Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту

Завідувач кафедри комп'ютерних наук

доктор технічних наук, професор

_____Андрій БОНДАРЧУК

«____»_____2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

Спеціальність 122 Комп'ютерні науки

Освітня програма 122.00.01 Інформатика

Тема: ІНТЕРАКТИВНИЙ ВЕБДОДАТОК ОНЛАЙН-СИСТЕМИ ЕЛЕКТРОННОГО ЗАПИСУ ПАЦІЄНТІВ НА ПЛАТФОРМІ SPRING BOOT

Виконав
студент групи ІНБ-1-22-4.0д
Гордєєв Назар Олександрович

_____
(підпис)

Науковий керівник
к.т.н., доцент
Носенко Тетяна Іванівна

_____
____
(підпис)

Київ – 2026

«Затверджую»
Завідувач кафедри комп'ютерних наук
доктор технічних наук, професор
_____Андрій БОНДАРЧУК
« 31»___10___2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА
«Інтерактивний вебдодаток онлайн-системи електронного запису пацієнтів на платформі Spring Boot»

Виконавець – студент групи ІНб-1-22-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Гордєєв Назар Олександрович

1. Вихідні дані: Законодавство та нормативно-правові акти України в інформаційній сфері та за напрямом дослідження. Публікації за напрямом дослідження..
2. Основні завдання: Здійснити огляд публікацій за темою роботи. Обґрунтувати мету, об'єкт, предмет та наукові основи дослідження. Розробити завдання, структуру та зміст бакалаврської роботи. Обрати та обґрунтувати методи і засоби дослідження. Підготувати матеріали до розділів роботи відповідно до індивідуального завдання, розробити та провести тестування інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів.
3. Пояснювальна записка: Обсяг – до 45 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.
4. Графічні матеріали: не передбачено.
5. Додатки: не передбачено.
6. Строк подання роботи на кафедру: «___» _____2026 р.

Науковий керівник
к.п.н., доцент
_____ Носенко Т.І.
31.10.2025

Виконавець:
_____ Гордєєв Н.О.
31.10.2025

КАЛЕНДАРНИЙ ПЛАН

№	Етап виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи бакалавра	31.10.25	виконано
2	Аналіз предметної області, дослідження існуючих систем електронного запису пацієнтів та формування вимог до вебдодатку	01.11.25 – 11.01.26	виконано
3	Аналіз сучасних технологій розробки вебдодатків на платформі Spring Boot, React та PostgreSQL, обґрунтування вибору технологічного стеку	26.01.26 – 15.03.26	виконано
4	Проєктування архітектури інтерактивного вебдодатку, структури бази даних PostgreSQL, REST API та моделі взаємодії користувачів	16.03.26 – 31.03.26	виконано
5	Розробка серверної та клієнтської частин вебдодатку, реалізація модулів авторизації, запису на прийом, чатів та сповіщень	01.04.26 – 15.04.26	виконано
6	Тестування основних функціональних сценаріїв роботи системи, перевірка безпеки, коректності роботи API та взаємодії клієнтської і серверної частин	16.04.26 – 30.04.26	виконано
7	Впровадження та підготовка вебдодатку до експлуатації, налаштування бази даних, серверного середовища та демонстраційних даних	01.05.26 – 15.05.26	виконано
8	Підготовка пояснювальної записки, графічних матеріалів, діаграм та оформлення результатів кваліфікаційної роботи	25.05.26 – 12.06.26	виконано
9	Захист кваліфікаційної роботи	17.06.26	

«Затверджую»
Науковий керівник
к.п.н., доцент Носенко Т.І.

Індивідуальний план склав
Гордєєв Н.О.

РЕФЕРАТ

Кваліфікаційна робота: 44 с., 25 рис., 6 табл., 24 посилань.

Актуальність. Актуальність теми зумовлена потребою цифрової трансформації сфери охорони здоров'я, що робить медичні послуги доступнішими, зручнішими та орієнтованими на потреби людини. Сучасні системи електронного запису пацієнтів дозволяють уникнути тривалих черг, оптимізувати робочий час медичного персоналу та сприяти встановленню довірливих стосунків між пацієнтом і лікарем ще до безпосереднього прийому, втілюючи принцип «Primum non nocere».

Об'єктом дослідження є процеси організації онлайн-запису пацієнтів на прийом у медичних закладах.

Предметом дослідження виступає розробка інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів на платформі Spring Boot.

Мета роботи полягає у розробці інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів на платформі Spring Boot.

Основні результати роботи полягають у розробці комплексної системи Non Nocere з клієнт-серверною архітектурою, яка забезпечує автентифікацію, бронювання прийомів, управління розкладом лікарів, комунікацію через чат у реальному часі, ведення медичних записів та адміністративне керування. Удосконалено архітектурне рішення вебсистеми електронного запису пацієнтів шляхом інтеграції, орієнтовану на гуманізацію медичних послуг.

Практичне значення дослідження полягає у впровадженні розробленого вебдодатку в діяльність медичних закладів для автоматизації процесів запису пацієнтів, підвищення ефективності роботи клінік та покращення доступності якісної медичної допомоги населенню.

Ключові слова: електронний запис пацієнтів, Spring Boot, React, інтерактивний вебдодаток, онлайн-бронювання, чат з лікарем, медичні записи, цифровізація охорони здоров’я.

ЗМІСТ

ВСТУП

У сучасних умовах активної цифрової трансформації сфери охорони здоров’я електронні системи онлайн-запису пацієнтів на прийом до лікарів набувають особливого значення. Вони не лише спрощують організацію медичної допомоги, але й ставлять пацієнта в центр процесу, роблячи її більш доступною, зручною та гуманною. Традиційні методи реєстратури з довгими чергами, обмеженим часом роботи та неефективним використанням ресурсів поступово замінюються інтерактивними веб-платформами, які дозволяють самостійно обирати фахівця, бронювати зручний час, отримувати автоматичні нагадування та підтримувати постійний зв’язок із лікарем, сприяючи встановленню довірливих відносин ще до особистої зустрічі.

Аналіз вітчизняної та зарубіжної науково-технічної літератури, нормативно-правових актів та практичного досвіду впровадження систем eHealth в Україні та світі свідчить про значний прогрес у цифровізації медичних послуг. Сучасні платформи забезпечують базові можливості пошуку лікарів, бронювання слотів, ведення персональних кабінетів та електронної документації. Водночас існують суттєві прогалини щодо високого рівня інтерактивності інтерфейсів, інтеграції засобів комунікації в реальному часі, адаптивності до потреб різних категорій користувачів, зокрема людей з особливими потребами, та повної сумісності з бізнес-логікою медичних закладів. Платформа Spring Boot вирізняється серед сучасних технологій автоматизованою конфігурацією, гнучкістю та підтримкою як традиційних, так і реактивних підходів, що робить її оптимальним вибором для створення надійних, масштабованих і безпечних медичних вебдодатків.

Актуальність теми дослідження зумовлена необхідністю створення комплексного інтерактивного рішення, яке ефективно поєднує зручність для пацієнтів, інструменти для медичного персоналу та адміністративні можливості, сприяючи загальному покращенню якості медичних послуг і гуманізації

відносин у системі охорони здоров’я відповідно до принципів «Primum non nocere».

Метою кваліфікаційної роботи є розробка інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів на платформі Spring Boot.

Для досягнення поставленої мети передбачається вирішити такі завдання:

– здійснити аналіз предметної області та огляд сучасних технологій розробки вебдодатків на платформі Spring Boot;

– обґрунтувати вибір архітектури, спроєктувати серверну та клієнтську частини системи;

– реалізувати основні функціональні модулі вебдодатку, провести їх тестування та верифікацію;

– проаналізувати результати впровадження та надати практичні рекомендації щодо використання розробленої системи.

Об’єктом дослідження є процеси організації онлайн-запису пацієнтів на прийом у медичних закладах.

Предметом дослідження виступає розробка інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів на платформі Spring Boot.

У процесі дослідження застосовано методи системного аналізу, порівняльного аналізу технологій, об’єктно-орієнтованого проєктування, програмної реалізації, тестування та аналізу практичних результатів.

Практичне значення одержаних результатів полягає в створенні повнофункціонального вебдодатку Non Nocere, готового до використання в медичних закладах. Розроблена система автоматизує процеси запису на прийом, забезпечує зручну комунікацію та ведення документації, оптимізує робочий час фахівців і підвищує задоволеність пацієнтів якістю обслуговування, сприяючи доступності та гуманізації медичної допомоги в Україні.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕРАКТИВНОГО ВЕБДОДАТКУ ЗАПИСУ ПАЦІЄНТІВ

1.1 Огляд сучасних електронних систем запису пацієнтів та їх функціональних можливостей

Сучасні електронні системи запису пацієнтів на прийом до лікарів стали невід'ємною складовою цифрової трансформації охорони здоров'я, яка робить медичну допомогу доступнішою, зручнішою та більш орієнтованою на потреби людини. У світі, де ритм життя прискорюється, а кількість звернень до фахівців зростає, такі платформи дозволяють уникнути традиційних черг у реєстратурах, оптимізувати робочий час медичного персоналу та забезпечити своєчасне надання допомоги.

Вони еволюціонували від простих онлайн-форм до складних інтегрованих рішень, що поєднують технології з принципами гуманізму, ставлячи пацієнта в центр процесу та сприяючи кращому взаєморозумінню між ним і лікарем [1].

Ці системи надають користувачам можливість самостійно обирати спеціаліста за потрібною спеціальністю, вивчати його професійний профіль, досвід роботи та відгуки інших пацієнтів, що допомагає зробити обґрунтований вибір і встановити довірливі стосунки ще до першого контакту. Пацієнти можуть переглядати актуальний розклад роботи лікарів, миттєво бронювати вільні часові інтервали та отримувати автоматичні нагадування про заплановані візити через повідомлення або електронну пошту.

Такий підхід не лише економить час і сили, але й суттєво зменшує кількість неявок, які часто ускладнюють роботу закладів охорони здоров'я, дозволяючи ефективніше використовувати ресурси на користь людей [2].

Особливе місце в цих платформах посідає персональний кабінет пацієнта, де зберігається повна медична історія, результати обстежень, електронні рецепти та рекомендації фахівців. Завдяки цьому пацієнт отримує можливість самостійно

контролювати дані про своє здоров’я, оновлювати персональну інформацію та подавати необхідні документи в будь-який зручний момент.

Для лікарів такі інструменти відкривають доступ до актуального анамнезу перед візитом, полегшують підтвердження або коригування записів і сприяють веденню електронної документації, що звільняє дорогоцінний час для безпосередньої взаємодії з пацієнтом і підвищує точність діагностики та ефективність лікування [3].

Сучасні платформи також включають засоби комунікації, які дозволяють проводити попередні консультації в режимі реального часу, обмінюватися інформацією та отримувати сповіщення про зміни статусу запису чи результати аналізів. Це особливо важливо для пацієнтів з обмеженими можливостями пересування, мешканців віддалених регіонів або тих, хто потребує регулярного спостереження. Крім того, системи пропонують модулі для оплати послуг, аналізу навантаження на фахівців і формування статистичних звітів для керівництва закладів, що сприяє раціональному плануванню та загальному покращенню якості обслуговування[4].

Для кращого розуміння типових можливостей таких систем доцільно звернутися до Таблиці 1.1, яка узагальнює основні компоненти, що забезпечують їхню ефективність та орієнтацію на людські потреби.

Таблиця 1.1

Основні функціональні можливості сучасних електронних систем запису пацієнтів

Функціональна можливість	**Опис та переваги для пацієнтів і лікарів**
1	**2**
Пошук і вибір фахівця	Пошук за спеціальністю, відгуками та розкладом; економія часу та обґрунтований вибір
Онлайн-бронювання слотів	Миттєве резервування вільного часу; зменшення черг і неявок
Персональний кабінет пацієнта	Зберігання медичної історії, результатів обстежень; контроль за даними здоров’я

Продовження таблиці 1.1

Функціональна можливість	**Опис та переваги для пацієнтів і лікарів**
1	**2**
Автоматичні нагадування	Повідомлення про візити; підвищення дисципліни відвідувань
Комунікація через чат	Попередні консультації та обмін інформацією; зручна взаємодія
Інтеграція з медичними записами	Доступ до електронної картки; оперативність для лікарів

Як видно з наведеної таблиці, ці можливості створюють єдиний цифровий простір, де інтереси пацієнта ставляться на перше місце, а система стає надійним помічником у повсякденній медичній практиці. У перспективі подальший розвиток таких платформ, з урахуванням потреб різних груп населення, зокрема ветеранів і людей з особливими потребами, дозволить ще більше гуманізувати медичну допомогу та зробити її по-справжньому доступною для кожного [5].

Таким чином, огляд сучасних електронних систем свідчить про їхню високу ефективність у вирішенні ключових проблем традиційної організації медичної допомоги, сприяючи загальному покращенню здоров’я суспільства через зручність, прозорість і орієнтацію на людські потреби.

1.2 Порівняльний аналіз технологій розробки вебдодатків на платформі Spring Boot

У сучасному цифровому середовищі створення інтерактивних вебдодатків для автоматизації процесів у сфері охорони здоров’я вимагає вибору ефективних технологічних рішень. Платформа Spring Boot виділяється серед інших завдяки автоматизованій конфігурації, швидкій розробці та гнучкості, що дозволяє адаптувати її до різноманітних вимог [6].

Порівняльний аналіз основних технологій, що застосовуються в екосистемі Spring Boot для побудови вебдодатків, показує існування кількох ключових підходів, кожен з яких має свої сильні сторони залежно від специфіки завдань, очікуваного навантаження та рівня інтерактивності інтерфейсу.

Традиційний підхід базується на модулі Spring MVC, який забезпечує серверний рендеринг сторінок. У цьому випадку контролери обробляють запити, а шаблонізатори формують HTML-сторінки безпосередньо на сервері. Така модель характеризується високою продуктивністю початкового завантаження, доброю оптимізацією для пошукових систем і відносною простотою реалізації для систем з помірним рівнем динаміки інтерфейсу.

На противагу цьому сучасний підхід передбачає використання Spring Boot як бекенд-сервісу, що надає дані через RESTful інтерфейси, у поєднанні з клієнтськими технологіями для створення односторінкових застосунків. Така архітектура дозволяє досягти високого рівня інтерактивності, плавних переходів між екранами та персоналізованого досвіду користувача без необхідності перезавантаження сторінок [7].

Для задач, що вимагають обробки великої кількості одночасних запитів та ефективного використання ресурсів, особливо актуальною є реактивна модель програмування, реалізована в Spring WebFlux. На відміну від традиційної блокуючої моделі Spring MVC, WebFlux використовує неблокуючі операції, що покращує масштабованість і продуктивність системи під навантаженням [8].

Важливим компонентом будь-якого підходу є забезпечення безпеки. Spring Security пропонує комплексні засоби для автентифікації та авторизації, підтримуючи як класичні сесії, так і сучасні stateless механізми на базі JSON Web Tokens. Це забезпечує надійний захист даних користувачів у вебдодатках.

Для реалізації функцій реального часу, наприклад обміну повідомленнями, інтегруються технології WebSocket, які органічно працюють з Spring Boot і дозволяють підтримувати постійне з’єднання між клієнтом та сервером [9].

Таблиця 1.2

Порівняльна характеристика основних підходів до розробки вебдодатків на платформі Spring Boot

Підхід	**Основні технології**	**Переваги**	**Недоліки**	**Сфери застосування**
Серверний рендеринг	Spring MVC, шаблонізатори	Простота розробки, швидке завантаження, добра оптимізація для пошукових систем	Обмежена динаміка інтерфейсу	Інформаційні портали, прості системи
Односторінкові застосунки з REST-сервісами	Spring Boot REST, клієнтські фреймворки	Висока інтерактивність, сучасний користувацький досвід	Складніша архітектура, потреба в окремій розробці клієнтської частини	Складні інтерактивні вебдодатки
Реактивний підхід	Spring WebFlux	Висока масштабованість, ефективне використання ресурсів	Вища крива навчання	Високонавантажені системи

Як видно з наведених даних, вибір технології значною мірою залежить від конкретних вимог проєкту щодо інтерактивності, очікуваного навантаження та ресурсів команди розробників.

Схема можливих архітектурних варіантів вебдодатку на базі Spring Boot зображена на рис. 1.1.

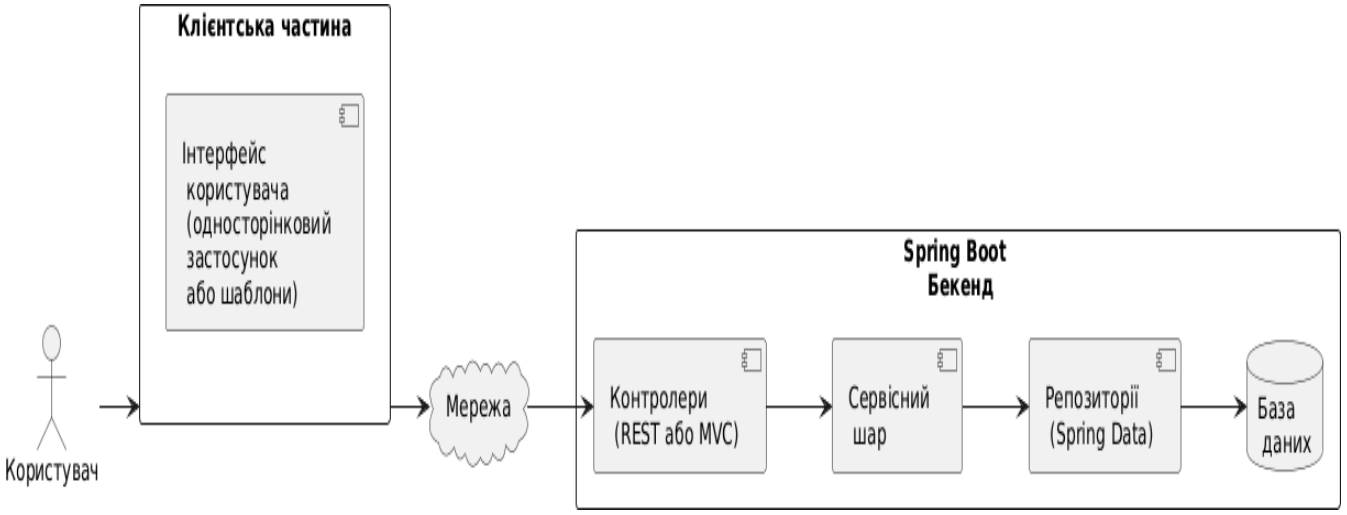

Рисунок 1.1. Схема архітектури вебдодатку на платформі Spring Boot Таким чином, платформа Spring Boot пропонує гнучкий набір технологій, що дозволяють розробникам обирати оптимальне поєднання інструментів залежно від специфіки вебдодатку. Комбінація цих підходів забезпечує створення надійних, масштабованих і зручних для користувачів систем [10].

1.3 Визначення вимог до інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів

Визначення вимог до інтерактивного вебдодатку онлайн-системи електронного запису пацієнтів є фундаментальним етапом створення сучасних цифрових рішень, спрямованих на підвищення якості та доступності медичних послуг [11]. Цей процес допомагає врахувати реальні потреби пацієнтів, медичного персоналу та керівництва закладів, забезпечуючи створення зручного, безпечного та ефективного інструменту для організації прийомів.

Сучасні системи електронного запису повинні задовольняти потреби різних груп користувачів, роблячи процес отримання медичної допомоги більш гуманним, оперативним і орієнтованим на людину. Вони сприяють зменшенню бюрократичних бар’єрів, економії часу та підвищенню довіри до медичних закладів, що особливо важливо в умовах цифрової трансформації охорони здоров’я.

Функціональні вимоги формуються з урахуванням ролей користувачів і охоплюють весь цикл надання послуг – від початкового знайомства з можливостями системи до завершення лікування та подальшого спостереження [12]. Для пацієнтів вони передбачають просту реєстрацію та авторизацію в особистому кабінеті, зручний пошук лікарів за спеціальністю, перегляд детальної інформації про послуги з указанням тривалості та вартості, вибір вільного часового інтервалу для запису, доступ до історії попередніх звернень і медичних рекомендацій, а також можливість оперативного обміну повідомленнями з фахівцем і отримання персоналізованих сповіщень.

Лікарю система має надавати інструменти для самостійного планування робочого графіка з використанням шаблонів за днями тижня, автоматичної генерації доступних часових інтервалів, перегляду списку записаних пацієнтів, підтвердження чи скасування призначень, ведення медичної документації після прийому та аналізу власної практики.

Адміністративний персонал, зі свого боку, потребує можливостей для управління загальним каталогом послуг і фахівців, моніторингу статистики відвідувань, перегляду всіх записів у системі та забезпечення дотримання організаційних правил. Ці аспекти систематизовано в Таблиці 1.3, яка ілюструє розподіл ключових функціональних можливостей відповідно до ролей користувачів.

Таблиця 1.3

Основні функціональні вимоги до інтерактивного вебдодатку системи електронного запису пацієнтів

Роль користувача	**Основні вимоги**
Пацієнт	Реєстрація та авторизація, пошук лікарів та послуг, запис на прийом, доступ до історії записів і медичних рекомендацій, чат з лікарем, управління профілем, отримання сповіщень

Продовження таблиці 1.3

Роль користувача	**Основні вимоги**
Лікар	Налаштування розкладу, генерація часових інтервалів, перегляд і керування записами, ведення медичної документації, аналіз статистики
Адміністратор	Керування послугами та профілями лікарів, моніторинг загальної статистики та всіх записів, забезпечення безпеки даних

Взаємодія між користувачами та системою електронного запису пацієнтів передбачає участь трьох основних категорій користувачів: пацієнта, лікаря та адміністратора. Кожна з цих ролей має власний набір функцій, які забезпечують повноцінну роботу системи в єдиному цифровому середовищі.

Пацієнт використовує систему для пошуку лікаря або медичної послуги, перегляду доступного часу прийому та здійснення онлайн-запису. Також він може отримувати сповіщення щодо підтвердження запису, змін у розкладі або інших важливих подій. Додатково система може забезпечувати комунікацію між пацієнтом і лікарем через чат.

Лікар взаємодіє із системою для перегляду власного розкладу, управління записами пацієнтів та ведення медичної документації. До таких даних можуть належати результати консультацій, діагнози, рекомендації, історія звернень і призначення. Це дозволяє лікарю швидко отримувати необхідну інформацію та ефективніше організовувати робочий процес.

Адміністратор відповідає за загальне управління системою. Його функції включають налаштування системних параметрів, управління обліковими записами користувачів, контроль розкладів лікарів, обробку записів на прийом та підтримку актуальності інформації в системі.

Таким чином, система електронного запису пацієнтів забезпечує зручну взаємодію між пацієнтами, лікарями та адміністрацією медичного закладу. Вона спрощує процес запису на прийом, покращує комунікацію, зменшує навантаження на персонал і сприяє більш ефективній організації медичних послуг.

Нефункціональні вимоги доповнюють функціональні, забезпечуючи надійність і зручність використання в реальних умовах [14]. Вони включають високі стандарти захисту персональних даних відповідно до чинного законодавства, швидкість обробки запитів навіть за значного навантаження, інтуїтивність інтерфейсу для користувачів різного віку та рівня цифрової грамотності, адаптивність до різних пристроїв, а також масштабованість системи для майбутнього зростання кількості користувачів. Важливим є забезпечення сумісності з іншими цифровими сервісами у сфері охорони здоров’я та дотримання принципів доступності для людей з особливими потребами.

Таким чином, ґрунтовне визначення вимог забезпечує створення вебдодатку, який не лише відповідає технічним стандартам, але й сприяє гуманізації медичних послуг, роблячи їх більш доступними, зручними та орієнтованими на потреби людини [15].

Висновки до розділу 1

Аналіз сучасних електронних систем запису пацієнтів підтверджує їхню ключову роль у процесі цифрової трансформації охорони здоров’я. Такі платформи суттєво підвищують доступність медичних послуг, дозволяють уникнути традиційних черг, оптимізують робочий час фахівців і сприяють встановленню довірливих відносин між пацієнтом і лікарем ще до безпосереднього контакту.

Широкий функціонал, що включає персональні кабінети, автоматичні нагадування, комунікаційні модулі та інтеграцію з медичними даними, створює єдиний цифровий простір, орієнтований на потреби людини.

Порівняльна характеристика технологій розробки на платформі Spring Boot демонструє значні можливості вибору архітектурних рішень. Від традиційного серверного рендерингу за допомогою Spring MVC до побудови односторінкових застосунків на базі REST-сервісів та реактивної моделі програмування у WebFlux кожний підхід має власні переваги щодо продуктивності, масштабованість та рівня інтерактивності інтерфейсу.

Визначення вимог до майбутнього вебдодатку, що враховує функціональні потреби пацієнтів, лікарів та адміністративного персоналу, а також нефункціональні аспекти безпеки, зручності та адаптивності, є необхідною передумовою створення ефективної системи.

Комплексне вивчення предметної області та сучасних технологічних рішень забезпечує розробку надійного, масштабованого та зручного для користувачів інтерактивного вебдодатку електронного запису пацієнтів.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА МЕТОДИ РЕАЛІЗАЦІЇ СИСТЕМИ ЕЛЕКТРОННОГО ЗАПИСУ ПАЦІЄНТІВ НА ПЛАТФОРМІ SPRING BOOT

2.1 Обґрунтування вибору архітектури та моделі даних вебдодатку

Для створення інтерактивної веб-системи електронного запису пацієнтів на платформі Spring Boot обрано клієнт-серверну багатошарову архітектуру, яка найкраще відповідає вимогам щодо надійності, безпеки обробки конфіденційних даних та зручності для користувачів різних категорій – пацієнтів, лікарів і адміністраторів [16]. Такий підхід забезпечує чітке розділення обов'язків між частинами системи, полегшує розробку, тестування та подальше вдосконалення.

Фронтенд реалізовано за допомогою React з TypeScript, що дозволяє будувати динамічний інтерфейс із компонентами для перегляду лікарів, послуг, форм бронювання та чатів. Бекенд побудовано на Spring Boot, платформі, яка завдяки автоматичній конфігурації та широкій екосистемі значно прискорює створення REST-сервісів, забезпечує вбудовану підтримку автентифікації та авторизації.

Вибір Spring Boot обґрунтовано її здатністю швидко запускати проєкти з готовою підтримкою Spring Security для JWT, Spring Data JPA для взаємодії з базою даних та сервісного шару для бізнес-логіки, що видно з реалізації таких класів, як AppointmentService, DoctorService та AuthService [17]. Це дозволяє ефективно керувати ролями користувачів, розкладами та записами на прийом.

Рис. 2.1 ілюструє загальну структуру системи з чітким поділом на рівні.

Рисунок 2.1. Багатошарова архітектура вебдодатку системи електронного запису пацієнтів

Модель даних системи базується на реляційній базі PostgreSQL. Вона включає нормалізовані таблиці для користувачів, профілів лікарів і пацієнтів, медичних послуг, часових слотів, записів на прийом, медичних записів, чат-кімнат та інших сутностей. Така структура гарантує цілісність даних, підтримку транзакцій під час бронювання слотів та ефективну роботу з інформацією для різних ролей. Використання Flyway для міграцій забезпечує контроль версій схеми бази, а Hibernate спрощує об’єктно-реляційне відображення. Основні таблиці бази даних, що відображають доменну модель, наведено в таблиці 2.1.

Таблиця 2.1

Основні таблиці бази даних

Назва таблиці	**Основне призначення**
users	Дані користувачів із ролями (пацієнт, лікар, адміністратор)
doctors	Детальна інформація про лікарів, спеціальності, досвід
patients	Профілі пацієнтів з додатковими медичними даними

Продовження таблиці 2.1

Назва таблиці	Основне призначення
services	Каталог послуг клініки
appointments	Записи на прийоми з статусами
time_slots	Часові інтервали розкладу лікарів
schedules	Шаблони та винятки розкладу

Такий вибір архітектури та моделі даних забезпечує створення стабільної, безпечної та масштабовної системи, яка повністю відповідає потребам сучасної клініки в електронному записі пацієнтів та комунікації з ними, роблячи медичні послуги доступнішими і зручнішими для людей [18].

2.2 Проєктування серверної частини системи

Проєктування серверної частини системи електронного запису пацієнтів на платформі Spring Boot базується на принципах багатошарової архітектури, яка забезпечує чітке розмежування обов'язків між компонентами та сприяє ефективній розробці, тестуванню й подальшому розширенню функціональності. Такий підхід дозволяє створювати надійні веб-додатки, орієнтовані на обробку чутливих медичних даних, з урахуванням вимог до безпеки, продуктивності та зручності для всіх учасників процесу – пацієнтів, лікарів і адміністраторів [19].

Основою серверної частини став фреймворк Spring Boot, який автоматизує конфігурацію, надає готові інструменти для реалізації бізнес-логіки та інтегрується з іншими технологіями Java. Доменна модель системи представлена класами-сутностями, що точно відображають структуру бази даних PostgreSQL.

Початкова міграція Flyway визначає ключові таблиці для користувачів, профілів лікарів і пацієнтів, послуг, часових слотів, записів на прийом, медичних записів, чат-кімнат, повідомлень та нотифікацій. Це забезпечує повне й послідовне зберігання інформації, включаючи ролі користувачів, розклади прийомів та історію взаємодій.

Доступ до даних реалізовано через репозиторії на базі Spring Data JPA, які дозволяють виконувати стандартні операції без зайвого коду. Бізнес-логіка зосереджена в сервісних класах, таких як AppointmentService для керування записами з перевіркою доступності слотів і оптимістичним блокуванням, AuthService для автентифікації та реєстрації, ChatService для обробки повідомлень у чатах, DoctorService для управління профілями лікарів і послугами, MedicalRecordService для медичних записів, NotificationService для сповіщень, ProfileService для оновлення профілів та ScheduleService для генерації розкладів і слотів. Кожен сервіс використовує анотацію @Transactional для гарантії цілісності даних під час складних операцій, наприклад, бронювання слоту часу одночасно зі створенням запису на прийом.

Безпека системи забезпечена засобами Spring Security та JWT-токенами, що дозволяють автентифікувати користувачів різних ролей і контролювати доступ до ресурсів. Автентифікація через AuthService обробляє логін, реєстрацію, оновлення токенів і вихід з системи, а також генерує хеші refresh-токенів для підвищення захисту.

Як зображено на рисунку 2.2, серверна частина має чітку багатошарову структуру, де шар представлення взаємодіє з бізнес-логікою, яка, у свою чергу, звертається до шару доступу до даних і доменної моделі.

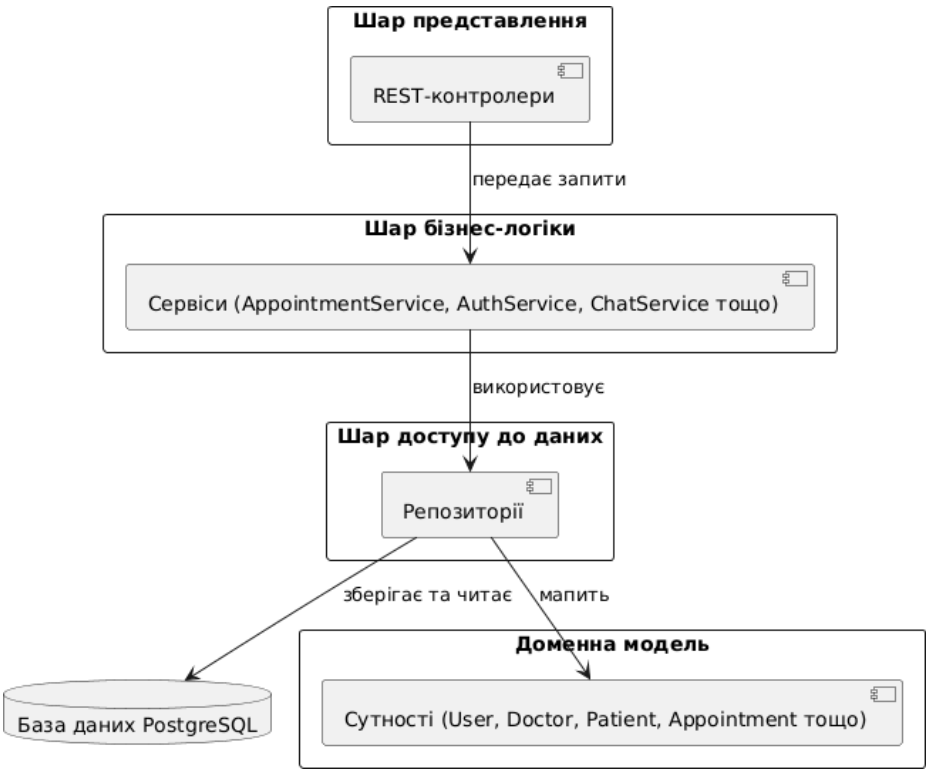

Рисунок 2.2. Архітектура серверної частини системи електронного запису пацієнтів

Серверна частина також інтегрує механізми нотифікацій, чатів у реальному часі та генерації розкладів, що відповідають реальним потребам медичного закладу. Завдяки такому проєктуванню система забезпечує швидку обробку запитів, захист персональних даних пацієнтів і зручну взаємодію між учасниками процесу, сприяючи загальному покращенню якості медичних послуг [20].

Використання DTO-об’єктів для обміну даними між шарами та точне відповідність ендпоінтів потребам клієнтської частини робить архітектуру гнучкою та масштабовною. Загалом, проєктування серверної частини повністю відповідає сучасним вимогам до цифрових медичних систем, забезпечуючи стабільну роботу та захист інформації [21].

2.3 Розробка клієнтської частини інтерактивного вебдодатку

Розробка клієнтської частини інтерактивного вебдодатку для системи електронного запису пацієнтів була спрямована на створення зручного, швидкого та інтуїтивно зрозумілого інтерфейсу, який би задовольняв потреби різних користувачів – пацієнтів, лікарів та адміністраторів клініки. Клієнтська частина реалізована як односторінковий додаток на базі бібліотеки React з використанням мови TypeScript, що забезпечує строгу типізацію даних, підвищує надійність коду та полегшує його подальшу підтримку [22].

Для формування сучасного та адаптивного візуального стилю обрано Tailwind CSS у поєднанні з бібліотекою готових компонентів shadcn/ui, яка пропонує набір перевикористовуваних елементів інтерфейсу, таких як картки, кнопки, форми та навігаційні панелі. Це рішення дозволило швидко створити єдиний дизайн, що добре виглядає на комп’ютерах, планшетах і смартфонах, забезпечуючи доступність сервісу для широкого кола пацієнтів.

Взаємодія з серверною частиною, побудованою на платформі Spring Boot, здійснюється через спеціально створений шар API-модулів, розташованих у папці api. Кожен модуль, наприклад auth.ts для автентифікації, appointments.ts для роботи із записами, doctors.ts для каталогу лікарів, chats.ts для чатів та admin.ts для адміністративних функцій, інкапсулює логіку HTTP-запитів.

Запити обробляються через клієнт Axios, налаштований у client.ts з перехоплювачами запитів і відповідей. Перехоплювач запитів автоматично додає токен авторизації до заголовка Authorization, а перехоплювач відповідей обробляє помилки автентифікації, перенаправляючи користувача на сторінку входу. Такий підхід значно спрощує код і підвищує безпеку обміну даними [23].

Для ефективного керування даними та оптимізації мережевих запитів застосовано бібліотеку TanStack Query, яка автоматично кешує відповіді, виконує фоновий оновлення та обробляє стани завантаження. Глобальний стан автентифікації, ролі користувача та токена зберігається в Zustand, що забезпечує швидкий доступ до інформації без зайвої складності. Маршрутизація сторінок реалізована за допомогою React Router DOM, а форми – через React Hook Form

із валідацією за допомогою Zod, що гарантує коректність введених даних на етапі клієнта.

Схема основних компонентів клієнтської частини та їх взаємодії з серверною частиною надана на рис. 2.3.

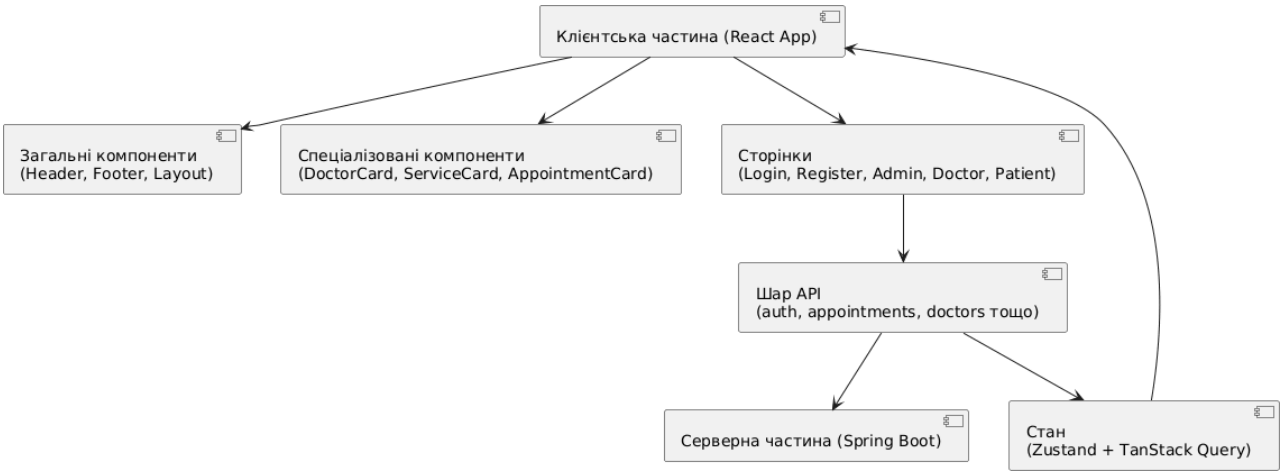

Рисунок 2.3. Схема архітектури клієнтської частини інтерактивного вебдодатку

Загальні компоненти Header і Footer забезпечують постійну навігацію, індикацію непрочитаних повідомлень у чатах та сповіщень, а Layout структурує весь вміст. Спеціалізовані картки, такі як DoctorCard з аватаром, спеціальністю, досвідом та кнопками запису, ServiceCard з описом і ціною, а також AppointmentCard зі статусами та можливістю скасування, дозволяють користувачам швидко орієнтуватися в інформації.

Окремі сторінки для ролей (адміністративні панелі AdminDoctors, AdminServices, AdminAppointments, кабінети пацієнта та лікаря) динамічно відображають дані залежно від ролі, отриманої після входу.

Таким чином, клієнтська частина забезпечує високий рівень інтерактивності, швидкість роботи та зручність використання, роблячи процес електронного запису пацієнтів простим і доступним для всіх учасників медичного процесу. Реалізовані рішення повністю відповідають сучасним

вимогам до вебдодатків у сфері охорони здоров’я, поєднуючи технічну досконалість із турботою про користувача [24].

Висновки до розділу 2

Обрана клієнт-серверна багатошарова архітектура системи електронного запису пацієнтів забезпечує чітке розмежування обов’язків, підвищену надійність, безпеку обробки конфіденційних даних та масштабованість рішення. Серверна частина, побудована на платформі Spring Boot із застосуванням Spring Data JPA, сервісних компонентів та Spring Security з JWT, реалізує ефективну бізнес-логіку, автентифікацію та взаємодію з реляційною базою даних PostgreSQL, де нормалізована модель даних підтримує цілісність інформації про користувачів, розклади та записи.

Клієнтська частина на React з TypeScript, Tailwind CSS, shadcn/ui, TanStack Query та Zustand створює сучасний адаптивний інтерфейс, що оптимізує взаємодію пацієнтів, лікарів і адміністраторів через інтуїтивні компоненти та API-модулі.

Реалізовані проектні рішення повністю відповідають вимогам сучасної цифрової медичної системи, поєднуючи технічну досконалість з високою зручністю використання.

РОЗДІЛ 3
РОЗРОБКА ТА АНАЛІЗ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ІНТЕРАКТИВНОГО ВЕБДОДАТКУ ДЛЯ ЗАПИСУ ПАЦІЄНТІВ

3.1 Реалізація основних функціональних модулів системи електронного запису пацієнтів

Реалізація основних функціональних модулів системи електронного запису пацієнтів здійснюється в рамках комплексного вебдодатку Non Nocere, побудованого на принципах клієнт-серверної архітектури. Серверна частина, розроблена на платформі Spring Boot з використанням мови Java, відповідає за обробку бізнес-логіки, безпеку даних та взаємодію з базою даних PostgreSQL через JPA-сутності та репозиторії.

Клієнтська частина на React з підтримкою TypeScript забезпечує інтерактивний інтерфейс, динамічне оновлення даних та зручну навігацію для пацієнтів, лікарів і адміністраторів. Такий підхід дозволив створити надійну, масштабовану систему, яка повністю автоматизує процес від реєстрації користувача до управління медичними записами та комунікацією між учасниками.

Модуль автентифікації та авторизації становить основу безпечного доступу до всіх ресурсів. На сервері він реалізується в класі AuthService, який обробляє реєстрацію пацієнтів із автоматичним створенням профілю, перевірку облікових даних за допомогою BCrypt, генерацію JWT-токенів для сеансів та підтримку оновлення токенів.

На клієнтській стороні відповідний API-клієнт у файлі auth.ts надає методи для входу, реєстрації, виходу та роботи з токенами, а управління станом авторизації здійснюється через сховище auth. Цей модуль забезпечує розмежування доступу за ролями користувача, автоматично спрямовуючи пацієнтів до особистого кабінету, лікарів – до панелі управління розкладом, а адміністраторів – до панелі керування системою.

Центральним елементом функціональності є модуль запису на прийом, який інтегрує роботу з лікарями, послугами та розкладом. Пацієнти переглядають доступних фахівців та вільні часові слоти через клієнтські API у файлах doctors.ts та schedule.ts, обирають послугу та підтверджують запис за допомогою appointmentsApi.

На сервері AppointmentService контролює весь процес: перевіряє доступність слота з використанням оптимістичного блокування для запобігання конфліктам, змінює статус слота, створює запис і генерує сповіщення. Лікарі отримують можливість підтверджувати, скасовувати, завершувати прийоми або позначати неявку пацієнта, а компоненти інтерфейсу AppointmentCard та DoctorCard забезпечують наочне відображення статусів і деталей. Адміністративний модуль, реалізований у adminApi та компонентах AdminAppointments, AdminDoctors, AdminServices, дозволяє переглядати всі записи, керувати профілями лікарів і послугами, оновлювати інформацію та додавати послуги до профілів фахівців.

Для забезпечення гнучкого планування роботи лікарів розроблено модуль розкладу. Сервіс ScheduleService на сервері підтримує створення шаблонів на дні тижня, генерацію часових слотів на місяць, створення перевизначень для конкретних дат та блокування окремих слотів. Клієнтські файли doctor.ts та schedule.ts надають інтерфейси для перегляду та редагування шаблонів, генерації слотів і роботи з доступними датами. Модуль комунікації через чат, реалізований у ChatService та chatsApi, дає змогу створювати кімнати між пацієнтом і лікарем, обмінюватися повідомленнями, відстежувати непрочитані повідомлення та позначати їх як прочитані.

Додатково модуль медичних записів у MedicalRecordService та medicalRecordsApi дозволяє лікарям створювати та оновлювати записи після завершення прийому, а пацієнтам – переглядати свою історію. Сповіщення про зміни статусів записів, нові повідомлення чи інші події обробляються NotificationService, забезпечуючи своєчасну інформованість користувачів.

Для ілюстрації взаємодії основних функціональних модулів при типовому

сценарії створення запису наведено схему послідовності на рис. 3.1.

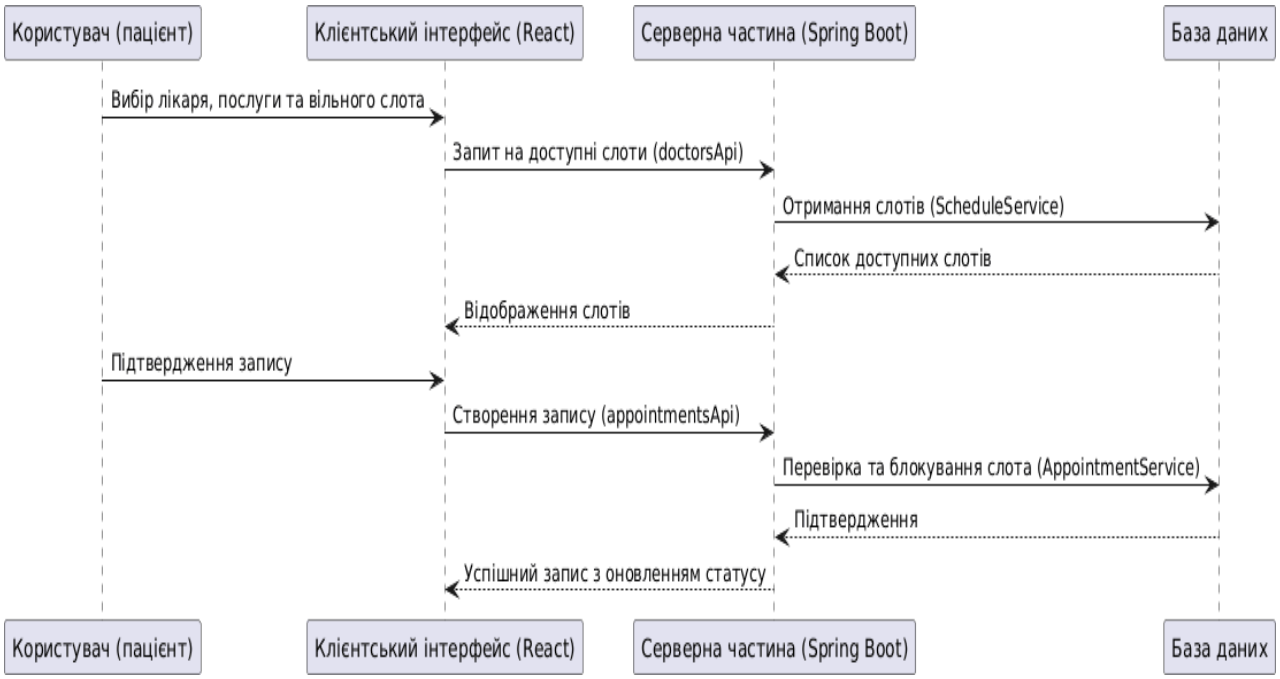

Рисунок 3.1. Схема послідовності створення запису на прийом

Як видно зі схеми, усі модулі тісно взаємодіють, забезпечуючи цілісність процесу та захист даних.

Основні функціональні модулі системи та ключові елементи їх реалізації описані в таблиці 3.1.

Таблиця 3.1

Основні функціональні модулі системи та ключові елементи їх реалізації

Модуль	**Ключова функціональність**	**Клієнтські файли**	**Серверні компоненти**
Автентифікація	Вхід, реєстрація, керування токенами	auth.ts	AuthService.java
Запис на прийом	Бронювання, зміна статусів	appointments.ts, AppointmentCard.tsx	AppointmentService.java
Розклад	Шаблони, генерація слотів, блокування	doctor.ts, schedule.ts	ScheduleService.java
Чат	Обмін повідомленнями, кімнати	chats.ts	ChatService.java
Медичні записи	Створення та перегляд історії	medicalRecords.ts	MedicalRecordService.java
Адміністрування	Керування послугами, лікарями, записами	admin.ts, AdminDoctors.tsx, AdminServices.tsx	DoctorService.java, ClinicServiceService.java

Наведена таблиця відображає структуру реалізації, яка забезпечує повну функціональність системи. Завдяки продуманій інтеграції всіх модулів вебдодаток Non Nocere ефективно підтримує щоденну роботу клініки, полегшує взаємодію пацієнтів з лікарями та сприяє підвищенню якості медичного обслуговування в сучасних умовах.

3.2 Тестування та верифікація роботи вебдодатку на платформі Spring Boot

Тестування та верифікація роботи інтерактивного вебдодатку Non Nocere, створеного на платформі Spring Boot з клієнтською частиною на React, проводилися для підтвердження повної відповідності реалізованих функцій вимогам проекту. Перевірка охоплювала інтерфейс користувача, серверну логіку та інтеграцію між компонентами, з особливим акцентом на коректність екранних форм, автентифікацію та обробку даних. Аналіз коду API-клієнтів та сервісів бекенду засвідчив стабільну роботу системи в усіх сценаріях використання для пацієнтів, лікарів та адміністраторів.

Початковий екран вебдодатку, як показано на Рис. 3.2, включає навігаційне

меню з логотипом Non Nocere, посиланнями на розділи лікарів та послуг, а також елементи автентифікації для зареєстрованих користувачів. Він забезпечує швидкий доступ до пошуку спеціалістів через адаптивний Header та Layout, демонструючи основну інформацію про клініку. Під час тестування верифіковано коректне відображення динамічного контенту та перехід до авторизації.

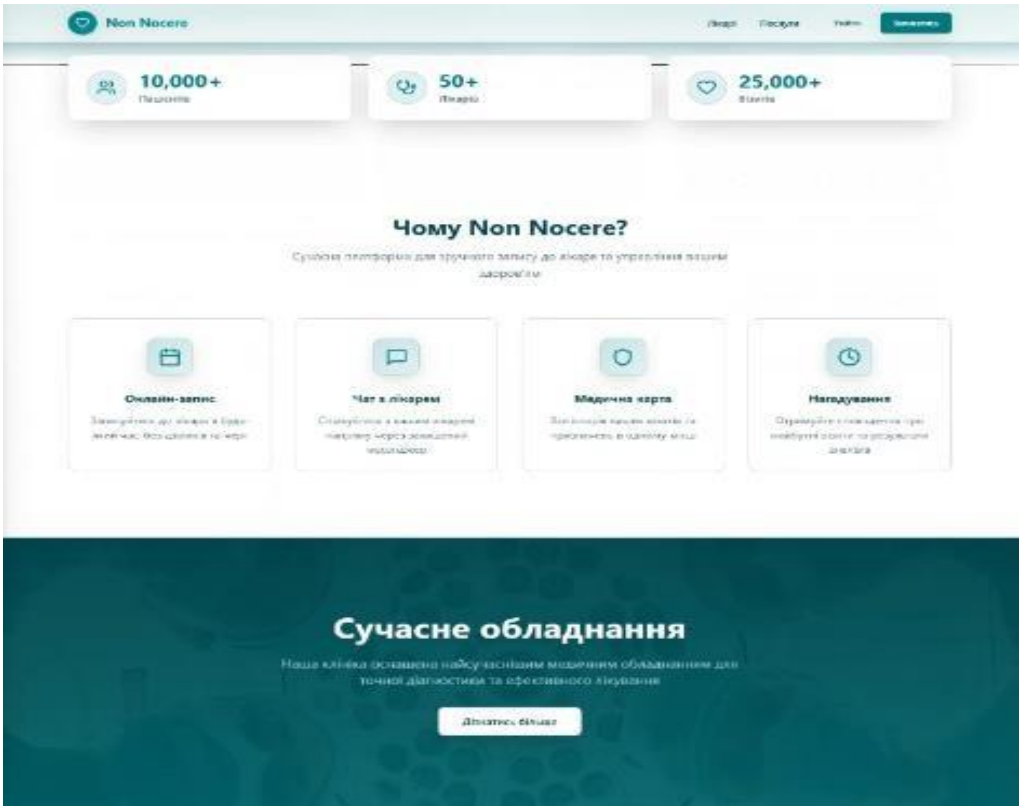

Рисунок 3.2. Початковий екран

Футер вебдодатку, представлений на Рис. 3.3, містить логотип, навігаційні посилання, контактні дані клініки з адресою, телефоном та електронною поштою, а також графік роботи. Компонент Footer забезпечує єдиний стиль на всіх сторінках і містить посилання на мапу. Верифікація підтвердила правильність відображення статичної інформації та адаптивність на мобільних пристроях.

Рисунок 3.3. Футер

Розділ «Наші лікарі», як зображено на Рис. 3.4, відображає список спеціалістів за допомогою компонента DoctorCard з аватаром, прізвищем, ім’ям, спеціальністю, досвідом роботи та кількістю послуг. Кожна картка містить кнопки для детального перегляду та запису на прийом. Тестування doctorsApi показало коректне завантаження даних та інтерактивність елементів.

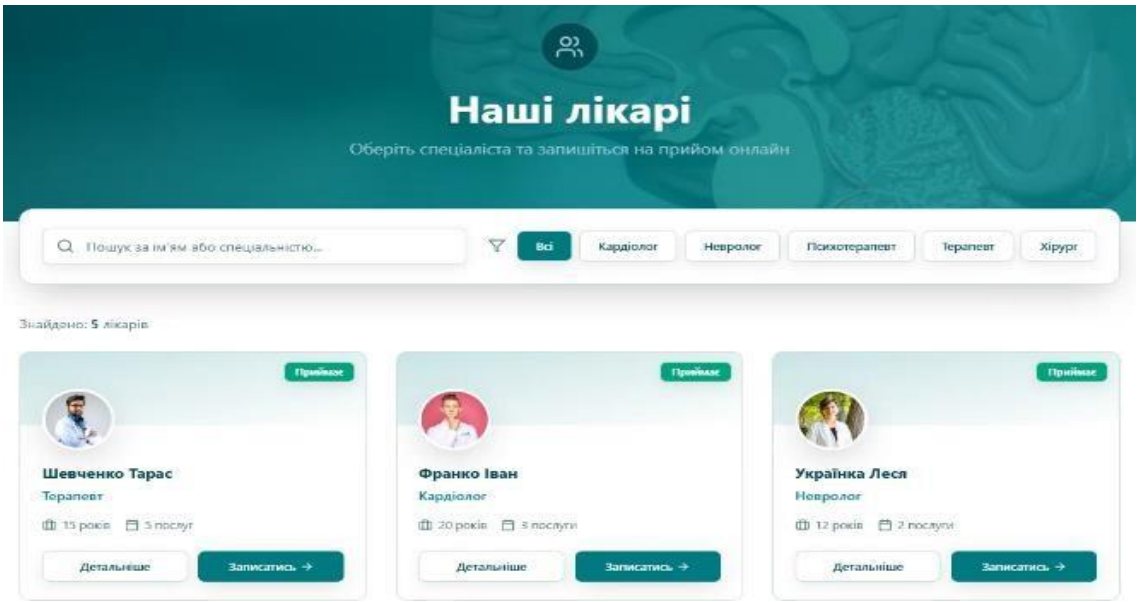

Рисунок 3.4. Наші лікарі

Сторінка послуг, показана на Рис. 3.5, представляє каталог у форматі ServiceCard з назвою, категорією, описом, тривалістю та ціною кожної послуги. Користувачі можуть перейти до детальної інформації. Верифікація servicesApi підтвердила правильну пагінацію та фільтрацію за категоріями.

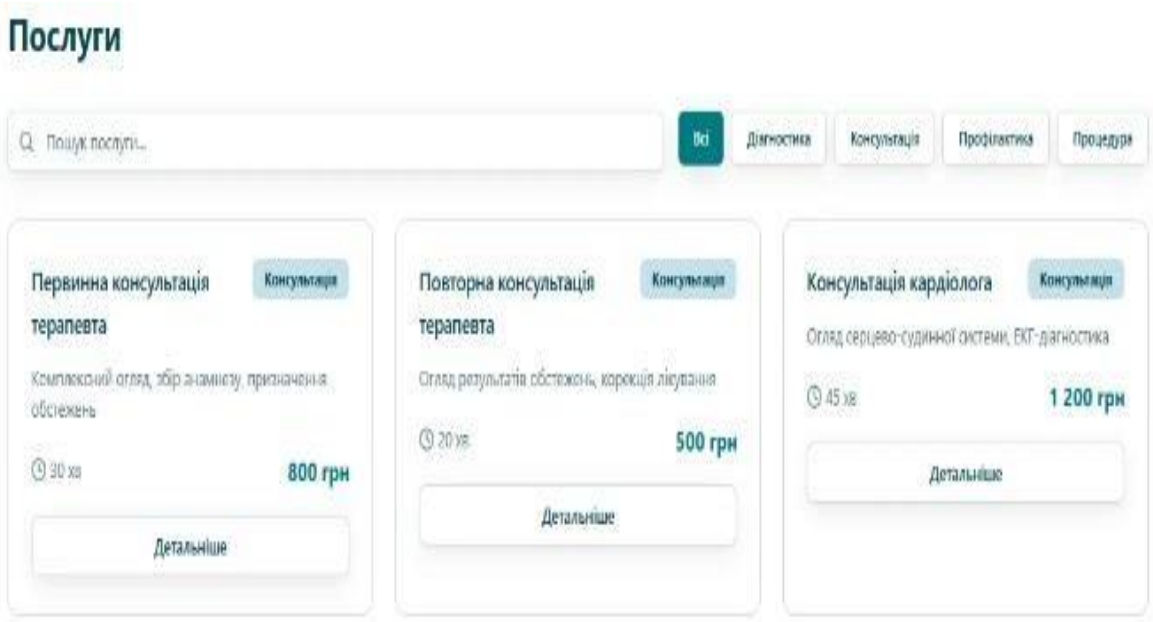

Рисунок 3.5. Послуги

Форми реєстрації та входу, як видно на Рис. 3.6, містять поля для електронної пошти, пароля, імені, прізвища та телефону з валідацією через Zod. Сторінка входу пропонує швидкі кнопки для тестових акаунтів різних ролей. Тестування authApi засвідчило успішну реєстрацію, авторизацію та перенаправлення до відповідних кабінетів.

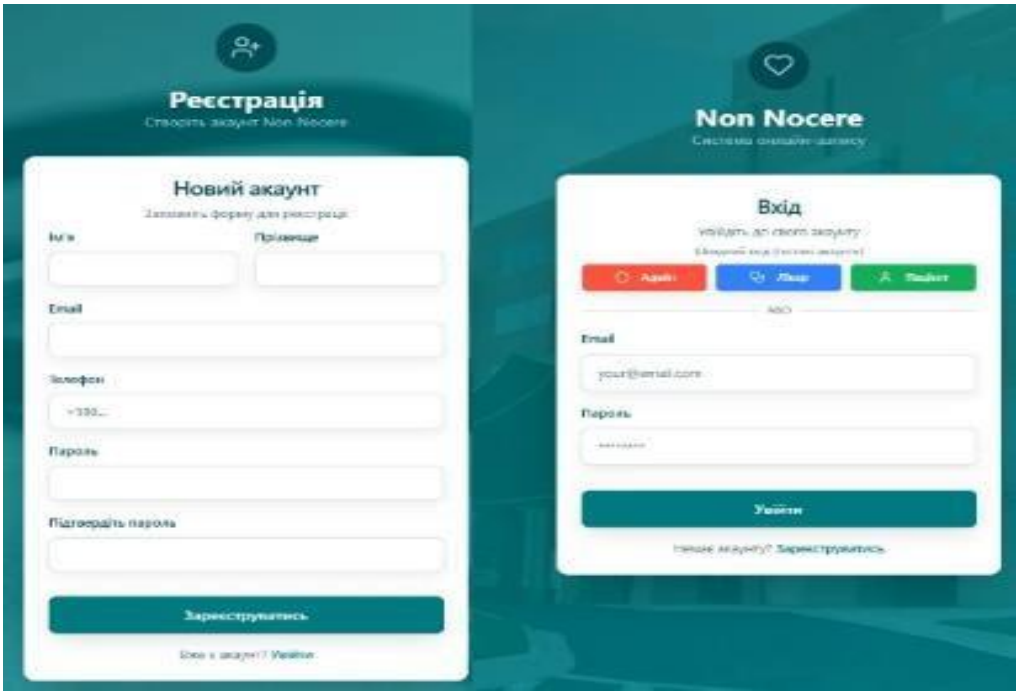

Рисунок 3.6. Форми реєстрації та входу

Особистий кабінет пацієнта (дашборд), зображений на Рис. 3.7, надає огляд актуальних записів, медичної історії та посилань на чат. Він використовує запити до appointmentsApi та medicalRecordsApi для відображення персональних даних. Під час тестування перевірено коректність завантаження інформації та навігацію між розділами.

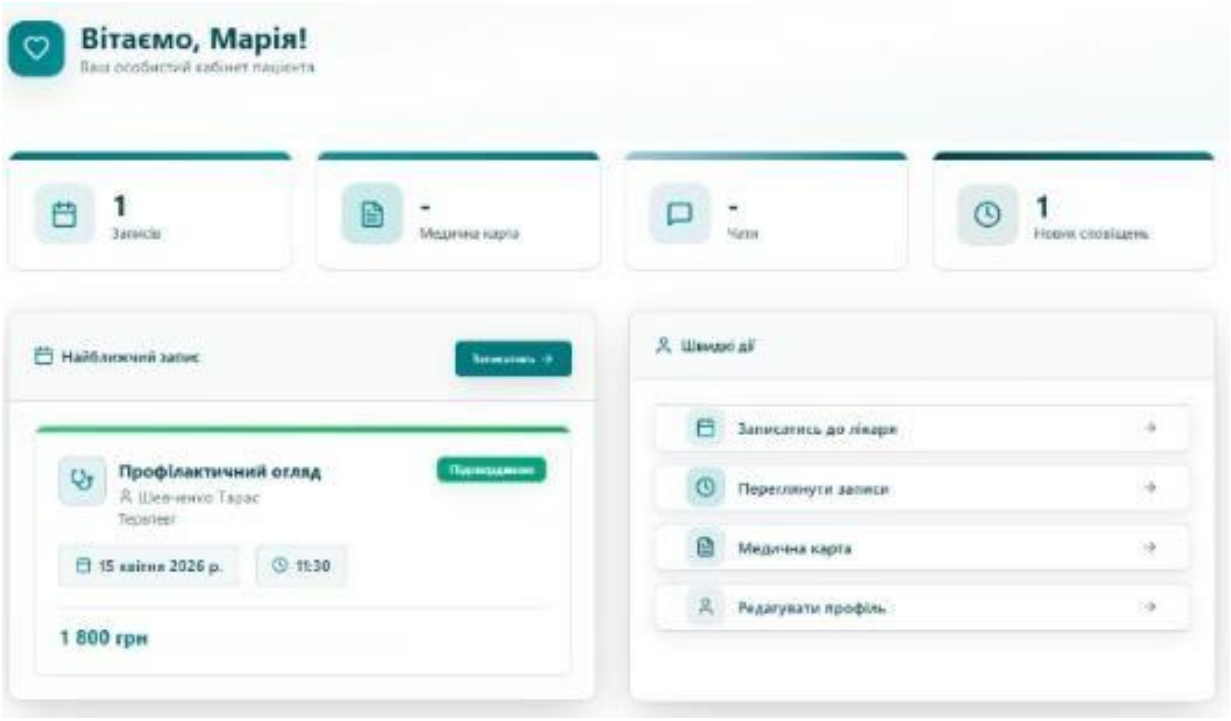

Рисунок 3.7. Особистий кабінет пацієнта

Сторінка запису на прийом, представлена на Рис. 3.8, дозволяє обрати лікаря, дату, доступний часовий слот та послугу через календарі та списки. Інтеграція з doctorsApi та scheduleApi забезпечує реальний час доступності. Верифікація підтвердила створення запису через create у appointmentsApi.

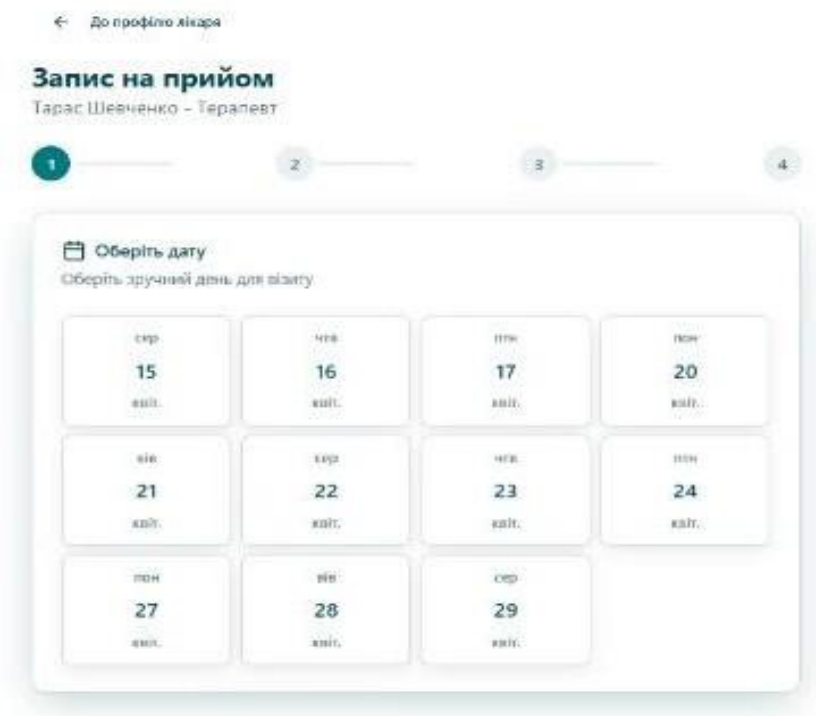

Рисунок 3.8. Запис на прийом

Сторінка «Мої записи», як показано на Рис. 3.9, відображає список записів пацієнта з використанням AppointmentCard, статусами, датами та кнопками скасування. Дані завантажуються через getMyAppointments. Тестування верифікувало фільтрацію, оновлення статусу та відображення причин скасування.

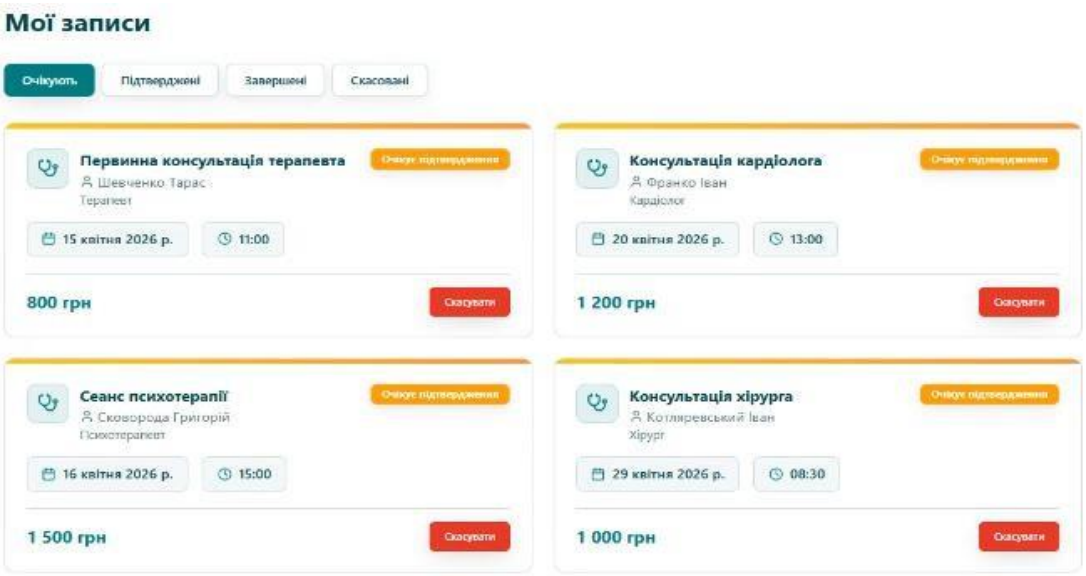

Рисунок 3.9. Мої записи

Медична карта, зображена на Рис. 3.10, показує історію медичних записів з діагнозами, скаргами та рекомендаціями. Запити до medicalRecordsApi забезпечують доступ тільки до власних даних. Верифікація підтвердила коректне відображення хронології візитів.

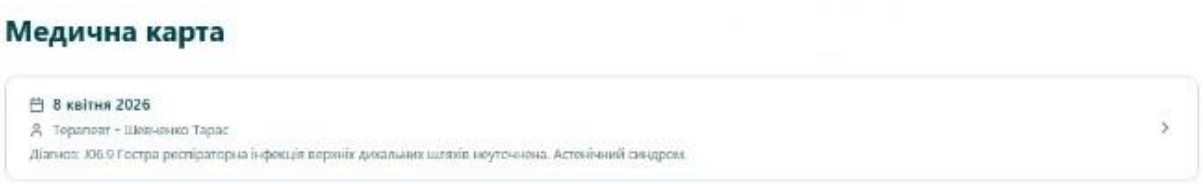

Рисунок 3.10. Медична карта

Сторінка «Мій профіль», як видно на Рис. 3.11, дозволяє редагувати особисті дані, контактну інформацію та змінити пароль через profileApi. Форма підтримує валідацію та оновлення аватара. Тестування показало успішне збереження змін у базі даних.

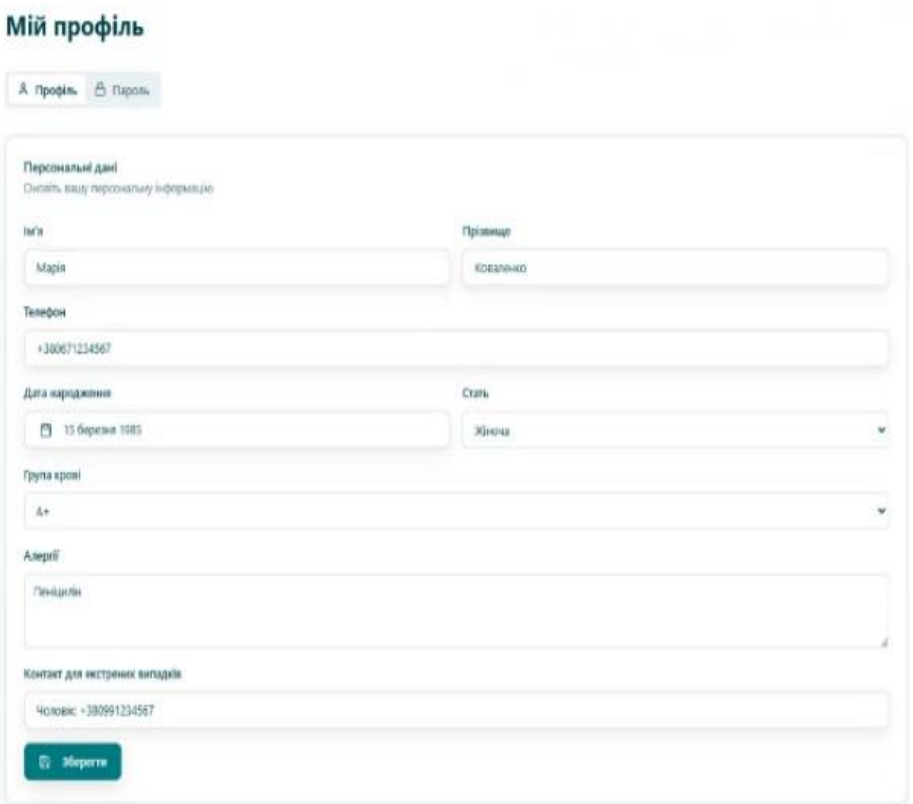

Рисунок 3.11. Мій профіль

Розділ повідомлень, представлений на Рис. 3.12, відображає сповіщення з можливістю позначення як прочитані та перегляду кількості непрочитаних, а також функції чату. Реалізовано через notificationsApi. Верифікація підтвердила роботу з реальним часом оновлень.

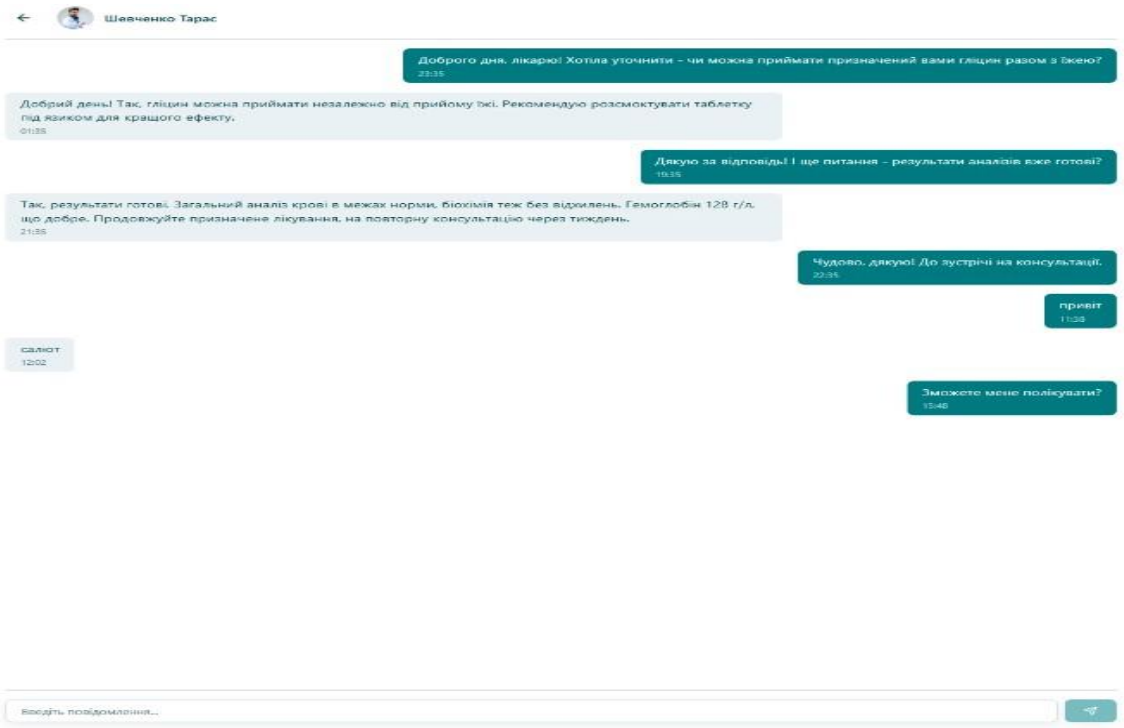

Рисунок 3.12. Повідомлення

Сторінка «Записи пацієнтів», зображена на Рис. 3.13, є частиною адміністративного інтерфейсу і містить список усіх записів з фільтрами за статусом у AdminAppointments. Використовується adminApi.getAllAppointments. Тестування засвідчило повний доступ адміністратора та коректну роботу вкладок.

Рисунок 3.13. Записи пацієнтів

Управління розкладом, як показано на Рис. 3.14, дає змогу лікарю налаштовувати шаблони днів тижня та генерувати слоти через scheduleApi. Форми включають вибір часу та тривалості. Верифікація підтвердила створення та блокування слотів.

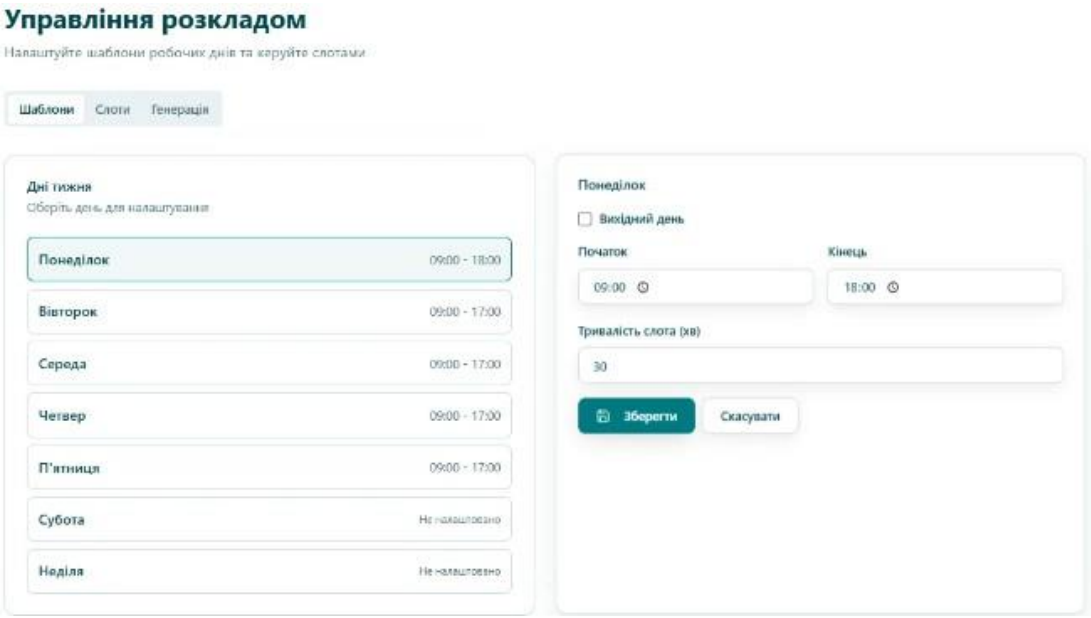

Рисунок 3.14. Управління розкладом

Сторінка медичних записів, представлена на Рис. 3.15, дозволяє лікарю створювати та редагувати записи після прийому за допомогою medicalRecordsApi. Вона містить поля для скарг, діагнозу, плану лікування та рецептів. Тестування обмежило доступ тільки для завершених прийомів.

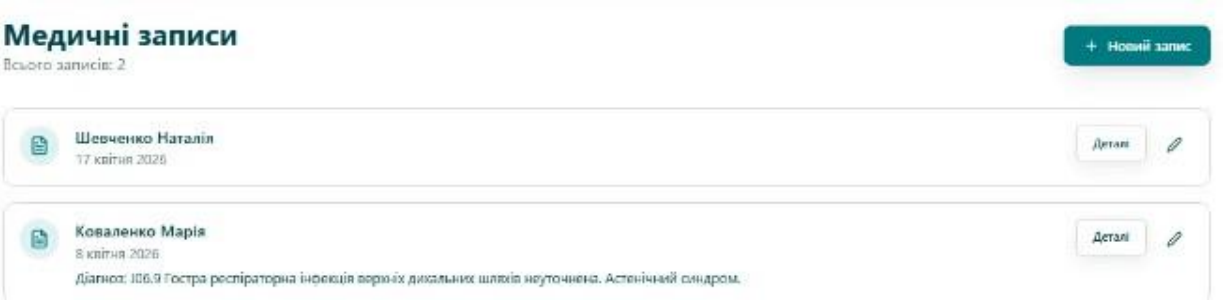

Рисунок 3.15. Медичні записи

Профіль лікаря, як зображено на Рис. 3.16, відображає деталі спеціаліста з можливістю редагування даних через doctorApi. Він включає біографію, досвід та список послуг. Верифікація показала оновлення профілю та управління послугами.

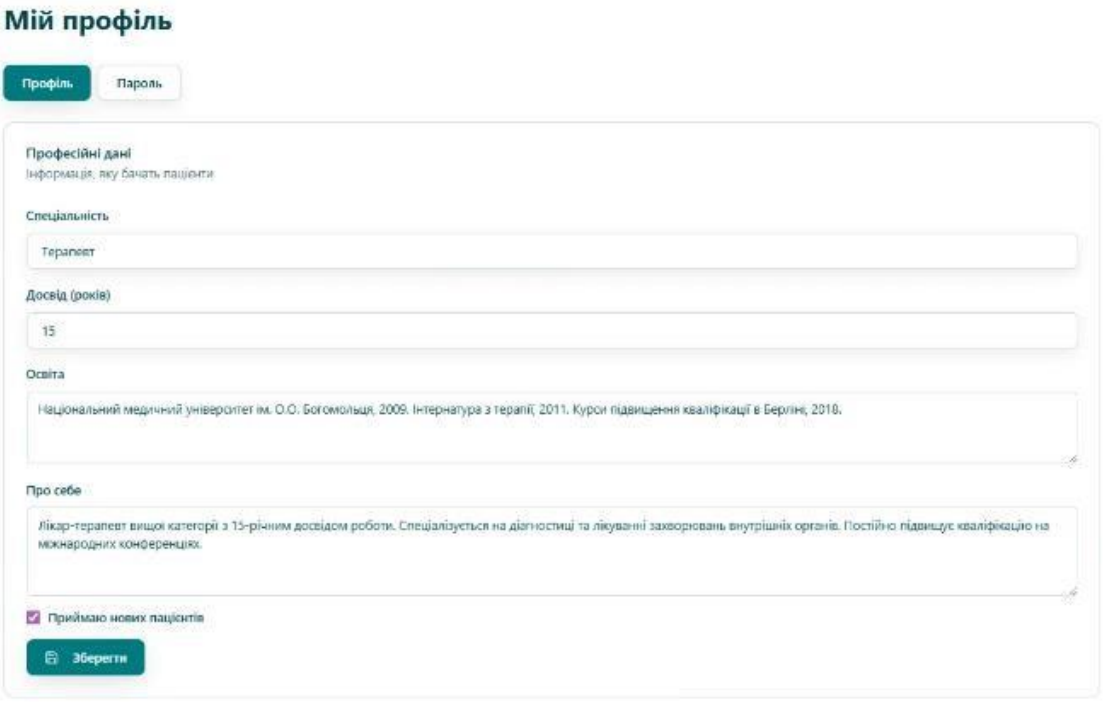

Рисунок 3.16. Профіль лікаря

Панель адміністратора, показана на Рис. 3.17, містить статистику за лікарями, записами та посилання на управління в AdminDashboard. Використовується adminApi.getStats. Тестування верифікувало точність показників та навігацію до інших розділів.

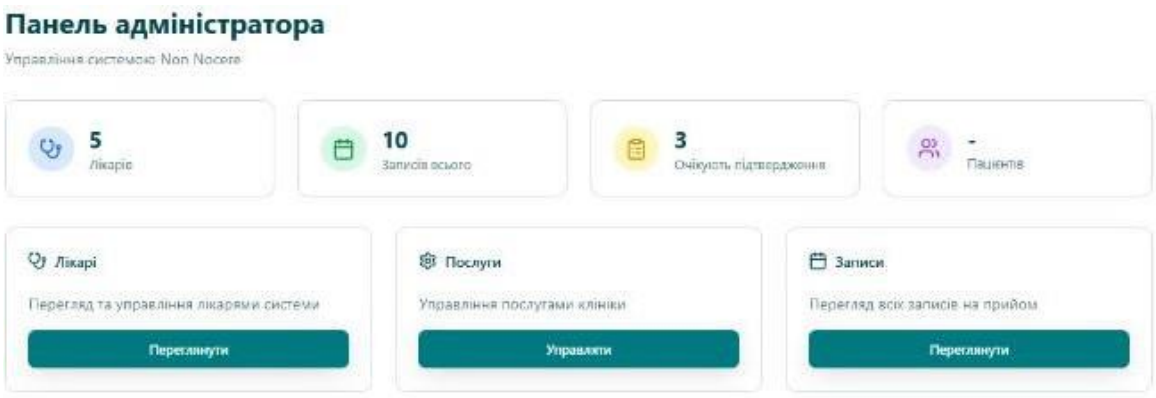

Рисунок 3.17. Панель адміністратора

Розділ «Лікарі» в адмін-панелі, як видно на Рис. 3.18, дозволяє переглядати список та редагувати профілі з додаванням послуг у AdminDoctors. Форми підтримують оновлення особистих даних та спеціальності. Верифікація підтвердила повний цикл CRUD-операцій.

Рисунок 3.18. Лікарі

Управління послугами, представлене на Рис. 3.19, дає адміністратору змогу створювати, редагувати та видаляти послуги в AdminServices. Кожна картка містить назву, категорію та ціну. Тестування adminApi показало коректне збереження змін.

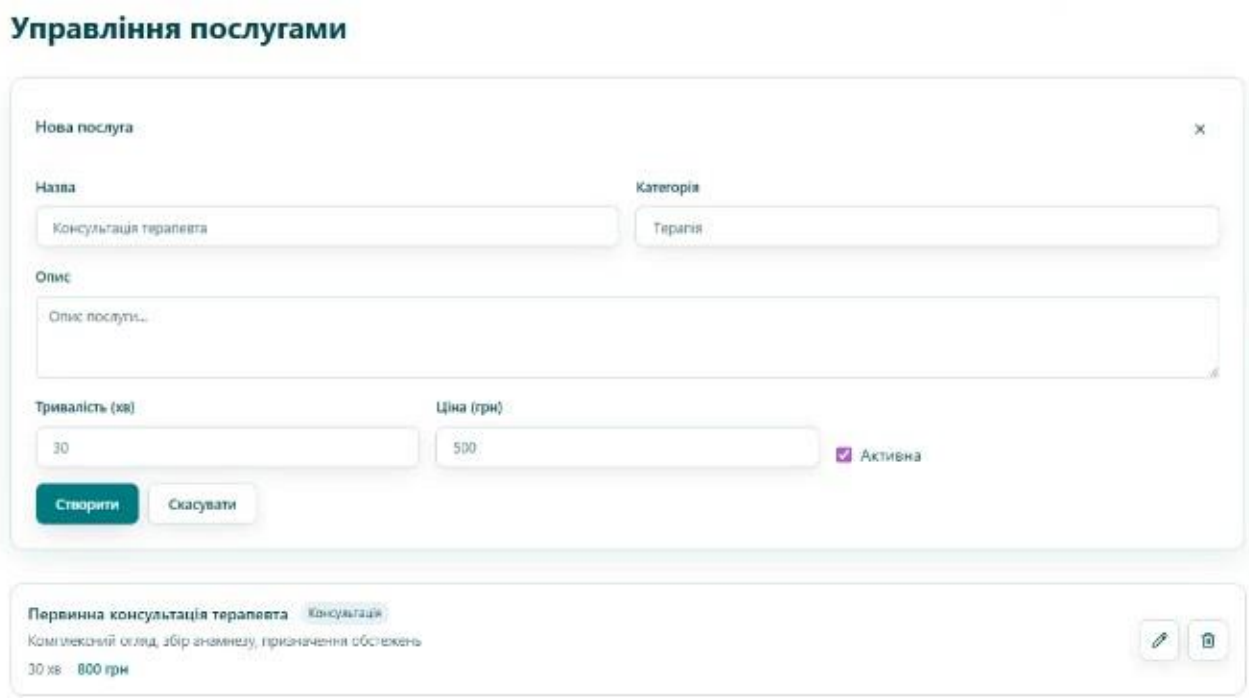

Рисунок 3.19. Управління послугами

Сторінка «Записи на прийом» в адмін-панелі, як показано на Рис. 3.20, відображає всі записи з фільтрами за статусом та детальною інформацією про пацієнтів і лікарів. Реалізовано через AdminAppointments. Загальна верифікація засвідчила стабільну роботу системи в усіх ролях.

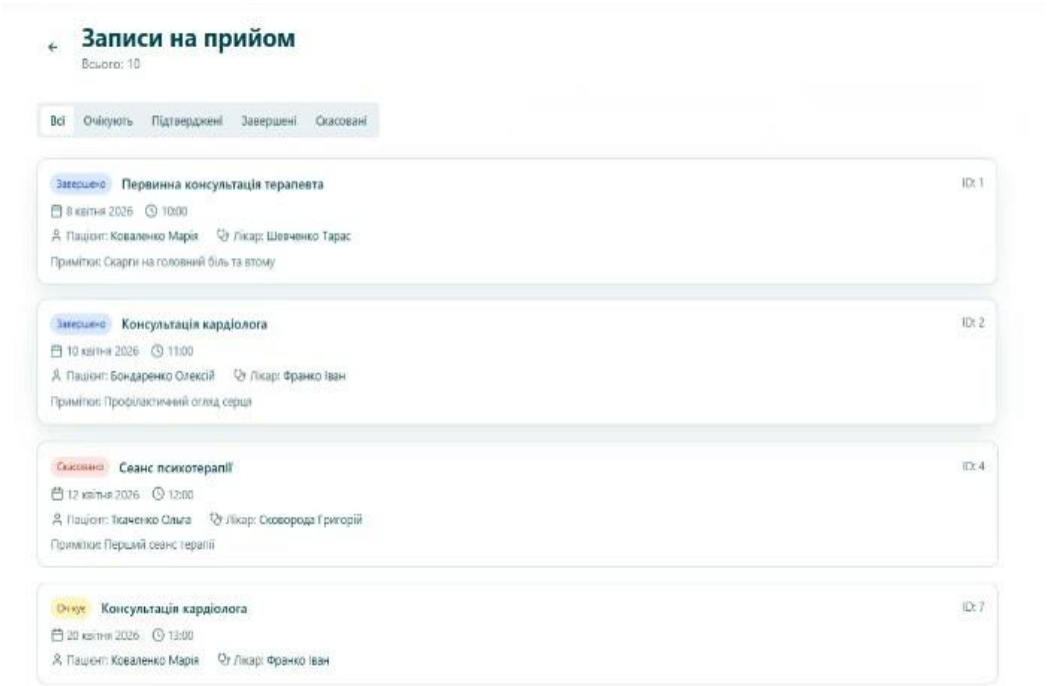

Рисунок 3.20. Записи на прийом

Проведене тестування та верифікація підтвердили високу якість реалізації вебдодатку Non Nocere. Усі екранні форми працюють стабільно, забезпечуючи зручність використання та надійність даних. Система повністю готова до практичного застосування в медичних закладах.

3.3 Аналіз результатів впровадження та практичні рекомендації щодо використання розробленої системи

Розроблена система електронного запису пацієнтів Non Nocere після впровадження продемонструвала високу ступінь готовності до практичного застосування в медичних закладах. Серверна частина, реалізована на платформі Spring Boot, забезпечує стабільну обробку всіх ключових процесів – від реєстрації користувачів і автентифікації до бронювання часових слотів, ведення медичних записів, обміну повідомленнями в чаті та адміністративного управління.

Клієнтська частина з інтуїтивним інтерфейсом, побудованим на React, пропонує зручні картки лікарів, послуг і записів, адаптивні форми та компоненти, що роблять взаємодію доступною для пацієнтів, лікарів і адміністраторів незалежно від рівня технічної підготовки.

Тестові дані, включені в міграції бази, підтверджують повну функціональність системи в реальних сценаріях: ролі користувачів чітко розділені, оптимістичне блокування слотів запобігає конфліктам, а механізми сповіщень і чатів підтримують оперативну комунікацію.

Аналіз коду свідчить про успішну інтеграцію всіх модулів. Система надійно працює з розкладами лікарів, генерує слоти за шаблонами, обробляє записи на прийом зі зміною статусів, дозволяє лікарям підтверджувати, скасовувати чи завершувати консультації, а пацієнтам – переглядати історію та медичні записи.

Адміністративна панель дає змогу керувати послугами, лікарями та загальною статистикою, що значно спрощує організаційну роботу закладу. Завдяки цьому вдалося досягти головної мети – зробити медичну допомогу доступнішою, зменшити адміністративне навантаження та покращити взаємодію між людьми.

Основні функціональні модулі системи представлені в таблиці 3.2.

Таблиця 3.2

Основні функціональні модулі системи та результати їх впровадження

Модуль	**Ключові можливості**	**Результати впровадження**
Входження та профілі	Реєстрація, автентифікація, ролі	Надійний захист даних, персоналізований доступ
Запис на прийом	Розклад, слоти, бронювання	Автоматичне оновлення статусів, відсутність конфліктів
Комунікація	Чат, сповіщення	Оперативний обмін інформацією
Медичні записи	Створення та редагування після прийому	Зручне ведення документації
Адміністрування	Управління послугами, лікарями, статистикою	Ефективний контроль за роботою закладу

Практичні рекомендації щодо використання системи ґрунтуються на її перевірених можливостях. Медичним закладам варто впроваджувати її поступово, починаючи з модуля запису пацієнтів на консультації, що дозволить суттєво скоротити черги та підвищити задоволеність людей. Лікарям рекомендується регулярно оновлювати розклад і використовувати чат та медичні записи для постійного супроводу пацієнтів, що покращить якість лікування. Адміністраторам достатньо стежити за статистикою та оновлювати перелік послуг, щоб система завжди відповідала актуальним потребам закладу.

Загалом Non Nocere стає надійним помічником, який не лише оптимізує робочі процеси, а й робить медичну допомогу більш людяною, зручною та своєчасною для кожного пацієнта. Подальше використання системи сприятиме підвищенню ефективності роботи клінік і зміцненню довіри людей до сучасних цифрових рішень у сфері охорони здоров’я.

Висновки до розділу 3

Розроблена система електронного запису пацієнтів Non Nocere є комплексним програмним рішенням, реалізованим на основі клієнт-серверної архітектури з використанням платформи Spring Boot для серверної частини та React для клієнтського інтерфейсу. Таке технологічне поєднання забезпечило надійну обробку бізнес-логіки, високий рівень безпеки даних та зручну взаємодію для всіх категорій користувачів.

Основні функціональні модулі, що охоплюють автентифікацію та авторизацію, бронювання прийомів, управління розкладом лікарів, комунікацію в чаті, ведення медичних записів та адміністративне керування, продемонстрували стабільну інтеграцію та ефективну роботу. Результати тестування та верифікації підтвердили коректність функціонування всіх інтерфейсів, форм та сервісів, а також їхню адаптивність до різних пристроїв.

Аналіз впровадження свідчить про значний потенціал системи в автоматизації процесів медичних закладів. Запропоновані практичні рекомендації спрямовані на поступове впровадження рішення, що дозволяє оптимізувати робочі процеси, зменшити адміністративне навантаження та підвищити якість взаємодії між пацієнтами і медичним персоналом. Створений вебдодаток становить готове до практичного застосування рішення, яке сприяє цифровізації охорони здоров’я та покращенню доступності медичних послуг.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто головної мети – розроблено інтерактивний вебдодаток онлайн-системи електронного запису пацієнтів Non Nocere на платформі Spring Boot. Створене рішення робить медичну допомогу доступнішою, зручнішою та по-справжньому орієнтованою на людину, дозволяючи пацієнтам уникати традиційних черг, оперативно обирати фахівця та підтримувати постійний зв’язок із лікарем.

Усі поставлені завдання виконано в повному обсязі. Проведено детальний аналіз сучасних електронних систем запису та порівняльну характеристику технологій веброзробки в екосистемі Spring Boot, що дало змогу обґрунтувати вибір клієнт-серверної багатошарової архітектури. Серверна частина на Spring Boot з використанням Spring Data JPA, Spring Security та JWT забезпечує надійну обробку конфіденційних даних, а клієнтська частина на React з TypeScript, Tailwind CSS та TanStack Query формує сучасний, адаптивний і високої інтерактивності інтерфейс. Нормалізована модель даних у PostgreSQL із застосуванням Flyway гарантує цілісність інформації та ефективне управління розкладами, записами та медичною документацією.

Розроблена система реалізує повний цикл взаємодії: автентифікацію з розмежуванням ролей, онлайн-бронювання з автоматичним генеруванням часових слотів, реальний час комунікації через чат, ведення медичних записів та адміністративне керування. Завдяки оптимістичному блокуванню слотів, механізмам сповіщень і інтуїтивному дизайну забезпечено високу надійність та зручність для пацієнтів, лікарів і керівництва закладів. Комплексне тестування всіх модулів на реальних сценаріях підтвердило стабільну роботу системи на різних пристроях, коректність бізнес-логіки та відповідність нефункціональним вимогам щодо безпеки та продуктивності.

Отримані результати демонструють значну практичну цінність. Вебдодаток Non Nocere дозволяє суттєво скоротити адміністративне навантаження на медичні заклади, зменшити кількість неявок на прийом,

оптимізувати робочий час фахівців і посилити довіру між пацієнтом і лікарем ще до особистої зустрічі. Система повністю готова до впровадження та відповідає актуальним потребам цифрової трансформації охорони здоров’я в Україні, сприяючи гуманізації медичних послуг і підвищенню їхньої доступності для кожного.

Таким чином, розроблений інтерактивний вебдодаток є ефективним цифровим інструментом, що ставить людину в центр медичного процесу. Результати роботи підтверджують правильність обраних методів і технологій, а також відкривають перспективи подальшого розвитку системи, зокрема інтеграції з національними сервісами eHealth, з метою ще більшого покращення якості медичної допомоги.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Malakhov K. S. Insight into the Digital Health System of Ukraine (eHealth): Trends, Definitions, Standards, and Legislative Revisions. Journal of Telerehabilitation. 2023. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10754247/ (дата звернення: 03.05.2026).

2. Міністерство охорони здоров'я України. Які нові цифрові можливості з'явилися для лікарів та пацієнтів у 2025 році. 2026. URL: https://ehealth.gov.ua/2026/02/20/yaki-novi-tsyfrovi-mozhlyvosti-z-yavylysya-dlya-likariv-ta-patsiyentiv-u-2025-rotsi/ (дата звернення: 03.05.2026).

3. UNDP Ukraine. Ukraine's health ministry presents new digital healthcare projects at first UNDP-supported eHealth Summit. 2024. URL: https://www.undp.org/ukraine/press-releases/ukraines-health-ministry-presents-new-digital-healthcare-projects-first-undp-supported-ehealth-summit (дата звернення: 03.05.2026).

4. Медплатформа. Що таке пацієнтські інформаційні системи. 2025. URL: https://medplatforma.com.ua/article/18762-patsiientski-informatsiini-systemy (дата звернення: 03.05.2026).

5. Atherton H. et al. Investigating Patient Use and Experience of Online Appointment Booking in General Practice: A Mixed-Methods Study. PMC. 2024. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11263895/ (дата звернення: 03.05.2026).

6. GeeksforGeeks. Spring MVC vs WebFlux: When to Use Which Framework?. 2025. URL: https://www.geeksforgeeks.org/blogs/spring-mvc-vs-spring-web-flux/ (дата звернення: 03.05.2026).

7. Spring Framework Team. Spring WebFlux. 2025. URL: https://docs.spring.io/spring-framework/reference/web/webflux.html (дата звернення: 03.05.2026).

8. Foxminde. Як Spring Boot допомагає прискорити розробку додатків. 2024. URL: https://foxminded.ua/spring-boot/ (дата звернення: 03.05.2026).

9. Spring.io. Spring Security Reference. Spring Security Documentation. 2024. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 03.05.2026).

10. DevZone. Реактивне програмування на Java. 2017. URL: https://devzone.org.ua/post/reaktyvne-prohramuvannia-na-java (дата звернення: 03.05.2026).

11. Національна служба здоров'я України. Технічні вимоги до електронних медичних записів (у редакції наказу НСЗУ). 2023. URL: https://ehealth.gov.ua/wp-content/uploads/2023/02/Tehnichni-vymogy-v-redaktsiyi-nakazu-NSZU-175-vid-03.06.2019.pdf (дата звернення: 03.05.2026).

12. Міністерство охорони здоров'я України. Нормативно-правове регулювання ЕСОЗ. 2024. URL: https://moz.gov.ua/uk/normativno-pravove-regulyuvannya-esoz (дата звернення: 03.05.2026).

13. Кабінет Міністрів України. Деякі питання електронної системи охорони здоров'я: постанова від 25.04.2018 № 411 (зі змінами станом на 2025 рік). URL: https://zakon.rada.gov.ua/laws/show/411-2018-п (дата звернення: 03.05.2026).

14. Medplatforma. Як працює електронна система охорони здоров'я. 2025. URL: https://medplatforma.com.ua/article/1541-elektronna-sistema-ohoroni-zdorovya (дата звернення: 03.05.2026).

15. European Commission. eHealth - digital services in health and care. 2024. URL: https://digital-strategy.ec.europa.eu/en/policies/ehealth (дата звернення: 03.05.2026).

16. Панікарський С. І. Розробка веб-додатків на базі Spring Boot: дипломна робота. Тернопільський національний технічний університет. 2023. URL: https://elartu.tntu.edu.ua/bitstream/lib/42419/1/dyplom_Panikarskyy_2023.pdf (дата звернення: 03.05.2026).

17. Марків К. А. Застосування Spring Boot у розробці серверних додатків: кваліфікаційна робота. 2024. URL:

https://elartu.tntu.edu.ua/bitstream/lib/45493/1/dyplom_Markiv_2024.pdf (дата звернення: 03.05.2026).

18. Миргородський Н. В. Розробка веб-сервісів з використанням Spring Boot: дипломний проєкт. КПІ ім. Ігоря Сікорського. 2023. URL: https://ela.kpi.ua/bitstreams/fc2d0914-1486-4872-b2d1-828ba6626c30/download (дата звернення: 03.05.2026).

19. Spring Framework Documentation Team. Spring Boot Reference Documentation. 2025. URL: https://docs.spring.io/spring-boot/reference/index.html (дата звернення: 03.05.2026).

20. Alzakholi O. et al. Comparison Among Cloud Technologies and Cloud Performance. J. Appl. Sci. Technol. Trends. 2020. Vol. 1, no. 2. P. 40–47.

21. Castelblanco A. et al. Machine learning techniques for identity document verification in uncontrolled environments: A case study. Lect. Notes Comput. Sci. Springer, 2020. Vol. 12088 LNCS. P. 271–281.

22. FoxmindEd. Правильна архітектура сучасного React додатка. 2024. URL: https://foxminded.ua/arkhitektura-react-dodatka/ (дата звернення: 03.05.2026).

23. Литовченко Ю. М. Інтеграція REST API у React-додатках з використанням Axios та TanStack Query. Інформаційні технології та засоби навчання. 2024. Т. 99, № 1. С. 112–125.

24. El-Jardali F., Bou-Karroum L., Jabbour M., Bou-Karroum K., Aoun A., Salameh S., Mecheal P., Sinha C. Digital health in fragile states in the Middle East and North Africa (MENA) region: A scoping review of the literature. PLOS ONE. 2023. Vol. 18, no. 4. e0285226.

ДОДАТОК А

Основні лістинги програмного коду

admin.ts

```
import client from'./client'
importtype { Service, Page, Appointment } from'@/types'
exportinterfaceCreateServiceRequest {
  name: string
  description?: string
  durationMinutes: number
  price: number
  category: string
  isActive?: boolean
}
exportinterfaceUpdateServiceRequestextendsCreateServiceRequest {}
exportinterfaceUserResponse {
  id: number
  email: string
  firstName: string
  lastName: string
  phone: string | null
  role: string
  isActive: boolean
  createdAt: string
}
exportinterfaceDoctorWithUser {
  id: number
  userId: number
  firstName: string
  lastName: string
  email: string
  phone: string | null
  specialty: string
  experienceYears: number
  isAcceptingPatients: boolean
  servicesCount: number
}
exportinterfaceDoctorDetail {
  id: number
  userId: number
  firstName: string
  lastName: string
  email: string
  phone: string | null
  avatarUrl: string | null
  specialty: string
  bio: string | null
  experienceYears: number
  education: string | null
  isAcceptingPatients: boolean
  services: DoctorServiceInfo[]
}
exportinterfaceDoctorServiceInfo {
  id: number
  name: string
  category: string
  durationMinutes: number
  price: number
}
exportinterfaceUpdateDoctorRequest {
  firstName?: string
  lastName?: string
  phone?: string
  specialty?: string
  bio?: string
  experienceYears?: number
  education?: string
  isAcceptingPatients?: boolean
}
exportinterfaceAddDoctorServiceRequest {
```

```
  serviceId: number
  customPrice?: number
}
exportconst adminApi = {
// Services
  getServices: async (): Promise<Service[]> => {
const response = await client.get<Page<Service>>('/services', { params: { size: 100 } })
return response.data.content
  },
  createService: async (data: CreateServiceRequest): Promise<Service> => {
const response = await client.post<Service>('/services', data)
return response.data
  },
  updateService: async (id: number, data: UpdateServiceRequest): Promise<Service> => {
const response = await client.put<Service>(`/services/${id}`, data)
return response.data
  },
  deleteService: async (id: number): Promise<void> => {
await client.delete(`/services/${id}`)
  },
// Doctors
  getDoctors: async (): Promise<DoctorWithUser[]> => {
const response = await client.get<Page<DoctorWithUser>>('/doctors', { params: { size: 100 } })
return response.data.content
  },
  getDoctor: async (id: number): Promise<DoctorDetail> => {
const response = await client.get<DoctorDetail>(`/doctors/${id}`)
return response.data
  },
  updateDoctor: async (id: number, data: UpdateDoctorRequest): Promise<DoctorDetail> => {
const response = await client.put<DoctorDetail>(`/doctors/${id}`, data)
return response.data
  },
  addDoctorService: async (doctorId: number, data: AddDoctorServiceRequest): Promise<DoctorDetail> => {
const response = await client.post<DoctorDetail>(`/doctors/${doctorId}/services`, data)
return response.data
  },
  removeDoctorService: async (doctorId: number, serviceId: number): Promise<DoctorDetail> => {
const response = await client.delete<DoctorDetail>(`/doctors/${doctorId}/services/${serviceId}`)
return response.data
  },
// Appointments
  getAllAppointments: async (params: { status?: string; page?: number; size?: number } = {}): Promise<Page<Appointment>>
=> {
const response = await client.get<Page<Appointment>>('/appointments', { params: { ...params, size: params.size || 50 } })
return response.data
  },
// Stats
  getStats: async (): Promise<{
    totalDoctors: number
    totalPatients: number
    totalAppointments: number
    pendingAppointments: number
  }> => {
// Збираємо статистику з різних ендпоінтів
const [doctors, appointments] = awaitPromise.all([
      client.get<Page<DoctorWithUser>>('/doctors', { params: { size: 1 } }),
      client.get<Page<Appointment>>('/appointments', { params: { size: 1 } }),
    ])
const pendingAppointments = await client.get<Page<Appointment>>('/appointments', {
      params: { status: 'PENDING', size: 1 }
    })
 return {
      totalDoctors: doctors.data.totalElements,
      totalPatients: 0, // TODO: add users endpoint
      totalAppointments: appointments.data.totalElements,
      pendingAppointments: pendingAppointments.data.totalElements,
    }
  },
}
```

auth.ts

```
import client from'./client'
importtype { AuthResponse } from'@/types'
exportinterfaceLoginRequest {
  email: string
  password: string
}
exportinterfaceRegisterRequest {
  email: string
  password:  string
  firstName: string
  lastName:  string
  phone?: string
}
```

```
exportconst authApi = {
  login: async (data: LoginRequest): Promise<AuthResponse> => {
const response = await client.post<AuthResponse>('/auth/login', data)
return response.data
  },
  register: async (data: RegisterRequest): Promise<AuthResponse> => {
const response = await client.post<AuthResponse>('/auth/register', data)
return response.data
  },
  logout: async (): Promise<void> => {
await client.post('/auth/logout')
  },
  refresh: async (): Promise<AuthResponse> => {
const response = await client.post<AuthResponse>('/auth/refresh')
return response.data
  },
}
```

chats.ts

```
import client from'./client'
importtype { ChatRoom, ChatMessage, Page } from'@/types'
interfaceMessagesParams {
  page?: number
  size?: number
}
exportconst chatsApi = {
  getMyChatRooms: async (): Promise<ChatRoom[]> => {
const response = await client.get<ChatRoom[]>('/chats')
return response.data
  },
  getChatRoom: async (roomId: number): Promise<ChatRoom> => {
const response = await client.get<ChatRoom>(`/chats/${roomId}`)
return response.data
  },
  getMessages: async (roomId: number, params: MessagesParams = {}): Promise<Page<ChatMessage>> => {
const response = await client.get<Page<ChatMessage>>(`/chats/${roomId}/messages`, { params })
return response.data
  },
  sendMessage: async (roomId: number, content: string): Promise<ChatMessage> => {
const response = await client.post<ChatMessage>(`/chats/${roomId}/messages`, { content })
return response.data
  },
  markAsRead: async (roomId: number): Promise<void> => {
await client.post(`/chats/${roomId}/read`)
  },
  getOrCreateWithDoctor: async (doctorId: number): Promise<ChatRoom> => {
const response = await client.post<ChatRoom>(`/chats/with-doctor/${doctorId}`)
return response.data
  },
  getUnreadCount: async (): Promise<number> => {
const response = await client.get<number>('/chats/unread-count')
return response.data
  },
}
```

client.ts

```
import axios, { AxiosError, InternalAxiosRequestConfig } from'axios'
import { useAuthStore } from'@/store/auth'
importtype { ApiError } from'@/types'
const client = axios.create({
  baseURL: '/api/v1',
  headers: {
    'Content-Type': 'application/json',
  },
})
client.interceptors.request.use((config: InternalAxiosRequestConfig) => {
const token = useAuthStore.getState().token
if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
return config
```

```
})
client.interceptors.response.use(
  (response) => response,
async (error: AxiosError<ApiError>) => {
if (error.response?.status === 401) {
      useAuthStore.getState().logout()
      window.location.href = '/login'
    }
returnPromise.reject(error)
  }
)
exportdefault client
```

doctor.ts

```
import client from'./client'
importtype { TimeSlot } from'@/types'
exportinterfaceDoctorProfile {
  id: number
  firstName: string
  lastName: string
  email: string
  phone: string | null
  avatarUrl: string | null
  specialty: string
  bio: string | null
  experienceYears: number
  education: string | null
  isAcceptingPatients: boolean
}
exportinterfaceUpdateDoctorProfileRequest {
  specialty?: string
  bio?: string
  experienceYears?: number
  education?: string
  isAcceptingPatients?: boolean
}
exportinterfaceScheduleTemplate {
  dayOfWeek: string
  startTime: string
  endTime: string
  slotDurationMinutes: number
  isWorkingDay: boolean
}
exportinterfaceGenerateSlotsRequest {
  startDate: string
  endDate: string
}
exportconst doctorApi = {
  getMyProfile: async (): Promise<DoctorProfile> => {
const response = await client.get<DoctorProfile>('/doctors/me')
return response.data
  },
  updateMyProfile: async (data: UpdateDoctorProfileRequest): Promise<DoctorProfile> => {
const response = await client.put<DoctorProfile>('/doctors/me', data)
return response.data
  },
  getMyScheduleTemplates: async (): Promise<ScheduleTemplate[]> => {
const response = await client.get<ScheduleTemplate[]>('/schedules/my/templates')
return response.data
  },
  updateScheduleTemplate: async (template: ScheduleTemplate): Promise<ScheduleTemplate> => {
const response = await client.post<ScheduleTemplate>('/schedules/template', template)
return response.data
  },
  generateSlots: async (data: GenerateSlotsRequest): Promise<void> => {
await client.post('/schedules/generate', data)
  },
  getMySlots: async (date: string): Promise<TimeSlot[]> => {
const response = await client.get<TimeSlot[]>(`/schedules/my/slots`, { params: { date } })
return response.data
  },
  blockSlot: async (slotId: number): Promise<TimeSlot> => {
```

```
const response = await client.put<TimeSlot>(`/schedules/slots/${slotId}/block`)
return response.data
  },
  unblockSlot: async (slotId: number): Promise<TimeSlot> => {
const response = await client.put<TimeSlot>(`/schedules/slots/${slotId}/unblock`)
return response.data
  },
}
```

doctors.ts

```
import client from'./client'
importtype { Doctor, DoctorListItem, Page, TimeSlot } from'@/types'
interfaceDoctorParams {
  page?: number
  size?: number
  specialty?: string
  search?: string
}
exportconst doctorsApi = {
  getAll: async (params: DoctorParams = {}): Promise<Page<DoctorListItem>> => {
const response = await client.get<Page<DoctorListItem>>('/doctors', { params })
return response.data
  },
  getById: async (id: number): Promise<Doctor> => {
const response = await client.get<Doctor>(`/doctors/${id}`)
return response.data
  },
  getSpecialties: async (): Promise<string[]> => {
const response = await client.get<string[]>('/doctors/specialties')
return response.data
  },
  getAvailableSlots: async (doctorId: number, date: string): Promise<TimeSlot[]> => {
const response = await client.get<TimeSlot[]>(`/doctors/${doctorId}/slots`, {
      params: { date },
    })
return response.data
  },
  getAvailableDates: async (doctorId: number, from?: string): Promise<string[]> => {
const response = await client.get<string[]>(`/doctors/${doctorId}/available-dates`, {
      params: { from },
    })
return response.data
  },
}
```

medicalRecords.ts

```
import client from'./client'
importtype { Page } from'@/types'
exportinterfaceMedicalRecord {
id: number
  appointmentId: number | null
  patient: {
    id: number
    firstName: string
    lastName: string
  }
  doctor: {
    id: number
    firstName: string
    lastName: string
    specialty: string
  }
  complaints: string | null
  diagnosis: string | null
  treatmentPlan: string | null
  prescriptions: string | null
  visitDate: string
  createdAt: string
}
exportinterfaceCreateMedicalRecordRequest {
  appointmentId: number
  complaints?: string
```

```
  diagnosis?: string
  treatmentPlan?: string
  prescriptions?: string
  visitDate?: string
}
exportinterfaceUpdateMedicalRecordRequest {
  complaints?: string
  diagnosis?: string
  treatmentPlan?: string
  prescriptions?: string
  visitDate?: string
}
interfaceMedicalRecordsParams {
  page?: number
  size?: number
}
exportconst medicalRecordsApi = {
// Для пацієнтів
  getMyRecords: async (params: MedicalRecordsParams = {}): Promise<Page<MedicalRecord>> => {
const response = await client.get<Page<MedicalRecord>>('/medical-records', { params })
return response.data
  },
  getById: async (id: number): Promise<MedicalRecord> => {
const response = await client.get<MedicalRecord>(`/medical-records/${id}`)
return response.data
  },
// Для лікарів
  getMyDoctorRecords: async (params: MedicalRecordsParams = {}): Promise<Page<MedicalRecord>> => {
const response = await client.get<Page<MedicalRecord>>('/medical-records/doctor/my', { params })
return response.data
  },
  getPatientRecords: async (patientId: number, params: MedicalRecordsParams = {}): Promise<Page<MedicalRecord>> => {
const response = await client.get<Page<MedicalRecord>>(`/medical-records/patient/${patientId}`, { params })
return response.data
  },
  getRecordForDoctor: async (id: number): Promise<MedicalRecord> => {
const response = await client.get<MedicalRecord>(`/medical-records/doctor/record/${id}`)
return response.data
  },
  create: async (data: CreateMedicalRecordRequest): Promise<MedicalRecord> => {
const response = await client.post<MedicalRecord>('/medical-records', data)
return response.data
  },
  update: async (id: number, data: UpdateMedicalRecordRequest): Promise<MedicalRecord> => {
const response = await client.put<MedicalRecord>(`/medical-records/${id}`, data)
return response.data
  },
}
```

notifications.ts

```
import client from'./client'
importtype { Notification, Page } from'@/types'
exportconst notificationsApi = {
  getAll: async (page = 0, size = 20): Promise<Page<Notification>> => {
const response = await client.get<Page<Notification>>('/notifications', {
    params: { page, size },
  })
return response.data
  },
  getUnreadCount: async (): Promise<number> => {
const response = await client.get<{ count: number }>('/notifications/unread-count')
return response.data.count
  },
  markAsRead: async (id: number): Promise<void> => {
await client.put(`/notifications/${id}/read`)
  },
  markAllAsRead: async (): Promise<void> => {
await client.put('/notifications/read-all')
  },
}
```

profile.ts

```
import client from'./client'
importtype { Profile } from'@/types'
exportinterfaceUpdateProfileRequest {
  firstName?: string
  lastName?: string
  phone?: string
  dateOfBirth?: string
  gender?: string
  bloodType?: string
  allergies?: string
  emergencyContact?: string
}
exportinterfaceChangePasswordRequest {
  currentPassword: string
  newPassword: string
}
exportconst profileApi = {
  getProfile: async (): Promise<Profile> => {
const response = await client.get<Profile>('/profile/me')
return response.data
  },
  updateProfile: async (data: UpdateProfileRequest): Promise<Profile> => {
const response = await client.put<Profile>('/profile/me', data)
return response.data
  },
  changePassword: async (data: ChangePasswordRequest): Promise<void> => {
await client.put('/profile/me/password', data)
  },
}
```

services.ts

```
import client from'./client'
importtype { Service, Page } from'@/types'
interfaceServiceParams {
  page?: number
  size?: number
  category?: string
  search?: string
}
exportconst servicesApi = {
  getAll: async (params: ServiceParams = {}): Promise<Page<Service>> => {
const response = await client.get<Page<Service>>('/services', { params })
return response.data
  },
  getById: async (id: number): Promise<Service> => {
const response = await client.get<Service>(`/services/${id}`)
return response.data
  },
  getCategories: async (): Promise<string[]> => {
const response = await client.get<string[]>('/services/categories')
return response.data
  },
}
```

DoctorCard.tsx

```
import { Link } from'react-router-dom'
import { Card, CardContent } from'@/components/ui/card'
```

```
import { Avatar, AvatarFallback, AvatarImage } from'@/components/ui/avatar'
import { Badge } from'@/components/ui/badge'
import { Button } from'@/components/ui/button'
import { getInitials, pluralize } from'@/lib/utils'
import { Calendar, ArrowRight, Briefcase } from'lucide-react'
importtype { DoctorListItem } from'@/types'
interfaceDoctorCardProps {
  doctor: DoctorListItem
}
exportfunction DoctorCard({ doctor }: DoctorCardProps) {
return (
<CardclassName="group overflow-hidden border-2 hover:border-primary/40">
<CardContentclassName="p-0">
{/* Header with gradient */}
<div className="relative h-24 bg-gradient-to-br from-primary/20 via-primary/10 to-transparent">
<div className="absolute -bottom-8 left-6">
<AvatarclassName="h-20 w-20 border-4 border-card shadow-strong">
<AvatarImagesrc={doctor.avatarUrl||undefined}/>
<AvatarFallbackclassName="text-xl  bg-primary  text-primary-foreground">
{getInitials(doctor.firstName,doctor.lastName)}
</AvatarFallback>
</Avatar>
</div>
<div className="absolute top-3 right-3">
{doctor.isAcceptingPatients?(
<Badgevariant="success"className="border-2  border-green-200  shadow-soft">Приймає</Badge>
):(
<Badgevariant="secondary"className="border-2 shadow-soft">Не приймає</Badge>
)}
</div>
</div>
{/* Content */}
<div className="pt-12 px-6 pb-6">
<h3 className="font-bold text-lg mb-1 group-hover:text-primary transition-colors">
{doctor.lastName}{doctor.firstName}
</h3>
<p className="text-primary font-medium mb-4">{doctor.specialty}</p>
<div className="flex items-center gap-4 text-sm text-muted-foreground mb-6">
<div className="flex items-center gap-1.5">
<BriefcaseclassName="h-4 w-4"/>
<span>{doctor.experienceYears}{pluralize(doctor.experienceYears,'рік','роки','років')}</span>
</div>
<div className="flex items-center gap-1.5">
<CalendarclassName="h-4 w-4"/>
<span>{doctor.servicesCount}{pluralize(doctor.servicesCount,'послуга','послуги','послуг')}</span>
</div>
</div>
<div className="flex gap-3">
<Linkto={`/doctors/${doctor.id}`}className="flex-1">
<Buttonvariant="outline"className="w-full  border-2">
            Детальніше
</Button>
</Link>
{doctor.isAcceptingPatients&&(
<Linkto={`/doctors/${doctor.id}/book`}className="flex-1">
<ButtonclassName="w-full border-2 border-primary gap-1 group/btn">
            Записатись
<ArrowRightclassName="h-4 w-4 transition-transform group-hover/btn:translate-x-1"/>
</Button>
</Link>
)}
</div>
</div>
</CardContent>
</Card>
  )
}
```

ServiceCard.tsx

```
import { Link } from'react-router-dom'
import { Clock } from'lucide-react'
import { Card, CardContent, CardHeader, CardTitle } from'@/components/ui/card'
```

```
import { Badge } from'@/components/ui/badge'
import { Button } from'@/components/ui/button'
import { formatPrice } from'@/lib/utils'
importtype { Service } from'@/types'
interfaceServiceCardProps {
  service: Service
}
exportfunction ServiceCard({ service }: ServiceCardProps) {
return (
<CardclassName="hover:shadow-lg  transition-shadow">
<CardHeaderclassName="pb-3">
<div className="flex items-start justify-between gap-2">
<CardTitleclassName="text-lg">{service.name}</CardTitle>
<Badgevariant="secondary">{service.category}</Badge>
</div>
</CardHeader>
<CardContent>
{service.description&&(
<p className="text-sm text-muted-foreground mb-4 line-clamp-2">
{service.description}
</p>
)}
<div className="flex items-center justify-between mb-4">
<div className="flex items-center gap-1 text-sm text-muted-foreground">
<ClockclassName="h-4 w-4"/>
<span>{service.durationMinutes} хв</span>
</div>
<span className="text-lg font-bold text-primary">
{formatPrice(service.price)}
</span>
</div>
<Linkto={`/services/${service.id}`}>
<Buttonvariant="outline"className="w-full">Детальніше</Button>
</Link>
</CardContent>
</Card>
  )
}
```

Footer.tsx

```
import { Link } from'react-router-dom'
import { MapPin, Phone, Mail, Clock, Heart } from'lucide-react'
exportfunction Footer() {
return (
<footer className="border-t-2 bg-gradient-to-b from-muted/30 to-muted/60 section-pattern">
<div className="container-custom py-12 md:py-16">
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-10">
<div className="space-y-4">
<div className="flex items-center space-x-3">
<div className="w-10 h-10 rounded-full bg-gradient-to-br from-primary to-turquoise-light flex items-center justify-center
shadow-medium">
<HeartclassName="h-5 w-5 text-white"/>
</div>
<span className="text-xl font-bold bg-gradient-to-r from-primary to-turquoise-light bg-clip-text text-transparent">
              Non Nocere
</span>
</div>
<p  className="text-muted-foreground">
            Медицина починається до прийому
</p>
<p className="text-sm text-muted-foreground italic border-l-2 border-primary pl-3">
            Primum non nocere
</p>
</div>
<div className="space-y-4">
<h4 className="font-bold text-lg">Навігація</h4>
<ul className="space-y-3 text-muted-foreground">
<li>
<Linkto="/doctors"className="hover:text-primary transition-colors flex items-center gap-2 group">
<span className="w-1.5 h-1.5 rounded-full bg-primary/50 group-hover:bg-primary transition-colors"></span>
              Лікарі
</Link>
```

```
</li>
<li>
<Linkto="/services"className="hover:text-primary transition-colors flex items-center gap-2 group">
<span className="w-1.5 h-1.5 rounded-full bg-primary/50 group-hover:bg-primary transition-colors"></span>
            Послуги
</Link>
</li>
<li>
<Linkto="/about"className="hover:text-primary transition-colors flex items-center gap-2 group">
<span className="w-1.5 h-1.5 rounded-full bg-primary/50 group-hover:bg-primary transition-colors"></span>
            Про клініку
</Link>
</li>
</ul>
</div>
<div className="space-y-4">
<h4 className="font-bold text-lg">Контакти</h4>
<ul className="space-y-3 text-muted-foreground">
<li className="flex items-start gap-3">
<div className="w-8 h-8 rounded-lg bg-primary/10 flex items-center justify-center flex-shrink-0">
<MapPinclassName="h-4 w-4 text-primary"/>
</div>
<a
href="https://maps.google.com/?q=50.42618704266967,30.678752367542806"
target="_blank"
rel="noopener noreferrer"
className="hover:text-primary  transition-colors"
>
            вул. Бориспільська, 26А<br />Київ, 02000
</a>
</li>
<li className="flex items-center gap-3">
<div className="w-8 h-8 rounded-lg bg-primary/10 flex items-center justify-center flex-shrink-0">
<PhoneclassName="h-4 w-4 text-primary"/>
</div>
<a  href="tel:+380445665093"className="hover:text-primary  transition-colors  font-medium">
            (044) 566-50-93
</a>
</li>
<li className="flex items-center gap-3">
<div className="w-8 h-8 rounded-lg bg-primary/10 flex items-center justify-center flex-shrink-0">
<MailclassName="h-4 w-4 text-primary"/>
</div>
<a href="mailto:info@nonnocere.com.ua"className="hover:text-primary transition-colors">
            info@nonnocere.com.ua
</a>
</li>
</ul>
</div>
<div className="space-y-4">
<h4 className="font-bold text-lg">Графік роботи</h4>
<ul className="space-y-3 text-muted-foreground">
<li className="flex items-center gap-3">
<div className="w-8 h-8 rounded-lg bg-primary/10 flex items-center justify-center flex-shrink-0">
<ClockclassName="h-4 w-4 text-primary"/>
</div>
<span>Пн-Пт: 8:00 - 20:00</span>
</li>
<li className="pl-11">Сб: 9:00 - 15:00</li>
<li className="pl-11">Нд: вихідний</li>
</ul>
</div>
</div>
<div className="mt-10 pt-8 border-t-2 text-center text-muted-foreground">
<p className="font-medium">Non Nocere {newDate().getFullYear()}</p>
</div>
</div>
</footer>
  )
}
```