити повертали правильні результати, навіть при комбінованих фільтрах.

Особливу увагу приділено сценарію надання доступу лікарю. Пацієнт вводив електронну адресу лікаря, обирав, до яких категорій даних надати дозвіл, після чого лікар, увійшовши до свого облікового запису, бачив список пацієнтів, які його авторизували. Було перевірено, що лікар не має доступу до даних інших користувачів, і що при відкликанні доступу пацієнтом усі записи зникають зі списку лікаря.

Також перевірявся сценарій перегляду медичних записів лікарем. Після авторизації лікар міг обрати пацієнта та побачити доступні йому записи з фільтрами за категорією й датою. Було переконливо доведено, що лікар бачить тільки ті записи, які дозволені пацієнтом, і не має змоги їх редагувати чи видаляти.

З технічної точки зору були протестовані випадки втрати інтернет-з’єднання, невалідних запитів, а також спроб доступу до ресурсів без токена авторизації. У кожному випадку система коректно обробляла ситуації: зберігала сесію при повторному підключенні, повертала повідомлення про помилку або статус «Недоступно» в разі відсутності авторизації.

Таким чином, тестування ключових сценаріїв довело, що система коректно реалізує всі основні функції, відповідає очікуваній логіці роботи користувачів і забезпечує високий рівень захисту та керованості даними. Система виявилася стабільною та готовою до практичного використання в рамках особистого збереження медичної інформації.

На рисунку 4.1 зображена діаграма тестового сценарію

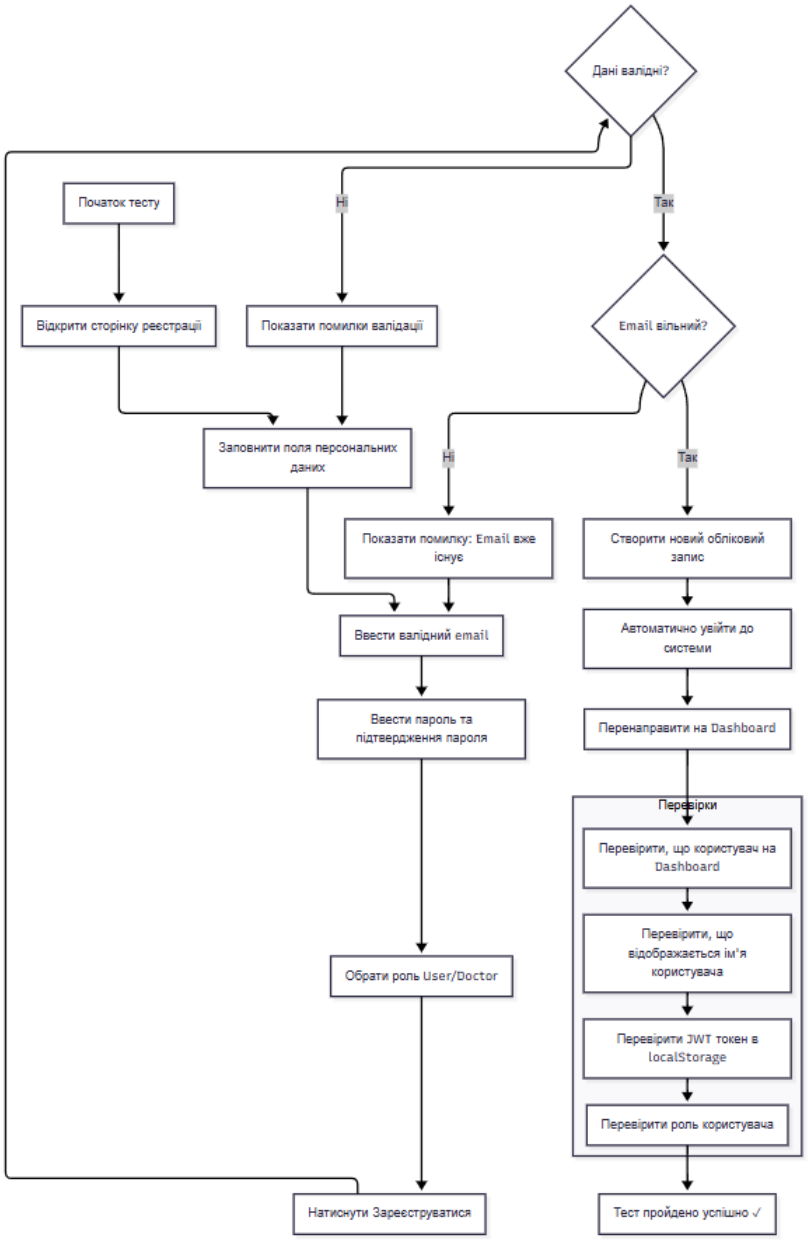


Рис. 4.1 Діаграма тестового сценарію реєстрації нового користувача

4.3 Аналіз отриманих результатів

Після завершення розробки програмного забезпечення для ведення персонального архіву медичних даних було проведено комплексне тестування, яке

мало на меті перевірку коректності реалізованої функціональності, відповідності системи поставленим вимогам та загальної готовності до практичного використання. Особлива увага приділялася як функціональним аспектам, так і ключовим нефункціональним характеристикам, зокрема безпеці, стабільності роботи та масштабованості. Загальні результати засвідчили, що система демонструє стабільну поведінку при виконанні всіх основних сценаріїв та коректно реагує на можливі виняткові ситуації.

У процесі перевірки було протестовано критично важливі функції: реєстрацію та аутентифікацію користувачів, створення, редагування та видалення медичних записів, завантаження файлів медичних документів, фільтрацію та пошук даних, а також механізм надання доступу лікарям. Усі зазначені можливості продемонстрували належну працездатність. Незначні недоліки, виявлені під час тестування (наприклад, відсутність обмежень на розмір завантажуваних файлів або валідацію окремих полів форм), були оперативно усунені. Всі перевірки проводилися як на рівні клієнтської, так і серверної частини застосунку.

Окрему увагу було приділено безпеці — критично важливому аспекту, враховуючи роботу з персональними медичними даними. Було перевірено коректність реалізації аутентифікації за допомогою JWT-токенів, захист API від неавторизованих запитів, а також правильність розмежування прав доступу відповідно до ролей «пацієнт» і «лікар». Система продемонструвала здатність ефективно захищати медичну інформацію та забезпечувати доступ до неї лише уповноваженим користувачам. У тестових сценаріях лікарі отримували доступ лише до тих записів, які були надані пацієнтами, що повністю відповідає вимогам до конфіденційності.

Таким чином, результати тестування підтверджують відповідність системи функціональним і нефункціональним вимогам, що були сформульовані на етапі проєктування. Система є стабільною, безпечною та логічно організованою, а її архітектура дозволяє подальше масштабування та розвиток.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено програмне забезпечення для ведення персонального архіву медичних даних, що вирішує актуальну проблему збереження, впорядкування та безпечного доступу до особистої медичної інформації. Проведений аналіз предметної області дозволив виявити основні обмеження та недоліки існуючих рішень, зокрема відсутність можливості самостійного додавання записів, обмежений набір категорій даних, нестачу контролю доступу до інформації та відсутність підтримки роботи з медичними документами.

Розроблене програмне забезпечення реалізовано з використанням сучасного стеку технологій: ASP.NET Core Web API, Microsoft SQL Server, Entity Framework Core, React з TypeScript та супутніх бібліотек. Архітектура системи побудована за принципами багатошарового розподілу, що забезпечує високу масштабованість, зручність супроводу та відповідність принципам SOLID. Значну увагу було приділено питанням безпеки, зокрема захищеній аутентифікації за допомогою JWT, хешуванню паролів, реалізації контрольованого доступу до медичних записів та захисту переданих даних.

У результаті тестування підтверджено коректну роботу всіх основних сценаріїв: реєстрація та автентифікація користувачів, створення медичних записів, завантаження документів, фільтрація, пошук, надання доступу лікарям та робота з базою даних. Система показала стабільність, відповідність функціональним і нефункціональним вимогам, а також здатність ефективно вирішувати поставлені задачі.

Таким чином, розроблене рішення повністю реалізує поставлену мету — створення надійного інструменту для персонального збереження медичної інформації з можливістю гнучкого керування доступом. Отримані результати підтверджують доцільність і практичну значущість створеної системи, яка може бути використана як самостійний продукт або стати основою для подальшого розвитку більш масштабованих медичних платформ.

Апробація результатів дослідження:

1. Сокальський Г.К., Довженко Т.П., “Розробка програмного забезпечення для ведення персонального архіву медичних даних”, VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 122-124.
2. Сокальський Г.К., Довженко Т.П.,“Інтелектуальна система ведення персонального медичного архіву на основі сучасних програмних засобів”, V Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.05.25, ДУІКТ, Київ, Україна, подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. MyChart is epic. MyChart: веб-сайт. URL: https://www.mychart.org/ (дата звернення: 03.03.2025)
2. Helsi. Helsi: веб-сайт. URL https://helsi.me/ (дата звернення: 03.03.2025)
3. ASP.NET documentation. Learn.microsoft : веб-сайт. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0 (дата звернення: 05.03.2025)
4. REST API як спосіб спілкування компонент веб-додатків. Foxminded: веб-сайт. URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 05.03.2025)
5. Introduction to JSON Web Tokens. JWT Debugger: веб-сайт. URL: https://jwt.io/introduction (дата звернення: 05.03.2025)
6. Entity Framework Core. Learn.microsoft: веб-сайт. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 06.03.2025)
7. EF Core Migrations. Learn Entity Framework Core: веб-сайт. URL: https://www.learnentityframeworkcore.com/migrations (дата звернення: 06.03.2025)
8. Microsoft SQL Server. Microsoft: веб-сайт. URL: https://www.microsoft.com/en-us/sql-server (дата звернення: 07.03.2025)
9. React The library for web and native user interfaces. React: веб-сайт. URL: https://react.dev/ (дата звернення: 07.03.2025)
10. Learn how to include React Bootstrap in your project. React Bootstrap: веб-сайт. URL: https://react-bootstrap.netlify.app/docs/getting-started/introduction (дата звернення: 08.03.2025)
11. Layered (N-Layer) Architecture. Medium: веб-сайт. URL: https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42 (дата звернення: 03.03.2025)
12. Lock, A. ASP.NET Core in Action, Third Edition. Manning Publications. 2023. 984 с.

13. Esposito, D. Programming ASP.NET Core (Developer Reference). Microsoft Press. 2020. 614 c.
14. Smith, J. P. Entity Framework Core in Action, Second Edition. Manning Publications. 2021. 624 c.
15. Rippon, C. Learn React with TypeScript: A Beginner's Guide to Reactive Web Development with React 18 and TypeScript. Packt Publishing. 2023. 474 c.
16. Goldberg, J. Learning TypeScript: Enhance Your Web Development Skills Using Type-Safe JavaScript. O'Reilly Media. 2022. 318 c.
17. Ben-Gan, I. T-SQL Fundamentals, 4th Edition. Microsoft Press. 2023. 608 c.
18. Richards, M., Ford, N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media. 2020. 422 c.
19. Bell, M. B. T. Software Architect. Wiley. 2023. 432 c.
20. Shaikh, A. Web Application Architecture: The Latest Guide 2025. *Peerbits*. 2024. URL: https://www.peerbits.com/blog/web-application-architecture.html
21. West R., Assaf W., Noble E., Longoria M., D'Antoni J., Davidson L.. SQL Server 2022 Administration Inside Out. Microsoft Press. 2023. 922 c.
22. Noback, M. Advanced Web Application Architecture. Leanpub. 2020. 539 p.
23. Ford, N., Richards, M., Sadalage, P., Dehghani, Z.. Software Architecture: The Hard Parts. O'Reilly Media. 2021. 464 c.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для ведення персонального архіву медичних даних

Виконав студент 4 курсу
Групи ПД-42
Сокальський Геннадій Костянтинович
Керівник роботи
Д.т.н., Професор ІПЗ Довженко Тимур Павлович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – покращення централізованого зберігання, впорядкування та контролю доступу до особистих медичних даних користувача.

Об'єкт дослідження - процес зберігання та управління персональною медичною інформацією.

Предмет дослідження - програмне забезпечення для ведення персонального архіву медичних даних із можливістю надання доступу лікарям.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз сучасних підходів до організації персональних медичних архівів.
2. Дослідити функціональні можливості наявних рішень і виявити їхні основні обмеження.
3. Розробити архітектуру системи з урахуванням вимог до безпеки, масштабованості та зручності використання.
4. Реалізувати основні функції: реєстрацію користувачів, створення та категоризацію медичних записів, завантаження документів.
5. Забезпечити можливість фільтрації, пошуку інформації та контрольованого доступу лікарів до даних користувача.
6. Виконати тестування реалізованої системи та оцінити її відповідність поставленим вимогам.

3

АНАЛІЗ АНАЛОГІВ

Критерій	MyChart	Helsi	MyHealthApp	MedicalArchive
Самостійне додавання записів	-	-	-	+
Категоризація записів	-	-	-	+
Завантаження сканів, фото документів	+	-	-	+
Керування доступом лікарів	-	-	-	+
Незалежність від конкретної медичної мережі	-	-	+	+
Централізована історія медичних даних	+	+	-	+
Фільтрація та пошук по записах	-	+	-	+
Орієнтованість на повноцінний архів	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та аутентифікація користувачів (пацієнт, лікар)
2. Управління медичними записами (створення, редагування, перегляд, видалення)
3. Категоризація медичних записів (прийоми, рецепти, направлення, довідки, щеплення)
4. Завантаження сканованих документів (PDF, JPG, PNG)
5. Пошук і фільтрація записів (за різними критеріями)
6. Контрольований доступ лікарям (через email)

Нефункціональні вимоги

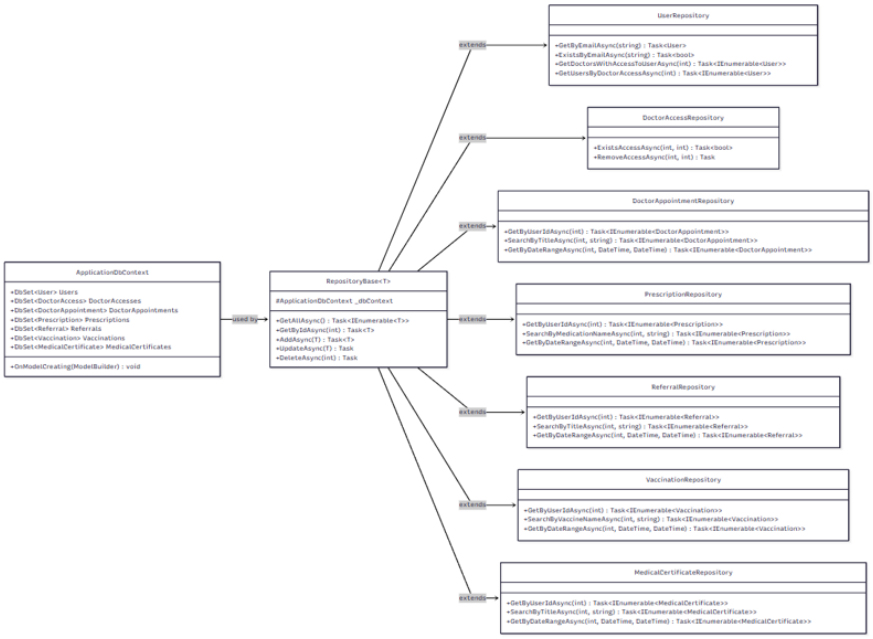
1. Забезпечення безпеки (HTTPS, хешування паролів, JWT)
2. Висока продуктивність та надійність
3. Підтримка масштабованості (модульна архітектура, SOLID)
4. Зручний адаптивний інтерфейс (різні пристрої)
5. Обробка помилок та логування (для діагностики)

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

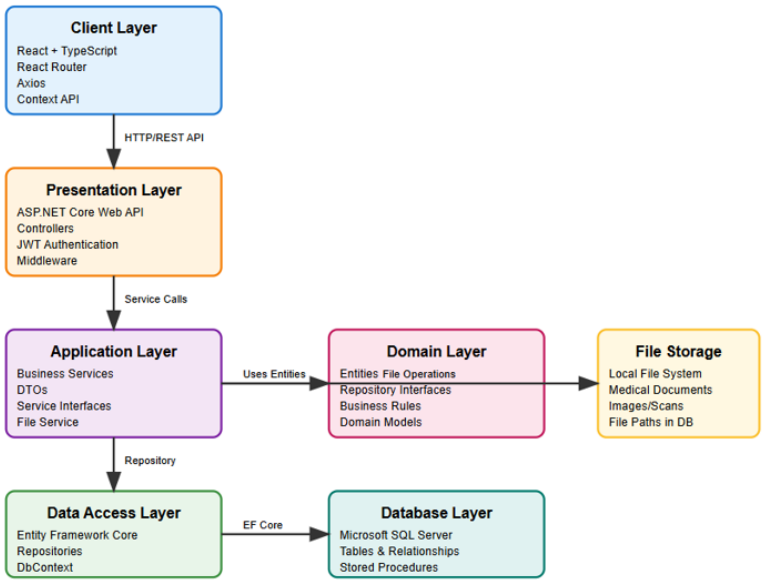
6

ДІАГРАМА КЛАСІВ



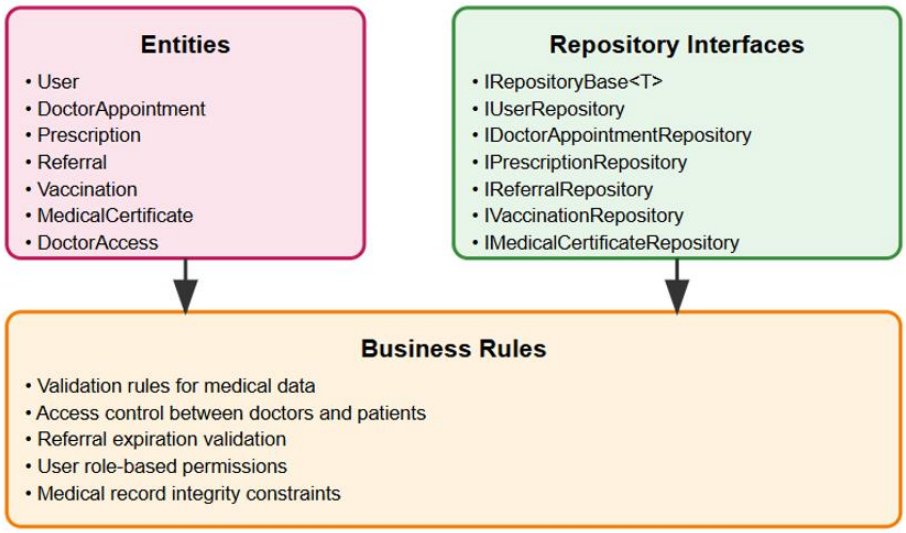
7

ДІАГРАМА ЗАГАЛЬНОЇ АРХІТЕКТУРИ СИСТЕМИ



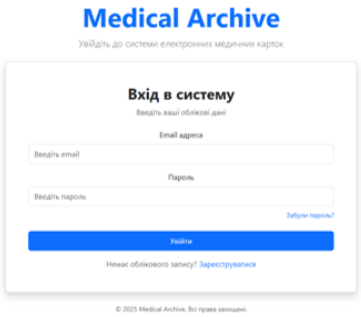
8

ДІАГРАМА СТРУКТУРИ

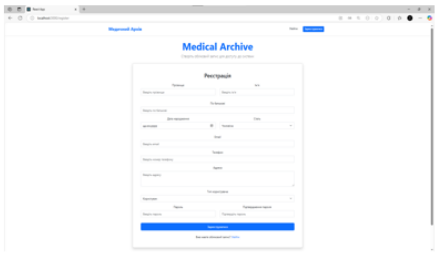


9

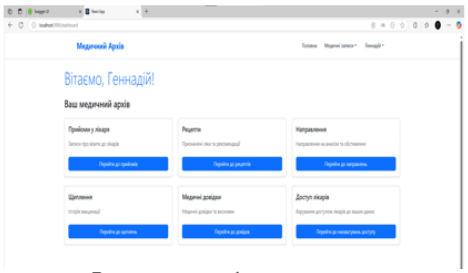
ЕКРАННІ ФОРМИ



Сторінка входу в систему

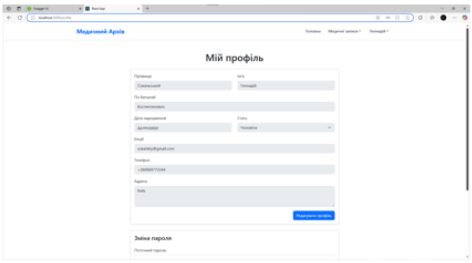


Сторінка реєстрації в системі

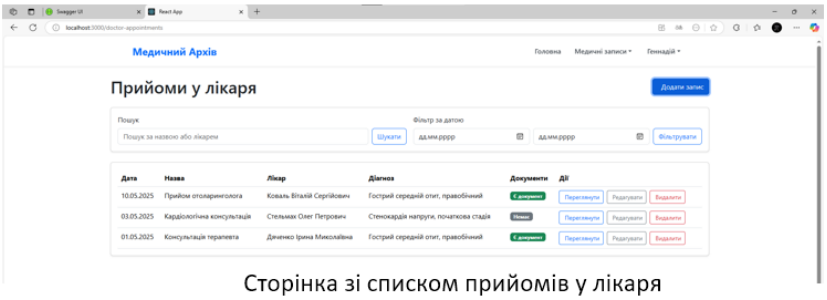


Головна сторінка системи

ЕКРАННІ ФОРМИ



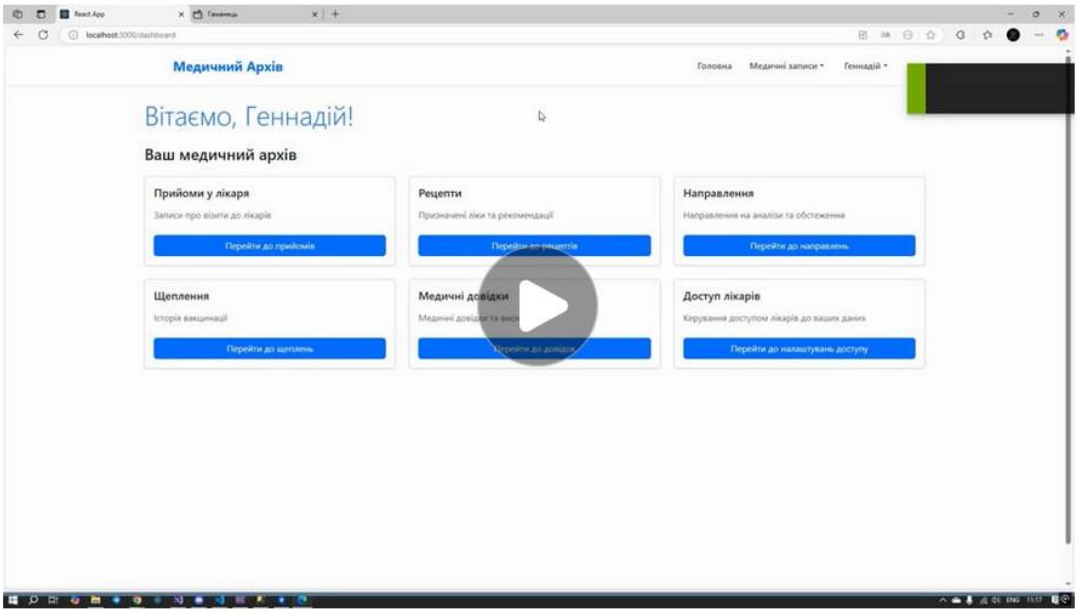
Сторінка профілю користувача



Сторінка зі списком прийомів у лікаря

11

ДЕМОНСТРАЦІЯ РОБОТИ ЗАСТОСУНКУ



12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Сокальський Г.К., Довженко Т.П., “Розробка програмного забезпечення для ведення персонального архіву медичних даних”, VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 122-124.
2. Сокальський Г.К., Довженко Т.П.,“Інтелектуальна система ведення персонального медичного архіву на основі сучасних програмних засобів”, V Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.05.25, ДУІКТ, Київ, Україна, подано до друку.

13

ВИСНОВКИ

1. Проведено ґрунтовний огляд сучасних рішень для організації персональних медичних архівів, що дозволило визначити ефективні підходи до структурування та захисту медичних даних.
2. Проаналізовано функціональні можливості наявних систем. Встановлено, що основними їхніми обмеженнями є відсутність гнучкого механізму контролю доступу, обмежений вибір категорій медичних записів, недостатньо зручний інтерфейс для кінцевого користувача та невисокий рівень забезпечення безпеки.
3. Розроблено модульну архітектуру програмного забезпечення, що відповідає вимогам до безпеки, масштабованості та зручності використання. Застосовано сучасні підходи до побудови серверної та клієнтської частин із використанням ASP.NET Core Web API, React з TypeScript та SQL Server.
4. Реалізовано ключовий функціонал системи: забезпечено можливість реєстрації та аутентифікації користувачів із розмежуванням ролей, створення та категоризації медичних записів, завантаження та збереження сканованих документів.
5. Налагоджено механізми фільтрації та пошуку даних, що дозволяє ефективно працювати з великими обсягами медичної інформації. Забезпечено функцію контрольованого доступу лікарів до обраних записів користувача за допомогою email-запрошень.
6. Проведено комплексне тестування розробленої системи. Перевірено коректність виконання основних сценаріїв використання, механізми захисту даних та відповідність розробленого рішення визначеним вимогам. Отримані результати підтвердили працездатність системи та готовність до практичного використання.

14

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using MedicalArchive.API.Application.DTOs.AuthDTOs;
using MedicalArchive.API.Application.DTOs.UserDTOs;
using MedicalArchive.API.Application.Interfaces;
using MedicalArchive.API.Domain.Entities;
using MedicalArchive.API.Domain.Interfaces;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;

namespace MedicalArchive.API.Application.Services
{
    public class AuthService : IAuthService
    {
        private readonly IUserRepository _userRepository;
        private readonly IConfiguration _configuration;

        public AuthService(IUserRepository userRepository, IConfiguration configuration)
        {
            _userRepository = userRepository;
            _configuration = configuration;
        }

        public async Task<TokenDto> LoginAsync(UserLoginDto loginDto)
        {
            var user = await _userRepository.GetByEmailAsync(loginDto.Email);

            if (user == null)
            {
                throw new Exception("Користувача з такою електронною адресою не знайдено");
            }

            if (!VerifyPasswordHash(loginDto.Password, user.PasswordHash))
            {
                throw new Exception("Невірний пароль");
            }
```

```
            var token = await GenerateJwtTokenAsync(MapToUserDto(user));

            return new TokenDto
            {
                AccessToken = token,
                UserEmail = user.Email,
                UserRole = user.Role,
                UserId = user.Id
            };
        }

        public async Task<UserDto> RegisterAsync(UserRegistrationDto registrationDto)
        {
            var existingUser = await _userRepository.ExistsByEmailAsync(registrationDto.Email);

            if (existingUser)
            {
                throw new Exception("Користувач з такою електронною адресою вже існує");
            }

            var passwordHash = HashPassword(registrationDto.Password);

            var user = new User
            {
                FirstName = registrationDto.FirstName,
                LastName = registrationDto.LastName,
                MiddleName = registrationDto.MiddleName,
                DateOfBirth = registrationDto.DateOfBirth,
                Gender = registrationDto.Gender,
                Email = registrationDto.Email,
                PhoneNumber = registrationDto.PhoneNumber,
                Address = registrationDto.Address,
                PasswordHash = passwordHash,
                Role = registrationDto.Role
            };

            await _userRepository.AddAsync(user);

            return MapToUserDto(user);
        }

        public async Task<string> GenerateJwtTokenAsync(UserDto user)
        {
            var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
            var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
                new Claim(JwtRegisteredClaimNames.Email, user.Email),
                new Claim(ClaimTypes.Role, user.Role),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            };

            var token = new JwtSecurityToken(
                issuer: _configuration["Jwt:Issuer"],
                audience: _configuration["Jwt:Audience"],
                claims: claims,
                expires: DateTime.Now.AddHours(24),
                signingCredentials: credentials
            );
```

```
            return new JwtSecurityTokenHandler().WriteToken(token);
        }

        private string HashPassword(string password)
        {
            using (var sha256 = SHA256.Create())
            {
                var hashedBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
                return BitConverter.ToString(hashedBytes).Replace("-", "").ToLower();
            }
        }

        private bool VerifyPasswordHash(string password, string hash)
        {
            var passwordHash = HashPassword(password);
            return passwordHash == hash;
        }

        private UserDto MapToUserDto(User user)
        {
            return new UserDto
            {
                Id = user.Id,
                FirstName = user.FirstName,
                LastName = user.LastName,
                MiddleName = user.MiddleName,
                DateOfBirth = user.DateOfBirth,
                Gender = user.Gender,
                Email = user.Email,
                PhoneNumber = user.PhoneNumber,
                Address = user.Address,
                Role = user.Role
            };
        }
    }
}
```

```
using MedicalArchive.API.Application.DTOs.DoctorAccessDTOs;
using MedicalArchive.API.Application.DTOs.UserDTOs;
using MedicalArchive.API.Application.Interfaces;
using MedicalArchive.API.Domain.Entities;
using MedicalArchive.API.Domain.Interfaces;

namespace MedicalArchive.API.Application.Services
{
    public class DoctorAccessService : IDoctorAccessService
    {
        private readonly IDoctorAccessRepository _doctorAccessRepository;
        private readonly IUserRepository _userRepository;

        public DoctorAccessService(IDoctorAccessRepository doctorAccessRepository, IUserRepository userRepository)
        {
            _doctorAccessRepository = doctorAccessRepository;
            _userRepository = userRepository;
        }

        public async Task<IEnumerable<UserDto>> GetDoctorsWithAccessToUserAsync(int userId)
        {
            var doctors = await _userRepository.GetDoctorsWithAccessToUserAsync(userId);
            return doctors.Select(MapToUserDto);
        }
```

```
        public async Task<IEnumerable<UserDto>> GetUsersByDoctorAccessAsync(int doctorId)
        {
            var users = await _userRepository.GetUsersByDoctorAccessAsync(doctorId);
            return users.Select(MapToUserDto);
        }

        public async Task<DoctorAccessDto> GrantAccessAsync(int userId, DoctorAccessCreateDto accessDto)
        {
            // Перевіряємо, чи існує користувач
            var user = await _userRepository.GetByIdAsync(userId);
            if (user == null)
            {
                throw new Exception($"Користувача з ID {userId} не знайдено");
            }

            // Знаходимо лікаря за email
            var doctor = await _userRepository.GetByEmailAsync(accessDto.DoctorEmail);
            if (doctor == null)
            {
                throw new Exception($"Лікаря з email {accessDto.DoctorEmail} не знайдено");
            }

            // Перевіряємо, чи є цей користувач лікарем
            if (doctor.Role != "Doctor")
            {
                throw new Exception($"Користувач з email {accessDto.DoctorEmail} не є лікарем");
            }

            // Перевіряємо, чи вже є доступ у цього лікаря
            var existingAccess = await _doctorAccessRepository.ExistsAccessAsync(userId, doctor.Id);
            if (existingAccess)
            {
                throw new Exception($"Лікар з email {accessDto.DoctorEmail} вже має доступ до ваших даних");
            }

            // Надаємо доступ
            var doctorAccess = new DoctorAccess
            {
                UserId = userId,
                DoctorId = doctor.Id,
                GrantedDate = DateTime.UtcNow
            };

            var createdAccess = await _doctorAccessRepository.AddAsync(doctorAccess);

            return new DoctorAccessDto
            {
                Id = createdAccess.Id,
                UserId = createdAccess.UserId,
                DoctorId = createdAccess.DoctorId,
                GrantedDate = createdAccess.GrantedDate,
                User = MapToUserDto(user),
                Doctor = MapToUserDto(doctor)
            };
        }

        public async Task<bool> RevokeAccessAsync(int userId, int doctorId)
        {
            // Перевіряємо, чи існує доступ
            var existingAccess = await _doctorAccessRepository.ExistsAccessAsync(userId, doctorId);
            if (!existingAccess)
            {
```

```
                throw new Exception($"Доступ лікаря з ID {doctorId} до користувача з ID {userId} не знайдено");
            }

            // Відкликаємо доступ
            await _doctorAccessRepository.RemoveAccessAsync(userId, doctorId);

            return true;
        }

        public async Task<bool> HasAccessAsync(int userId, int doctorId)
        {
            return await _doctorAccessRepository.ExistsAccessAsync(userId, doctorId);
        }

        // Мапінг з доменної моделі в DTO
        private UserDto MapToUserDto(User user)
        {
            return new UserDto
            {
                Id = user.Id,
                FirstName = user.FirstName,
                LastName = user.LastName,
                MiddleName = user.MiddleName,
                DateOfBirth = user.DateOfBirth,
                Gender = user.Gender,
                Email = user.Email,
                PhoneNumber = user.PhoneNumber,
                Address = user.Address,
                Role = user.Role
            };
        }
    }
}
using MedicalArchive.API.Application.DTOs.MedicalCertificateDTOs;
using MedicalArchive.API.Application.Interfaces;
using MedicalArchive.API.Domain.Entities;
using MedicalArchive.API.Domain.Interfaces;

namespace MedicalArchive.API.Application.Services
{
    public class MedicalCertificateService : IMedicalCertificateService
    {
        private readonly IMedicalCertificateRepository _certificateRepository;
        private readonly IFileService _fileService;

        public MedicalCertificateService(
            IMedicalCertificateRepository certificateRepository,
            IFileService fileService)
        {
            _certificateRepository = certificateRepository;
            _fileService = fileService;
        }

        public async Task<MedicalCertificateDto> GetByIdAsync(int id, int userId)
        {
            var certificate = await _certificateRepository.GetByIdAsync(id);

            if (certificate == null || certificate.UserId != userId)
            {
                throw new Exception($"Медичну довідку з ID {id} не знайдено");
            }

            return MapToDto(certificate);
        }

        public async Task<IEnumerable<MedicalCertificateDto>> GetAllByUserIdAsync(int userId)
```

```
        {
            var certificates = await _certificateRepository.GetByUserIdAsync(userId);
            return certificates.Select(MapToDto);
        }

        public async Task<IEnumerable<MedicalCertificateDto>> SearchByTitleAsync(int userId, string searchTerm)
        {
            var certificates = await _certificateRepository.SearchByTitleAsync(userId, searchTerm);
            return certificates.Select(MapToDto);
        }

        public async Task<IEnumerable<MedicalCertificateDto>> GetByDateRangeAsync(int userId, DateTime startDate, DateTime endDate)
        {
            var certificates = await _certificateRepository.GetByDateRangeAsync(userId, startDate, endDate);
            return certificates.Select(MapToDto);
        }

        public async Task<MedicalCertificateDto> CreateAsync(int userId, MedicalCertificateCreateDto certificateDto)
        {
            // Завантаження файлу, якщо він є
            string filePath = string.Empty;
            if (certificateDto.Document != null)
            {
                filePath = await _fileService.SaveFileAsync(certificateDto.Document, "certificates", userId);
            }

            // Створення об'єкту медичної довідки
            var certificate = new MedicalCertificate
            {
                Title = certificateDto.Title,
                IssueDate = certificateDto.IssueDate,
                Description = certificateDto.Description,
                UserId = userId,
                DocumentFilePath = filePath
            };

            // Збереження в репозиторій
            var createdCertificate = await _certificateRepository.AddAsync(certificate);

            return MapToDto(createdCertificate);
        }

        public async Task<MedicalCertificateDto> UpdateAsync(int id, int userId, MedicalCertificateUpdateDto certificateDto)
        {
            // Перевіряємо, чи існує медична довідка
            var certificate = await _certificateRepository.GetByIdAsync(id);

            if (certificate == null || certificate.UserId != userId)
            {
                throw new Exception($"Медичну довідку з ID {id} не знайдено");
            }

            // Якщо є новий файл, видаляємо старий і завантажуємо новий
            if (certificateDto.Document != null)
            {
                if (!string.IsNullOrEmpty(certificate.DocumentFilePath))
                {
                    await _fileService.DeleteFileAsync(certificate.DocumentFilePath);
                }
```

```
                certificate.DocumentFilePath = await
_fileService.SaveFileAsync(certificateDto.Document,
"certificates", userId);
            }

            // Оновлюємо дані
            certificate.Title = certificateDto.Title;
            certificate.IssueDate =
certificateDto.IssueDate;
            certificate.Description =
certificateDto.Description;

            // Зберігаємо зміни
            await
_certificateRepository.UpdateAsync(certificate);

            return MapToDto(certificate);
        }

        public async Task<bool> DeleteAsync(int
id, int userId)
        {
            // Перевіряємо, чи існує медична
довідка
            var certificate = await
_certificateRepository.GetByIdAsync(id);

            if (certificate == null || certificate.UserId
!= userId)
            {
                throw new Exception($"Медичну
довідку з ID {id} не знайдено");
            }

            // Видаляємо файл, якщо він є
            if
(!string.IsNullOrEmpty(certificate.DocumentFilePath))
            {
                await
_fileService.DeleteFileAsync(certificate.DocumentFileP
ath);
            }

            // Видаляємо довідку
            await
_certificateRepository.DeleteAsync(id);

            return true;
        }

        // Мапінг з доменної моделі в DTO
        private MedicalCertificateDto
MapToDto(MedicalCertificate certificate)
        {
            return new MedicalCertificateDto
            {
                Id = certificate.Id,
                Title = certificate.Title,
                IssueDate = certificate.IssueDate,
                Description = certificate.Description,
                UserId = certificate.UserId,
                DocumentFilePath =
certificate.DocumentFilePath
            };
        }
    }
}
using
MedicalArchive.API.Application.DTOs.AuthDTOs;
using
MedicalArchive.API.Application.DTOs.UserDTOs;
using
MedicalArchive.API.Application.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace MedicalArchive.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService
_authService;

        public AuthController(IAuthService
authService)
        {
            _authService = authService;
```

```
        }

        [HttpPost("register")]
        public async Task<ActionResult<UserDto>> Register([FromBody] UserRegistrationDto registrationDto)
        {
            try
            {
                var result = await _authService.RegisterAsync(registrationDto);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPost("login")]
        public async Task<ActionResult<TokenDto>> Login([FromBody] UserLoginDto loginDto)
        {
            try
            {
                var result = await _authService.LoginAsync(loginDto);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
using MedicalArchive.API.Application.DTOs.UserDTOs;
using MedicalArchive.API.Application.Interfaces;
using MedicalArchive.API.Domain.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace MedicalArchive.API.Controllers
{
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    public class UsersController : ControllerBase
    {
        private readonly IUserService _userService;

        public UsersController(IUserService userService)
        {
            _userService = userService;
        }

        [HttpGet("profile")]
        public async Task<ActionResult<UserDto>> GetProfile()
        {
            try
            {
                var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");
                var result = await _userService.GetByIdAsync(userId);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPut("profile")]
        public async Task<ActionResult<UserDto>> UpdateProfile([FromBody] UserUpdateDto updateDto)
        {
            try
            {
```

```
                var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");
                var result = await _userService.UpdateUserAsync(userId, updateDto);
                return Ok(result);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPost("change-password")]
        public async Task<ActionResult> ChangePassword([FromBody] ChangePasswordDto model)
        {
            try
            {
                var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");
                var result = await _userService.ChangePasswordAsync(userId, model.CurrentPassword, model.NewPassword);
                return Ok(new { success = result });
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpDelete("delete-account")]
        public async Task<ActionResult> DeleteAccount()
        {
            try
            {
                var userId = int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ?? "0");
                var result = await _userService.DeleteUserAsync(userId);
                return Ok(new { success = result });
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }

    public class ChangePasswordDto
    {
        public required string CurrentPassword { get; set; }
        public required string NewPassword { get; set; }
    }
}
```