Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут
комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: Платформа планування медичних консультацій для приватних клінік

Виконав: студент 6 курсу групи КН- 61м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Чеков Д.П.
(прізвище та ініціали)

Керівник Крошний І.М.
(прізвище та ініціали)

Рецензент Демчук М.В.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"

(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри КН

_____Борецька І.Б.

„10" грудня 2025 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чеков Денис Петрович

(прізвище, ім'я, по батькові)

1. Тема роботи: Платформа планування медичних консультацій для приватних клінік

керівник роботи *Крошний Ігор Миколайович, к.т.н, доцент*

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

затверджені наказом вищого навчального закладу від "29" квітня 2025 року № С-288

2. Термін подання студентом роботи 10 грудня 2025 року

3. Вихідні дані до роботи Розробити платформу планувань медичних консультацій для приватних клінік. Реалізувати можливість підключення чат-боту для надсилання сповіщень про запланований прийом.

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

1.Стан проблемної області

2.Інформаційне забезпечення

3.Математичне забезпечення

4.Програмне забезпечення

5.Розроблення стартап-проєкту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалів до доповіді

6. Дата видачі завдання 1 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Розділ 1. Стан проблемної області	02.05.2025 – 12.05.2025	Виконано
2	Розділ 2. Інформаційне забезпечення	13.05.2025 – 13.06.2025	Виконано
3	Розділ 3. Математичне забезпечення	14.06.2025 – 09.07.2025	Виконано
4	Розділ 4. Програмне забезпечення	10.07.2025 – 21.09.2025	Виконано
5	Розділ 5. Розроблення стартап-проєкту	22.09.2025 – 19.10.2025	Виконано
6	Оформлення дипломної магістерської роботи	20.10.2025 – 10.12.2025	Виконано

Студент _____ **Чеков Д.П.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крошний І.М.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 68 сторінок пояснювальної записки, 24 рисунків, 4 таблиці, 1 додаток, 25 джерел.

Основною метою роботи є розробка веб-орієнтованої платформи «Healthcare» для автоматизації процесів планування та бронювання медичних консультацій у приватних клініках. Використання архітектури SPA (Single Page Application) дозволяє забезпечити миттєвий відгук інтерфейсу та уникнути перезавантаження сторінок при виборі слотів часу.

Інтеграція з Telegram Bot API забезпечує автоматизований канал комунікації для надсилання нагадувань пацієнтам про візити та сповіщень про зміни у розкладі. Для збереження, обробки даних та реалізації серверної частини застосовано технологічний стек MERN: нереляційну базу даних MongoDB, середовище виконання Node.js та фреймворк Express.js.

Ключові слова: медична інформаційна система, онлайн-запис, MERN stack, React, Node.js, TypeScript, MongoDB, Telegram Bot API, SPA.

ABSTRACT

The thesis contains 68 pages of explanatory note, 24 figures, 4 tables, 1 appendices, 25 sources.

The main goal of the work is to develop a web-oriented platform "Healthcare" for automating the processes of scheduling and booking medical consultations in private clinics. The use of SPA (Single Page Application) architecture allows ensuring instant interface response and avoiding page reloads when selecting time slots.

Integration with the Telegram Bot API provides an automated communication channel for sending visit reminders to patients and notifications about schedule changes. The MERN technology stack is used to store, process data, and implement the server side: MongoDB database, Node.js runtime environment, and Express.js framework.

Keywords: medical information system, online appointment, MERN stack, React, Node.js, TypeScript, MongoDB, Telegram Bot API, SPA.

ТЕХНІЧНЕ ЗАВДАННЯ

Для забезпечення ефективного функціонування платформи планування медичних консультацій у приватних клініках необхідно виконати наступні технічні завдання:

1. Проаналізувати поточний стан методів планування медичних консультацій та виявити їхні недоліки, а також здійснити огляд існуючих програмних рішень на ринку для визначення їхніх сильних сторін та прогалин.
2. Розробити масштабовану та гнучку архітектуру платформи, включаючи проектування її клієнтської (Frontend), серверної (Backend) частин та рівня даних, з обґрунтуванням вибору технологічного стеку.
3. Спроектувати структуру бази даних для ефективного зберігання інформації про користувачів, лікарів, їхні розклади та записи на прийом, з урахуванням оптимізації та безпеки даних.
4. Обґрунтувати та розробити математичні моделі та алгоритми для ключових функціональних модулів, зокрема для оптимізації розкладу, перевірки доступності слотів, безпечної автентифікації та авторизації.
5. Моделювати функціональні модулі системи та сценарії взаємодії користувачів (пацієнтів, лікарів) з інтерфейсом платформи, а також визначити принципи реалізації ключових програмних компонентів.
6. Передбачити та спроектувати інтеграцію платформи із зовнішніми сервісами, зокрема з Telegram Bot API, для реалізації автоматизованих сповіщень та нагадувань про заплановані візити.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних.

MERN – це абривіатура яка означає стек тохнологій для розробки веб-додатків: Mongodb, Express, React, Node.

UML (Unified Modeling Language) – це стандартизована графічна мова яка використовується для візуалізації, проектування та документування програмних систем, UML використовується для створення діаграм які чітко показуються структуру системи.

SPA (Single Page Application) – веб-додаток на одну сторінку, це додаток який загружає одну HTML-сторінку та динамічно її оновлює без необхідності переключення на іншу сторінку, так додаток працює швидше і більш інтерактивно.

MPA (Multi-Page Application) – веб додаток на багато сторінок, це додаток у якому кожна нова сторінка створюється з оновленням сторінки.

NPM (Node package manager) – це менеджер пакетів для мови програмування JavaScript, встановлюється по замовчуванню з Node.js, дозволяє легко встановлювати та керувати залежностями проекту.

ВСТУП

Актуальність теми. В умовах динамічного розвитку ринку приватних медичних послуг ключовим фактором конкурентоспроможності клініки стає не лише якість лікування, але й рівень сервісу та ефективність адміністративних процесів. Інтеграція цифрових екосистем у діяльність медичних закладів дозволяє трансформувати підхід до взаємодії з клієнтами, забезпечуючи високий рівень лояльності та оптимізацію використання ресурсів.

Традиційні підходи до організації візитів, що базуються на телефонній комунікації або застарілих локальних системах, виявляють свою неефективність: вони призводять до виникнення організаційних колізій у розкладі, нерівномірного навантаження на спеціалістів та втрати потенційних клієнтів через незручні канали комунікації. Для приватного сектору медицини це означає прямі фінансові втрати та зниження репутаційних показників.

Розробка спеціалізованої платформи планування медичних консультацій є актуальним завданням, оскільки дозволяє створити єдиний цифровий простір для координації роботи лікарів та пацієнтів. Таке рішення забезпечує прозорість графіків, мінімізує вплив людського фактору при бронюванні, автоматизує нагадування та вивільняє час адміністративного персоналу для виконання більш пріоритетних завдань. Впровадження веб-орієнтованої платформи відповідає сучасним вимогам до eHealth-систем та очікуванням користувачів щодо мобільності та доступності сервісів 24/7.

Метою роботи є підвищення ефективності роботи приватних медичних закладів шляхом розробки веб-платформи, що забезпечує автоматизацію процесів планування, бронювання та супроводу медичних консультацій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз стану проблемної області та існуючих програмних продуктів для медичного менеджменту.
2. Здійснити системний аналіз об’єкта дослідження, побудувати дерево цілей та діаграми потоків даних (DFD).

3. Розробити інформаційне забезпечення системи, включаючи концептуальну та логічну моделі бази даних.
4. Обґрунтувати математичне забезпечення для прийняття рішень при виборі архітектури системи.
5. Виконати програмну реалізацію платформи з використанням сучасних веб-технологій (MERN stack, TypeScript).

Об’єктом дослідження є процес організації та управління потоком пацієнтів у приватних медичних клініках.

Предметом дослідження є методи та інформаційні технології автоматизації планування медичних консультацій та створення веб-орієнтованих платформ для сфери охорони здоров’я.

Наукова новизна одержаних результатів полягає у подальшому розвитку методів проєктування медичних інформаційних систем шляхом інтеграції модуля автоматизованого сповіщення через месенджери в єдиний контур планування ресурсів клініки, що дозволяє зменшити кількість неявок пацієнтів (no-show rate).

Практичне значення отриманих результатів визначається створенням працездатного програмного продукту, який дозволяє:

- забезпечити пацієнтам зручний інтерфейс для самостійного вибору спеціаліста та часу візиту;
- надати лікарям інструментарій для динамічного управління власним графіком (слотами прийому);
- автоматизувати комунікацію з клієнтами через Telegram-бот;
- зменшити операційне навантаження на рецепцію клініки.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис проблемної області

Ефективне функціонування системи охорони здоров'я є критичним показником якості життя суспільства. Протягом останнього десятиліття галузь переживає фундаментальні зміни, зумовлені стрімкою інтеграцією інноваційних інформаційно-комунікаційних технологій. Ці процеси докорінно змінюють механізми взаємодії між надавачами медичних послуг та пацієнтами, трансформуючи традиційну модель медицини у цифрову екосистему [1].

Ключовим викликом для сучасних медичних установ, особливо у приватному секторі, залишається оптимізація адміністративних процесів, зокрема менеджменту візитів. Традиційні методи організації запису, такі як телефонна комунікація або особистий візит до реєстратури, сьогодні демонструють свою неефективність. Вони призводять до виникнення фізичних черг, нерівномірного навантаження на медичний персонал, помилок у графіках та нераціонального використання часових ресурсів як лікарів, так і пацієнтів.

Впровадження спеціалізованих медичних інформаційних систем (MIC) дозволяє вирішити ці проблеми шляхом автоматизації рутинних операцій. Зменшення паперового документообігу та перехід на електронні рейки дозволяє закладам заощаджувати значні матеріальні та людські ресурси, знижуючи операційні витрати. Сучасні системи класу eHealth надають пацієнтам можливість самостійно керувати своїми візитами: переглядати актуальний розклад лікаря, заплановувати прийом в режимі онлайн, отримувати автоматичні нагадування та скасовувати запис без необхідності контакту з реєстратурою .

Світовий досвід підтверджує ефективність такого підходу. Країни з розвиненою цифровою медициною, такі як Естонія (де 99% даних оцифровано), Велика Британія (сервіс NHS App) та Польща (система e-Recepta), демонструють значне підвищення якості послуг та економію бюджетних коштів завдяки впровадженню національних систем eHealth .

В Україні процес цифровізації активізувався з 2017 року із запровадженням реформи та системи eHealth, що базується на відкритій архітектурі [2] та використанні API. Це створило конкурентне середовище для розробників МІС, сприяючи появі різноманітних програмних рішень для автоматизації медичної сфери . Статистичні дані підтверджують попит на такі послуги: ще на етапі старту реформи понад півмільйона українців скористалися онлайн-записом, а відвідуваність популярних платформ обчислюється мільйонами користувачів щомісяця. Як видно з наведеної статистики (рис. 1.1), ринок насичений різними рішеннями, кожне з яких має свої особливості. Для обґрунтування вимог до нової системи доцільно провести детальний аналіз функціональних можливостей лідерів ринку.

Показники відвідуваності медичних веб-сервісів

	helsi	DOC.UA	Лікарні	Health 24
Відвідуваність на місяць, млн	1.57	0.95	0.46	0.29
Тривалість одного відвідування, хв	6.5	2.29	2.59	11.24
Сторінок за візит	8.19	2.03	2.37	7.96
Частка відвідувачів з України, %	99	83	94	98
Джерело трафіку, %				
прямий	58	15	11	24
пошуковий	32	81	86	72

Джерело: дані SimilarWeb

Рисунок 1.1 — Дані SimilarWeb по відвідуваності медичних веб-сервісів

1.2 Аналіз існуючих рішень

На українському ринку програмного забезпечення для медицини представлено широкий спектр рішень, які можна класифікувати як комплексні медичні інформаційні системи (МІС) та сервіси-агрегатори. Розглянемо детально найбільш поширені з них.

1.2.1. Helsi.me

Система Helsi (рис. 1.2) є найпопулярнішою та найбільш впізнаваною системою в Україні, вона охоплює як державні комунальні заклади, так і приватні клініки. Вона

виступає ключовим інструментом у реалізації медичної реформи, забезпечуючи взаємодію з Центральною базою даних eHealth.

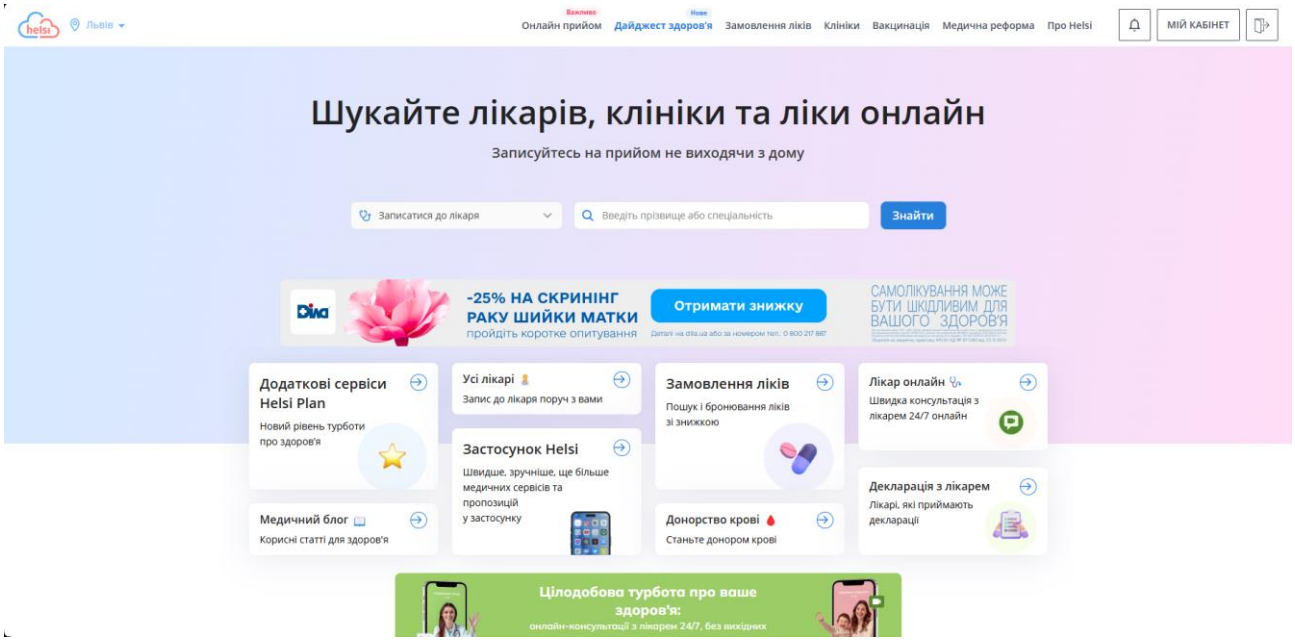


Рисунок 1.2 — Сторінка Helsi.me

Архітектура системи спрямована на комплексну автоматизацію робочих процесів у поліклініках, стаціонарах, лабораторіях, лікарнях та аптеках. Важливою перевагою є наявність вбудованого конструктора медичних документів та інструментів для формування статистичної звітності відповідно до вимог НСЗУ.

Функціональні можливості для пацієнтів: Інтерфейс кабінету пацієнта надає широкі можливості для самообслуговування. Користувачі можуть здійснювати пошук лікарів за спеціалізацією або геолокацією та записуватися на прийом у режимі реального часу. Система надає доступ до Електронної медичної картки (ЕМК), де зберігається історія відвідувань, діагнози, результати аналізів та призначені плани лікування. Це забезпечує прозорість медичного обслуговування та дозволяє пацієнту мати під рукою всю необхідну інформацію .

Функціональні можливості для лікарів та закладів: для медичного персоналу Helsi пропонує інструменти ведення електронної документації, що дозволяє відмовитися від паперових карток. Лікарі мають доступ до інтегрованих клінічних протоколів, історії хвороб пацієнтів та результатів діагностики. Адміністративний

модуль дозволяє керувати реєстрацією декларацій, моніторити завантаженість персоналу та генерувати звітність для державних органів .

Критичний аналіз: попри масштабність, Helsi має суттєві недоліки для невеликих приватних клінік. Система перевантажена функціоналом, необхідним для державної звітності, що ускладнює інтерфейс та вимагає тривалого навчання персоналу. Крім того, жорстка структура розкладу не завжди дозволяє гнучко налаштовувати слоти прийому, що є критичним для приватної практики.

1.2.2. Doc.ua

Платформа Doc.ua (рис. 1.3) позиціонується як провідний медичний онлайн-хаб та агрегатор послуг. На відміну від класичних МІС, основний акцент тут зроблено на маркетинговій складовій та зручності пошуку медичних послуг для кінцевого споживача.

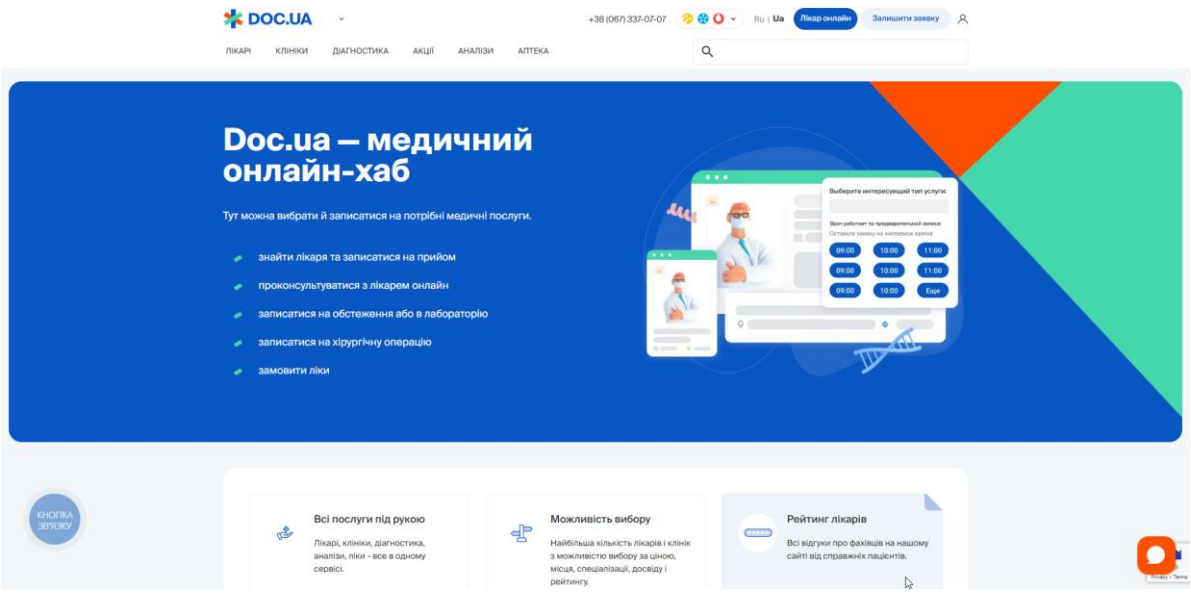


Рисунок 1.3 — Сторінка Doc.ua

Сервіс об'єднує тисячі лікарів та клінік, надаючи користувачам інструменти для вибору фахівця на основі реальних відгуків та рейтингів. Це сприяє підвищенню прозорості ринку та формуванню довіри до приватної медицини .

Функціональні можливості для пацієнтів: користувачі платформи отримують доступ до зручного каталогу з розширеними фільтрами: за спеціалізацією, районом міста, ціною прийому та рейтингом лікаря. Окрім класичного запису на очний прийом, реалізовано можливість замовлення телемедичних консультацій, запису на

лабораторні дослідження та діагностику. Важливою особливістю є наявність бонусної програми, знижок та акційних пропозицій, що підвищує лояльність аудиторії .

Функціональні можливості для лікарів та клінік: для медичних закладів Doc.ua виступає потужним каналом залучення нових клієнтів (лідогенерації). Особистий кабінет дозволяє керувати записами, налаштовувати онлайн-календар та обробляти заявки. Платформа надає маркетингові інструменти для просування послуг клініки та підвищення впізнаваності бренду лікаря .

Критичний аналіз: Doc.ua є ефективним інструментом маркетингу, але не може повністю замінити внутрішню систему управління клінікою. Модель монетизації, що передбачає комісію за кожного пацієнта, може бути фінансово обтяжливою для малих кабінетів. Крім того, функціонал управління внутрішніми процесами (склад, фінанси, кадри) тут відсутній.

1.2.3. Likarni.com

Онлайн-сервіс Likarni.com (рис. 1.4) є ще одним популярним агрегатором, який фокусується на наданні актуальної інформації про приватні клініки та діагностичні центри України.

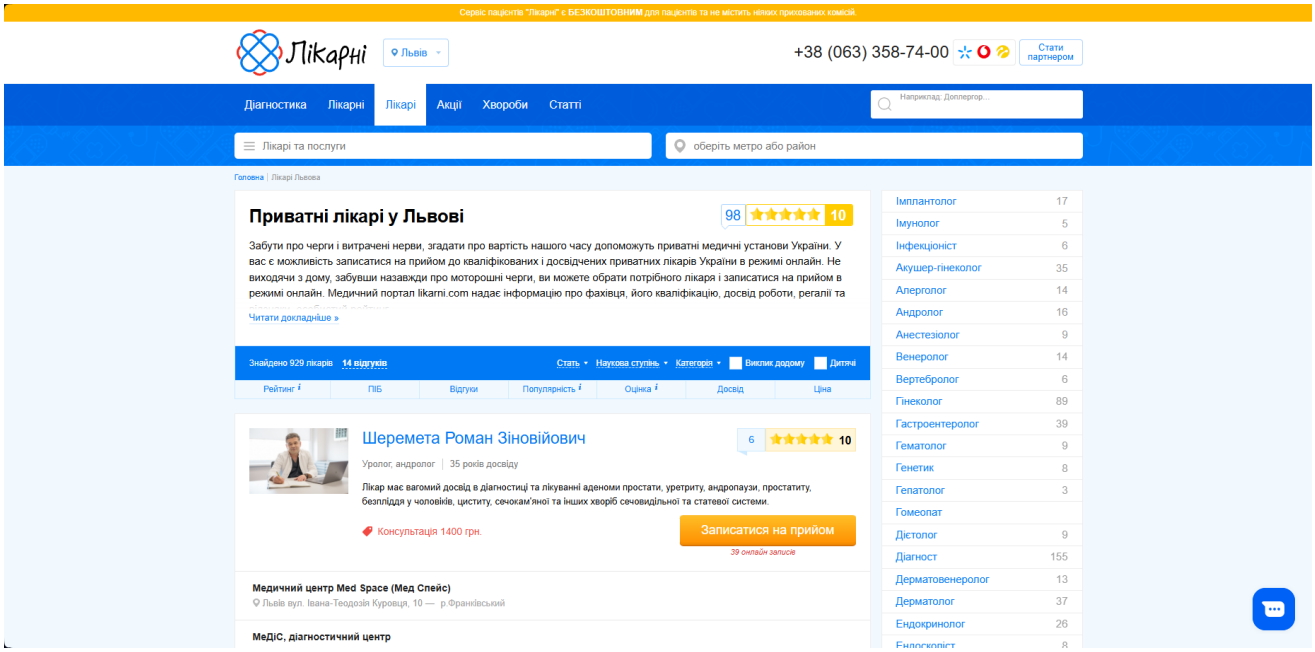


Рисунок 1.4 — Сторінка Likarni.com

Ресурс дозволяє пацієнтам порівнювати умови прийому в різних закладах, ознайомлюватися з переліком послуг та ціновою політикою. Ключовою особливістю є відкрита система відгуків, що допомагає користувачам приймати зважені рішення .

Функціональні можливості для пацієнтів: Система пропонує інтуїтивний пошук за симптомами, захворюваннями або методами діагностики (МРТ, УЗД тощо). Пацієнт може записатися на прийом онлайн, поставити запитання лікарю через форму зворотного зв'язку та отримати персональні рекомендації. Окремий акцент зроблено на інформаційному наповненні: статті та медичні довідники допомагають користувачам краще орієнтуватися у питаннях здоров'я .

Функціональні можливості для лікарів та клінік: Партнери сервісу отримують можливість розміщувати детальну інформацію про свої послуги, графік роботи та обладнання. Особистий кабінет надає базові функції для управління вхідним потоком пацієнтів та налаштування розкладу. Рейтингова система стимулює заклади покращувати якість обслуговування .

Критичний аналіз: Як і попередній аналог, Likarni.com є переважно інформаційно-пошуковою системою. Вона відмінно справляється з задачею первинного контакту пацієнта з клінікою, проте не забезпечує інструментів для утримання клієнтів (CRM), автоматизації повторних візитів та глибокої інтеграції з робочим процесом лікаря.

1.2.4. Health24

Health24 (рис. 1.5) представляє собою сучасну хмарну медичну інформаційну систему, яка поєднує функції МІС та порталу для пацієнтів. Вона сертифікована для роботи з eHealth та підтримує стандарти захисту інформації.

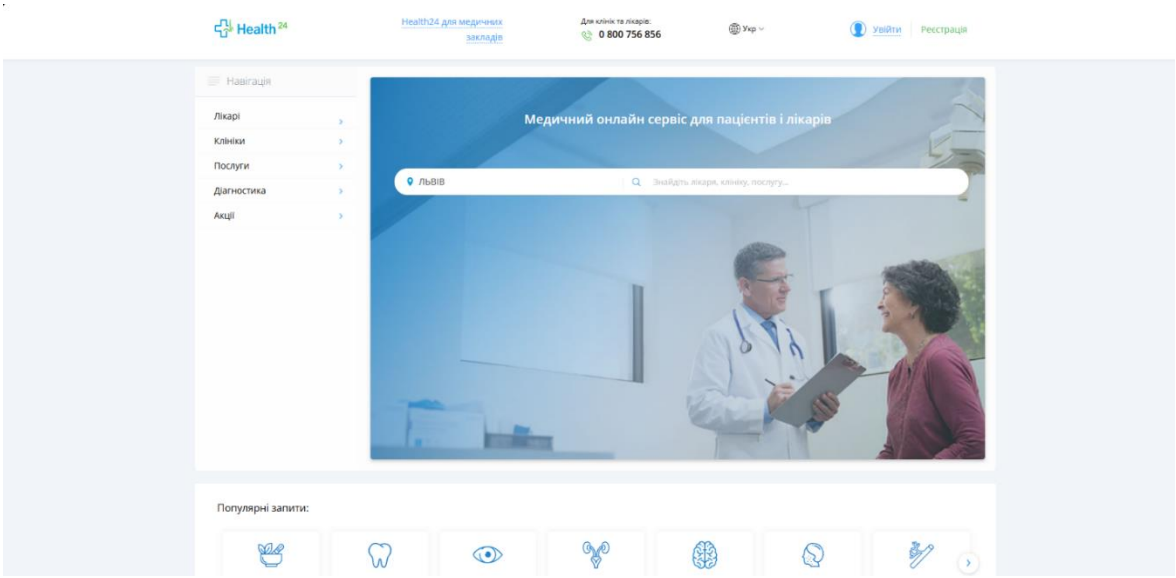


Рисунок 1.5 — Сторінка Health24.ua

Система розроблена для повної цифровізації діяльності медичного закладу, забезпечуючи автоматизацію як клінічних, так і адміністративних процесів.

Функціональні можливості: для пацієнтів доступний повний цикл електронних сервісів: від запису до лікаря до перегляду результатів аналізів та історії хвороби. Для лікарів реалізовано потужний функціонал ведення ЕМК, створення шаблонів оглядів, виписування направлень та рецептів. Для адміністрації доступні модулі кадрового обліку, складського обліку, фінансової звітності та бізнес-аналітики. Важливим аспектом є інтеграція з лабораторними системами та підтримка телемедицини.

Захист даних: Health24 приділяє значну увагу безпеці: використовується шифрування даних, дворівнева авторизація та розмежування прав доступу, що відповідає вимогам законодавства про захист персональних даних [3].

Критичний аналіз: незважаючи на потужний функціонал, Health24 є складним та вартісним рішенням. Для індивідуальних лікарів-практиків або невеликих кабінетів вартість впровадження та підтримки такої системи може бути невиправданою. Крім того, інтерфейс системи вимагає часу на освоєння, що може сповільнювати роботу персоналу на початкових етапах.

1.2.5. Порівняльна характеристика аналогів

Для наочного відображення переваг та недоліків розглянутих систем у порівнянні з платформою «Healthcare» було складено порівняльну таблицю (Таблиця 1.1).

Таблиця 1.1 — Порівняльна характеристика функціональних можливостей систем-аналогів

Критерій порівняння	**Helsi.me**	**Doc.ua**	**Likarni**	**Health24**	**Healthcare**
Основна цільова аудиторія	Державні заклади, великі мережі	Пацієнти (пошук лікаря)	Пацієнти (пошук лікаря)	Великі приватні та держ. клініки	Приватні кабінети, малий бізнес
Наявність мобільного додатку	Так	Так	Ні	Так	Ні
Інтеграція з Telegram (сповіщення)	Ні	Ні	Ні	Ні	Так
Складність інтерфейсу	Висока	Середня	Середня	Висока	Низька (Мінімалістичний)
Гнучкість налаштування графіку	Низька	Н/Д (зовнішній запис)	Н/Д	Середня	Висока (Динамічні слоти)
Необхідність навчання персоналу	Так	Ні	Так	Ні	Ні

Як видно з таблиці 1.1, існуючі на ринку рішення (Helsi, Health24) мають широкий функціонал, але характеризуються високою складністю та вартістю, що робить їх надлишковими для малого бізнесу. агрегатори (Doc.ua) вирішують проблему пошуку пацієнтів, але не надають інструментів для внутрішнього

управління розкладом. Система «Healthcare» заповнює цю нішу, пропонуючи баланс між простотою, вартістю та сучасними комунікаційними можливостями через Telegram.

1.3 Висновки до розділу

У першому розділі було проведено комплексний аналіз предметної області та стану ринку цифрових рішень в охороні здоров'я України. Встановлено, що автоматизація процесу запису до лікаря є необхідною умовою ефективної роботи сучасного медичного закладу. Детальний огляд існуючих платформ (Helsi, Doc.ua, Likarni.com, Health24) дозволив виявити наступні закономірності:

Існуючі рішення здебільшого орієнтовані на великі державні заклади або мережеві клініки.

Агрегатори послуг працюють за комісійною моделлю, що не завжди вигідно для малого бізнесу.

Відчувається дефіцит легких, гнучких та доступних рішень, які б фокусувалися на зручності комунікації (зокрема через месенджери) та простоті управління графіком.

Це обґрунтовує доцільність розробки власної інформаційної системи, яка поєднає в собі функції онлайн-запису, автоматизованих нагадувань через Telegram та інтуїтивного управління розкладом, задовольняючи специфічні потреби приватних клінік та індивідуальних лікарів.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Характеристика та класифікація інформаційних потоків

Для забезпечення коректного функціонування платформи та прийняття ефективних рішень щодо архітектури бази даних, необхідно провести детальний аналіз вхідної та вихідної інформації. Специфіка медичної сфери вимагає особливої уваги до точності, актуальності та цілісності даних.

2.1.1. Вхідна інформація

Вхідна інформація – це сукупність даних, що надходять до системи від зовнішніх об'єктів (користувачів, адміністраторів, зовнішніх API) і підлягають реєстрації, обробці або збереженню. У системі «Healthcare» вхідні дані можна класифікувати за наступними категоріями:

1. Реєстраційно-облікова інформація:
    - *Дані ідентифікації:* прізвище, ім'я, по батькові користувачів, адреси електронної пошти, номери телефонів.
    - *Дані автентифікації:* паролі (що підлягають хешуванню), ідентифікатори сесій [4].
    - *Профільні дані лікаря:* інформація про спеціалізацію, кваліфікацію, опис професійного досвіду.
2. Планова інформація (Дані розкладу):
    - Керуючі параметри робочого часу, що вносяться лікарем: дати робочих днів, час початку та завершення зміни, тривалість одного прийому (слоту).
    - Дані про винятки у розкладі (відпустки, лікарняні, форс-мажорні зміни).
3. Оперативна інформація (Транзакційні дані):
    - Параметри запиту на бронювання: обрана дата, конкретний часовий слот, ідентифікатор пацієнта.
    - Команди управління статусом: запити на скасування візиту, підтвердження виконання послуги.
4. Комунікаційна інформація:

- Дані для налаштування сповіщень: Telegram Chat ID, налаштування частоти нагадувань.

2.1.2. Вихідна інформація

Вихідна інформація є результатом логічної та аналітичної обробки вхідних даних системою. Вона призначена для візуалізації користувачам або передачі зовнішнім сервісам.

- Інформаційні дашборди: персоналізовані переліки запланованих візитів в особистих кабінетах лікаря та пацієнта.
- Візуалізація зайнятості: графічне відображення календаря, де чітко розмежовано доступні (вільні) та заброньовані слоти.
- Електронні підтвердження: згенеровані системою "талони" з деталями запису (ПІБ лікаря, адреса, час).
- Системні сповіщення: автоматично сформовані текстові повідомлення, що передаються через шлюз Telegram API (нагадування про візит, повідомлення про зміни в розкладі).

2.2 Об'єктно-орієнтоване моделювання системи (UML)

Для формалізації функціональних вимог та опису логіки взаємодії акторів із системою використано методологію об'єктно-орієнтованого аналізу на базі мови UML (Unified Modeling Language) [5]. Ключовим етапом проєктування є розробка діаграми прецедентів (Use Case Diagram), яка дозволяє визначити межі системи та сценарії її використання (рис. 2.1).

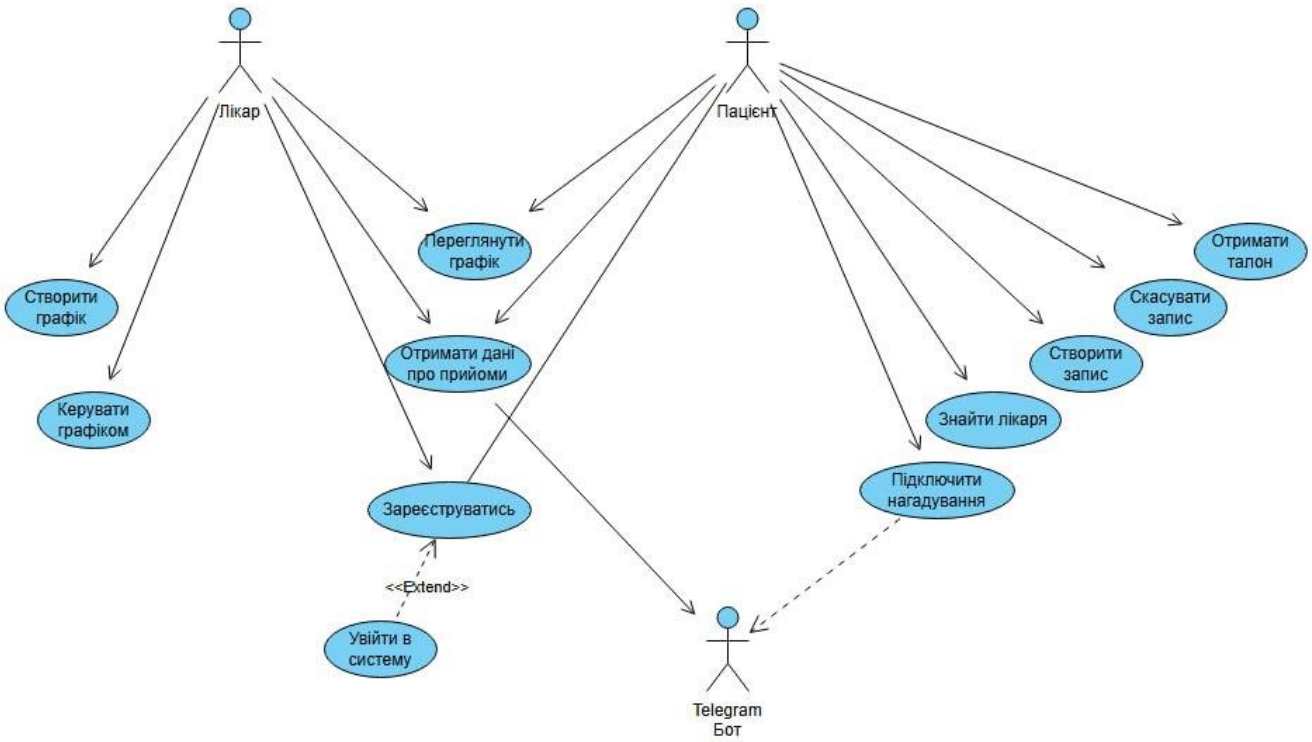


Рисунок 2.1 — Діаграма прецежентів системи «Healthcare»

2.2.1. Характеристика акторів системи

На діаграмі виділено три основні типи акторів, кожен з яких має специфічну роль та набір прав доступу:

1. Лікар (Doctor): активний користувач системи, що виступає в ролі постачальника послуг. Його основна мета — ефективне управління власним часовим ресурсом та отримання інформації про потік пацієнтів.
2. Пацієнт (Patient): кінцевий споживач послуг. Його взаємодія із системою спрямована на пошук необхідного спеціаліста та резервування часу для отримання медичної допомоги.
3. Telegram Бот (External Actor): зовнішній системний актор, що виконує роль посередника у комунікації. Він не ініціює процеси самостійно, але реагує на події в системі, забезпечуючи доставку інформації користувачу.

2.2.2. Детальний опис прецедентів (Use Cases)

Діаграма відображає основні сценарії використання, які покривають бізнес-процеси платформи.

Блок ідентифікації та доступу:

- *Зареєструватись / Увійти в систему:* це базові прецеденти, обов'язкові для початку роботи. На діаграмі показано відношення <<Extend>>, що демонструє логічний зв'язок: процес реєстрації може бути розширений процесом входу (або навпаки, вхід можливий тільки після реєстрації). Це забезпечує захист персональних даних та розмежування простору пацієнта і лікаря.

Функціональний блок "Управління розкладом" (зона відповідальності Лікаря):

- *Створити графік:* лікар ініціює генерацію слотів на певний період. Система повинна перевірити коректність дат та відсутність перетинів з існуючими записами.
- *Керувати графіком:* цей прецедент включає можливості редагування (зміна часу) або видалення слотів у разі зміни життєвих обставин лікаря.
- *Отримати дані про прийоми:* Перегляд списку записаних пацієнтів. Це дозволяє лікарю підготуватися до прийому заздалегідь.

Функціональний блок "Бронювання та пошук" (зона відповідальності Пацієнта):

- *Знайти лікаря:* пацієнт використовує пошукові фільтри (спеціалізація, прізвище) для знаходження потрібного фахівця.
- *Переглянути графік:* система відображає актуальний стан розкладу обраного лікаря, фільтруючи вже зайняті слоти.
- *Створити запис:* центральний прецедент системи. Користувач ініціює транзакцію бронювання, яка змінює статус слоту з "Вільний" на "Зайнятий" та створює зв'язок між пацієнтом і лікарем.
- *Отримати талон:* після успішного запису система генерує підтвердження, яке слугує гарантією бронювання.
- *Скасувати запис:* пацієнт має можливість скасувати візит. Ця дія запускає ланцюжок подій: звільнення слоту в розкладі лікаря та відправку сповіщення лікарю.

Блок "Сповіщення":

- *Підключити нагадування:* пацієнт ініціює взаємодію з Telegram-ботом. Цей прецедент встановлює зв'язок між обліковим записом на веб-платформі та

акаунтом у месенджері, що є необхідною умовою для роботи системи автоматичних нагадувань.

2.3 Концептуальна модель даних

На основі аналізу інформаційних потоків та сценаріїв використання було розроблено концептуальну модель предметної області. Вона визначає основні сутності, їх атрибути та логічні зв'язки, абстрагуючись від конкретної фізичної реалізації в СУБД. Така модель є необхідною для забезпечення цілісності даних та уникнення дублювання інформації.

Сутність «User» (Користувач) Це узагальнена сутність, що представляє будь-яку зареєстровану особу в системі. Вона містить атрибути, спільні для всіх ролей, що дозволяє уніфікувати процес авторизації.

- *Атрибути:* унікальний ідентифікатор, email (логін), хешований пароль, роль (Лікар/Пацієнт), Telegram Chat ID.

Сутність «Schedule» (Розклад) Описує ресурс часу, який пропонується лікарем. Враховуючи специфіку роботи приватних клінік, графік може бути нерегулярним, тому доцільно моделювати його як набір окремих часових інтервалів (слотів).

- *Атрибути:* дата, час початку, час завершення, статус доступності (Вільний/Забронюваний).
- *Зв'язок:* має відношення "багато-до-одного" з сутністю Лікаря (один лікар володіє багатьма слотами часу).

Сутність «Appointment» (Запис) Транзакційна сутність, яка фіксує факт домовленості між двома сторонами. Вона виникає в момент бронювання.

- *Атрибути:* дата створення запису, дата візиту, статус запису (Активний, Виконаний, Скасований), коментар.
- *Зв'язки:* ця сутність є сполучною ланкою, що має посилання на конкретного Пацієнта та конкретного Лікаря.

2.4 Інструментальні засоби проєктування

Для реалізації етапу інформаційного проєктування було обрано набір сучасних інструментів, що забезпечують ефективність розробки та якість документації.

1. Засоби візуального моделювання: для побудови UML-діаграм використано спеціалізоване програмне забезпечення (наприклад, Visual Paradigm або Draw.io). Це дозволило створити наочні моделі прецедентів, які є зрозумілими як розробникам, так і замовнику [6].
2. Система контролю версій Git: для управління змінами в коді та документації обрано розподілену систему Git. Вона дозволяє фіксувати історію розвитку проєкту, працювати з різними версіями (гілками) функціоналу та забезпечує можливість командної роботи.
3. Платформа GitHub: використовується як віддалений репозиторій для надійного зберігання вихідного коду та документації.

2.5 Висновки до розділу

У другому розділі здійснено комплексне проєктування інформаційного забезпечення платформи «Healthcare».

1. Проведено детальний аналіз та класифікацію вхідної та вихідної інформації, що дозволило чітко окреслити інформаційний периметр системи.
2. За допомогою уніфікованої мови моделювання UML розроблено діаграму прецедентів, яка формалізує функціональні вимоги та описує сценарії взаємодії основних акторів (Лікаря, Пацієнта, Бота).
3. Розроблено концептуальну модель даних, що визначає структуру інформаційних об'єктів (Користувач, Розклад, Запис) та зв'язки між ними.
4. Спроєктована інформаційна архітектура створює надійну основу для подальшої програмної реалізації системи, забезпечуючи її логічну цілісність та відповідність потребам користувачів.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору архітектури системи

Для вибору оптимальної архітектури веб-платформи було використано метод аналізу ієрархій (MAI), розроблений Т. Сааті [7]. Задача полягає у виборі між двома альтернативами:

• Альтернатива A_1: SPA (Single Page Application) — обраний нами підхід (React).

• Альтернатива A_2: MPA (Multi-Page Application) — класичний підхід з перезавантаженням сторінок.

Критерії порівняння (K):

• K_1: Швидкодія інтерфейсу (User Experience).

• K_2: Швидкість розробки (Time to Market).

• K_3: Масштабованість (Scalability).

• K_4: Навантаження на сервер.

3.1.1. Побудова матриці попарних порівнянь

Для визначення ваг критеріїв будується матриця A, де елемент a_{ij} відображає перевагу критерію i над критерієм j.

$$A = \begin{pmatrix} 1 & 3 & 2 & 5 \\ 1/3 & 1 & 1/2 & 2 \\ 1/2 & 2 & 1 & 3 \\ 1/5 & 1/2 & 1/3 & 1 \end{pmatrix} \quad (3.1)$$

Для даної матриці розраховано власний вектор W (ваги критеріїв):

• W_1 (Швидкодія) ≈ 0.48

• W_2 (Швидкість розробки) ≈ 0.14

• W_3 (Масштабованість) ≈ 0.27

• W_4 (Навантаження) ≈ 0.11

3.1.2. Оцінка альтернатив

Для кожного критерію проведено порівняння альтернатив A_1 та A_2.

• За критерієм K_1 (Швидкодія) SPA має значну перевагу над MPA.

• За критерієм K_3 (Масштабованість) SPA також переважає завдяки розділенню фронтенду і бекенду [8].

3.1.3. Синтез глобальних пріоритетів

Інтегральна оцінка для кожної альтернативи розраховується за формулою:

$$S(A_i) = \sum_{j=1}^{n} w_j \cdot v_{ij} \quad (3.2)$$

де w_j — вага критерію, а v_{ij} — оцінка альтернативи за цим критерієм.

У результаті розрахунків отримано:

$$S(A_{SPA}) = 0.72, \quad S(A_{MPA}) = 0.28 \quad (3.3)$$

Оскільки $0.72 > 0.28$, математично обґрунтованим є вибір архітектури SPA (Single Page Application), що підтверджує доцільність використання бібліотеки React.

3.2 Математична модель планування розкладу

Ключовою функцією системи є запобігання колізіям (подвійному бронюванню) при записі на прийом. Для формалізації цієї задачі використано апарат теорії множин та часової логіки.

Нехай $D = \{d_1, d_2, \dots, d_n\}$ — множина лікарів у системі.

Нехай T — дискретний простір часу.

Графік роботи лікаря d_i можна представити як множину доступних часових інтервалів (слотів):

$$S_i = \{[t_{start}^k, t_{end}^k] \mid k \in N\} \quad (3.4)$$

де t_{start}^k — час початку k-го слоту, t_{end}^k — час завершення.

Умова коректності розкладу:

Інтервали всередині графіку одного лікаря не повинні перетинатися:

$$\forall [t_a, t_b], [t_c, t_d] \in S_i : ([t_a, t_b] \neq [t_c, t_d]) \backslash implies([t_a, t_b] \cap [t_c, t_d] = \emptyset) \quad (3.5)$$

Алгоритм бронювання:

Коли пацієнт намагається створити запис на інтервал $R = [t_{req\_start}, t_{req\_end}]$, система виконує перевірку належності цього інтервалу до множини вільних слотів S_{free}:

$$S_{free} = S_i \setminus A_i \tag{3.6}$$

де A_i — множина вже існуючих записів (Appointments) лікаря.

Бронювання можливе тоді і тільки тоді, коли:

$$\exists s \in S_{free} : R \subseteq s \tag{3.7}$$

Ця математична модель реалізована у програмному коді на рівні контролера AppointmentController, що гарантує цілісність даних розкладу.

3.3 Математичні основи захисту даних

Для забезпечення конфіденційності даних користувачів у системі використовуються криптографічні перетворення [9].

Модель хешування паролів:

Для зберігання паролів не використовується відкритий текст P. Замість цього зберігається хеш-значення H, отримане за допомогою алгоритму bcrypt.

Математично процес описується функцією:

$$H = E_{bcrypt}(P, Salt, Cost) \tag{3.8}$$

де:

- P — пароль користувача.
- $Salt$ — випадкова послідовність біт (128 біт), що додається до пароля для захисту від атак через райдужні таблиці.
- $Cost$ — параметр складності (кількість ітерацій алгоритму), що робить підбір пароля методом грубої сили обчислювально неможливим за прийнятний час.

Перевірка пароля при вході здійснюється шляхом порівняння хешів:

$$Check(P_{input}, H_{stored}) = \begin{cases} True, \text{якщо } E_{bcrypt}(P_{input}, Salt_{stored}) = H_{stored} \\ False, \quad \text{в іншому випадку} \end{cases} \tag{3.9}$$

3.4 Алгоритми роботи системи

Для реалізації функціональних вимог платформи та забезпечення коректності обробки даних було розроблено низку алгоритмів. Вони базуються на теорико-множинних моделях, описаних у попередніх підрозділах, і визначають логіку поведінки системи в критичних точках (бронювання, пошук, сповіщення).

3.4.1. Алгоритм перевірки доступності слоту

Процес бронювання візиту вимагає суворої перевірки даних для уникнення накладок у розкладі. Алгоритм отримує на вхід бажаний час запису та перевіряє його на перетин з усіма існуючими записами у базі даних для обраного лікаря (рис. 3.1).

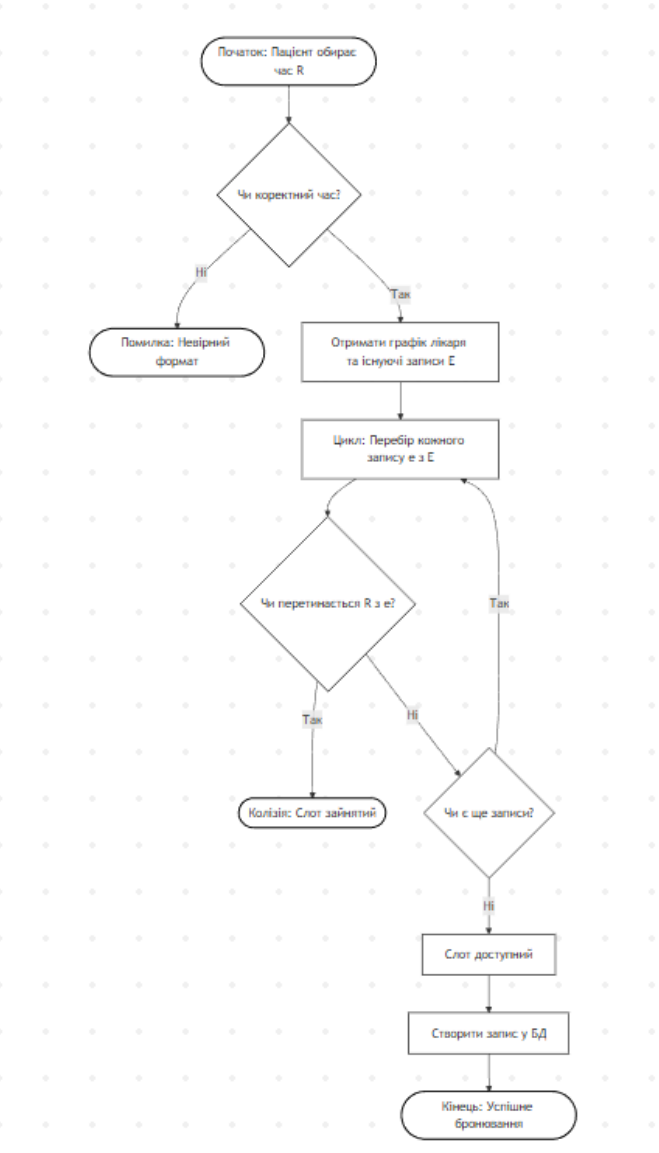


Рисунок 3.1 — Алгоритм переверіки доступності бронювання

Як видно зі схеми, процес включає етап валідації формату введених даних, завантаження актуального розкладу з БД та ітеративну перевірку умов перетину. Якщо жодної колізії не виявлено, система переходить до транзакції створення запису.

3.4.2. Алгоритм пошуку та фільтрації лікарів

Для забезпечення швидкого доступу пацієнтів до необхідних спеціалістів розроблено алгоритм фільтрації. Він працює на стороні бази даних MongoDB, використовуючи індексовані поля. Алгоритм аналізує вхідний пошуковий запит. Якщо запит порожній, система повертає повний список лікарів. У іншому випадку відбувається розгалуження логіки: пошук за підрядком у полях "Прізвище" та "Спеціалізація" (рис. 3.2). Даний підхід дозволяє зменшити навантаження на клієнтську частину, оскільки фільтрація відбувається на сервері, і на фронтенд передається лише релевантний набір даних D_{result}

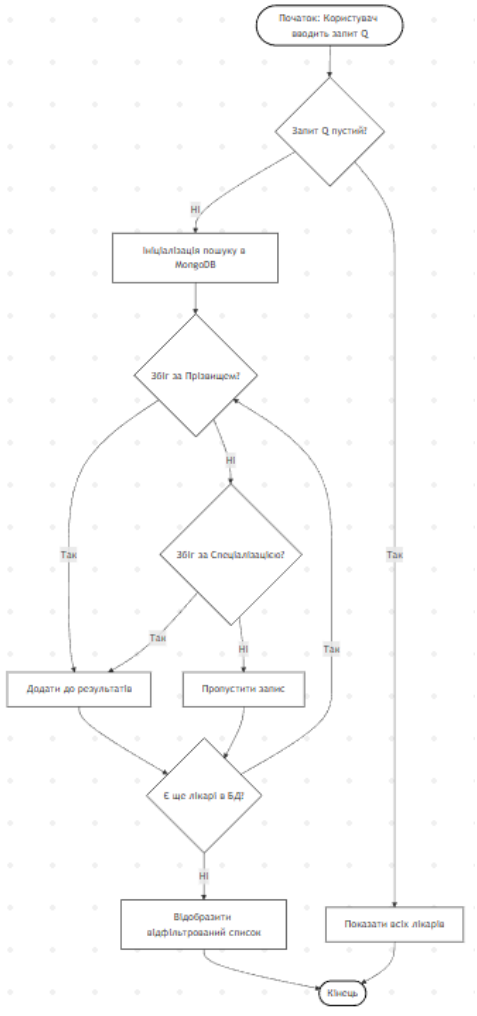


Рисунок 3.2 — Алгоритм пошуку лікарів

3.4.3. Алгоритм диспетчера сповіщень

Підсистема сповіщень реалізована як фоновий процес, що працює незалежно від дій користувача. Алгоритм базується на періодичному опитуванні бази даних. Диспетчер отримує поточний системний час T_{now}, вибирає всі активні записи та розраховує різницю часу Δt до початку візиту. Якщо Δt дорівнює контрольним значенням (24 години або 6 годин), ініціюється процедура відправки повідомлення через Telegram API (рис. 3.3).

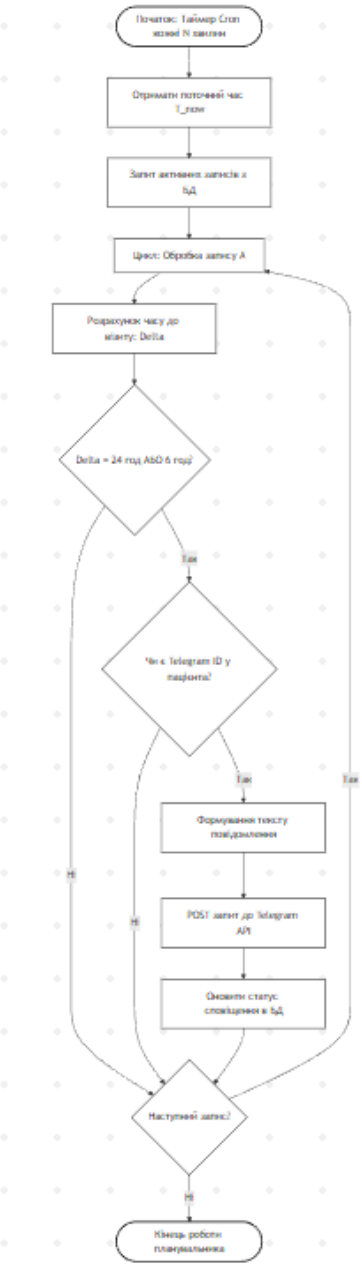


Рисунок 3.3 — Алгоритм надсилання сповіщень

3.5 Висновки до розділу

1. За допомогою методу аналізу ієрархій проведено математичне обґрунтування вибору архітектури SPA, довівши її перевагу над MPA за сукупністю критеріїв швидкодії та масштабованості.

2. Розроблено теоретико-множинну модель планування розкладу, яка формалізує умови валідності часових слотів та виключає можливість накладання записів.

3. Описано математичні принципи криптографічного захисту даних, реалізовані в системі через алгоритм bcrypt, що гарантує безпеку облікових записів користувачів.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Обґрунтування вибору засобів розробки та технологічного стека

Вибір програмних засобів для реалізації платформи «Healthcare» базувався на необхідності створення високопродуктивної, масштабованої та безпечної системи, здатної працювати в режимі реального часу. Архітектурне рішення проєкту побудовано за принципом SPA (Single Page Application). На відміну від традиційних багатосторінкових сайтів (MPA), де при кожній дії користувача відбувається повне перезавантаження сторінки, SPA завантажує лише один HTML-документ і динамічно оновлює необхідний контент. Такий підхід забезпечує плавність інтерфейсу, що є критичним для комфортної роботи пацієнтів та лікарів із розкладом.

В якості основного інструментарію обрано стек технологій MERN [10] (MongoDB [11, 12], Express.js [13, 14], React [15, 16], Node.js [17]), посилений типізованою мовою TypeScript [18, 19]. Цей вибір дозволяє використовувати єдину мову програмування (JavaScript/TypeScript) на всіх рівнях розробки — від бази даних до інтерфейсу користувача, що спрощує підтримку коду та інтеграцію компонентів.

Нижче наведено детальну характеристику кожного компонента системи.

4.1.1. Серверна платформа (Backend) Серверна частина реалізована на базі середовища виконання Node.js [11, 12].

- Технічні особливості: Node.js побудований на двигуні V8 від Google Chrome і використовує подієво-орієнтовану модель (event-driven architecture) з неблокуючим введенням-виведенням (Non-blocking I/O). Це означає, що сервер не створює новий потік для кожного клієнта, а обробляє тисячі з'єднань в одному потоці, використовуючи механізм Event Loop.
- Переваги для проєкту: Для системи бронювання, яка передбачає велику кількість легких запитів (перевірка вільних слотів, запис на прийом, надсилання сповіщень), така архітектура є найбільш ефективною, оскільки дозволяє мінімізувати затримки та споживання оперативної пам'яті.

Для організації маршрутизації та побудови REST API використано веб-фреймворк Express.js [13]. Це мінімалістичний та гнучкий інструмент, який надає

зручні методи для обробки HTTP-запитів, роботи з параметрами URL та управління сесіями. Важливою особливістю є система middleware (проміжного програмного забезпечення), яка дозволяє легко інтегрувати модулі для логування, аутентифікації та обробки помилок.

4.1.2. Клієнтська частина (Frontend)

Інтерфейс користувача розроблено за допомогою бібліотеки React.js [14, 15].

- Віртуальний DOM: React використовує технологію Virtual DOM — легку копію реального DOM-дерева. При зміні стану застосунку (наприклад, при бронюванні часу) бібліотека спочатку оновлює віртуальну структуру, порівнює її з попередньою версією і вносить лише необхідні точкові зміни в реальний інтерфейс. Це забезпечує високу швидкість рендерингу складних компонентів, таких як інтерактивний календар.
- Мова TypeScript: Для підвищення надійності коду використано TypeScript — надмножину JavaScript зі статичною типізацією. Це дозволяє описувати чіткі інтерфейси для сутностей (наприклад, IUser, IAppointment), що допомагає виявляти потенційні помилки ще на етапі компіляції, а не під час виконання програми.
- UI-бібліотека: Для створення візуального стилю використано Material-UI (MUI) [20]. Ця бібліотека надає набір готових компонентів (кнопки, форми, модальні вікна), які відповідають принципам Material Design від Google. Це гарантує адаптивність дизайну, тобто коректне відображення платформи як на моніторах комп'ютерів, так і на екранах мобільних пристроїв, що забезпечує узгодженість із новітніми підходами до проєктування користувацького досвіду.

4.1.3. Система управління базами даних (СУБД)

Для збереження інформаційних масивів було надано перевагу нереляційній базі даних MongoDB.

- Формат даних: MongoDB зберігає інформацію у форматі BSON (бінарний JSON). Це дозволяє природним чином відображати об'єкти програмного коду в базу даних без складних перетворень, характерних для SQL-систем.
- Гнучкість схеми: медичні дані можуть мати складну ієрархічну структуру (наприклад, вкладені об'єкти розкладу всередині профілю лікаря). MongoDB дозволяє

зберігати такі дані ефективно, без необхідності розбивати їх на безліч зв'язаних таблиць, що спрощує масштабування системи при зростанні навантаження.

Для взаємодії з базою даних у середовищі Node.js використано бібліотеку Mongoose. Вона виконує роль ODM (Object Data Modeling), дозволяючи описувати схеми даних, валідувати їх перед збереженням та будувати зручні запити до БД.

4.1.4. Середовище розробки та інструментальні засоби

Для написання програмного коду, налагодження та тестування системи було обрано інтегроване середовище розробки (IDE) Visual Studio Code (VS Code) від компанії Microsoft (рис. 4.1). Цей вибір обумовлений низкою технічних переваг, які роблять його стандартом де-факто для розробки веб-застосунків на JavaScript/TypeScript:

1. Глибока підтримка TypeScript: оскільки VS Code та TypeScript розробляються однією компанією, редактор забезпечує найкращу на ринку підтримку типізації "з коробки". Функція IntelliSense надає інтелектуальне автодоповнення коду, інформацію про параметри функцій та швидкий доступ до документації API, що значно знижує ймовірність синтаксичних та логічних помилок ще на етапі написання коду.

2. Розширюваність: архітектура редактора дозволяє встановлювати спеціалізовані плагіни, що адаптують середовище під потреби конкретного проєкту. Для розробки платформи «Healthcare» було використано розширення:

   - *ES7+ React/Redux/React-Native snippets* — для швидкого створення компонентів React.
   - *Prettier & ESLint* — для автоматичного форматування коду та дотримання єдиного стилю (Code Style), що є критичним для підтримуваності проєкту.
   - *MongoDB for VS Code* — для перегляду та редагування даних у базі безпосередньо з інтерфейсу редактора.

3. Вбудовані засоби налагодження (Debugging): VS Code має потужний вбудований дебагер для Node.js, який дозволяє встановлювати точки зупинки (breakpoints), переглядати стек викликів та стан змінних у реальному часі. Це суттєво пришвидшує процес пошуку та усунення дефектів у серверній частині застосунку.

4. Інтегрований термінал та Git: наявність вбудованого терміналу дозволяє виконувати команди NPM (менеджера пакетів), запускати локальні сервери та скрипти збірки, не перемикаючись між вікнами. Графічний інтерфейс для роботи з Git спрощує процеси контролю версій: перегляд змін (diff), створення комітів та розв'язання конфліктів злиття (merge conflicts).

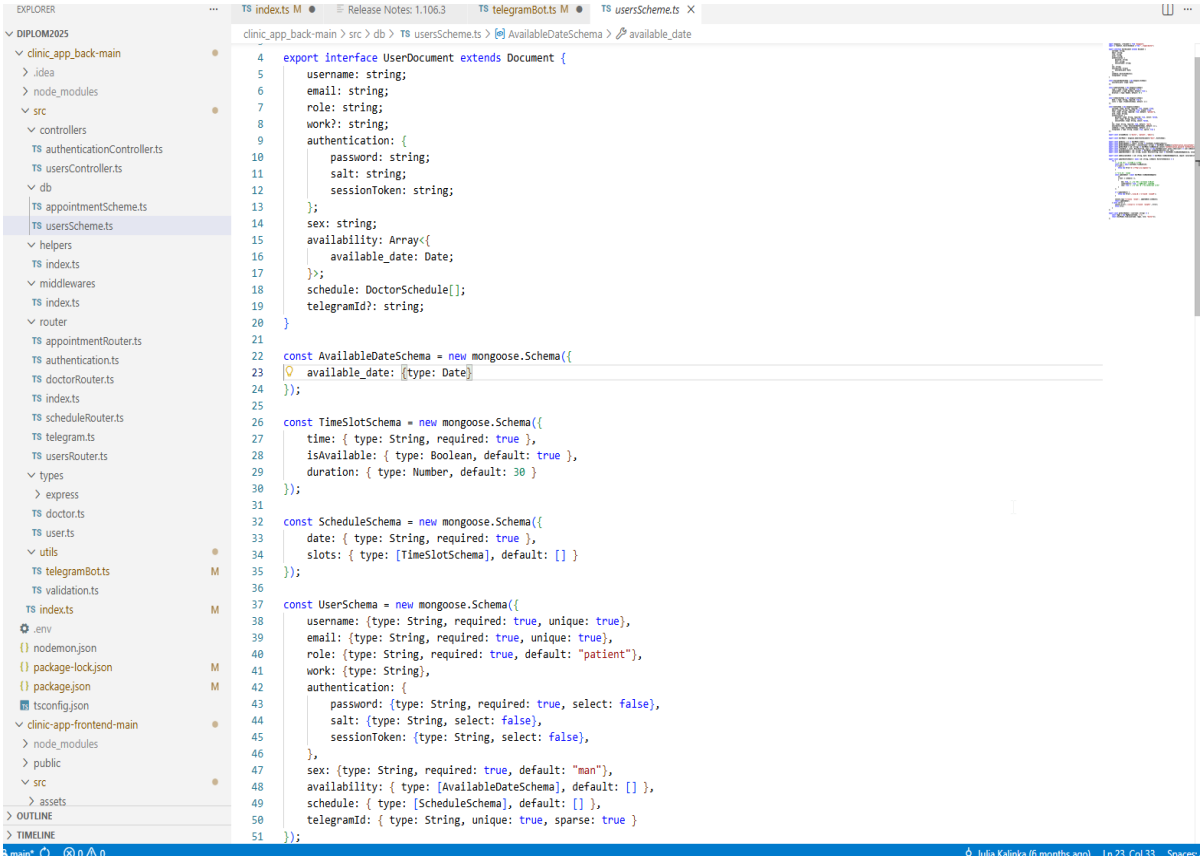


Рисунок 4.1 — Інтерфейс розробленої програми в середовищі Visual Studio Code

4.2 Опис роботи бази даних

4.2.1. Структура бази даних:

Застосунок використовує дві колекції в базі даних MongoDB, спроєктовані для забезпечення цілісності даних та швидкого доступу.

1. Колекція Users (Користувачі): зберігає дані всіх учасників системи. Завдяки поліморфній структурі, всі дані пацієнтів та лікарів знаходяться в одній колекції, але мають різні набори полів.
    - _id: унікальний ідентифікатор.
    - username, email: особисті дані.

- role: ким є користувач (doctor або patient).
- work: (тільки для лікарів): професія лікаря.
- sex: стать людини.
- authentication: об'єкт з даними безпеки (хеш пароля password, сіль salt, токен сесії sessionToken).
- schedule (тільки для лікарів): масив об'єктів, що описує робочі слоти лікаря.
- availability: масив доступних дат для запису.
- telegramId: ідентифікатор для надсилання сповіщень у Telegram.

2. Колекція Appointments (Записи): містить дані записів і інформацію про заплановані прийоми до лікаря.
   - userId: посилання на пацієнта.
   - doctorId: посилання на лікаря.
   - date, time: часові параметри візиту.
   - status: стан візиту (активний, скасований, виконаний).

4.2.2. Логічна структура проєкту

Проєкт розділено на клієнтську та серверну частини з чітким розмежуванням відповідальності.

Frontend (Клієнт):

- pages/: компоненти сторінок (Login, Register, Dashboard, Appointments).
- components/: перевикористовувані елементи інтерфейсу (картки лікарів DoctorCard, поля вводу ControlledTextField).
- services/: модулі для виконання HTTP-запитів до API через бібліотеку Axios [21].
- utils/: допоміжні функції, включаючи валідацію форм (validation.ts).

Backend (Сервер):

- controllers/: обробка вхідних запитів (наприклад, authenticationController.ts для реєстрації, usersController.ts для управління профілями).
- models/: схеми даних Mongoose (usersScheme.ts, appointmentScheme.ts).
- routes/: визначення маршрутів API.

- middleware/: перевірка токенів доступу та прав користувачів.

4.3 Приклад роботи системи

Для верифікації функціональності розробленої платформи «Healthcare» та підтвердження її відповідності технічному завданню було проведено комплексне тестування основних сценаріїв використання. Нижче наведено детальний опис логіки взаємодії користувачів із системою, який демонструє реалізацію ключових бізнес-процесів платформи.

Етап 1: авторизація та реєстрація. Взаємодія з програмним комплексом розпочинається зі сторінки автентифікації (рис. 4.2). Цей етап є критично важливим з точки зору безпеки, оскільки система повинна чітко ідентифікувати користувача для надання йому відповідних прав доступу (роль «Пацієнт» або «Лікар»).

Інтерфейс форми входу спроєктовано в стилі мінімалізму, щоб не відволікати користувача від основної дії. Користувач вводить свою електронну пошту та пароль у відповідні поля. На цьому етапі спрацьовують механізми валідації на боці клієнта (Client-side validation), які перевіряють, чи не є поля порожніми. Після натискання кнопки "Увійти" дані відправляються на сервер, де відбувається звірка хешу введеного пароля з хешем, що зберігається в базі даних MongoDB.

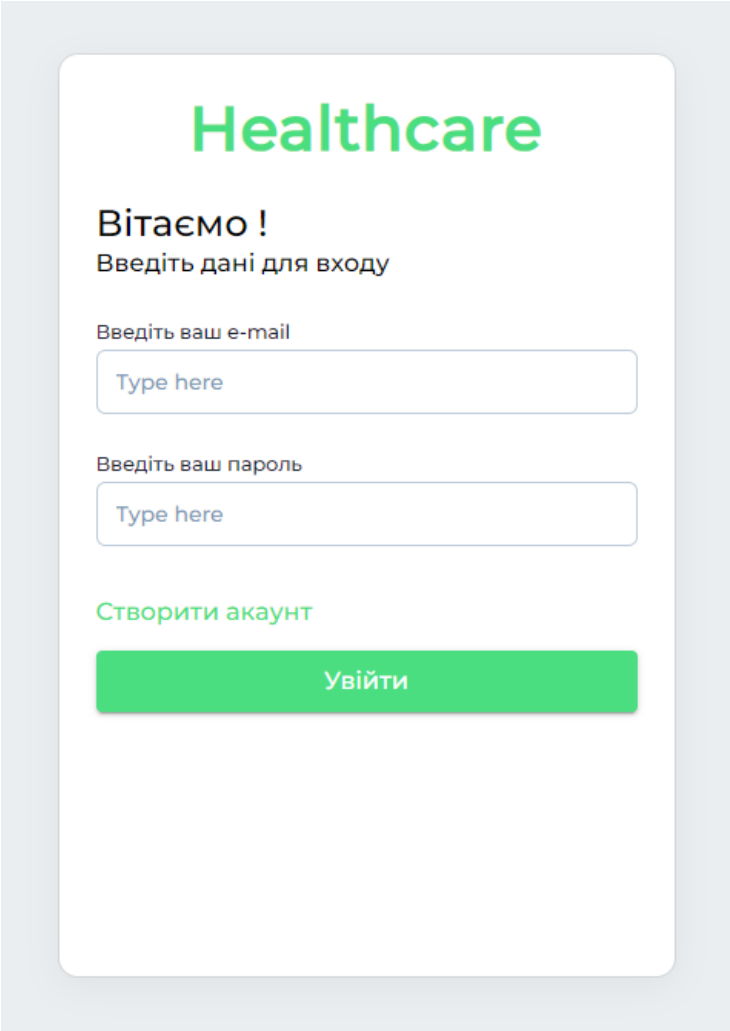


Рисунок 4.2 — Форма входу в акаунт

У разі, якщо користувач ще не має облікового запису, система пропонує йому пройти процедуру реєстрації (рис. 4.3). Перехід до форми реєстрації здійснюється за допомогою інтуїтивно зрозумілого посилання "Створити акаунт". Форма реєстрації є більш розширеною і збирає необхідний мінімум даних для створення профілю: прізвище, ім'я, контактний email та пароль. Особливу увагу при розробці було приділено UX/UI аспектам вибору ролі користувача. Через випадаючий список користувач визначає свій статус у системі: «Пацієнт» або «Лікар». Це архітектурне рішення дозволяє використовувати єдину точку входу для всіх типів користувачів, але розмежовувати функціонал відразу після створення акаунту. Для ролі «Лікар» інтерфейс динамічно адаптується, додаючи поле вибору медичної спеціалізації, що є необхідним для подальшої індексації лікаря в пошуковій системі платформи. Варто зазначити, що поля вводу оснащені "живою" валідацією: наприклад, система

перевіряє формат електронної пошти за допомогою регулярних виразів (RegExp) та контролює складність пароля (мінімальна довжина, наявність цифр), що підвищує загальний рівень безпеки системи ще на етапі введення даних.

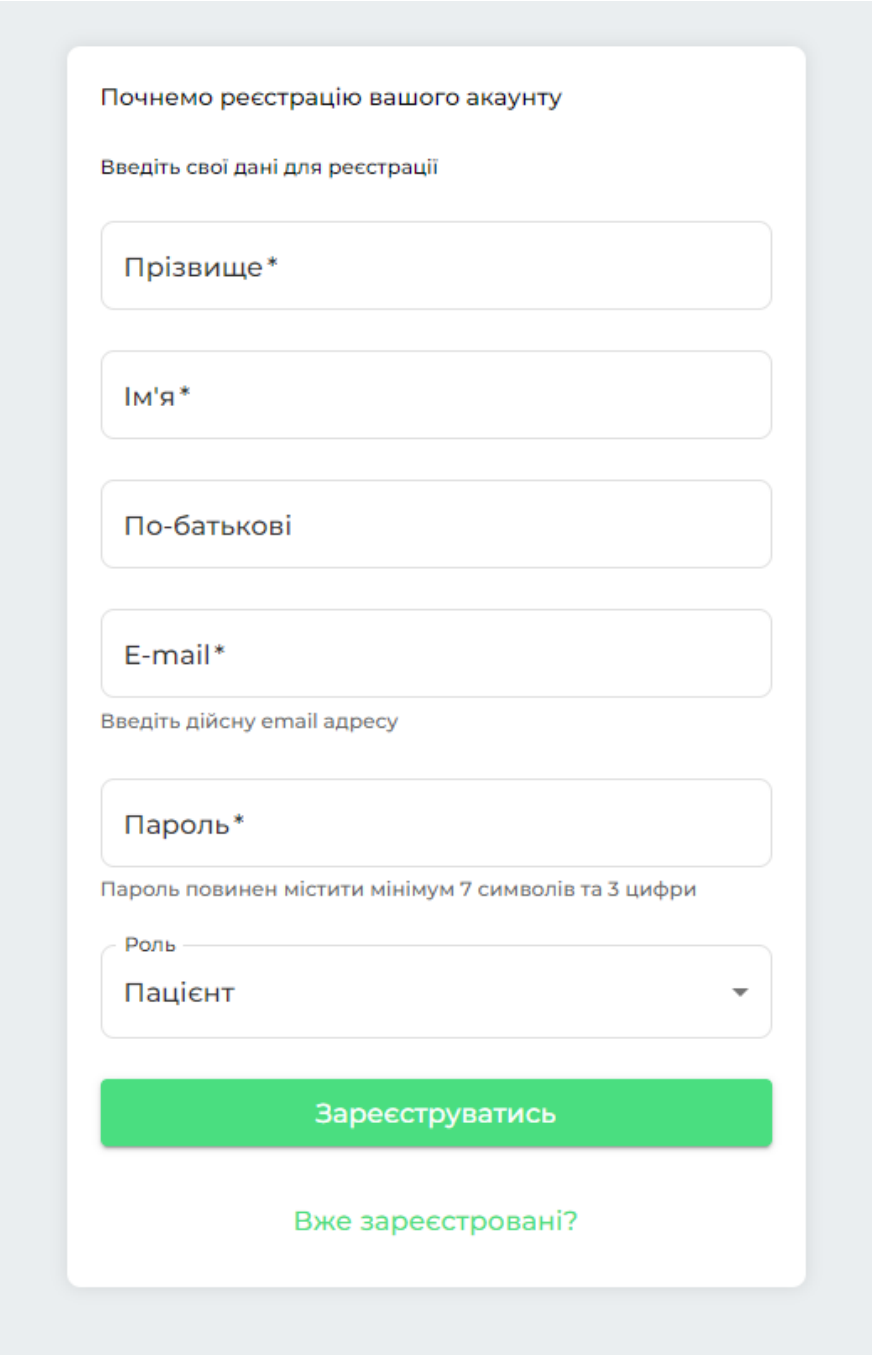


Рисунок 4.3 — Форма реєстрації

Етап 2: пошук лікаря (Функціонал пацієнта). Після успішної авторизації пацієнт потрапляє на головну сторінку платформи (рис. 4.4). Головною метою користувача на цьому етапі є швидкий пошук необхідного медичного спеціаліста. Для реалізації цієї

потреби розроблено зручний пошуковий модуль, розташований у центральній частині екрана.

Модуль пошуку підтримує два основні сценарії:

1. Пошук за спеціалізацією: користувач може обрати напрямок медицини (наприклад, "Стоматологія", "Кардіологія") зі списку або почати вводити назву, і система запропонує релевантні варіанти.
2. Пошук за іменем: якщо пацієнт шукає конкретного лікаря, він може ввести його прізвище.

Пошуковий алгоритм реалізовано з використанням оптимізованих запитів до бази даних, що забезпечує миттєвий відгук інтерфейсу навіть при великій кількості зареєстрованих лікарів.

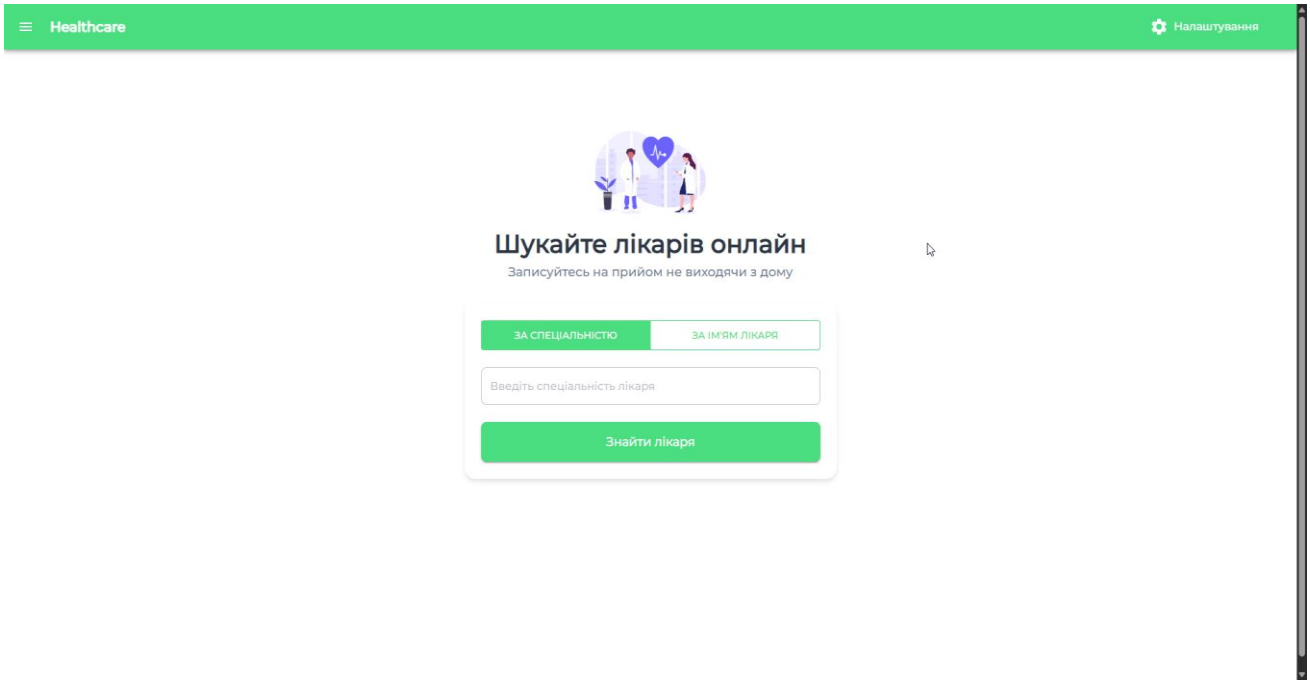


Рисунок 4.4 — Головне вікно з пошуком лікаря

Результати пошуку візуалізуються у вигляді списку інтерактивних карток лікарів (рис 4.5). Кожна картка містить ключову інформацію: фотографію, ПІБ, спеціалізацію та статус доступності. При виборі конкретного лікаря відкривається інтерфейс бронювання. Центральним елементом тут є інтерактивний календар та панель вибору часових слотів. Система автоматично підвантажує графік лікаря на поточний тиждень. Дні, в які лікар не приймає, позначаються неактивними, що унеможливлює помилковий запис. Після вибору дати користувачу демонструються лише вільні

години прийому. Алгоритм на сервері фільтрує слоти, виключаючи ті, що вже були заброньовані іншими пацієнтами або час яких вже минув. Це гарантує актуальність даних у реальному часі та запобігає виникненню колізій (ситуацій, коли два пацієнти намагаються записатися на один час).

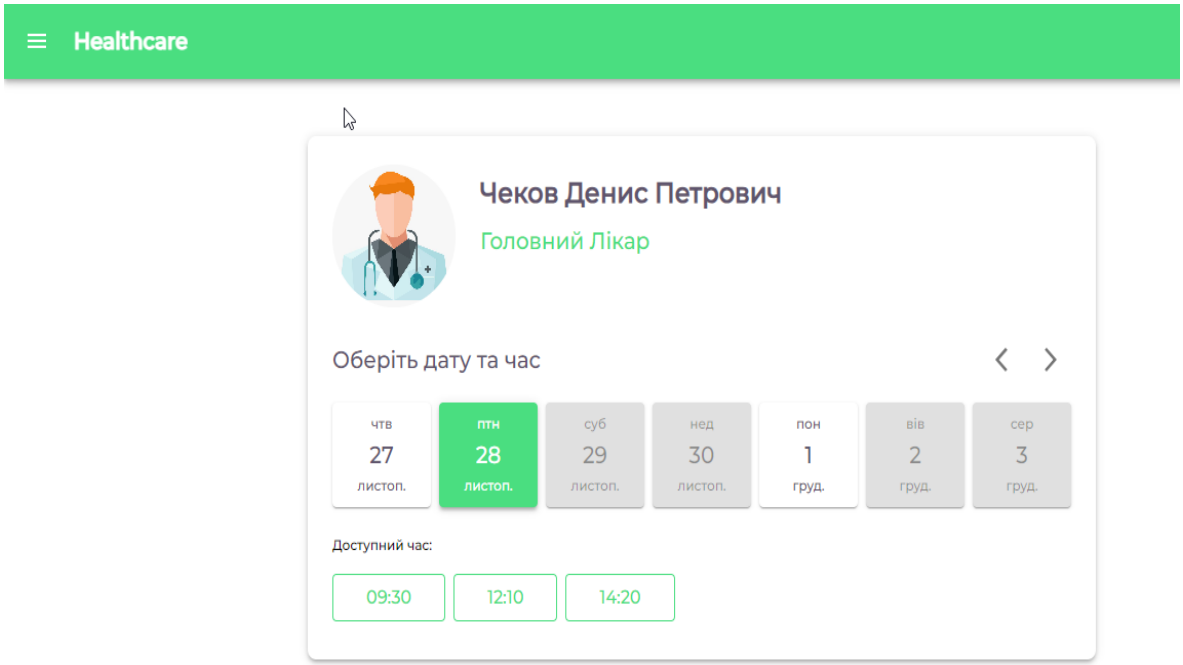


Рисунок 4.5 — Графік прийому у лікаря

Етап 3: управління записами. Для забезпечення контролю над своїм лікуванням пацієнт має доступ до розділу «Мої записи» в особистому кабінеті (рис. 4.6). Цей розділ виконує роль органайзера, де в хронологічному порядку відображаються всі заплановані візити. Інтерфейс картки запису спроєктовано таким чином, щоб надати користувачу максимум інформації: дату, час, ім'я лікаря та поточний статус візиту («Активний», «Виконаний», «Скасований»). Кольорове кодування статусів (зелений для активних, сірий для виконаних, червоний для скасованих) дозволяє швидко оцінити стан своїх записів. Важливою функцією є можливість скасування візиту. Натискання кнопки «Скасувати» ініціює запит до API, який не лише змінює статус запису в базі даних пацієнта, але й миттєво звільняє відповідний часовий слот у графіку лікаря, роблячи його доступним для інших користувачів. Це демонструє високий рівень зв'язності даних у системі.

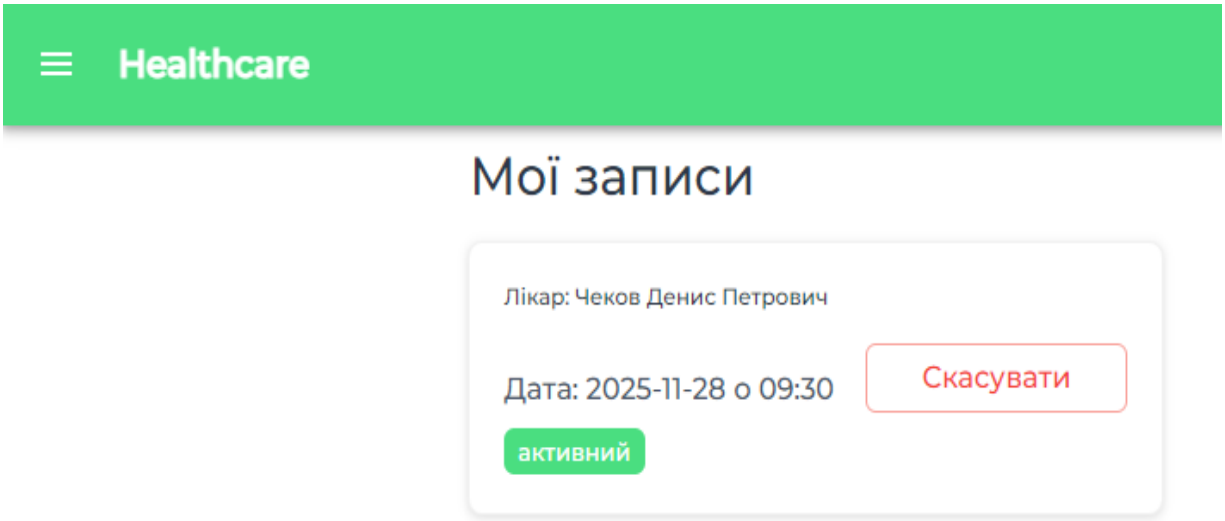


Рисунок 4.6 — Історія записів пацієнта

Етап 4: управління графіком (Функціонал лікаря). Лікар має доступ до розширеного функціоналу управління розкладом (рис. 4.7). На сторінці налаштувань графіку він бачить свій календар по днях. Лікар може видаляти існуючі слоти або додавати нові.

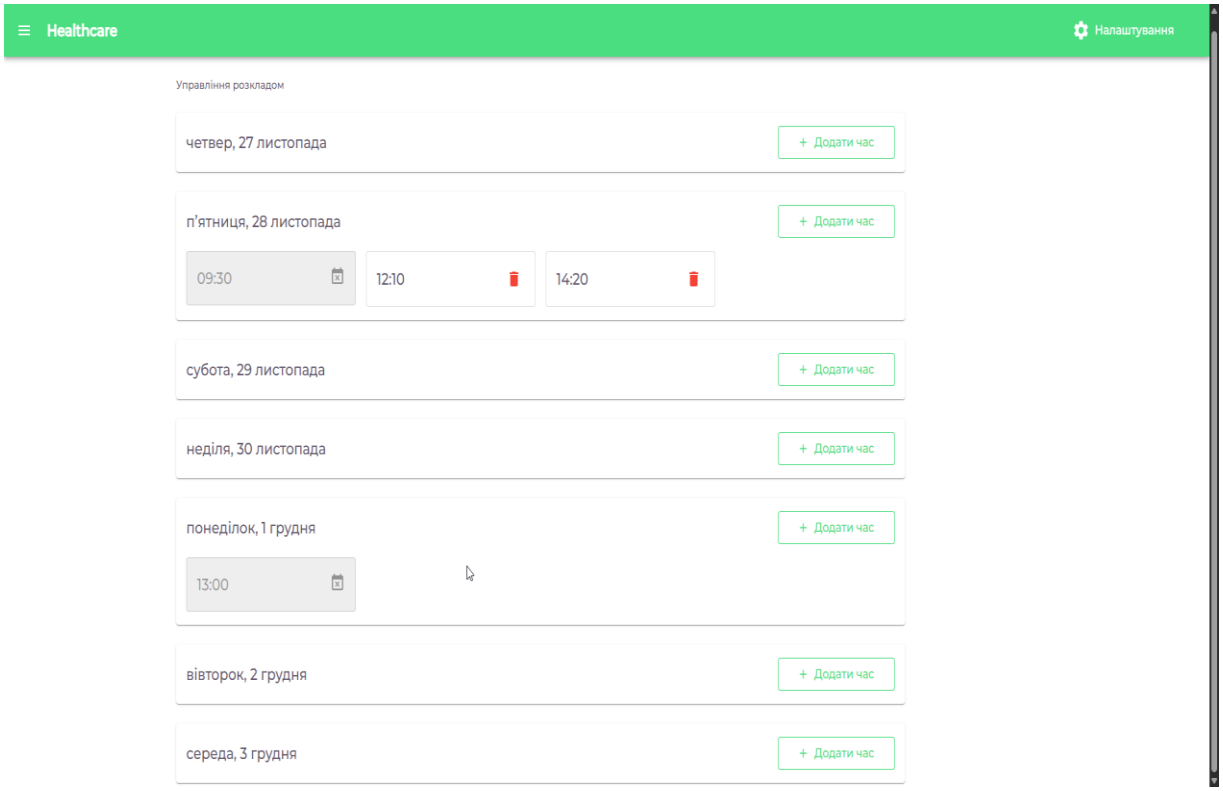


Рисунок 4.7 — Сторінка керування записами у лікаря

Для забезпечення зручності управління графіком лікаря було реалізовано модальне вікно (діалог), яке дозволяє додавати нові часові слоти без перезавантаження сторінки. Цей компонент розроблено з використанням бібліотеки Material-UI, що гарантує єдність стилю та адаптивність інтерфейсу (рис. 4.8).

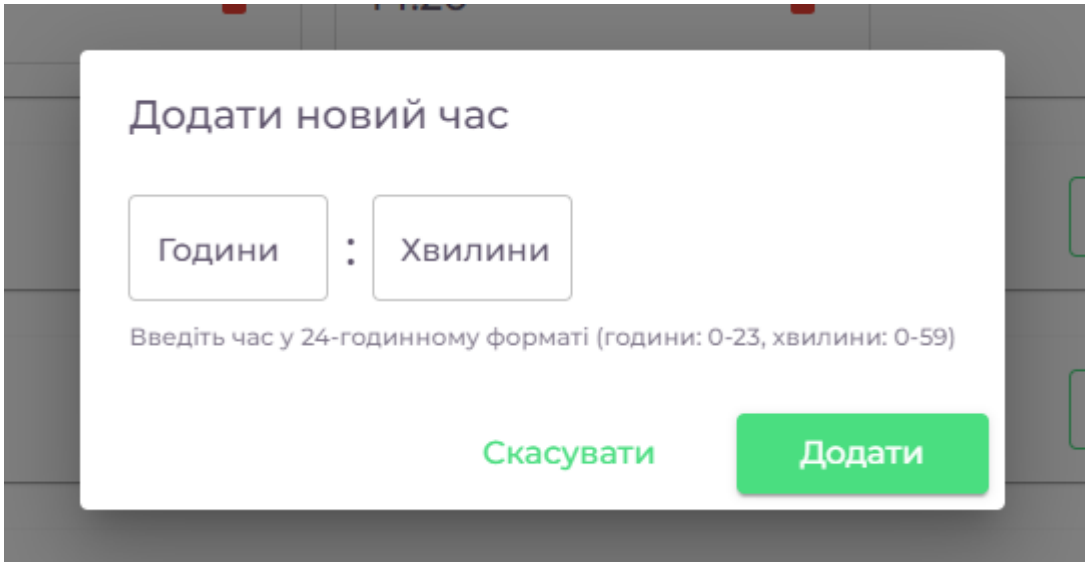


Рисунок 4.8 - Спливаюче вікно для створення запису

У наведеному нижче фрагменті коду (рис. 4.9) показано структуру компонента Dialog. Він містить поля введення для годин та хвилин, які мають вбудовану валідацію (наприклад, години обмежені діапазоном 0-23, хвилини — 0-59). Стан полів керується за допомогою хуків React useState, а при натисканні кнопки "Додати" викликається функція-обробник, яка формує новий об'єкт слоту та відправляє його на сервер.

```
const Schedule: React.FC = () => {
    const navigate = useNavigate();
    const [schedule, setSchedule] = useState<DoctorSchedule[]>([]);
    const [loading, setLoading] = useState(true);
    const [openDialog, setOpenDialog] = useState(false);
    const [selectedDate, setSelectedDate] = useState('');
    const [hours, setHours] = useState('');
    const [minutes, setMinutes] = useState('');
    const [userId, setUserId] = useState<string>('');

    const fetchSchedule = async () => {
        try {
            setLoading(true);
            const userData = await getMyUserData();
            if (userData.role !== 'doctor') {
                navigate('/main');
                return;
            }
            setUserId(userData._id);
            const response = await getDoctorSchedule(userData._id);
            console.log('Отримано відповідь від сервера:', response);

            if (response.success && Array.isArray(response.schedule)) {
                console.log('Встановлюємо новий розклад:', response.schedule);
                setSchedule(response.schedule);
            } else {
                console.error('Неправильний формат розкладу:', response);
                throw new Error(response.message || 'Неправильний формат розкладу');
            }
        } catch (error: any) {
            console.error('Помилка при отриманні розкладу:', error);
            toast.error(error.message || 'Помилка при отриманні розкладу');
        } finally {
            setLoading(false);
        }
    };

    useEffect(() => {
        fetchSchedule();
    }, []);

    const handleAddTimeClick = (date: string) => {
        setSelectedDate(date);
        setHours('');
        setMinutes('');
        setOpenDialog(true);
    };
```

Рисунок 4.9 – Приклад компонента Dialog

На сторінці «Мої записи» лікар бачить перелік пацієнтів, які записалися на прийом, із зазначенням дати та часу (рис. 4.10). Він може змінювати статус запису (наприклад, підтвердити виконання або скасувати у разі форс-мажору).

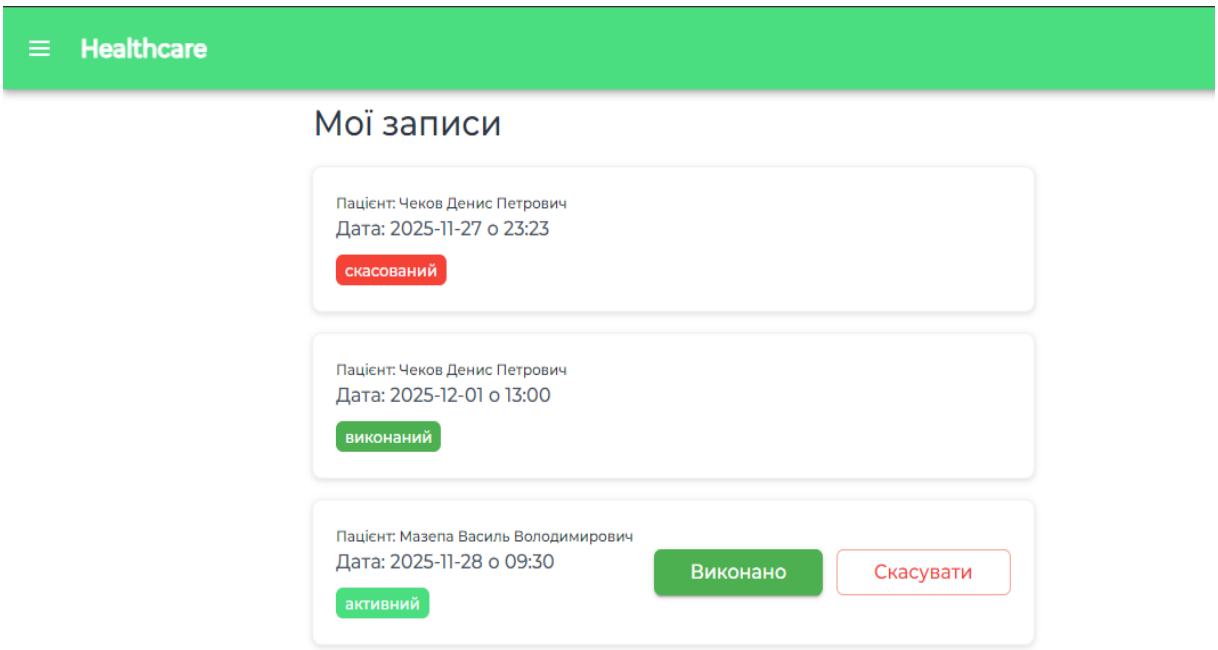


Рисунок 4.10 — Сторінка створених записів у лікаря

Етап 5: налаштування профілю та сповіщення. Всі користувачі мають доступ до панелі налаштувань через бокове меню, де також доступна опція виходу з системи (рис. 4.11).

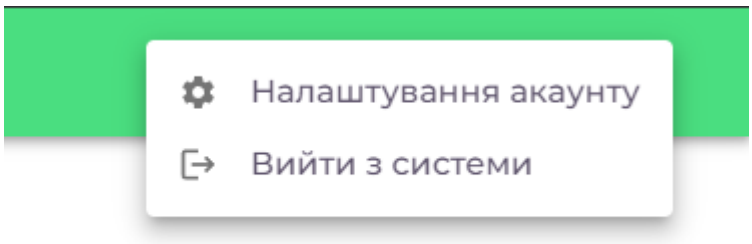


Рисунок 4.11 — Опції меню “Налаштування”

На сторінці профілю користувач може редагувати особисту інформацію. Ключовою особливістю системи «Healthcare» є інтеграція з Telegram. На цій сторінці розміщено інструкцію з підключення бота для отримання автоматичних нагадувань (рис. 4.12).

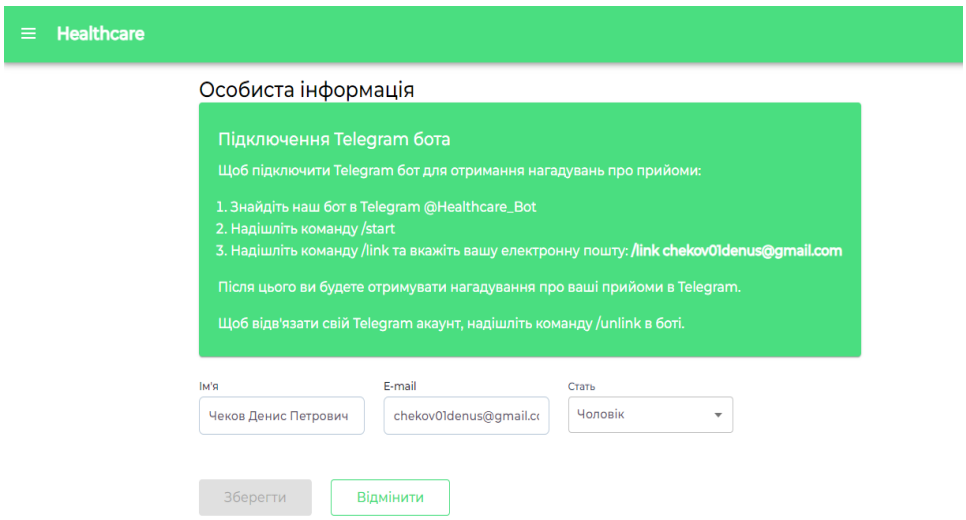


Рисунок 4.12 — Сторінка для налаштування аккаунту та сповіщень

Ключовою перевагою системи є інтеграція з месенджером Telegram, яка реалізована на базі бібліотеки Telegraf у середовищі Node.js (рис. 4.13). Це дозволяє забезпечити надійний канал доставки сповіщень користувачам. Замість складної процедури авторизації через сторонні сервіси, було обрано простий та надійний механізм «рукостискання» через команду. Інтерфейс надає користувачу чітку покрокову інструкцію: знайти бота, запустити його та ввести унікальну команду прив'язки, яка містить ідентифікатор користувача (в даному випадку — email). Це забезпечує однозначну відповідність між акаунтом у веб-системі та Telegram-акаунтом.

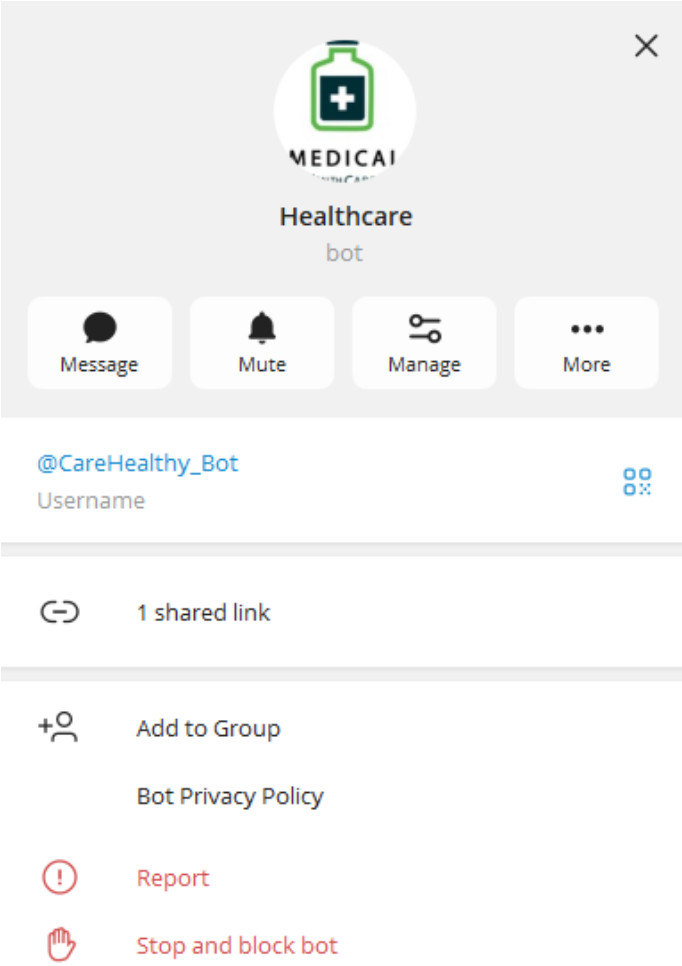
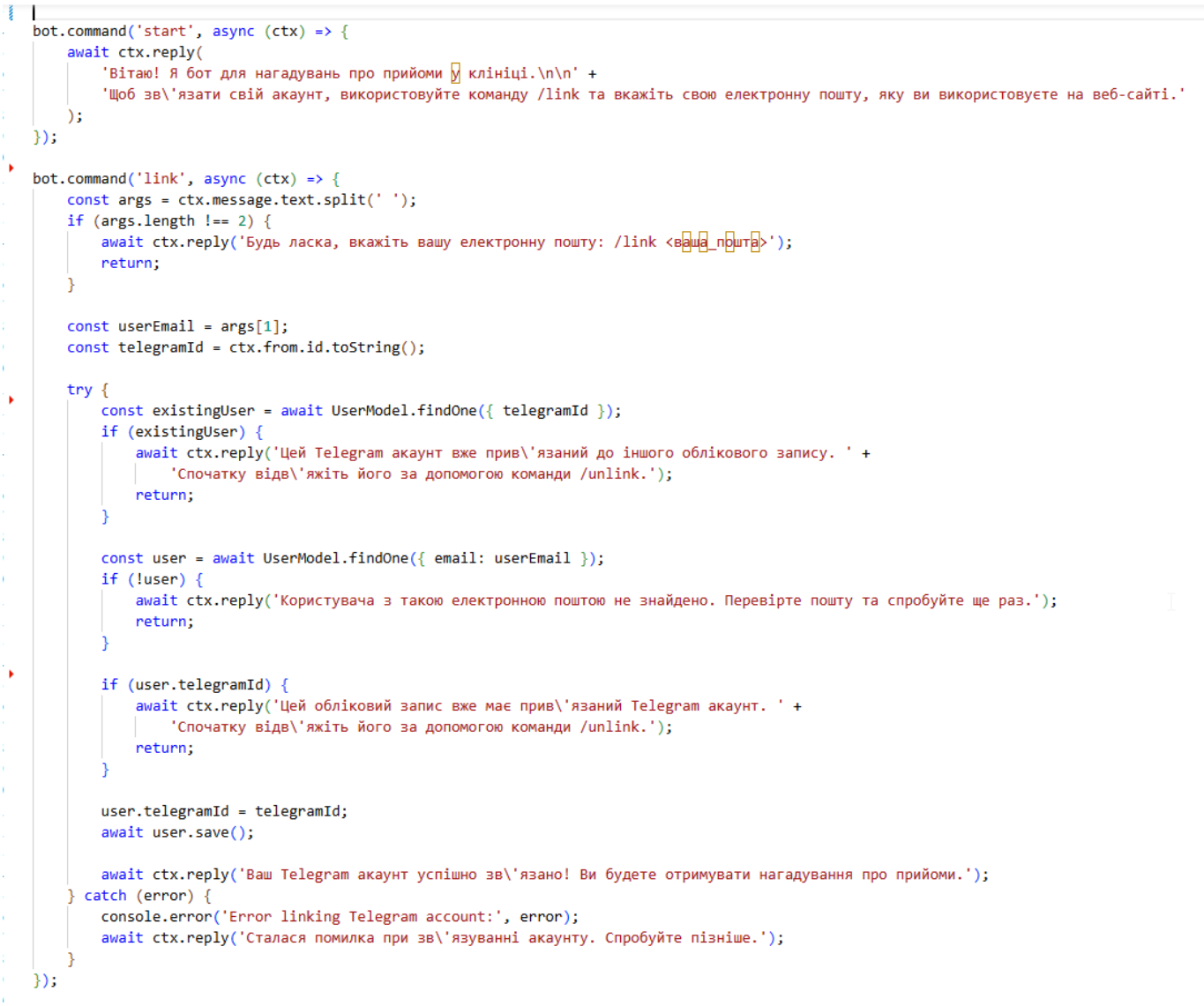


Рисунок 4.13 — Зовнішній вигляд бота у Telegram

Нижче наведено фрагмент коду, що демонструє обробку основних команд бота (рис. 4.14). Команда /start ініціалізує діалог і надає користувачу інструкції. Команда /link приймає email користувача як аргумент, перевіряє його наявність у базі даних MongoDB (через UserModel) та, у разі успішної валідації, зберігає унікальний telegramId у профілі користувача. Це створює зв'язок між акаунтом на веб-платформі та чатом у месенджері.



```
bot.command('start', async (ctx) => {
    await ctx.reply(
        'Вітаю! Я бот для нагадувань про прийоми у клініці.\n\n' +
        'Щоб зв\'язати свій акаунт, використовуйте команду /link та вкажіть свою електронну пошту, яку ви використовуєте на веб-сайті.'
    );
});

bot.command('link', async (ctx) => {
    const args = ctx.message.text.split(' ');
    if (args.length !== 2) {
        await ctx.reply('Будь ласка, вкажіть вашу електронну пошту: /link <ваша_пошта>');
        return;
    }

    const userEmail = args[1];
    const telegramId = ctx.from.id.toString();

    try {
        const existingUser = await UserModel.findOne({ telegramId });
        if (existingUser) {
            await ctx.reply('Цей Telegram акаунт вже прив\'язаний до іншого облікового запису. ' +
                'Спочатку відв\'яжіть його за допомогою команди /unlink.');
            return;
        }

        const user = await UserModel.findOne({ email: userEmail });
        if (!user) {
            await ctx.reply('Користувача з такою електронною поштою не знайдено. Перевірте пошту та спробуйте ще раз.');
            return;
        }

        if (user.telegramId) {
            await ctx.reply('Цей обліковий запис вже має прив\'язаний Telegram акаунт. ' +
                'Спочатку відв\'яжіть його за допомогою команди /unlink.');
            return;
        }

        user.telegramId = telegramId;
        await user.save();

        await ctx.reply('Ваш Telegram акаунт успішно зв\'язано! Ви будете отримувати нагадування про прийоми.');
    } catch (error) {
        console.error('Error linking Telegram account:', error);
        await ctx.reply('Сталася помилка при зв\'язуванні акаунту. Спробуйте пізніше.');
    }
});
```

Рисунок 4.14 — Приклад реалізації чат-бота

Для активації сповіщень користувач знаходить бота в Telegram, запускає його за допомогою кнопки або команди /start. Після цього користувачу необхідно прив'язати акаунт платформи до бота за допомогою команди /link <його email аккаунту>.

Після успішної прив'язки бот автоматично надсилає сповіщення про важливі події: успішне створення запису, нагадування про візит (за 24 та 6 годин), а також повідомлення про скасування прийому лікарем або пацієнтом (рис. 4.15). Така інтеграція значно підвищує рівень сервісу та зменшує навантаження на адміністраторів клініки, оскільки відпадає потреба в ручному нагадуванні про візити.

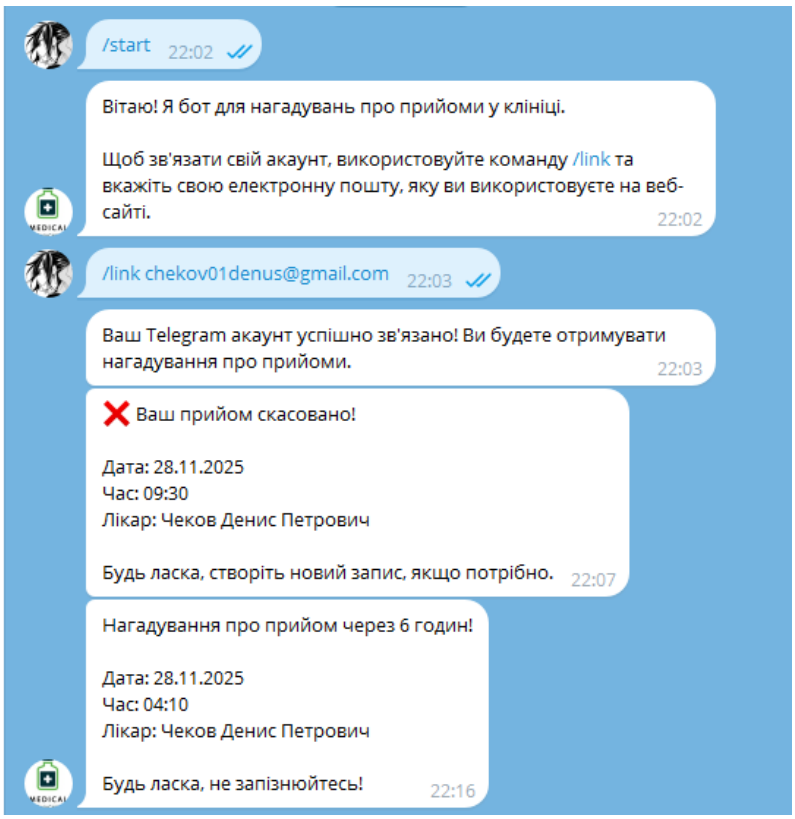


Рисунок 4.15 — Види сповіщень у чаті з Telegram ботом

4.4 Висновки до розділу

В даному розділі описано практичну реалізацію інформаційної системи «Healthcare». Обґрунтовано доцільність використання стека MERN та TypeScript для забезпечення надійності та швидкодії додатку. Розроблена структура бази даних MongoDB дозволяє ефективно зберігати та обробляти інформацію про користувачів і записи. Програмний продукт має чіткий поділ на клієнтську та серверну частини, реалізований з дотриманням принципів модульності. Аналіз контрольного прикладу підтвердив коректну роботу всіх ключових функцій системи: реєстрації, пошуку лікарів, динамічного управління графіком та механізму сповіщень через Telegram. Створений інтерфейс є інтуїтивно зрозумілим, що мінімізує час на навчання користувачів.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту та аналіз конкурентних переваг

Сутність ідеї проєкту Стартап-проєкт «Healthcare» являє собою спеціалізовану веб-платформу, що працює за моделлю SaaS (Software as a Service). Основна ідея полягає у створенні доступного, легкого у налаштуванні та ефективного інструменту для автоматизації адміністративних процесів у малих та середніх приватних медичних закладах (стоматологічні кабінети, косметологічні клініки, приватні практики сімейних лікарів тощо).

Основні напрями використання:

1. Автоматизація запису: заміна паперових журналів та Excel-таблиць на хмарну систему бронювання, доступну 24/7.
2. Управління пацієнтопотоком: візуалізація завантаженості лікарів, уникнення накладок у розкладі.
3. Комунікація з клієнтом: автоматичне інформування пацієнтів про візити через месенджер Telegram, що дозволяє відмовитися від ручного обдзвону адміністраторами.

Відмінність від конкурентів На ринку України існують потужні гравці (Helsi, Health24, Doc.ua), проте система «Healthcare» має чітке позиціювання, що відрізняє її від аналогів:

Таблиця 5.1 — Порівняльний аналіз конкурентних переваг стартап-проєкту (порівняння з Helsi/Health24).

Характеристика	**Конкуренти(Helsi, Health24)**	**Стартап «Healthcare»**
Цільова аудиторія	Великі клініки, держсектор	Приватні кабінети, малий бізнес
Складність впровадження	Висока (навчання персоналу, налаштування серверів)	Низька (реєстрація та початок роботи за 5 хв)
Комунікація	Переважно SMS (платно) або мобільний додаток	Telegram-бот (безкоштовно, зручно)

Продовження таблиці 5.1

Характеристика	Конкуренти (Helsi, Health24)	Стартап «Healthcare»
Вартість	Висока абонплата або комісія за пацієнта	Доступна фіксована підписка
Інтерфейс	Перевантажений функціями звітності	Мінімалістичний, фокус на розкладі

Унікальна торгова пропозиція (УТП): «Healthcare» — це найпростіший спосіб для лікаря організувати свій час та позбутися неявок пацієнтів завдяки інтеграції з Telegram, без переплат за зайвий функціонал.

5.2 Аналіз технологічних можливостей реалізації

Для успішного запуску стартапу необхідно оцінити технічну здійсненність проєкту. Проведений аудит показує, що обрані технології дозволяють реалізувати продукт з мінімальними капіталовкладеннями.

1. Доступність технологій:
   - Проєкт базується на стеку MERN (MongoDB, Express, React, Node.js), який є відкритим (Open Source) і не потребує купівлі дорогих ліцензій.
   - Використання Telegram API є безкоштовним, що суттєво знижує собівартість комунікації з клієнтами порівняно з SMS-шлюзами.
2. Ресурсне забезпечення:
   - Для розгортання MVP (Minimal Viable Product) достатньо оренди хмарного сервера (VPS) початкового рівня, вартість якого не перевищує $10-20 на місяць.
   - База даних MongoDB може масштабуватися горизонтально, що дозволяє системі працювати стабільно навіть при зростанні кількості клінік-клієнтів.
3. Кадровий потенціал:
   - Розробка та підтримка системи може здійснюватися невеликою командою (1-2 Full-stack розробники), що робить проєкт фінансово стійким на старті.

5.3 Аналіз ринкових можливостей

Характеристика ринку Ринок приватної медицини в Україні демонструє стійке зростання. За оцінками експертів, частка приватних медичних закладів складає близько 25-30%, при цьому сегмент малих кабінетів є найменш цифровізованим. Багато лікарів-ФОП досі використовують блокноти, що створює значний потенціал для впровадження простих CRM-систем.

Таблиця 5.2 — Аналіз сильних та слабких сторін стартап-проєкту (Внутрішні фактори)

Сильні сторони	Слабкі сторони
Сучасний технологічний стек: Використання MERN (MongoDB, Express, React, Node.js) забезпечує швидкодію та легку масштабованість системи.	Відсутність бренду: На етапі запуску проєкт не має репутації та клієнтської бази, що ускладнює первинні продажі.
Мобільність: Глибока інтеграція з Telegram-ботом дозволяє пацієнтам та лікарям взаємодіяти з системою без встановлення «важких» мобільних додатків.	Обмежений функціонал: У порівнянні з комплексними МІС (наприклад, Health24), у MVP-версії відсутні модулі складського обліку та фінансової звітності.
Економічна ефективність: Використання Open Source технологій мінімізує собівартість розробки та підтримки.	Технологічна залежність: Робота підсистеми сповіщень критично залежить від стабільності роботи API месенджера Telegram.
UX/UI: Інтуїтивно зрозумілий інтерфейс знижує поріг входження для персоналу та не потребує тривалого навчання.	

Таблиця 5.3 — Аналіз можливостей та загроз зовнішнього середовища

Можливості	Загрози
Зростання ринку: Збільшення частки приватних медичних кабінетів та індивідуальних практик, які потребують автоматизації.	Зміна законодавства: Посилення вимог регулятора до зберігання медичних даних (КСЗІ) може вимагати додаткових витрат на сертифікацію.
Розширення ніші: Адаптація системи для суміжних сфер послуг (ветеринарні клініки, косметологічні салони, психологічні студії).	Політика платформ: Ризик зміни умов використання API Telegram або його можливе блокування регулятором.
Функціональний розвиток: Потенціал для додавання модуля телемедицини та відеоконсультацій.	Конкурентний тиск: Великі гравці ринку можуть застосувати демпінг цін, щоб витіснити нові стартапи.
Фінансові інструменти: Можливість інтеграції з платіжними шлюзами (LiqPay, WayForPay) для впровадження передоплати візитів.	Кібербезпека: Зростання активності хакерських атак на медичні сервіси та ризик витоку персональних даних.

5.4 Розроблення стратегії ринкового впровадження

Для виведення продукту на ринок пропонується використати стратегію диференційованого маркетингу, фокусуючись на потребах малого бізнесу.

1. Товарна політика: пропонується веб-платформа з адаптивним дизайном, що не потребує встановлення. Основний акцент робиться на надійності роботи Telegram-бота, який виступає "персональним асистентом" для пацієнта.

2. Цінова політика: обрано модель монетизації за підпискою (Subscription Model).

- Тариф «Start»: безкоштовно (до 50 записів на місяць) — для ознайомлення та залучення користувачів.

- Тариф «Pro»: фіксована плата (наприклад, 400 грн/місяць за лікаря) — повний безліміт та пріоритетна підтримка. Це значно дешевше, ніж комісійна модель агрегаторів.

3. Політика просування:

- Таргетована реклама в соціальних мережах (Facebook, Instagram) з націленням на лікарів приватної практики.
- Контент-маркетинг: публікація статей про ефективність тайм-менеджменту в медицині.
- Партнерські програми з постачальниками медичного обладнання.

5.5 Висновки до розділу

У п'ятому розділі розроблено концепцію стартап-проєкту «Healthcare». Проведений аналіз показав, що проєкт має високий потенціал для комерціалізації в ніші малого медичного бізнесу завдяки поєднанню низької вартості, простоти впровадження та ефективної комунікації через месенджери. Технологічна реалізація не потребує значних капіталовкладень, а обрана бізнес-модель підписки забезпечує прогнозований дохід. Реалізація проєкту дозволить клінікам зменшити операційні витрати та підвищити лояльність пацієнтів.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальне науково-практичне завдання підвищення ефективності управління пацієнтопотоком у приватних медичних закладах шляхом розробки веб-орієнтованої платформи «Healthcare».

Основні наукові та практичні результати роботи полягають у наступному:

1. Проведено аналіз стану проблемної області. Встановлено, що існуючі на ринку рішення (Helsi, Doc.ua, Health24) мають суттєві обмеження для малого та середнього приватного бізнесу: високу вартість впровадження, надлишковий функціонал державної звітності або орієнтацію виключно на лідогенерацію. Виявлено потребу у створенні спеціалізованої SaaS-платформи, яка фокусується на гнучкості графіків та автоматизації комунікації через месенджери.
2. Здійснено системний аналіз та моделювання. Побудовано дерево цілей та діаграми потоків даних (DFD), що дозволило формалізувати процеси взаємодії між лікарем, пацієнтом та системою. Розроблено інформаційну модель, яка включає гнучку структуру бази даних для зберігання різнорідної медичної інформації.
3. Обґрунтовано математичне забезпечення. Використано метод аналізу ієрархій (MAI) Т. Сааті для прийняття багатокритеріального рішення щодо вибору архітектури системи. Це дозволило математично підтвердити доцільність використання архітектури SPA (Single Page Application) порівняно з традиційними багатосторінковими сайтами за критеріями швидкодії, масштабованості та зручності користувача.
4. Виконано програмну реалізацію системи «Healthcare».
    - Розроблено клієнт-серверний додаток на базі стека технологій MERN (MongoDB, Express.js, React, Node.js) із використанням мови TypeScript, що забезпечило високу надійність та продуктивність коду.
    - Реалізовано модуль динамічного управління розкладом, який дозволяє лікарям самостійно налаштовувати слоти прийому "на льоту".

- Впроваджено інтелектуальну систему сповіщень на базі Telegram Bot API, яка автоматизує процес нагадування пацієнтам про візити, що дозволяє знизити рівень неявок (no-show rate).
- Забезпечено захист персональних даних користувачів шляхом хешування паролів (bcrypt) та використання токенів доступу (JWT) з урахуванням сучасних рекомендацій безпеки веб-застосунків (OWASP).

5. Розроблено стартап-проєкт. Проведено маркетинговий аналіз та розроблено бізнес-модель комерціалізації платформи за моделлю підписки (SaaS). Визначено цільову аудиторію (приватні кабінети, стоматології, косметологічні клініки) та стратегію виходу на ринок. Доведено економічну ефективність впровадження системи для клінік за рахунок економії часу адміністраторів та зменшення простоїв лікарів.

Практична цінність роботи полягає у створенні повністю функціонального програмного продукту, готового до впровадження у діяльність приватних медичних закладів. Система «Healthcare» забезпечує автоматизацію запису на прийом у режимі 24/7, покращує комунікацію з пацієнтами та надає власникам клінік інструмент для прозорого контролю завантаженості персоналу.

Перспективи подальшого розвитку системи вбачаються у розширенні функціоналу модулем телемедицини, інтеграції з платіжними шлюзами для передоплати візитів та впровадженні аналітичних інструментів на базі штучного інтелекту для прогнозування попиту на медичні послуги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Global strategy on digital health 2020–2025. — Geneva : World Health Organization, 2021. — 116 p.
2. Про затвердження Змін до деяких нормативно-правових актів Міністерства охорони здоров'я України (щодо функціонування електронної системи охорони здоров’я) : Наказ МОЗ України № 1971. — URL: https://moz.gov.ua (дата звернення: 15.05.2025).
3. Закон України «Про захист персональних даних» : за станом на 14 січ. 2024 р. / Верховна Рада України. — URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 15.05.2025).
4. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection — Information security management systems — Geneva : International Organization for Standardization, 2022. — 12 p.
5. Fowler M. UML Distilled : A Brief Guide to the Standard Object Modeling Language / M. Fowler. — 3rd ed. — Boston : Addison-Wesley Professional, 2018. — 208 p.
6. Booch G. The Unified Modeling Language User Guide / G. Booch, J. Rumbaugh, I. Jacobson. — 2nd ed. — Boston : Addison-Wesley Professional, 2017. — 496 p.
7. Saaty T. L. The Analytic Hierarchy Process : Planning, Priority Setting, Resource Allocation / T. L. Saaty. — McGraw-Hill International, 2018. — 287 p.
8. Martin R. C. Clean Architecture : A Craftsman's Guide to Software Structure and Design / R. C. Martin. — London : Prentice Hall, 2017. — 432 p.
9. Katz J. Introduction to Modern Cryptography / J. Katz, Y. Lindell. — 3rd ed. — Boca Raton : CRC Press, 2020. — 600 p.
10. Freeman A. Pro MERN Stack : Full Stack Web App Development with Mongo, Express, React, and Node / A. Freeman. — 2nd ed. — New York : Apress, 2019. — 450 p.
11. Bradshaw S. MongoDB: The Definitive Guide : Powerful and Scalable Data Storage / S. Bradshaw, E. Brazil, K. Chodorow. — 3rd ed. — Sebastopol : O'Reilly Media, 2019. — 514 p.

12. Mongoose Documentation [Електронний ресурс]. — URL: https://mongoosejs.com/docs/guide.html (дата звернення: 22.05.2025).

13. Hogan B. Web Development with Node and Express : Leveraging the JavaScript Stack / B. Hogan. — 2nd ed. — Sebastopol : O'Reilly Media, 2019. — 340 p.

14. Express.js Routing and Middleware [Електронний ресурс]. — URL: https://expressjs.com/en/guide/routing.html (дата звернення: 21.05.2025).

15. Banks A. Learning React : Modern Patterns for Developing React Apps / A. Banks, E. Porcello. — 2nd ed. — Sebastopol : O'Reilly Media, 2020. — 296 p.

16. React Documentation [Електронний ресурс]. — URL: https://react.dev/reference/react (дата звернення: 20.05.2025).

17. Node.js Documentation [Електронний ресурс]. — URL: https://nodejs.org/en/docs/ (дата звернення: 21.05.2025).

18. Cherny B. Programming TypeScript : Making Your JavaScript Applications Scale / B. Cherny. — Sebastopol : O'Reilly Media, 2019. — 324 p.

19. TypeScript Documentation [Електронний ресурс]. — URL: https://www.typescriptlang.org/docs/ (дата звернення: 20.05.2025).

20. Material UI (MUI) Components [Електронний ресурс]. — URL: https://mui.com/material-ui/getting-started/ (дата звернення: 23.05.2025).

21. Axios HTTP Client Documentation [Електронний ресурс]. — URL: https://axios-http.com/docs/intro (дата звернення: 23.05.2025).

22. Telegram Bot API Documentation [Електронний ресурс]. — URL: https://core.telegram.org/bots/api (дата звернення: 22.05.2025).

23. JSON Web Token (JWT) : RFC 7519 [Електронний ресурс] / IETF. — URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 24.05.2025).

24. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks [Електронний ресурс]. — URL: https://owasp.org/www-project-top-ten/ (дата звернення: 24.05.2025).

25. Bohr A. Artificial Intelligence in Healthcare / A. Bohr, K. Memarzadeh. — London : Academic Press, 2020. — 356 p.

ДОДАТОК А

Index.ts:

```
import express from 'express';
import {createServer} from 'http';
import "dotenv/config";
import bodyParser from 'body-parser';
import cookieParser from 'cookie-parser';
import cors from 'cors';
import compression from 'compression';
import mongoose from 'mongoose';
import router from "./router";
import appointmentRoutes from './router/appointmentRouter';
import doctorRouter from './router/doctorRouter';
import telegramRoutes from './router/telegram';
import { startTelegramBot } from './utils/telegramBot';

mongoose.Promise = Promise;

const MONGODB_URI = process.env.MONGODB_CONNECTION_STRING;

mongoose
  .connect(MONGODB_URI)
  .then(() => {
    console.log("✅ Connected to MongoDB database!");
    console.log("Database URL:", MONGODB_URI.replace(/\/\/.*:.*@/, '//***:***@'));
    startTelegramBot();
  })
  .catch((error) => {
    console.error("❌ MongoDB connection error:", error.message);
    console.error("Connection string used:", MONGODB_URI.replace(/\/\/.*:.*@/, '//***:***@'));
    process.exit(1);
  });

const app = express();
```

```
app.use(cors({
    origin: 'http://localhost:5173',
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With', 'Accept']
}));

// Інші middleware
app.use(compression());
app.use(cookieParser());
app.use(bodyParser.json());
app.use(express.json());

// Роути
app.use('/api', doctorRouter);
app.use('/api', appointmentRoutes);
app.use('/api/telegram', telegramRoutes);
app.use('/', router());

// Обробка помилок
app.use((err: any, req: express.Request, res: express.Response, next: express.NextFunction) => {
    console.error(err.stack);
    res.status(500).json({ message: 'Щось пішло не так!' });
});

const server = createServer(app);

server.listen(8080, () => {
    console.log('Server is running on http://localhost:8080');
});
```

telegram.ts:

```
import express from 'express';
import { isAuthenticated } from '../middlewares';
import { UserModel, updateUserById } from '../db/usersScheme';

const router = express.Router();

router.post('/link', isAuthenticated, async (req, res) => {
   try {
      const { telegramId } = req.body;
      const userId = req.identity._id;

      if (!telegramId) {
         return res.status(400).json({ message: 'Telegram ID is required' });
      }

      const user = await UserModel.findById(userId);
      if (!user) {
         return res.status(404).json({ message: 'User not found' });
      }

      await updateUserById(String(userId), { telegramId });

      res.json({ message: 'Telegram account linked successfully' });
   } catch (error) {
      console.error('Error linking Telegram account:', error);
      res.status(500).json({ message: 'Internal server error' });
   }
});

export default router;
```

scheduleRouter.ts:

```
import express from 'express';
```

```
import { isAuthenticated } from '../middlewares';
import { getUserById, updateUserSchedule } from '../db/usersScheme';
import Appointment, { IAppointment } from '../db/appointmentScheme';
import { TimeSlot, DoctorSchedule } from '../types/doctor';

const router = express.Router();

// Get doctor's schedule
router.get('/:doctorId', async (req: express.Request, res: express.Response) => {
    try {
        const { doctorId } = req.params;
        console.log('GET /schedule/:doctorId - Отримання розкладу для лікаря:', doctorId);

        const doctor = await getUserById(doctorId);

        if (!doctor) {
            console.log('Лікаря не знайдено:', doctorId);
            return res.status(404).json({ success: false, message: 'Лікаря не знайдено' });
        }

        if (!doctor.schedule) {
            console.log('Розклад порожній, повертаємо пустий масив');
            return res.json({
                success: true,
                message: 'Розклад успішно отримано',
                schedule: []
            });
        }

        console.log('Отримано лікаря з бази:', JSON.stringify(doctor, null, 2));
        console.log('Поточний розклад лікаря:', JSON.stringify(doctor.schedule, null, 2));

        // Отримуємо всі активні записи для цього лікаря
        const appointments = await Appointment.find({
            doctorId,
```

```
            status: 'активний'
        }).lean();

        console.log('Активні записи:', JSON.stringify(appointments, null, 2));

        // Додаємо інформацію про записи до розкладу
        const scheduleWithAppointments = doctor.schedule.map(day => ({
            ...day,
            slots: day.slots.map((slot: TimeSlot) => {
                const hasAppointment = appointments.some((app: IAppointment) =>
                    app.date === day.date &&
                    app.time === slot.time
                );
                return {
                    ...slot,
                    hasAppointment
                };
            })
        }));

                                    console.log('Підготовлений розклад для відправки:',
JSON.stringify(scheduleWithAppointments, null, 2));

        res.json({
            success: true,
            message: 'Розклад успішно отримано',
            schedule: scheduleWithAppointments
        });
    } catch (error) {
        console.error('Помилка при отриманні розкладу:', error);
        res.status(500).json({ success: false, message: 'Помилка при отриманні розкладу' });
    }
});

// Update doctor's schedule
```

```
router.post('/', isAuthenticated, async (req: express.Request, res: express.Response) => {
  try {
    if (!req.identity) {
      console.log('POST /schedule - Користувач не авторизований');
      return res.status(401).json({ success: false, message: 'Користувач не авторизований' });
    }

    const userId = req.identity._id;
    const userRole = req.identity.role;

    console.log('POST /schedule - Оновлення розкладу для користувача:', { userId, userRole });

    if (userRole !== 'doctor') {
      console.log('Спроба оновити розклад не-лікарем:', userRole);
      return res.status(403).json({ success: false, message: 'Немає прав на зміну розкладу' });
    }

    const { schedule } = req.body;

    console.log('Отриманий розклад:', JSON.stringify(schedule, null, 2));

    if (!Array.isArray(schedule)) {
      console.log('Невірний формат розкладу:', schedule);
      return res.status(400).json({ success: false, message: 'Невірний формат розкладу' });
    }

    // Валідація формату дати та часу
    for (const day of schedule) {
      if (!day.date || !day.slots || !Array.isArray(day.slots)) {
        return res.status(400).json({
          success: false,
          message: 'Невірний формат даних розкладу'
        });
      }
```

```
    // Перевірка формату дати (YYYY-MM-DD)
    if (!/^\d{4}-\d{2}-\d{2}$/.test(day.date)) {
      return res.status(400).json({
        success: false,
        message: `Невірний формат дати: ${day.date}. Очікується формат YYYY-MM-DD`
      });
    }

    // Перевірка слотів
    for (const slot of day.slots) {
      if (!slot.time || typeof slot.isAvailable !== 'boolean' || !slot.duration) {
        return res.status(400).json({
          success: false,
          message: 'Невірний формат даних слота'
        });
      }

      // Перевірка формату часу (HH:mm)
      if (!/^([0-1]?[0-9]|2[0-3]):[0-5][0-9]$/.test(slot.time)) {
        return res.status(400).json({
          success: false,
          message: `Невірний формат часу: ${slot.time}. Очікується формат HH:mm`
        });
      }
    }
}

// Перевіряємо, чи не видаляються слоти з активними записами
const existingAppointments = await Appointment.find({
    doctorId: userId,
    status: 'активний'
});

console.log('Існуючі активні записи:', existingAppointments);
```

```
        for (const appointment of existingAppointments) {
          const daySchedule = schedule.find(day => day.date === appointment.date);
          if (daySchedule) {
                    const timeSlot = daySchedule.slots.find((slot: TimeSlot) => slot.time === appointment.time);
            if (!timeSlot || !timeSlot.isAvailable) {
              console.log('Спроба видалити час з активним записом:', { date: appointment.date, time: appointment.time });
              return res.status(400).json({
                success: false,
                message: `Неможливо видалити час ${appointment.time} ${appointment.date}, оскільки на нього вже є запис`
              });
            }
          }
        }

        try {
          // Зберігаємо розклад в базу даних
          const updatedUser = await updateUserSchedule(userId, schedule);

          if (!updatedUser || !updatedUser.schedule) {
            console.log('Помилка при оновленні розкладу в базі даних');
            return res.status(500).json({
              success: false,
              message: 'Помилка при оновленні розкладу'
            });
          }

          console.log('Розклад успішно оновлено:', updatedUser.schedule);

          res.json({
            success: true,
            message: 'Розклад успішно оновлено',
            schedule: updatedUser.schedule
```

```
            });
        } catch (error: any) {
            console.error('Помилка при збереженні розкладу:', error);
            return res.status(500).json({
                success: false,
                message: error.message || 'Помилка при оновленні розкладу'
            });
        }
    } catch (error) {
        console.error('Помилка при оновленні розкладу:', error);
        res.status(500).json({ success: false, message: 'Помилка при оновленні розкладу' });
    }
});

export default router;
```