МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МАТІЯШ МИКОЛА ВЯЧЕСЛАВОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р техн. наук, професор
_____ Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ВЕБЗАСТОСУНКУ ДЛЯ МЕДИЧНИХ ЦЕНТРІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Ірина ФРИЗ, старший викладач
кафедри інформаційних технологій,
канд. фіз.-мат. наук
_____

Оцінка: _____ / _____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____

Вінниця - 2026

АНОТАЦІЯ

Матіяш М. В. Розробка клієнт-серверного вебзастосунку для медичних центрів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено проблематику цифрової трансформації процесів запису та управління в приватних медичних центрах і розроблено повнофункціональний вебзастосунок «Ескулап». Система забезпечує онлайн-запис до лікарів, підключення AI-помічника на базі Gemini для рекомендації спеціаліста за симптомами, управління графіками та слотами, персональні кабінети для трьох ролей (пацієнт, менеджер, адміністратор), завантаження фото лікарів і аватарок, моніторинг активності, звіти та управління акціями. Застосунок побудовано за допомогою стеку React + Vite + Zustand + Tailwind CSS (фронтенд) та Express + SQLite + PostgreSQL + JWT + Multer (бекенд).

Ключові слова: вебзастосунок, медичний центр, онлайн-запис, AI-помічник, управління графіками, React, Express, SQLite, PostgreSQL.

70 ст., 55 рис., 8 табл., 3 дод., 41 джерело.

ABSTRACT

Matiiash M. Development of a client-server web application for medical centers. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification (bachelor's) thesis examines the challenges of digital transformation in appointment booking and management processes of private medical centers and develops the full-featured web application «Aesculapius». The system provides online doctor booking, Gemini-based AI assistant for specialist recommendation by symptoms, schedule and slot management, personalized dashboards for three roles (patient, manager, administrator), doctor photo and user avatar uploads, activity monitoring, reports and promotions management. The application is built using the React + Vite + Zustand + Tailwind CSS stack (frontend) and Express + SQLite + PostgreSQL + JWT + Multer (backend).

Keywords: web application, medical center, online booking, AI assistant, schedule management, React, Express, SQLite, PostgreSQL.

ЗМІСТ

ВСТУП

Сучасна система охорони здоров’я в Україні переживає активну цифрову трансформацію, що дозволяє медичним центрам оптимізувати процеси запису на прийом, підвищувати доступність послуг та суттєво покращувати взаємодію з пацієнтами. Онлайн-системи запису до лікарів стають невід’ємною складовою ефективної роботи закладів, забезпечуючи автоматизацію розкладів, зменшення черг, персоналізовані рекомендації та цілодобовий доступ до сервісу через будь-який пристрій. Впровадження таких рішень сприяє зниженню адміністративного навантаження на реєстрацію на десятки відсотків, підвищенню завантаженості лікарів та зростанню задоволеності пацієнтів завдяки зручності та швидкості процесу.

Актуальність теми дослідження зумовлена стрімким зростанням попиту на цифрові медичні сервіси в Україні. Багато невеликих та середніх медичних центрів досі покладаються на телефонний запис або паперові журнали, що спричинює помилки у графіках, простої спеціалістів та втрати пацієнтів. Існуючі популярні платформи (Helsi.me, Health24 та інші) пропонують потужний функціонал, однак часто мають суттєві обмеження для приватних центрів: високі комісії, недостатню гнучкість налаштувань під конкретний заклад, обмежену інтеграцію з внутрішніми процесами управління та слабку підтримку додаткових сервісів (AI-рекомендації, аналітика, управління акціями). Таким чином, залишається потреба у власних рішеннях, які поєднують сучасний інтерфейс, реальний час оновлення даних, повноцінне управління графіками та адаптацію до специфіки роботи українських приватних медичних центрів.

Мета роботи полягає у розробці клієнт-серверного вебзастосунку для медичних центрів, який забезпечує комплексну автоматизацію онлайн-запису на прийом, управління розкладами лікарів, персоналізовані рекомендації через AI-помічник, адміністрування системи та аналітику діяльності закладу.

Об’єктом дослідження є процес розробки клієнт-серверного вебзастосунку для автоматизації діяльності медичних центрів в умовах цифрової трансформації охорони здоров’я.

Предмет дослідження становлять методи, архітектурні рішення, моделі даних та програмні засоби реалізації вебзастосунку, що забезпечують ефективне управління записами, графіками, профілями лікарів, ролями користувачів та додатковими сервісами (AI-рекомендації, звіти, завантаження фото, моніторинг активності).

Визначено такі завдання дослідження:

– проаналізувати предметну область медичних інформаційних систем та сформувати ключові вимоги до програмного забезпечення для приватних медичних центрів;

– дослідити існуючі вебзастосунки для управління записами та медичними центрами, виявивши їхні переваги й недоліки;

– розробити архітектуру системи, структуру бази даних та бізнес-процеси основних модулів;

– реалізувати ключові алгоритми роботи онлайн-запису, управління графіками, автентифікації та AI-рекомендацій;

– виконати програмну реалізацію клієнтської та серверної частин, забезпечити адаптивний інтерфейс та провести комплексне тестування;

– оцінити отримані результати та визначити напрями подальшого розвитку продукту.

Практичне значення полягає у створенні повноцінного вебзастосунку, адаптованого до потреб українських приватних медичних центрів середнього та малого сегменту. Розроблена система забезпечує зручний онлайн-запис, AI-помічник для підбору спеціаліста, управління графіками та акціями, рольова модель доступу, аналітику завантаженості та експорт звітів, що сприяє підвищенню операційної ефективності, зменшенню адміністративних витрат та покращенню клієнтського досвіду. Отриманий продукт може використовуватися

як готове рішення для конкретного закладу або як основа для подальшої комерціалізації та масштабування.

Апробація результатів дослідження: результати дослідження доповідалися на VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології», 15 травня 2026 р., Донецький національний університет імені Василя Стуса, м. Вінниця, тема доповіді: «ВЕБЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ МЕДИЧНИМ ЦЕНТРОМ З AI-ПОМІЧНИКОМ НА БАЗІ GOOGLE GEMINI».

Робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються теоретичні основи, особливості предметної області та аналіз існуючих рішень. Другий розділ присвячений проектуванню архітектури, моделям даних та алгоритмам функціонування. Третій розділ містить опис реалізації, особливості інтерфейсу, заходи забезпечення безпеки та результати тестування. Робота включає 54 рисунки, 9 таблиць, 2 додутки та 40 використаних джерел.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБЗАСТОСУНКІВ ДЛЯ МЕДИЧНИХ ЦЕНТРІВ

1.1 Особливості предметної області медичних інформаційних систем

Предметна область медичних інформаційних систем вирізняється унікальною комбінацією технічних, етичних і правових викликів, що випливають із необхідності обробки надзвичайно чутливих даних про стан здоров'я людини. На відміну від інших галузей, де інформаційні системи оперують переважно фінансовими чи комерційними даними, медичні системи мають справу з персональною інформацією, яка безпосередньо впливає на життя та благополуччя пацієнтів.

Ця особливість зумовлює високий рівень відповідальності розробників і операторів таких систем, оскільки будь-яка помилка в обробці даних або порушення конфіденційності може призвести не лише до матеріальних збитків, а й до непоправних наслідків для здоров'я людини [1].

У сучасних умовах цифрової трансформації охорони здоров'я особливе значення набуває інтеграція різних рівнів даних – від електронних медичних карт пацієнтів до результатів лабораторних досліджень і рекомендацій щодо лікування. Водночас предметна область характеризується необхідністю врахування багаторольової структури користувачів: пацієнти потребують зручного інтерфейсу для самостійного запису на прийом, лікарі – інструментів для ведення історії хвороби, а адміністратори закладів – модулів для аналітики та звітності [2]. Ці вимоги диктують використання гнучких архітектурних рішень, здатних масштабуватися залежно від розміру медичного закладу та обсягу оброблюваної інформації.

Важливим аспектом є дотримання медичної таємниці та принципів етики, які в умовах цифровізації набувають нових форм. Медичні інформаційні системи в Україні та країнах Європейського Союзу будуються з урахуванням

національного законодавства про захист персональних даних та європейських стандартів, що передбачає створення єдиних реєстрів медичних записів [3]. Медичні інформаційні системи повинні передбачати механізми шифрування даних у спокої та під час передачі, а також аудиту всіх дій користувачів. Це особливо актуально в контексті розвитку телемедицини та віддаленого моніторингу стану пацієнтів, коли інформація циркулює між різними пристроями та платформами [4].

Крім того, предметна область вимагає інтеграції стандартизованих класифікаторів хвороб і процедур, що забезпечує сумісність даних між різними закладами охорони здоров'я. Така стандартизація дозволяє проводити ефективний аналіз епідеміологічної ситуації та формувати обґрунтовані рекомендації для політики в галузі охорони здоров'я.

Таблиця 1.1 відображає порівняльну характеристику ключових вимог до захисту даних у медичних інформаційних системах, що допомагає зрозуміти відмінності між традиційними підходами та сучасними цифровими рішеннями.

Таблиця 1.1 – Порівняння вимог до захисту даних у традиційних паперових та цифрових медичних системах

Аспект захисту	**Традиційна паперова система**	**Цифрова медична інформаційна система**
Конфіденційність	Фізичний доступ до документів	Шифрування та контроль доступу на рівні ролей
Цілісність даних	Ризик втрати або пошкодження паперів	Автоматичні резервні копії та перевірка цілісності
Доступність	Обмежена фізичним місцезнаходженням	Доступ з будь-якого пристрою за умови автентифікації
Аудит дій	Відсутній або ручний журнал	Автоматичний лог усіх операцій з даними

Як видно з наведеного порівняння, цифрові системи значно підвищують

рівень захисту, але водночас вимагають більш складної інфраструктури та постійного моніторингу.

Рис. 1.1 демонструє типову архітектуру медичної інформаційної системи, де чітко видно взаємодію основних компонентів, що забезпечують безперервний обмін даними між користувачами та серверною частиною.

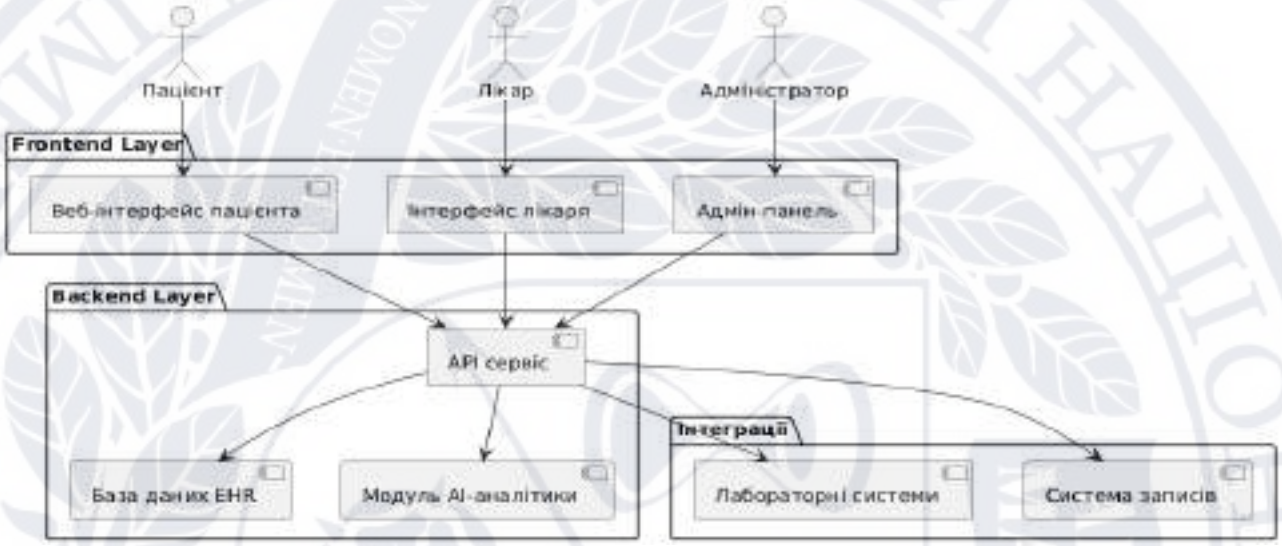

Рисунок 1.1 – Архітектура типової медичної інформаційної системи

Наведена схема ілюструє багатошарову структуру, де frontend забезпечує зручність взаємодії, backend – обробку та зберігання даних, а інтеграційні модулі – обмін інформацією з зовнішніми системами. Така архітектура є ключовою для забезпечення безперервності медичних процесів і відповідає вимогам сучасних стратегій цифрової трансформації [5].

Особливої уваги заслуговує інтеграція штучного інтелекту в медичні інформаційні системи, яка дозволяє автоматизувати процеси рекомендацій щодо спеціалістів, аналізу симптомів і прогнозування навантаження на заклади. Однак впровадження таких технологій вимагає суворого дотримання етичних норм і прозорості алгоритмів, щоб уникнути упереджень у прийнятті рішень щодо лікування [6]. Предметна область також передбачає обов’язкову адаптацію систем до національних особливостей, зокрема врахування регіональних

відмінностей у доступності медичних послуг і специфіки законодавства про електронний документообіг у сфері охорони здоров'я.

Рис. 1.2 ілюструє послідовність основних етапів процесу онлайн-запису на прийом, що є одним із найпоширеніших функціональних блоків сучасних медичних вебсистем.

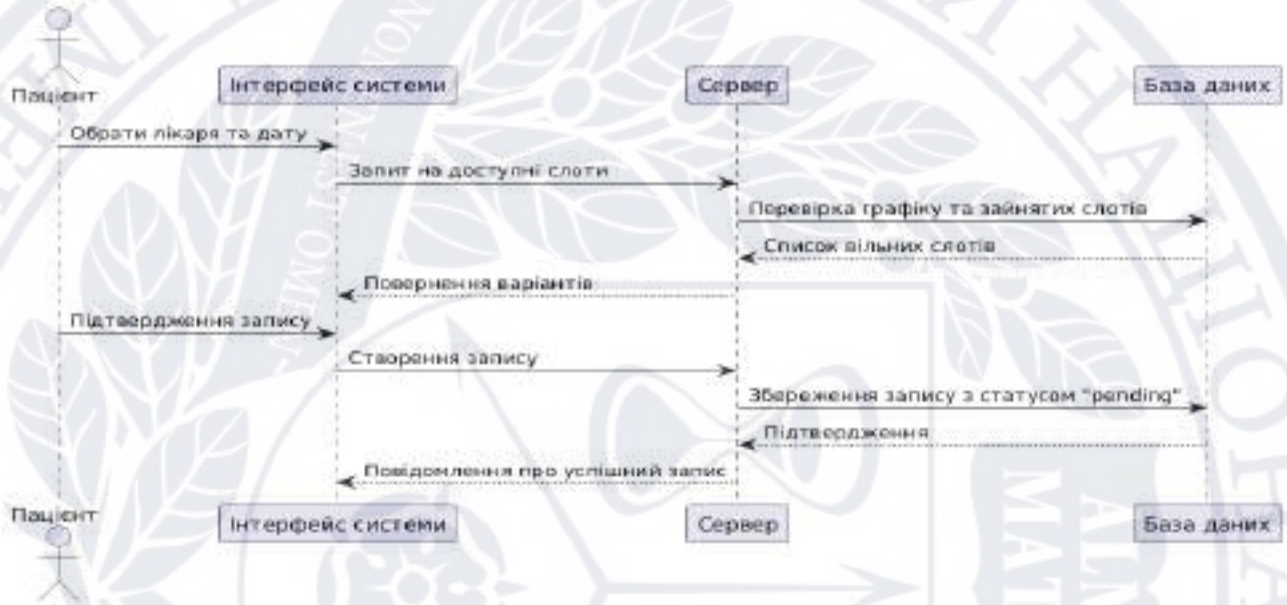

Рисунок 1.2 – Послідовність процесу онлайн-запису на прийом

Цей процес підкреслює важливість реального часу перевірки доступності ресурсів і автоматичного оновлення статусів записів, що мінімізує людський фактор і підвищує ефективність роботи закладів. У цілому предметна область медичних інформаційних систем вимагає комплексного підходу, де технічні рішення тісно переплітаються з правовими, етичними та медичними аспектами, забезпечуючи не тільки зручність користування, але й найвищий рівень безпеки та надійності даних про здоров'я громадян [7].

Таблиця 1.2 містить класифікацію медичних інформаційних систем за рівнем інтеграції та функціональності, що дозволяє краще зрозуміти їхню еволюцію в сучасних умовах.

Таблиця 1.2 – Класифікація медичних інформаційних систем за рівнем інтеграції

Рівень інтеграції	**Характеристика системи**	**Приклади застосування**
Локальний	Обробка даних у межах одного закладу	Внутрішній облік пацієнтів
Регіональний	Обмін даними між закладами одного регіону	Єдина система записів області
Національний	Інтеграція з державними реєстрами	Електронна система охорони здоров’я
Міжнародний	Сумісність зі світовими стандартами	Телемедицина з зарубіжними партнерами

Така класифікація демонструє поступовий перехід від ізольованих рішень до глобально інтегрованих платформ, що є актуальним трендом розвитку галузі. У підсумку особливості предметної області медичних інформаційних систем зумовлюють необхідність постійного балансу між інноваційністю технологій і жорсткими вимогами до захисту прав пацієнтів, що робить їх розробку одним із найвідповідальніших напрямків сучасної інформаційної інженерії.

1.2 Аналіз існуючих вебзастосунків для управління медичними центрами

Сучасний ринок цифрових рішень для медичних центрів характеризується значною різноманітністю вебзастосунків, які пропонують функціональність від простого онлайн-запису до комплексних систем управління всією діяльністю закладу. Аналіз наявних платформ дозволяє виявити основні тенденції розвитку, сильні та слабкі сторони різних підходів, а також визначити прогалини, які залишаються актуальними для нових розробок. Більшість комерційних рішень орієнтовані на ринки США та Західної Європи, де домінують платформи з

високим рівнем інтеграції з національними електронними системами охорони здоров’я та страхових компаній [8].

Однією з найпоширеніших категорій є системи типу Practice Management Software (PMS), які поєднують функції онлайн-запису, управління графіком лікарів, ведення електронних медичних карт, білінгу та базової звітності [9]. Такі платформи, як Zocdoc, Doctolib (Європа) чи Practo (Індія та Південно-Східна Азія), роблять основний акцент на зручності для пацієнтів, пропонуючи швидкий пошук лікаря за спеціалізацією, відгуками, розташуванням та доступним часом.

Водночас для медичних центрів ці системи надають інструменти автоматичного нагадування про візити через SMS та email, інтеграцію з Google Calendar та базовий аналітичний модуль. Проте в більшості випадків глибока клінічна функціональність (детальна історія хвороби, призначення ліків, інтеграція з лабораторними інформаційними системами) залишається обмеженою або доступною лише в преміум-версіях [10].

Інший значний сегмент ринку займають комплексні електронні медичні інформаційні системи (EMR/EHR), які орієнтовані переважно на лікарів та клінічний персонал. Прикладами є Epic Systems, Cerner, Allscripts (США), CompuGroup Medical (Німеччина) та MedDream (Литва та країни Балтії). Такі системи пропонують повноцінне ведення структурованої медичної документації відповідно до міжнародних стандартів (HL7 FHIR, SNOMED CT, ICD-11), потужні інструменти клінічного прийняття рішень, інтеграцію з пристроями моніторингу та підтримку телемедицини. Однак їхня вартість, складність впровадження та потреба в значній IT-інфраструктурі роблять ці рішення практично недоступними для невеликих та середніх медичних центрів, особливо в країнах з перехідною економікою [11].

Таблиця 1.3 наводить порівняльну характеристику основних функціональних блоків, які найчастіше зустрічаються в сучасних вебзастосунках для медичних центрів.

Таблиця 1.3 – Порівняння ключових функціональних модулів у популярних медичних вебплатформах

Функціональний модуль	**Zocdoc / Doctolib**	**Epic / Cerner**	**CompuGroup Medical**	**MedDream / OpenEMR**
Онлайн-запис пацієнтом	Повноцінний	Обмежений	Повноцінний	Повноцінний
Управління графіком лікарів	Базове	Розширене	Розширене	Середнє
Електронна медична карта (EHR)	Відсутня / базова	Повноцінна	Повноцінна	Повноцінна
Інтеграція з лабораторіями	Обмежена	Повна	Повна	Середня
Телемедицина	Так	Так	Так	Так (залежить від версії)
AI-рекомендації спеціалістів	Так (простий)	Так (клінічний)	Так (клінічний)	Ні / плагіни
Аналітика та звіти для керівництва	Базова	Розширена	Розширена	Середня
Вартість впровадження (середня)	Низька–середня	Висока	Висока	Низька–середня

Як видно з таблиці, існує чіткий розподіл: платформи, орієнтовані на пацієнта, пропонують зручний інтерфейс запису, але слабку клінічну частину, тоді як професійні EHR-системи забезпечують глибоку медичну функціональність за значно вищу ціну та складність.

Рис. 1.3 ілюструє типову функціональну архітектуру сучасного комерційного вебзастосунку для управління медичним центром середнього розміру.

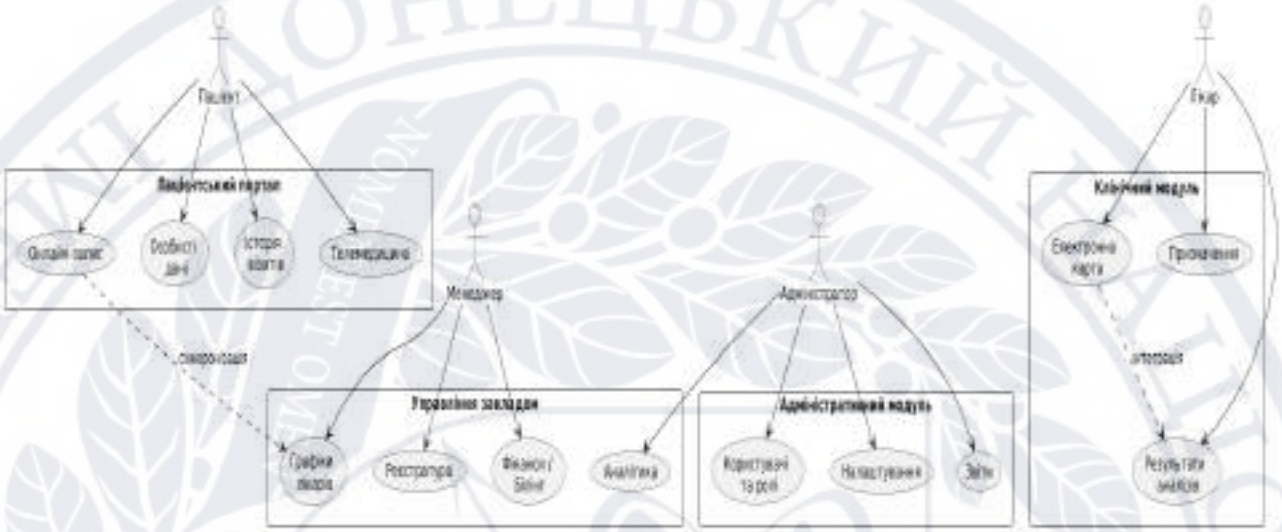

Рисунок 1.3 – Типова функціональна архітектура комерційного медичного вебзастосунку

Наведена схема демонструє, що більшість рішень будуються за модульним принципом, де пацієнтський портал та клінічний модуль часто функціонують як відносно незалежні підсистеми, об’єднані через центральний сервер управління записами та графіками.

Окремий напрямок розвитку пов’язаний із впровадженням штучного інтелекту. Платформи, такі як Ada Health, Babylon Health (до 2023 року) та сучасні модулі в Epic та Cerner, пропонують симптом-чекери та первинні рекомендації щодо спеціалізації. Однак рівень точності таких систем залишається предметом дискусій, а етичні та юридичні аспекти їх використання регулюються дуже жорстко, особливо в країнах ЄС [12].

В Україні та країнах Східної Європи спостерігається зростання інтересу до відкритих рішень (наприклад, OpenEMR, Open Hospital) та локальних розробок, які намагаються поєднати доступну вартість із базовою відповідністю національним вимогам електронного документообігу в охороні здоров’я [13].

Водночас більшість таких систем суттєво поступаються комерційним аналогам у зручності інтерфейсу, швидкості роботи та підтримці сучасних технологій (PWA, real-time оновлення, мобільна адаптивність) [14].

Рис. 1.4 показує еволюцію функціональних можливостей медичних вебзастосунків за останні п’ять років.

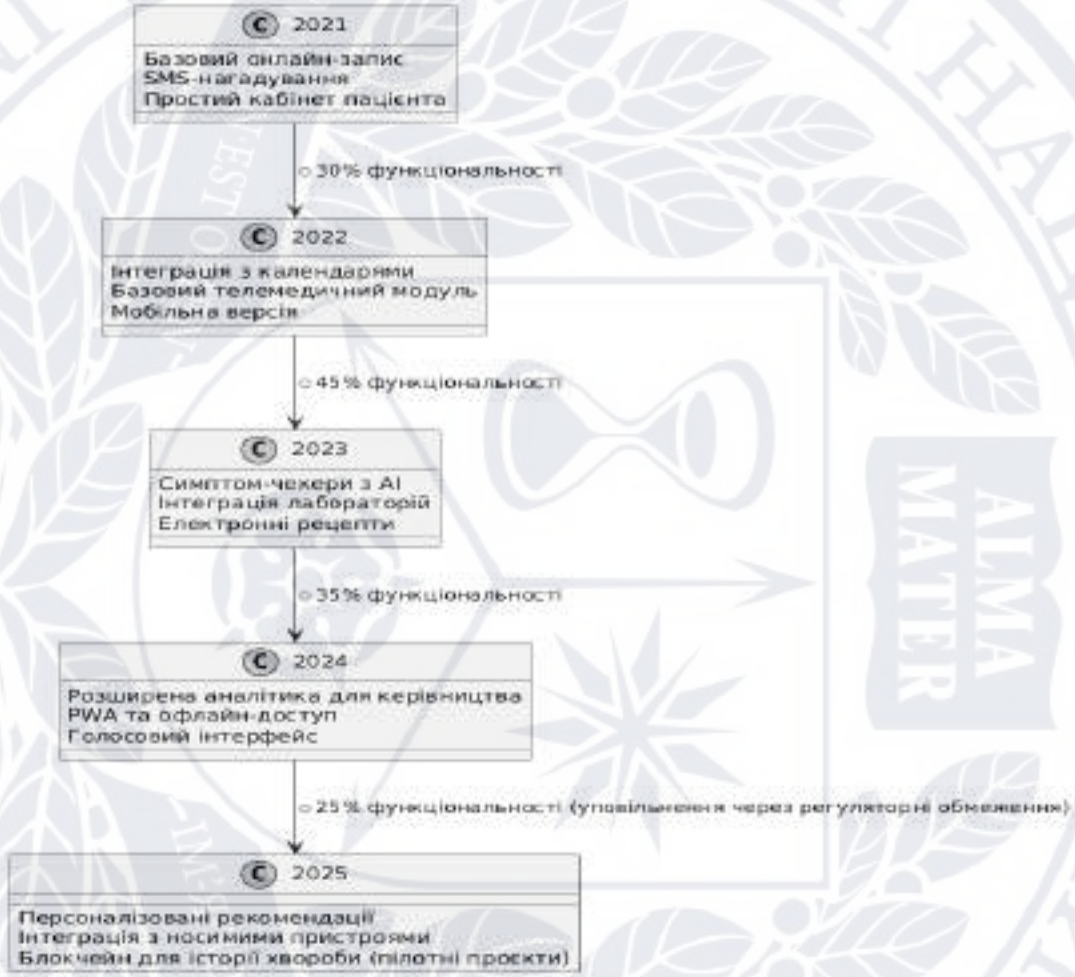

Рисунок 1.4 – Еволюція функціональних можливостей медичних вебзастосунків

Аналіз існуючих рішень показує, що ринок розділений між двома основними підходами: доступні платформи з акцентом на зручність пацієнта та дорогі комплексні системи з потужною клінічною складовою. Водночас залишається значна ніша для гнучких, помірно коштовних рішень, які поєднують сучасний інтерфейс, базову AI-функціональність та відповідність локальним регуляторним вимогам[15].

Саме в цій ніші з’являються нові можливості для розробки спеціалізованих вебзастосунків, адаптованих до потреб невеликих та середніх медичних центрів країн із перехідною економікою.

1.3 Технології та інструменти розробки клієнт серверних вебзастосунків

Розробка сучасних клієнт-серверних вебзастосунків для медичних центрів вимагає ретельного вибору технологічного стеку, який одночасно забезпечує високу продуктивність, безпеку обробки персональних даних, зручність інтерфейсу для різних категорій користувачів та можливість швидкого масштабування системи. У 2024–2026 роках ринок веброзробки остаточно закріпив домінування JavaScript-екосистеми як на клієнтській, так і на серверній стороні, що дозволяє створювати єдиний стек (full-stack JavaScript / TypeScript) і значно скорочує час на розробку та підтримку [16].

На клієнтській стороні беззаперечним лідером залишається React – бібліотека, яка завдяки компонентному підходу, віртуальному DOM та потужній екосистемі дозволяє створювати складні інтерактивні інтерфейси з високою швидкодією. Для прискорення розробки та оптимізації часу збірки широко застосовується Vite – сучасний інструмент, що замінив Create React App у більшості нових проєктів завдяки миттєвому запуску dev-сервера та значно швидшій продакшн-збірці.

Управління станом у великих медичних системах найчастіше реалізується через рішення на кшталт Zustand або Jotai, які забезпечують простоту та передбачуваність порівняно з Redux, але при цьому добре масштабовані [17].

Для стилізації інтерфейсів переважає підхід CSS-in-JS або використання сучасних препроцесорів з підтримкою змінних та nested rules. Популярність набирають утиліти на кшталт Tailwind CSS та його похідні (UnoCSS, Panda CSS), які дозволяють швидко створювати адаптивні дизайни без написання великої кількості власного CSS [18]. Водночас для проєктів, де критична семантична

чистота та повторне використання стилів, зберігається використання модульних стилів (CSS Modules) або Styled Components.

Анімації та мікроінтеракції реалізуються переважно за допомогою бібліотек Framer Motion та GSAP, що забезпечують плавність і доступність навіть на мобільних пристроях [19].

Для серверної частини Node.js з фреймворком Express залишається одним із найпоширеніших стеків для створення RESTful та GraphQL API завдяки простоті, великій кількості готових middleware та високій продуктивності при правильному масштабуванні. Для автентифікації та авторизації стандарт де-факто – JSON Web Tokens (JWT) у поєднанні з ролевою моделлю доступу (RBAC). Захист від типових вебвразливостей (XSS, CSRF, SQL-ін’єкції) забезпечується комбінацією helmet, express-rate-limit, express-validator та правильним налаштуванням CORS. Для обробки завантаження файлів (аватарки, медичні зображення) найчастіше використовується multer з жорсткими обмеженнями розміру та типів файлів [20].

База даних у медичних вебзастосунках обирається з урахуванням обсягу даних, вимог до транзакційності та відповідності регуляторним нормам. SQLite залишається популярним вибором для прототипів, невеликих центрів та автономних систем завдяки нульовій конфігурації та вбудованій підтримці в багатьох середовищах.

Для середніх та великих проєктів перевагу надають PostgreSQL (завдяки розширеній підтримці JSONB, повнотекстовому пошуку та геопросторовим даним) або MySQL 8+ з підтримкою JSON. У випадках, коли потрібна висока доступність та горизонтальне масштабування, використовуються MongoDB або розподілені рішення на кшталт CockroachDB [21].

Таблиця 1.4 демонструє порівняння основних технологій клієнтської та серверної частин, які найчастіше застосовуються у вебзастосунках для медичних центрів.

Таблиця 1.4 – Порівняння ключових технологій клієнт-серверної розробки медичних вебзастосунків

Категорія	**Лідер**	**Альтернативи (зростаючі)**	**Переваги лідера**	**Недоліки лідера**
Клієнтський фреймворк	React	Solid.js, Svelte	Величезна екосистема, велика спільнота	Висока крива навчання для новачків
Збірка та dev-сервер	Vite	Turbopack, esbuild	Найшвидший HMR, простота конфігурації	Менше плагінів порівняно з Webpack
Управління станом	Zustand / Jotai	Redux Toolkit, Recoil	Легковагість, мінімальний boilerplate	Менше готових патернів для дуже складних станів
Стилізація	Tailwind CSS	UnoCSS, Panda CSS	Швидкість розробки, консистентність	Великий розмір бандла без purge
Серверний фреймворк	Express	Fastify, NestJS	Простота, величезна кількість middleware	Менша продуктивність порівняно з Fastify
Автентифікація	JWT + RBAC	OAuth 2.1, Passkeys	Простота реалізації, широка сумісність	Необхідність ручного управління токенами
База даних (малий/середній)	PostgreSQL / SQLite	MongoDB, Supabase	Надійність, ACID-транзакції	Складніша міграція порівняно з NoSQL

Рис. 1.5 ілюструє типову сучасну архітектуру клієнт-серверного вебзастосунку для медичного центру з урахуванням найкращих практик.

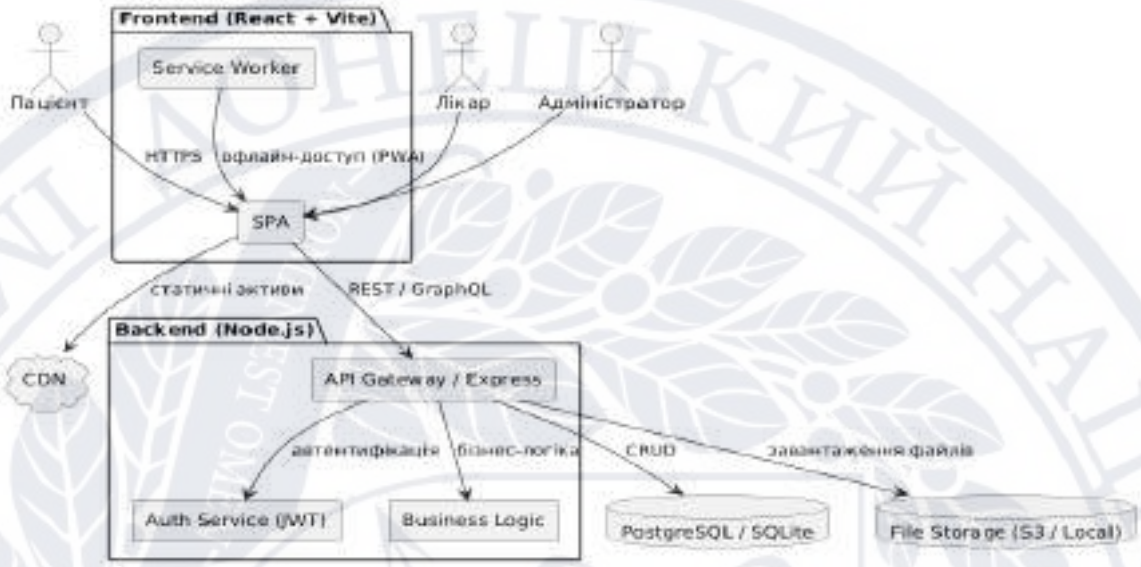

Рисунок 1.5 – Сучасна архітектура клієнт-серверного медичного вебзастосунку

Інтеграція штучного інтелекту та машинного навчання стає невід'ємною частиною нових проєктів. Для чат-ботів та симптом-чекерів широко застосовуються API великих мовних моделей (Google Gemini, OpenAI GPT-4o, Anthropic Claude), тоді як для клінічних рекомендацій та прогнозування навантаження використовуються легковагі моделі на базі TensorFlow.js (клієнтська сторона) або PyTorch / scikit-learn (серверна сторона). Важливим трендом є використання серверних компонентів (React Server Components) та edge-функцій (Vercel Edge, Cloudflare Workers), що дозволяють значно зменшити затримки та обсяг даних, які передаються клієнту [22].

Безпека залишається критичним аспектом. Окрім стандартних заходів (HTTPS, CSP, secure cookies), у медичних системах обов'язково застосовується шифрування даних у спокої (AES-256), токенізація чутливих полів, аудит всіх дій користувачів та регулярне пентестування. Для відповідності GDPR, HIPAA та Дотримання Закону України «Про захист персональних даних» 2297-VI від 14.06.2025 використовуються рішення на кшталт Vault (HashiCorp) або вбудовані механізми шифрування PostgreSQL [23].

Рис. 1.6 ілюструє типовий стек технологій, який найчастіше зустрічається у проєктах для медичних вебзастосунків середнього розміру.

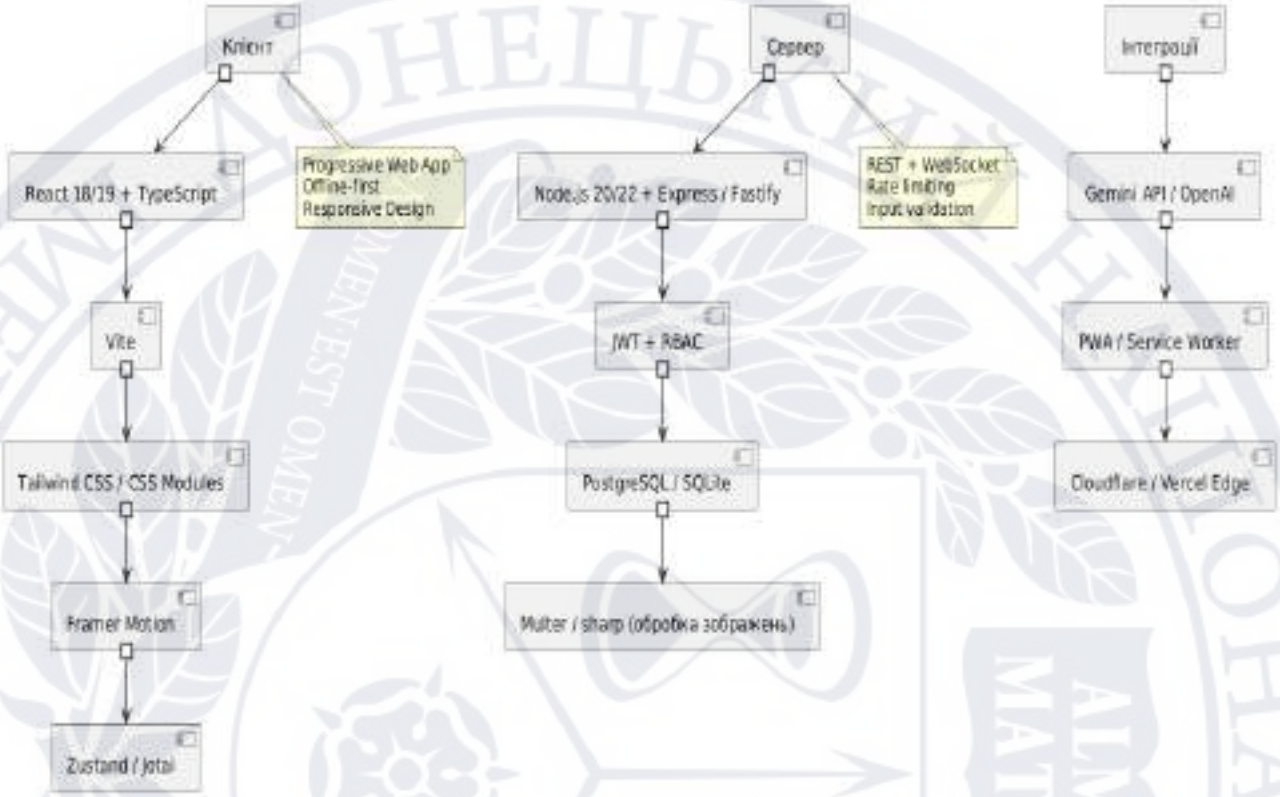

Рисунок 1.6 – Типовий технологічний стек медичного вебзастосунку

У цілому вибір технологій для клієнт-серверних вебзастосунків медичного призначення визначається необхідністю досягти балансу між швидкістю розробки, безпекою даних, зручністю для всіх категорій користувачів та можливістю подальшого розширення системи. Сучасний стек, побудований навколо React + Vite + Node.js + PostgreSQL, є одним із найбільш оптимальних рішень для середніх та великих проєктів, оскільки забезпечує високу продуктивність, широку екосистему та відносно низьку вартість підтримки.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КЛІЄНТ СЕРВЕРНОГО ВЕБЗАСТОСУНКУ ДЛЯ МЕДИЧНИХ ЦЕНТРІВ

2.1 Формування функціональних вимог до вебзастосунку

Формування функціональних вимог є одним із найважливіших етапів проєктування клієнт-серверного вебзастосунку для медичних центрів, оскільки саме на цьому етапі визначаються межі системи, ключові сценарії використання та очікування всіх груп стейкхолдерів. Вимоги мають бути чіткими, вимірюваними, досяжними, релевантними та обмеженими в часі, що відповідає принципам SMART, широко застосовуваним у сучасній практиці розробки медичних інформаційних систем [24].

Процес формування функціональних вимог починається з аналізу потреб основних груп користувачів: пацієнтів, лікарів, реєстраторів/менеджерів та адміністраторів закладу. Пацієнти очікують максимально простого та інтуїтивного інтерфейсу для самостійного запису на прийом, перегляду власної історії візитів, отримання нагадувань та, за можливості, первинної консультації через чат-бот або AI-помічник. Лікарів цікавить швидкий доступ до актуальної інформації про пацієнта, зручне ведення прийому, автоматичне формування призначень та інтеграція з електронними рецептами.

Менеджери потребують інструментів для управління графіками, підтвердження/скасування записів, контролю навантаження на лікарів та базової фінансової аналітики. Адміністратори, у свою чергу, вимагають повноцінного контролю доступу, управління ролями, перегляду детальної звітності та моніторингу активності користувачів системи [25].

Однією з ключових особливостей формування вимог для медичних вебзастосунків є необхідність балансу між зручністю користування та високим рівнем захисту персональних даних. Згідно з чинним законодавством України та європейськими регуляціями (GDPR, HIPAA та Дотримання Закону України «Про

захист персональних даних»), система повинна забезпечувати чітке розмежування прав доступу, повний аудит дій, шифрування даних у спокої та під час передачі, а також можливість відкликання згоди на обробку даних пацієнтом у будь-який момент [26].

Таблиця 2.1 представляє узагальнену класифікацію функціональних вимог за групами користувачів та рівнем критичності.

Таблиця 2.1 – Класифікація функціональних вимог за групами користувачів та критичністю

Група користувачів	**Вимога**	**Критичність**	**Пріоритет**
Пацієнт	Самостійний запис на прийом	Висока	1
Пацієнт	Перегляд історії власних візитів	Середня	2
Пацієнт	Отримання AI-рекомендацій спеціаліста	Середня	3
Лікар	Доступ до електронної карти пацієнта	Висока	1
Лікар	Формування призначень та направлень	Висока	1
Менеджер	Управління графіком роботи лікарів	Висока	1
Менеджер	Підтвердження/скасування записів	Висока	1
Адміністратор	Управління ролями та правами доступу	Висока	1
Адміністратор	Генерація звітів за період	Середня	2
Усі користувачі	Двофакторна автентифікація	Висока	1

Як видно з таблиці, більшість критичних вимог пов’язана з безпекою доступу та основними бізнес-процесами закладу – записом та веденням прийому.

Рис. 2.1 ілюструє Use Case діаграму основних функціональних сценаріїв типового вебзастосунку для медичного центру.

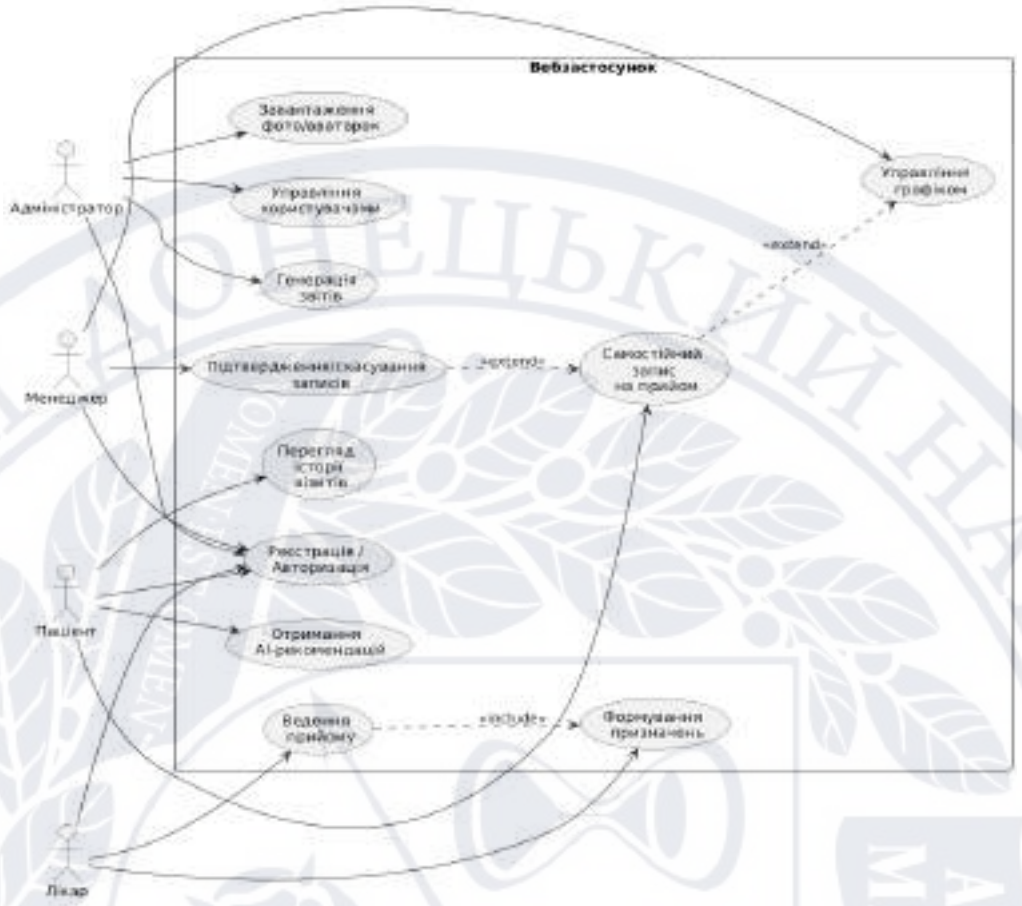

Рисунок 2.1 – Use Case діаграма основних функціональних сценаріїв

Наведена діаграма демонструє, що більшість сценаріїв пацієнта та менеджера тісно пов’язані через підсистему управління записами, тоді як адміністративні функції є відносно ізольованими, що полегшує реалізацію розмежування доступу.

Особливу увагу під час формування вимог варто приділяти нефункціональним аспектам, які безпосередньо впливають на прийнятність системи. Серед них – продуктивність (час відповіді API не більше 800 мс за 95% запитів), доступність (uptime не нижче 99,5%), адаптивність інтерфейсу до мобільних пристроїв, підтримка офлайн-режиму для перегляду базової інформації та локалізація інтерфейсу щонайменше українською мовою [27].

Окремим блоком вимог є інтеграційні сценарії: підключення до зовнішніх лабораторних інформаційних систем, державних реєстрів (наприклад, eHealth), платіжних шлюзів та сервісів відправки повідомлень. Кожен такий інтерфейс

повинен мати чітко визначені формати обміну даними, механізми обробки помилок та резервні сценарії [28].

Рис. 2.2 відображає контекстну діаграму взаємодії вебзастосунку з зовнішніми системами.

Рисунок 2.2 – Контекстна діаграма взаємодії з зовнішніми системами

Формування повного переліку функціональних вимог завершується створенням детальної специфікації, яка стає основою для подальшого архітектурного проєктування, розробки та тестування. Правильно сформульовані вимоги дозволяють уникнути значних переробок на пізніх етапах, забезпечують відповідність кінцевого продукту реальним потребам медичного закладу та підвищують ймовірність успішного впровадження системи в реальних умовах.

2.2 Розробка архітектури системи та структури бази даних

Проєктування клієнт-серверного вебзастосунку для медичних центрів вимагає ретельного балансу між функціональними потребами різних груп користувачів, вимогами до безпеки персональних даних та можливостями масштабування системи в умовах обмежених ресурсів типового медичного закладу. Сучасні підходи до архітектури таких систем ґрунтуються на принципах мікросервісної або модульної монолітної структури з чітким розділенням

відповідальностей між клієнтською та серверною частинами [29]. У розглядуваному випадку обрано класичну трирівневу архітектуру (presentation layer – application layer – data layer), доповнену проксі-шаром для обробки CORS та автентифікації, що дозволяє ефективно організувати взаємодію між React-фронтендом та Express-бекендом.

Рис. 2.3 ілюструє загальну архітектуру системи, де видно основні шари та потоки даних між ними.

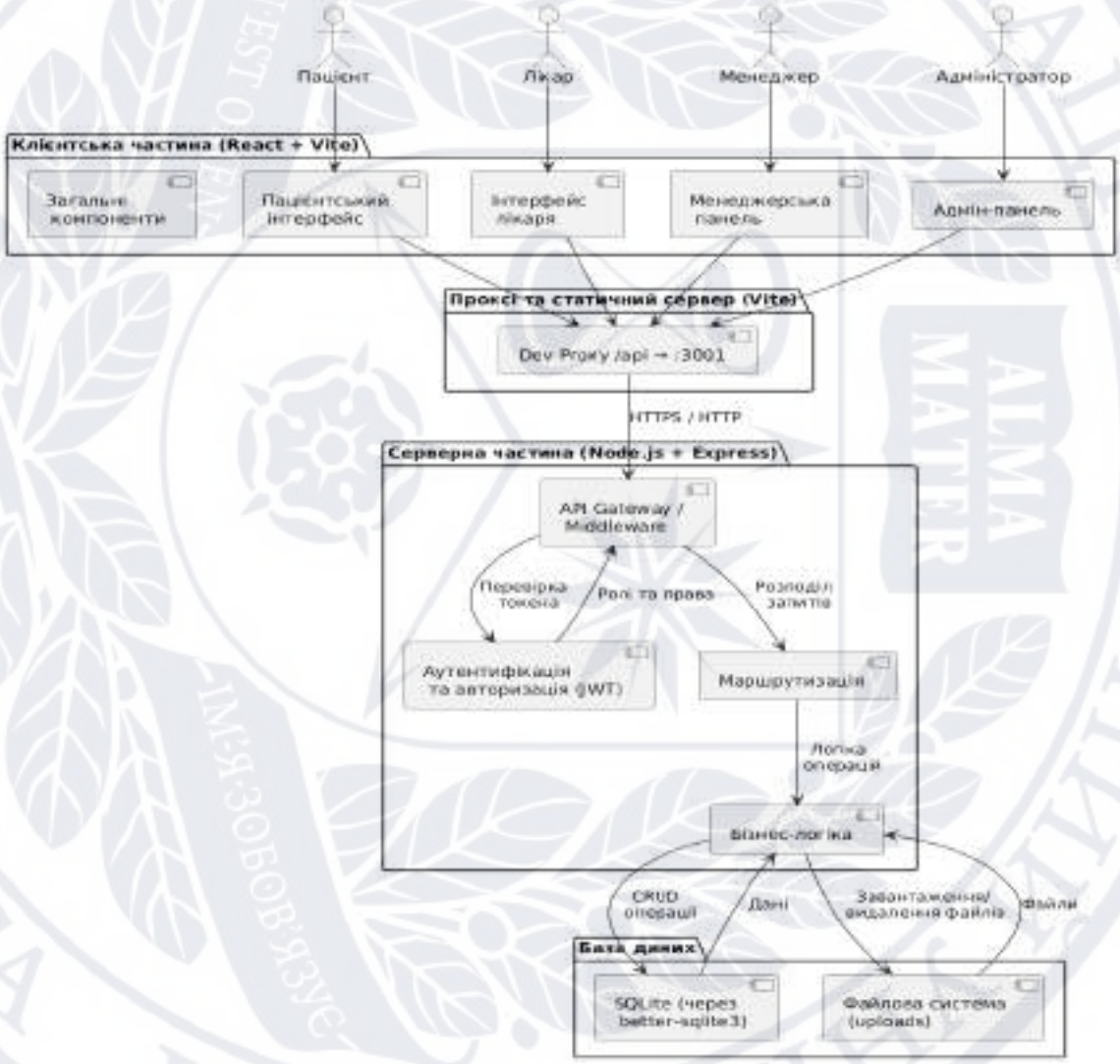

Рисунок 2.3 – Загальна архітектура клієнт-серверного вебзастосунку

У клієнтській частині використовується Vite як інструмент збірки та розробки, що забезпечує швидкий hot-module-replacement та ефективну продакшн-збірку. Усі взаємодії з сервером відбуваються через централізований

Axios-інстанс із перехоплювачами запитів та відповідей, що дозволяє автоматично додавати JWT-токен та обробляти помилки 401/403 [30]. Серверна частина побудована на Express з модульною структурою маршрутів, де кожен ресурс (doctors, appointments, schedules тощо) має власний файл маршрутизації. При цьому, автентифікація реалізується через JWT, а авторизація – через middleware перевірки ролей (verifyToken + requireRole).

Рис. 2.4 демонструє компонентну структуру системи з акцентом на взаємодію основних модулів.

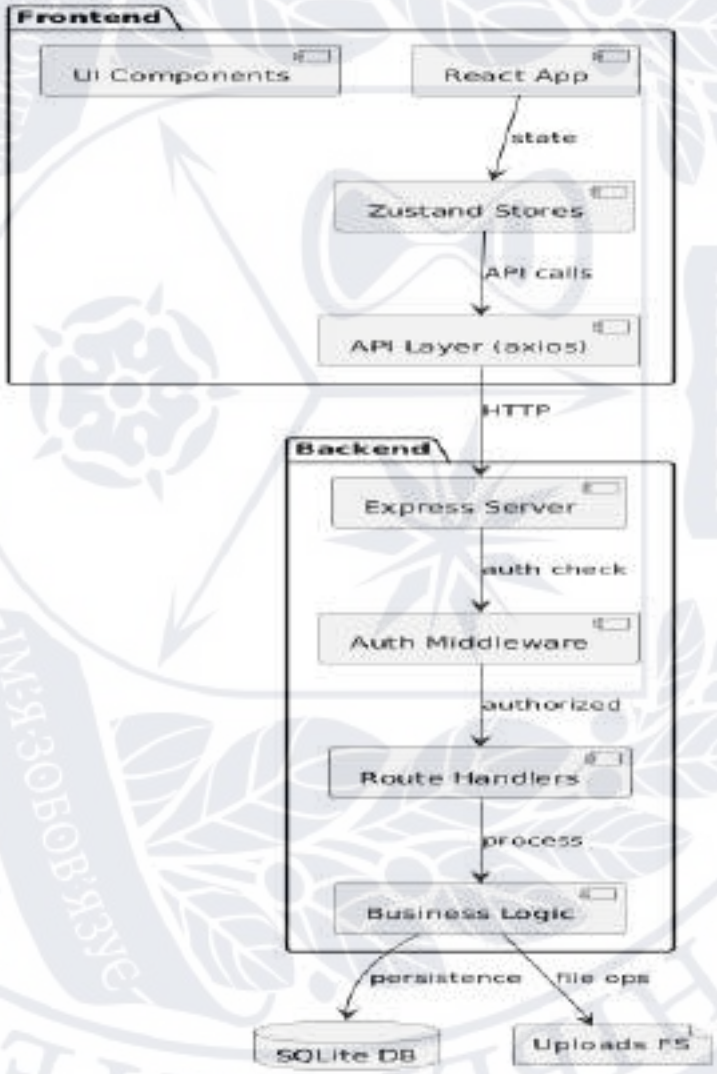

Рисунок 2.4 – Діаграма компонентів системи

Структура бази даних спроєктована з урахуванням нормалізації третьої форми та оптимізації для частого читання графіків і записів. Рис. 2.5 відображає

ER-діаграму структури бази даних. Використовується SQLite+ завдяки його легкості, відсутності потреби в окремому сервері БД та достатній продуктивності для середнього медичного центру, Основні сутності включають користувачів (users), лікарів (doctors), спеціалізації (specializations), графіки роботи (schedules), слоти (генеруються динамічно), записи (appointments), промоакції (promotions) та журнали активності (activity logs) [31, 32]. SQLite використовується для локальної розробки та тестів, PostgreSQL закладено в архітектуру, як основна СУБД для серверу.

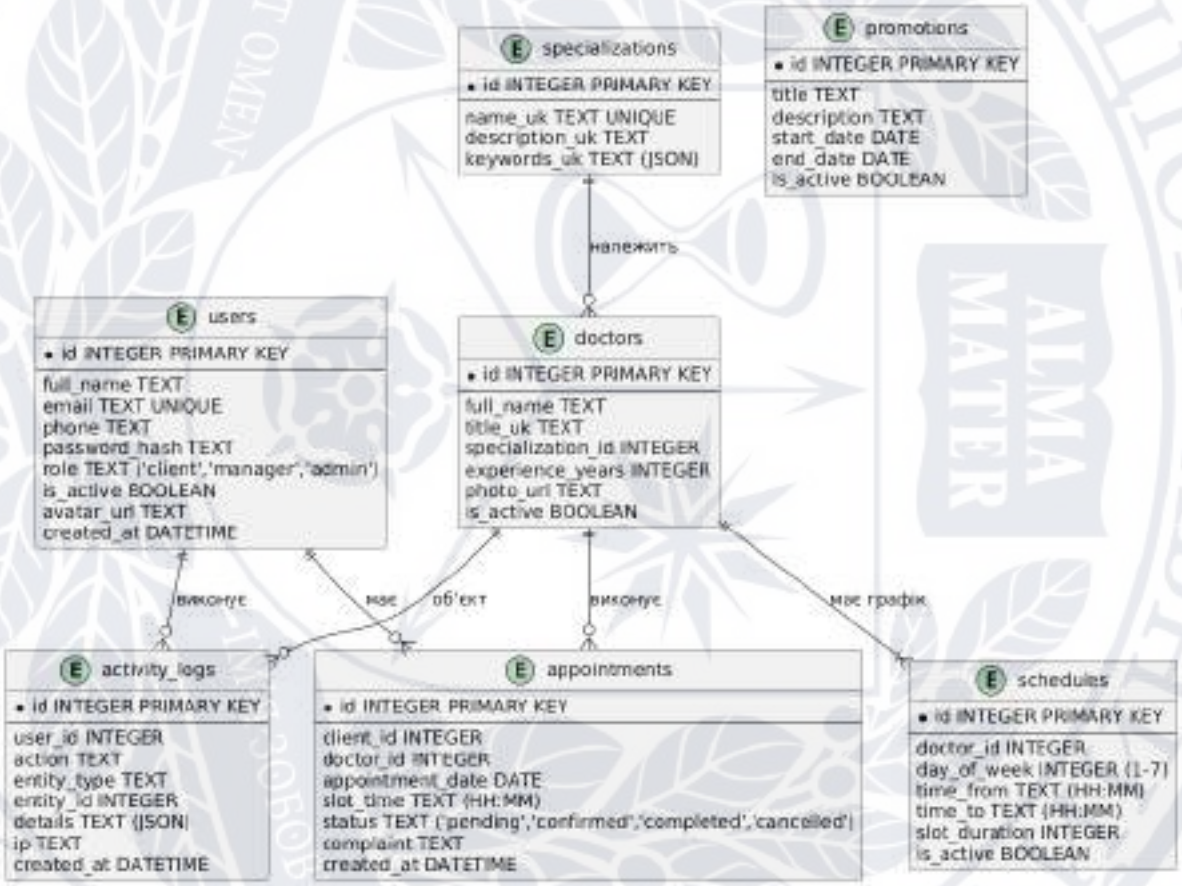

Рисунок 2.5 – ER-діаграма структури бази даних

Життєвий цикл запису на прийом доцільно реалізувати через чітко визначені стани та переходи між ними [33], що і застосовано в даній розробці. Рис. 2.6 показує діаграму станів сутності appointment.

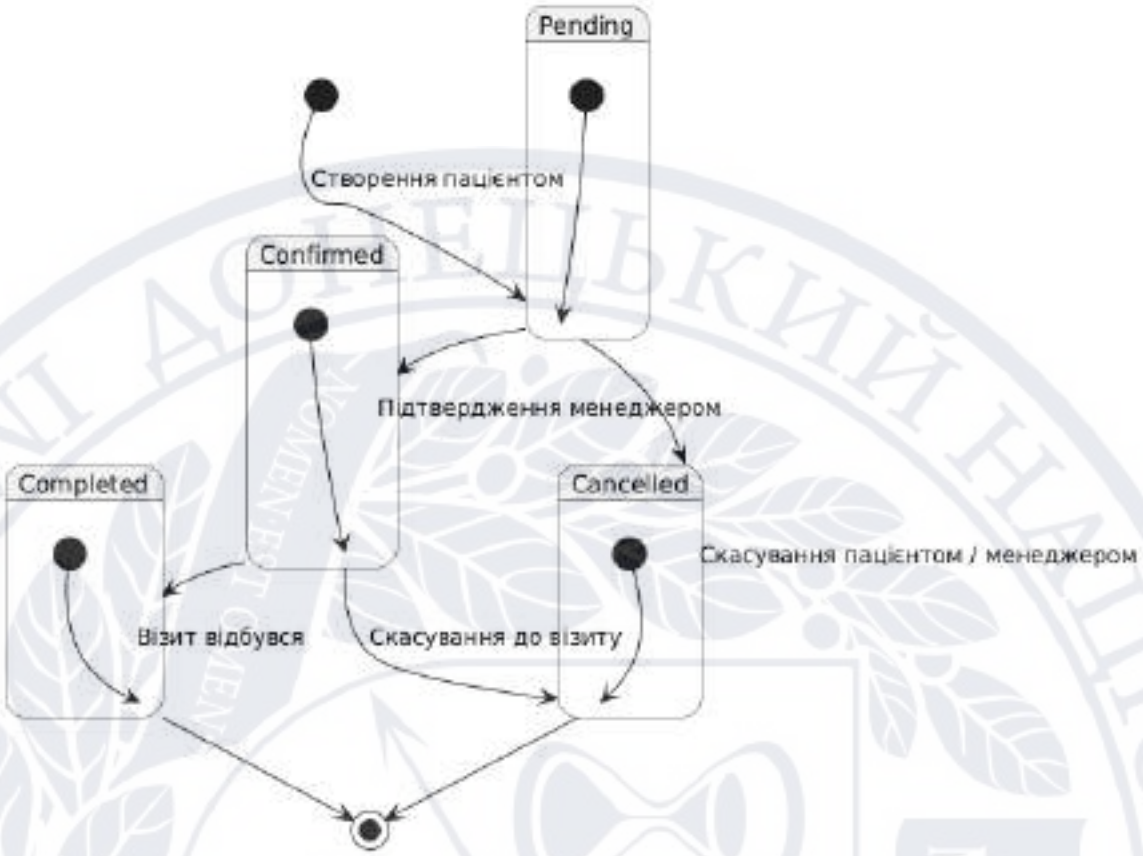

Рисунок 2.6 – Діаграма станів сутності «Запис на прийом»

Запропонована архітектура та структура бази даних дозволяють ефективно реалізувати ключові сценарії використання: самостійний запис пацієнтів, управління графіками та записами менеджерами, контроль доступу та аналітику для адміністраторів, при цьому зберігаючи прийнятну продуктивність навіть на відносно слабкому серверному обладнанні.

Використання SQLite та PostgreSQL як основного сховища даних значно спрощує розгортання та адміністрування системи в реальних умовах невеликих і середніх медичних центрів, де відсутня можливість утримувати окремий штат адміністраторів баз даних [34]. Водночас чітке розділення логіки на клієнтську та серверну частини, централізоване управління автентифікацією та авторизацією, а також продумана модель даних створюють надійний фундамент для подальшого розширення функціональності, зокрема інтеграції з зовнішніми лабораторними системами, державними реєстрами чи платіжними сервісами.

2.3 Розробка алгоритмів роботи основних модулів системи

Проєктування алгоритмів основних модулів клієнт-серверного вебзастосунку для медичних центрів вимагає чіткого врахування специфіки предметної області, де критичними є безпека даних, висока доступність, швидкість обробки запитів та коректність клінічних процесів.

Алгоритми повинні забезпечувати узгодженість стану системи в умовах одночасного доступу багатьох користувачів, мінімізувати ризик втрати даних та гарантувати відповідність нормативним вимогам щодо захисту персональної медичної інформації [35].

Центральним модулем системи є підсистема онлайн-запису на прийом, яка реалізує взаємодію між клієнтським інтерфейсом та серверною логікою перевірки доступності ресурсів. Алгоритм починається з вибору лікаря та дати пацієнтом, після чого клієнт надсилає запит на отримання вільних слотів. Сервер перевіряє графік роботи лікаря на вказаний день тижня, генерує теоретичні слоти на основі тривалості прийому, виключає вже зайняті інтервали та, якщо дата збігається з поточною, додатково фільтрує минулі часові інтервали.

Отриманий список повертається клієнту, де пацієнт обирає конкретний слот. Підтвердження запису супроводжується транзакційним збереженням запису зі статусом «очікування підтвердження», що дозволяє уникнути подвійного бронювання в умовах високого навантаження [36].

Рис. А.1 (Додаток А) ілюструє детальний алгоритм роботи модуля вибору та підтвердження слоту запису.

Ще одним критичним модулем є підсистема рекомендації спеціаліста на основі симптомів, яка поєднує правило-базований підхід із можливістю використання зовнішнього AI-сервісу. Алгоритм починається з отримання текстового опису скарг від пацієнта.

Спочатку виконується базова обробка: токенізація, видалення стоп-слів, пошук ключових симптомів за словником. Якщо виявлено достатньо чітких відповідностей, система повертає рекомендацію на основі заздалегідь

визначених правил (наприклад, «головний біль + нудота → невролог»). У разі неоднозначності або складного опису запит передається до AI-моделі разом з історією попередньої розмови.

Отримана відповідь доповнюється списком рекомендованих спеціалізацій та конкретних лікарів з урахуванням їхньої завантаженості та відгуків. Такий гібридний підхід дозволяє зберегти швидкість роботи при низькій якості інтернет-з'єднання та забезпечити високу точність рекомендацій у складних випадках [37].

Таблиця 2.2 демонструє порівняння основних етапів алгоритмів двох ключових модулів – запису на прийом та AI-рекомендації спеціаліста.

Таблиця 2.2 – Порівняння ключових етапів алгоритмів основних модулів

Етап алгоритму	**Модуль онлайн-запису**	**Модуль AI-рекомендації спеціаліста**
Вхідні дані	Лікар, дата, бажаний час	Текстовий опис скарг, історія чату
Перевірка коректності	Дата не в минулому, лікар активний	Текст не порожній, довжина достатня
Основна обробка	Генерація слотів, виключення зайнятих	Токенізація + правило-базовий пошук / AI-запит
Транзакційна критичність	Висока (запобігання подвійному бронюванню)	Низька (рекомендація не змінює стан системи)
Асинхронні дії	Сповіщення після підтвердження	Логування запиту для подальшого аналізу
Обробка помилок	Повернення чіткого повідомлення + лог	Повернення базової рекомендації або помилки

Модуль управління графіками лікарів реалізує алгоритм, який забезпечує масове та поодиноке редагування робочих інтервалів. Під час створення графіка

перевіряється відсутність конфліктів за лікарем і днем тижня, нормалізуються часові формати, обчислюється тривалість слотів за замовчуванням.

Масове додавання використовує транзакцію для забезпечення атомарності: або всі дні додаються, або жоден. Оновлення графіка включає валідацію нового інтервалу (час початку < часу закінчення) та автоматичне перегенерування потенційних слотів у кеші, якщо він використовується [38].

Рис. 2.7 відображає узагальнений алгоритм оновлення графіка роботи лікаря з урахуванням транзакційної цілісності.

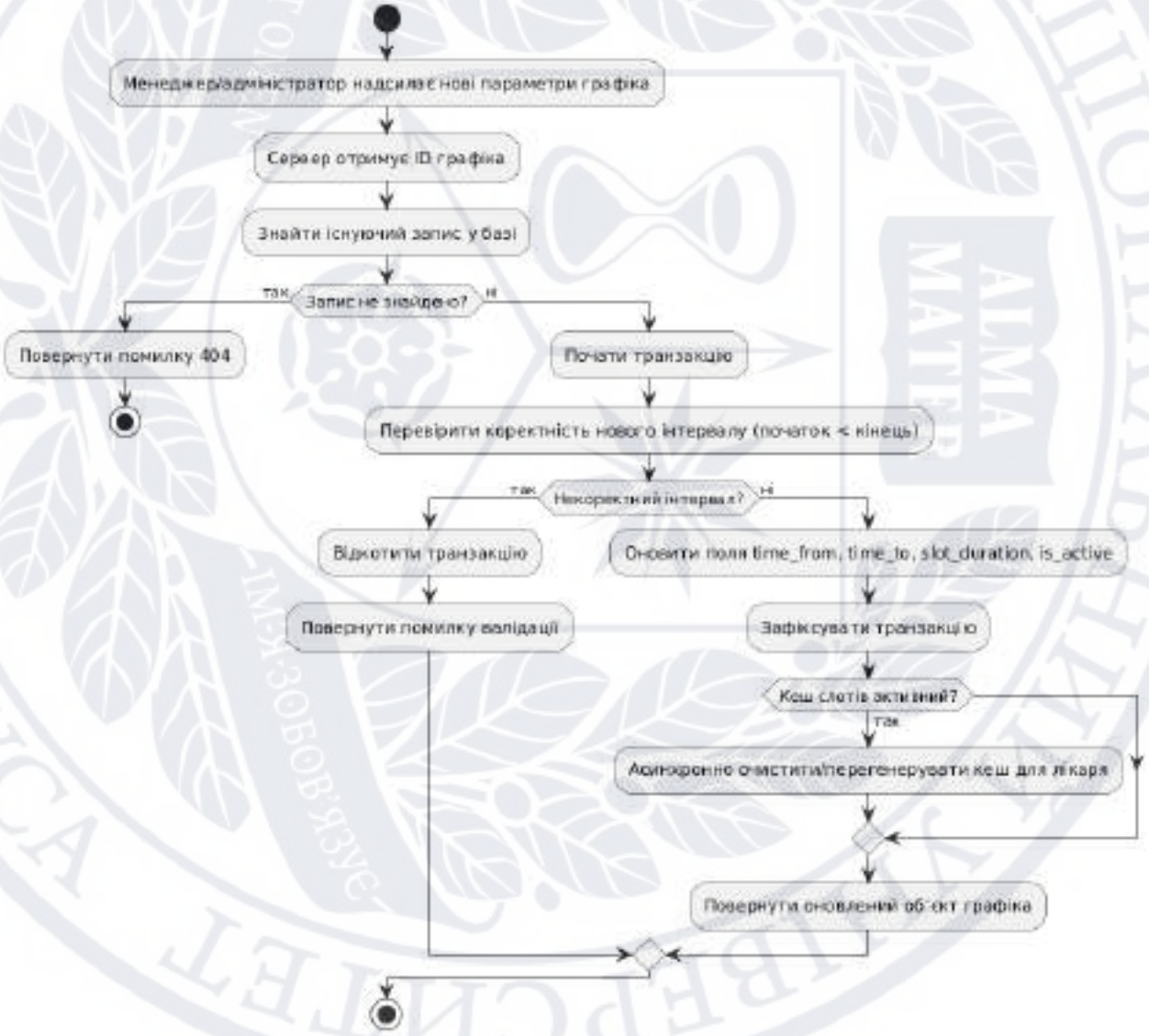

Рисунок 2.7 – Алгоритм оновлення графіка роботи лікаря

Алгоритми автентифікації та авторизації побудовані на основі JWT-

токенів з короткостроковим часом життя та механізмом автоматичного очищення сесії після отримання 401/403 статусів. Клієнт зберігає токен у localStorage через стан-менеджер, сервер перевіряє підпис та термін дії для кожного незахищеного запиту. Під час виявлення блокування акаунта повертається спеціальний код, який ініціює примусовий вихід [39,40].

Розроблені алгоритми забезпечують баланс між продуктивністю, безпекою та зручністю користувачів, відповідають принципам сучасного проєктування розподілених систем та дозволяють масштабувати рішення від невеликих кабінетів до мереж медичних центрів середнього розміру. Їх реалізація враховує як поточні технічні можливості, так і перспективу подальшого розширення функціональності без радикальної переробки ядра системи.

Проєктування клієнт-серверного вебзастосунку для медичних центрів охопило формування функціональних вимог відповідно до потреб пацієнтів, лікарів, менеджерів та адміністраторів з обов’язковим урахуванням вимог безпеки персональних даних і нормативних стандартів. Запропонована трирівнева архітектура з React-фронтендом, Express-бекендом та базою даних SQLite для локальної розробки та тестів та PostgreSQL, закладеної в архітектуру, як основної СУБД для серверу, забезпечує ефективне розмежування відповідальностей, високу продуктивність і простоту розгортання в умовах типових медичних закладів. Розроблені алгоритми основних модулів, зокрема онлайн-запису на прийом, рекомендації спеціаліста та управління графіками, гарантують узгодженість даних, запобігання конфліктам і дотримання принципів безпеки. Такий комплексний підхід створює надійну основу для подальшої реалізації системи, її масштабування та успішного впровадження в практику медичних установ.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ ДЛЯ МЕДИЧНИХ ЦЕНТРІВ

3.1 Розробка користувацького інтерфейсу та основних модулів

Розробка користувацького інтерфейсу вебзастосунку для медичних центрів передбачає створення сучасної, адаптивної та інтуїтивно зрозумілої системи, яка забезпечує комфортну взаємодію для всіх категорій користувачів – від пацієнтів, які записуються на прийом, до лікарів та адміністраторів, які керують графіками та даними.

Основою інтерфейсу є реактивна архітектура, побудована на компонентній моделі, де кожен елемент системи представлений як незалежний модуль з чітко визначеними пропсами та станом. Це дозволяє досягти високої гнучкості, повторного використання коду та швидкого оновлення візуальних елементів без порушення загальної структури застосунку. Головний контейнер застосунку забезпечує централізоване управління маршрутизацією та перевіркою авторизації, що гарантує безпеку доступу до персональних даних пацієнтів і адміністративних функцій.

Користувацький інтерфейс реалізовано з використанням сучасних технологій візуального дизайну, що включають детально продуману палітру кольорів, градієнти та ефекти скла. Центральні семантичні змінні визначають основні відтінки – від насиченого блакитного для акцентів до теплих перламутрових тонів для фонових поверхонь.

Такий підхід створює спокійну, довірливу атмосферу, характерну для медичних закладів, і водночас підкреслює професійність системи. Анімаційні ефекти, реалізовані через вбудовані переходи та бібліотеку для плавних рухів, забезпечують природність взаємодії: картки лікарів піднімаються під час наведення курсора, дні тижня в календарі підсвічуються з м’яким ефектом тіні, а форми введення даних реагують миттєво на фокус. Скролбар і селекція тексту

також стилізовані відповідно до загальної палітри, що підвищує естетичну цілісність інтерфейсу.

Головним елементом навігації є верхній заголовок, який адаптивно змінюється залежно від розміру екрана. На десктопі він відображає логотип, основні посилання та блок авторизації з ролевими бейджами, тоді як на мобільних пристроях активується зручне гамбургер-меню з плавною анімацією розгортання.

Нижня частина інтерфейсу містить футер з хвилею декоративного елемента, контактною інформацією та посиланнями на правові документи, що забезпечує повну орієнтацію користувача в системі. Така структура навігації дозволяє пацієнтам швидко переходити до пошуку лікарів, а адміністраторам – до панелі управління без зайвих кліків.

Таблиця 3.1 відображає основні компоненти користувацького інтерфейсу та їх функціональне призначення, що дає уявлення про модульну структуру системи (див. Додаток Б).

Таблиця 3.1 – Основні компоненти користувацького інтерфейсу та їх призначення

Компонент	**Основне призначення**	**Ключові особливості візуалізації**
1	**2**	**3**
Header	Навігація та авторизація	Адаптивне меню, ролеві бейджі, sticky-позиціонування
Footer	Контакти та правова інформація	Декоративна хвиля, соціальні іконки, градієнтний фон
DoctorCard	Відображення профілю лікаря	Фото з градієнтним фоном, анімація hover, компактний режим
SlotPicker	Вибір дати та часу запису	Тижневий календар, динамічні слоти, індикатори завантаження

Продовження таблиці 3.1

1	2	3
AppointmentCard	Картка запису на прийом	Статусні кольорові бейджі, анімація появи, дії скасування
AIAssistant	Чат з AI-помічником	Повідомлення з аватарами, рекомендації лікарів, індикатор набору
ProfileCard	Редагування профілю пацієнта	Завантаження аватарки, форми редагування з анімацією

Наведена таблиця демонструє, що кожен компонент виконує конкретну роль у загальній архітектурі, забезпечуючи логічну послідовність взаємодії користувача з системою.

Рис. 3.1 ілюструє ієрархію основних компонентів користувацького інтерфейсу, де чітко видно взаємозв’язки між модулями навігації, картками даних та інтерактивними формами.

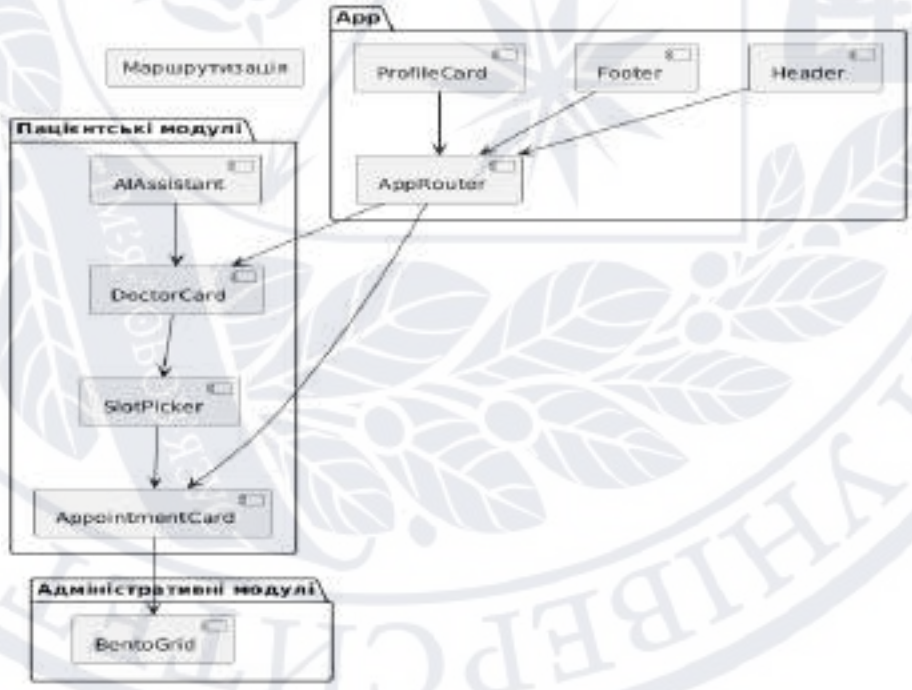

Рисунок 3.1 – Ієрархія основних компонентів користувацького інтерфейсу

Схема підкреслює централізоване управління через головний роутер, що

дозволяє динамічно змінювати вміст сторінок залежно від ролі користувача та стану авторизації.

Розробка модуля вибору слотів для запису на прийом є одним із ключових елементів інтерфейсу. Компонент динамічно завантажує дані про робочі дні тижня та вільні часові інтервали, відображаючи їх у вигляді інтерактивної сітки. Користувач може перемикати тижні за допомогою стрілок навігації, бачити кількість доступних слотів і вибирати зручний час. Під час завантаження даних відображається індикатор прогресу, а після вибору дати автоматично оновлюється список доступних часових інтервалів. Такий підхід забезпечує максимальну зручність і зменшує кількість помилок під час бронювання.

Модуль картки запису на прийом надає пацієнтам та менеджерам повну інформацію про запланований візит: статус, дані лікаря, спеціалізацію, дату та час, а також поле зі скаргами пацієнта. Картка підтримує різні кольорові позначки статусу та дозволяє скасувати запис за певних умов. Для менеджерів додатково відображається ім’я пацієнта. Анімація появи та hover-ефекти роблять інтерфейс живим і приємним для сприйняття.

Інтеграція AI-помічника у вигляді чат-інтерфейсу дозволяє пацієнтам отримувати рекомендації щодо спеціалістів безпосередньо в застосунку. Чат підтримує історію повідомлень, індикатор набору тексту, приклади запитів та рекомендації лікарів з можливістю переходу до їх профілів. Повідомлення форматуються з підтримкою markdown-подібних елементів, а відповіді AI виділяються спеціальним бейджем. Дисклеймер про не медичну консультацію завжди присутній, що відповідає етичним вимогам.

Профіль користувача реалізовано як окрему картку з можливістю редагування персональних даних і завантаження аватарки. Форма редагування з’являється з плавною анімацією, підтримує валідацію та сповіщення про успішні зміни. Завантаження зображення супроводжується індикатором прогресу та можливістю видалення аватарки.

Таблиця 3.2 показує основні стилістичні рішення, застосовані в інтерфейсі для забезпечення єдності дизайну.

Таблиця 3.2 – Основні стилістичні рішення інтерфейсу

Елемент стилю	**Опис реалізації**	**Вплив на користувацький досвід**
Градієнти та фони	Лінійні градієнти основної палітри	Створює глибинний, сучасний вигляд
Ефект скла	Напівпрозорі картки з блюром	Легкість сприйняття, сучасний тренд
Анімації	Framer Motion для переходів і hover	Плавність, приємна взаємодія
Адаптивність	Медіа-запити для мобільних пристроїв	Зручність на будь-якому екрані
Типографіка	Inter та Montserrat з чіткими розмірами	Читабельність і професійний вигляд

Ці рішення забезпечують єдиний візуальний стиль і високу ергономіку інтерфейсу.

Рис. 3.2 демонструє потік взаємодії користувача з модулем AI-помічника та рекомендаціями лікарів.

Рисунок 3.2 – Потік взаємодії з модулем AI-помічника

Наведений потік ілюструє інтеграцію чату з переходом до профілів лікарів, що значно спрощує процес вибору спеціаліста.

У цілому розробка користувацького інтерфейсу та основних модулів забезпечує гармонійне поєднання естетики, функціональності та технічної надійності. Кожен компонент ретельно адаптовано під потреби різних ролей користувачів, а сучасні технології анімації та адаптивності роблять роботу із застосунком комфортною і приємною.

Такий підхід дозволяє медичним центрам ефективно керувати процесами обслуговування пацієнтів, підвищувати якість надання послуг і забезпечувати високий рівень довіри до цифрової платформи.

3.2 Програмна реалізація системи

Розробка клієнт-серверного вебзастосунку для медичних центрів базується на сучасному стеку технологій, що забезпечує високу продуктивність, зручність підтримки та адаптивність до потреб користувачів різних ролей. Архітектура системи побудована за принципом розділення відповідальностей: клієнтська частина (frontend) реалізовано на React з використанням Vite як інструменту збірки, серверна частина (backend) – на Express.js з легкою вбудованою базою даних SQLite для локальної розробки та тестів та PostgreSQL, закладеної в архітектуру, як основної СУБД для серверу.

Такий підхід дозволяє швидко розгортати застосунок, мінімізувати залежності від зовнішніх сервісів баз даних на етапі розробки та тестування, а також забезпечити швидкий старт для невеликих і середніх медичних закладів.

Клієнтська частина побудована з використанням функціональних компонентів та хуків React, що відповідає сучасним практикам розробки. Центральним елементом є файл App.jsx, який відповідає за ініціалізацію маршрутизації та перевірку стану автентифікації при кожному завантаженні сторінки. Використання Zustand як менеджера стану забезпечує легке збереження та синхронізацію токена авторизації між вкладками браузера. Лістинг 3.1 містить фрагмент коду, що демонструє логіку початкової перевірки автентифікації. Цей підхід дозволяє уникнути зайвих запитів до сервера у

випадку, коли користувач уже не авторизований, та забезпечує швидке реагування системи на зміну стану авторизації.

Лістинг 3.1 – Ініціалізація автентифікації під час завантаження застосунку

```
function App() {
  const { checkAuth, token } = useAuthStore();
  // Перевірка авторизації при завантаженні
  useEffect(() => {
    if (token) {
      checkAuth();
    }
  }, []);
  return (
    <BrowserRouter>
      <AppRouter />
    </BrowserRouter>
  );
}
```

Серверна частина реалізовано на Express.js з чітким розділенням маршрутів за функціональними областями. Кожен модуль (doctors, appointments, schedules, slots тощо) має власний файл маршрутизації, що полегшує масштабування та підтримку коду. Важливою особливістю є використання middleware для перевірки токенів та ролей, що гарантує розмежування доступу між пацієнтами, менеджерами та адміністраторами. У лістингу 3.2 показано приклад захищеного маршруту для отримання списку лікарів. Така структура запитів дозволяє легко розширювати функціонал, додаючи нові фільтри без зміни логіки основного запиту.

Лістинг 3.2 – Отримання графіків роботи лікарів (фрагмент schedules.js)

```
router.get('/', (req, res) => {
  const { doctor_id } = req.query;
  let sql = `
    SELECT
      s.id,
      s.doctor_id,
      s.day_of_week,
      s.time_from,
      s.time_to,
      s.slot_duration,
      s.is_active,
```

```
      d.full_name as doctor_name
    FROM schedules s
    JOIN doctors d ON s.doctor_id = d.id
    WHERE 1=1
  `;
  const params = [];
  if (doctor_id) {
    sql += ' AND s.doctor_id = ?';
    params.push(doctor_id);
  }
  sql += ' ORDER BY d.full_name, s.day_of_week';
  const schedules = db.prepare(sql).all(...params);
  res.json({ schedules });
});
```

Одним із ключових компонентів клієнтської частини є механізм вибору вільних слотів для запису на прийом. Компонент SlotPicker реалізує інтерактивний календар на тиждень з можливістю навігації та відображенням доступності. Використання двох ефектів useEffect забезпечує асинхронне завантаження даних тижня та конкретних слотів вибраної дати. У лістингу 3.3 наведено фрагмент логіки завантаження даних тижня, що дозволяє мінімізує кількість запитів до сервера та забезпечує плавне оновлення інтерфейсу при зміні тижня.

Лістинг 3.3 – Завантаження доступності лікаря на тиждень (SlotPicker.jsx)

```
useEffect(() => {
  if (!doctorId) return;
  const loadWeek = async () => {
    setLoading(true);
    try {
      const { data } = await slotsApi.getWeek(doctorId,
weekStart.toISOString().split('T')[0]);
      setWeekData(data.days || []);
    } catch (err) {
      console.error('Error loading week:', err);
    } finally {
      setLoading(false);
    }
  };
  loadWeek();
}, [doctorId, weekStart]);
```

Інтеграція штучного інтелекту реалізовано через окремий модуль AIAssistant, який використовує API Gemini для генерації рекомендацій спеціалістів на основі опису скарг пацієнта. Історія чату зберігається в стані компонента, що дозволяє вести контекстну розмову. У лістингу 3.4 наведено фрагмент, який відповідає за обробку відправлення повідомлення та отримання відповіді від AI. Цей код демонструє стійкість до помилок та збереження контексту розмови, що є критично важливим для якісної взаємодії з пацієнтом.

Лістинг 3.4 – Обробка чату з AI-помічником (AIAssistant.jsx)

```
const handleSubmit = async (e) => {
  e.preventDefault();
  if (!input.trim() || loading) return;
  const userMessage = input.trim();
  setInput('');
  const newMessages = [...messages, { role: 'user', content:
userMessage }];
  setMessages(newMessages);
  setLoading(true);
  try {
    const history = newMessages.slice(1).map(m => ({
      role: m.role,
      content: m.content
    }));
    const { data } = await assistantApi.chat(userMessage,
history);
    setAiPowered(data.ai_powered);
    setMessages(prev => [...prev, {
      role: 'assistant',
      content: data.response,
      recommendations: data.recommendations || [],
      aiPowered: data.ai_powered,
      model: data.model
    }]);
  } catch (err) {
    const errorMsg = err.response?.data?.error || 'Сталася
помилка. Спробуйте ще раз.';
    setMessages(prev => [...prev, {
      role: 'assistant',
      content: errorMsg,
      isError: true
    }]);
  } finally {
    setLoading(false);
  }
};
```

Система завантаження файлів (аватарки користувачів та фото лікарів) реалізована з використанням multer на сервері та FormData на клієнті. Важливою особливістю є автоматичне видалення попередніх файлів під час завантаження нових, що дозволяє економити дисковий простір. У лістингу 3.5 наведено серверну логіку завантаження фото лікаря. Така реалізація забезпечує чистоту файлової системи та актуальність даних у базі.

Лістинг 3.5 – Завантаження та заміна фото лікаря (upload.js):

```
router.post('/doctor/:id', verifyToken, (req, res, next) => {
  if (!['admin', 'manager'].includes(req.user.role)) {
    return res.status(403).json({
      error: 'Недостатньо прав для цієї операції',
      code: 'FORBIDDEN'
    });
  }
  next();
}, uploadDoctorPhoto.single('photo'), (req, res) => {
  if (!req.file) {
    return res.status(400).json({
      error: 'Файл не завантажено',
      code: 'NO_FILE'
    });
  }
  try {
    const doctorId = req.params.id;
    const doctor = db.prepare('SELECT photo_url FROM doctors WHERE
id = ?').get(doctorId);
    if (!doctor) {
      fs.unlinkSync(req.file.path);
      return res.status(404).json({
        error: 'Лікаря не знайдено',
        code: 'DOCTOR_NOT_FOUND'
      });
    }
    if (doctor.photo_url) {
      const oldPath = path.join(__dirname, '..',
doctor.photo_url);
      if (fs.existsSync(oldPath)) {
        fs.unlinkSync(oldPath);
      }
    }
    const photoUrl = `/uploads/doctors/${req.file.filename}`;
    db.prepare('UPDATE doctors SET photo_url = ? WHERE id =
?').run(photoUrl, doctorId);
    res.json({
      success: true,
      photo_url: photoUrl
    });
```

```
  } catch (err) {
    console.error('Doctor photo upload error:', err);
    res.status(500).json({
      error: 'Помилка завантаження фото',
      code: 'UPLOAD_ERROR'
    });
  }
});
```

Глобальні стилі системи визначені через CSS-змінні, що дозволяє легко змінювати кольорову схему та типографіку. Використання градієнтів, ефектів скла (glassmorphism) та плавних анімацій створює сучасний та довірливий інтерфейс, що особливо важливо для медичного застосунку.

Рис. 3.3 ілюструє загальну архітектуру клієнт-серверної взаємодії системи.

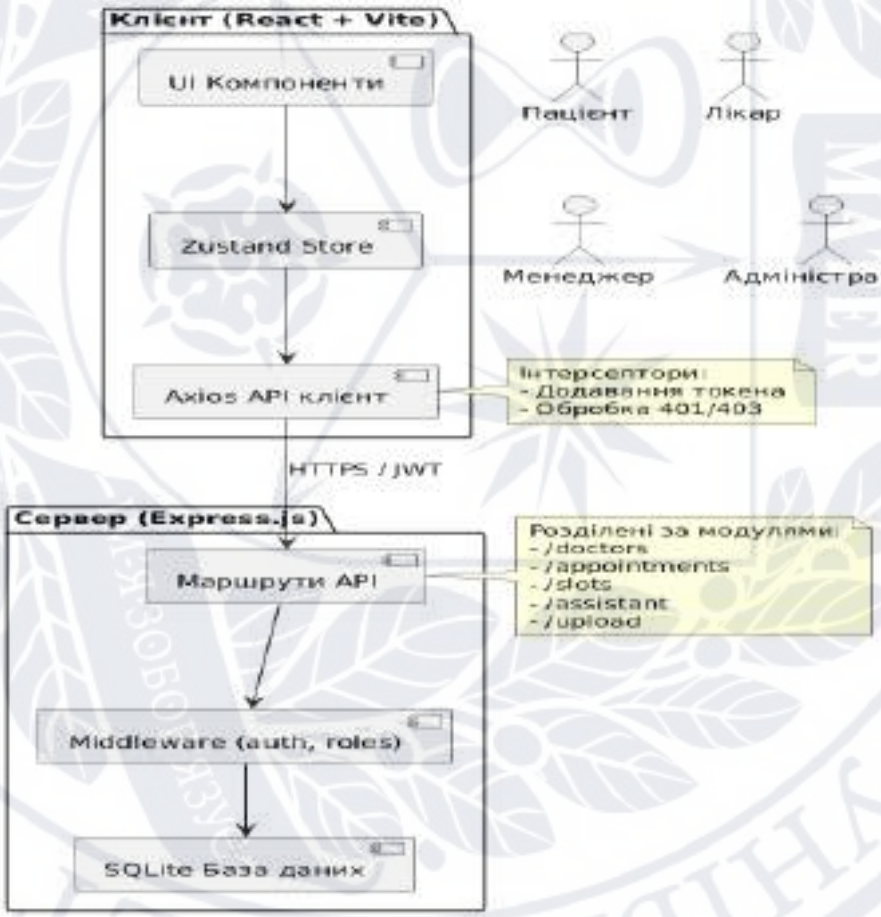

Рисунок 3.3 – Загальна архітектура клієнт-серверної взаємодії

Окрім того, система містить механізм логування активності користувачів, експорт звітів у CSV, управління спеціалізаціями та акціями, що робить її

повноцінним інструментом для автоматизації роботи медичного центру. Використання SQLite як основної бази даних на етапі розробки та тестування значно спрощує розгортання, а перехід на PostgreSQL або MySQL можливий у майбутньому без суттєвих змін у бізнес-логіці.

Таким чином, програмна реалізація поєднує сучасні підходи до розробки інтерфейсів, безпечну обробку автентифікації, ефективне управління ресурсами лікарів та інтеграцію елементів штучного інтелекту, створюючи цілісну екосистему для цифровізації процесів запису, управління та взаємодії з пацієнтами в медичних центрах.

3.3 Тестування програмного продукту та аналіз результатів

Тестування вебзастосунку проводилося з метою перевірки відповідності реалізованого функціоналу заявленим вимогам, оцінки стабільності роботи, зручності інтерфейсу та коректності обробки даних у всіх сценаріях використання.

Особлива увага приділялася перевірці ключових користувацьких сценаріїв: анонімного перегляду інформації, авторизації, запису на прийом, роботи AI-помічника, управління записами та адміністрування системи. Тестування охоплювало функціональне, інтерфейсне, адаптивне та навантажувальне тестування на реальних браузерах (Chrome, Firefox, Safari, Edge) та різних розмірах екранів.

Верхня панель навігації, що показана на рис. 3.4, є єдиним елементом, що присутній на всіх сторінках після завантаження застосунку. Вона містить логотип клініки, основне меню (Головна, Лікарі, Про нас, Контакти), перемикач теми (світла/темна), кнопку входу/виходу та аватар профілю після авторизації.

Рисунок 3.4 – Верхня панель навігації

Головна сторінка, зображена на рис. 3.5, відкривається за замовчуванням і виконує роль інформаційно-презентаційного лендінгу. Безпосередньо під верхньою панеллю розташований великий герой-банер з градієнтним фоном, заголовком «Ескулап – ваш шлях до здоров'я», підзаголовком та двома основними кнопками: «Записатися на прийом» та «AI-помічник».

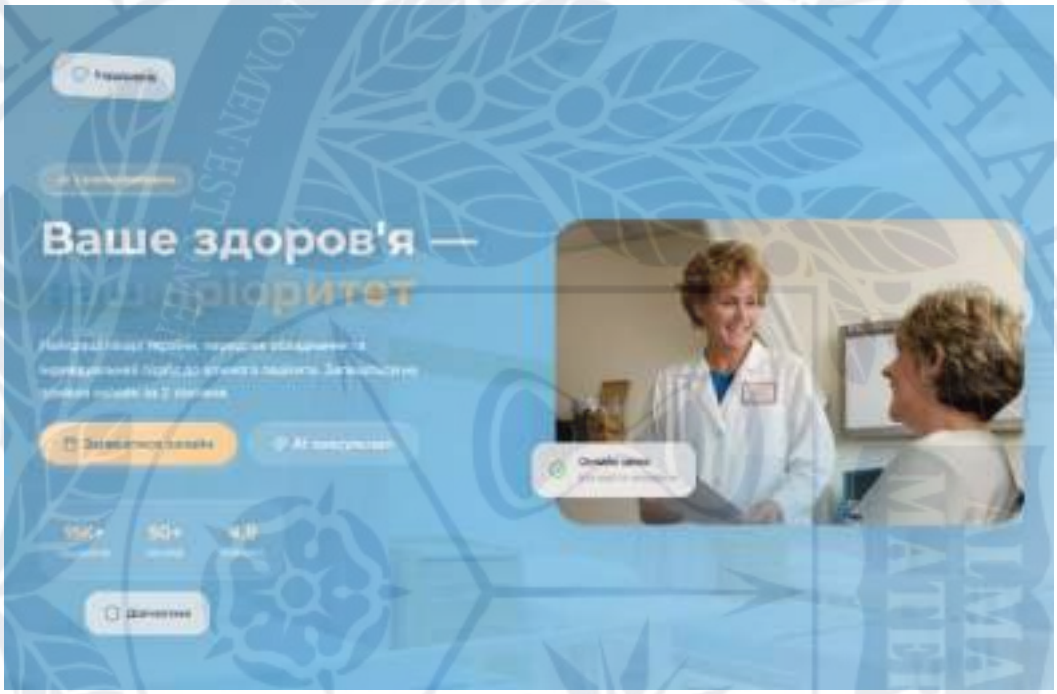

Рисунок 3.5 – Банер головної сторінки

Наступний блок головної сторінки складається з чотирьох інформаційних карток, що показані у десктопному представленні на рис.3.6, розташованих у рядок на великих екранах та у колонку на мобільних пристроях. Кожна картка має іконку, заголовок та короткий опис: «Безпека даних», «Підтримка 24/7», «Швидкий запис онлайн», «Турбота про кожного».

Рисунок 3.6 – Головна сторінка. Блок інформаційних карток

Нижче розміщено блок «Наші спеціалізації», що демонструється на рис. 3.7, який відображає картки спеціалізацій у вигляді сітки з іконкою, назвою та коротким описом. Кожна картка є клікабельною та веде на сторінку детального перегляду спеціалізації.

Рисунок 3.7 – Головна сторінка. Блок «Наші спеціалізації»

Ще нижче розташовано блок статистичних карток, який візуально демонструє досягнення клініки: кількість пацієнтів на рік, кількість лікарів, роки роботи та середній рейтинг. Картки виконані у стилі скляного ефекту з анімацією при скролі, що проілюстровано на рис 3.8.

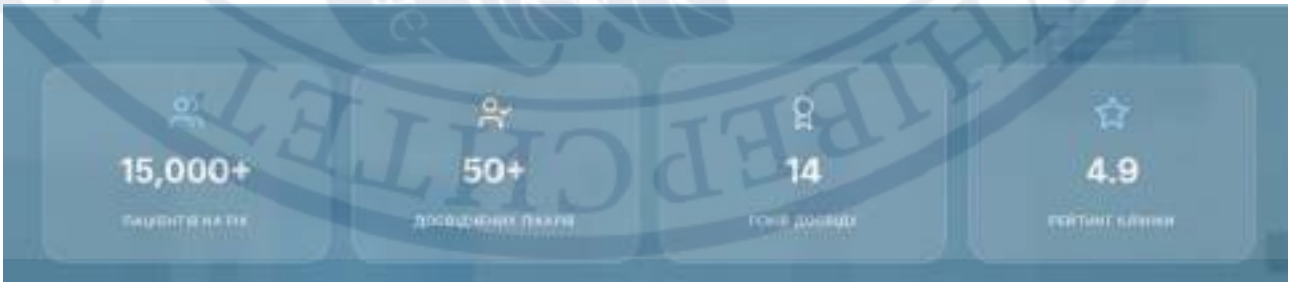

Рисунок 3.8 – Головна сторінка. Блок статистичних показників

Блок «Наші лікарі», що показаний на рис. 3.9 містить горизонтальний слайдер (або сітку на мобільних) з картками лікарів: фото, ПІБ, спеціалізація, стаж, рейтинг та кнопка «Записатися».

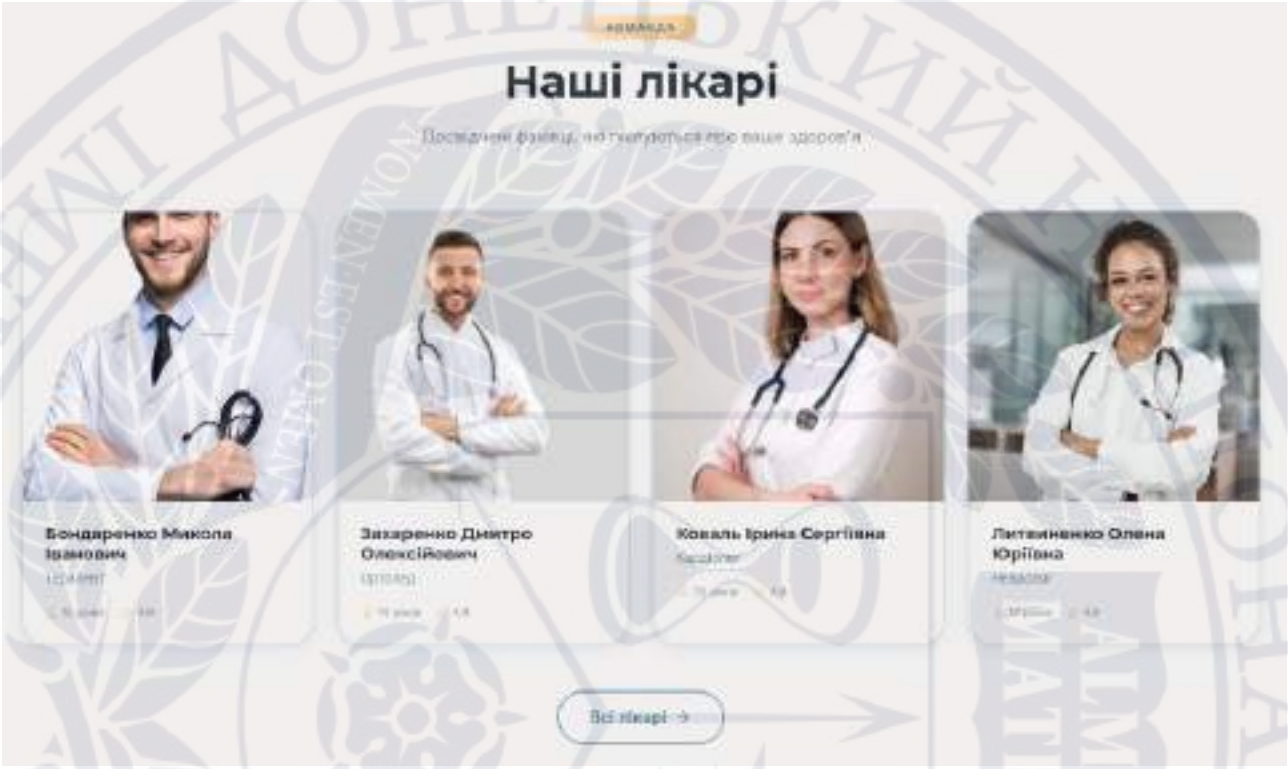

Рисунок 3.9 – Головна сторінка. Блок «Наші лікарі»

Блок «Спеціальні пропозиції», продемонстрований на рис. 3.10, відображає актуальні акції та знижки у вигляді карток з фото, назвою, описом та терміном дії.

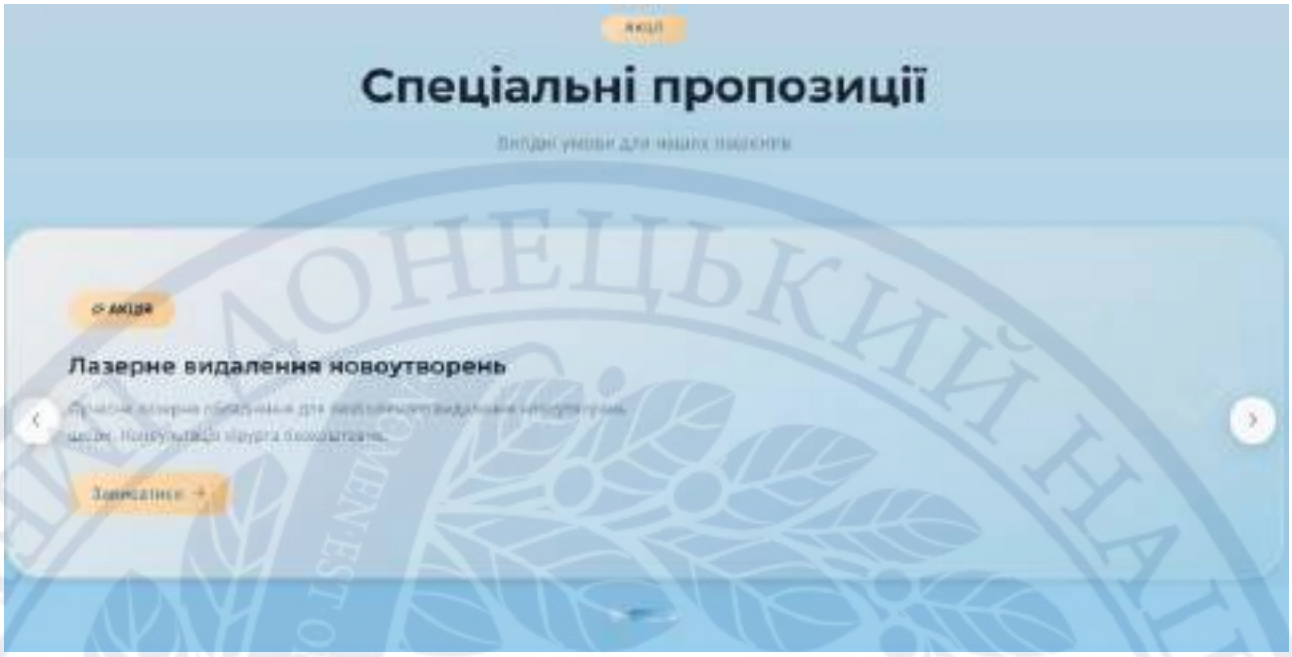

Рисунок 3.10 – Головна сторінка. Блок спеціальних пропозицій

Блок «Часті питання», що покананий на рис. 3.11 з можливістю розкритя пунктів, реалізований у вигляді акордеону, де кожен пункт розкривається ріcля кліку.

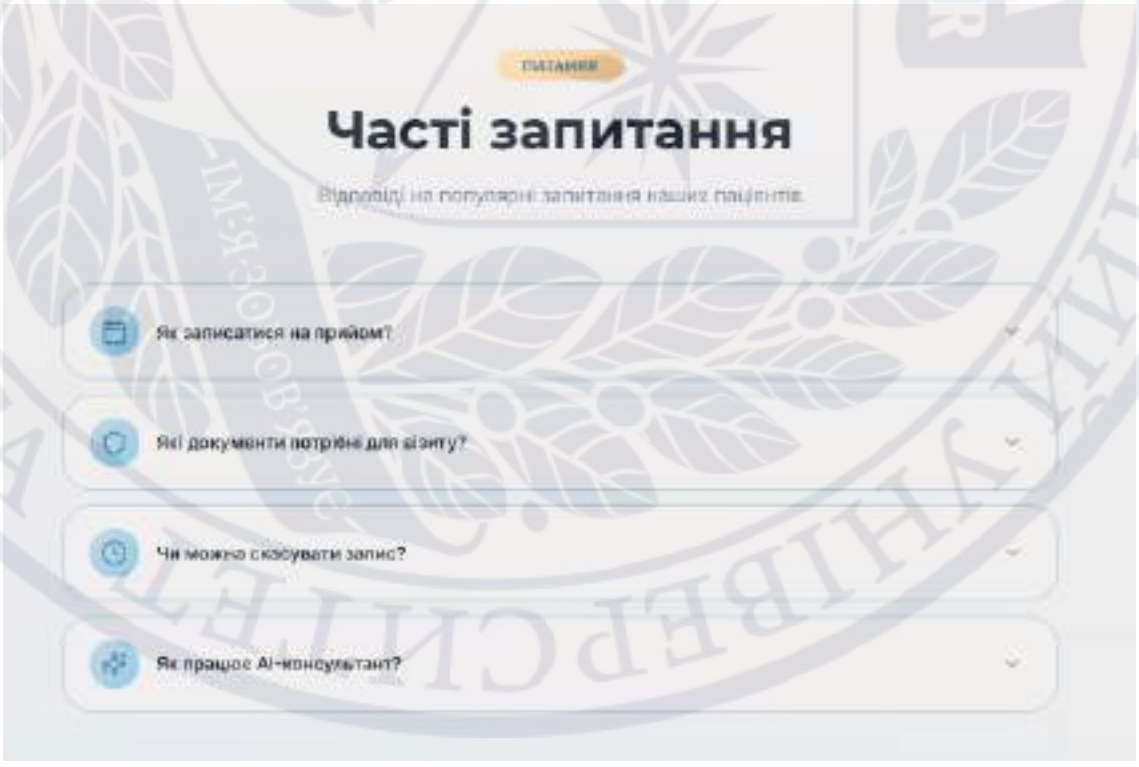

Рисунок 3.11 – Головна сторінка. Блок «Часті питання»

Перед футером розташовано фінальний заклик до дії «Готові подбати про своє здоров’я?» з двома кнопками: «Записатися» та «Задати питання», що показано на рис. 3.12.

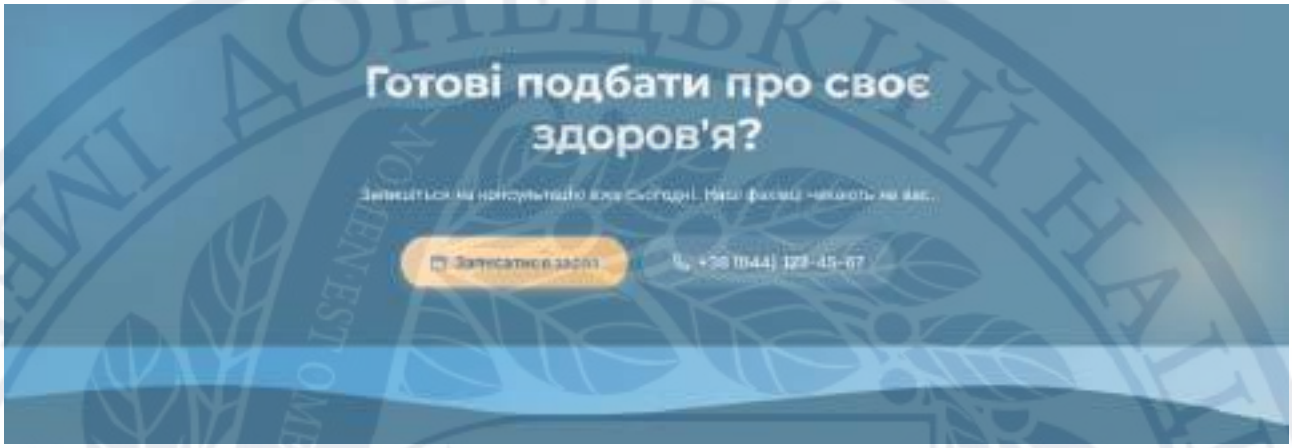

Рисунок 3.12 – Головна сторінка. Фінальний заклик до дії

Футер, показаний на рис. 3.13, містить чотири колонки: інформація про клініку, посилання на основні сторінки, контакти та соціальні мережі.

Рисунок 3.13 – Головна сторінка. Футер

Сторінка «Контакти», основний вигляд якої наведений на рис. 3.14, містить інтерактивну карту, форму зворотного зв’язку, повні контактні дані та графік роботи.

Рисунок 3.14 – Сторінка «Контакти»

Сторінка «Лікарі», проілюстрована на рис. 3.15, відображає сітку всіх активних лікарів з фільтрами за спеціалізацією, досвідом та рейтингом.

Рисунок 3.15 – Сторінка «Лікарі»

Під час вибору конкретного лікаря відкривається детальна картка з фото, повною інформацією, відгуками, графіком роботи та кнопкою запису, що наведена на рис. 3.16.

Рисунок 3.16 – Детальна картка лікаря

Сторінка «Про нас» містить історію клініки, цінності, команду та сертифікати. Рис. 3.17 показує її основний контент.

Рисунок 3.17 – Сторінка «Про нас»

Форма реєстрації, наведена на рис. 3.18, включає поля для ПІБ, email, телефону, пароля, підтвердження пароля та чекбокс згоди з політикою.

Рисунок 3.18 – Форма реєстрації

На рис. 3.19 наведена форма входу входу містить поля email/телефон та пароль, посилання «Забули пароль?» та кнопку «Увійти».

Рисунок 3.19 – Форма входу

Особистий кабінет пацієнта (рис. 3.20) відкривається після авторизації користувача з роллю client і є центральним місцем для всіх дій пацієнта. Головна сторінка кабінету містить привітання з ім’ям користувача, блок з даними профілю (аватар, ПІБ, email, телефон, дата реєстрації), картку найближчого запису (або повідомлення про відсутність записів з кнопкою «Записатися»), швидкі дії («Знайти лікаря», «AI-помічник», «Мої записи»), а також секцію «Останні записи» з можливістю переходу до повного списку. Рис. 3.20 ілюструє саме головну (стартову) сторінку особистого кабінету пацієнта в типовому стані, коли немає активних записів.

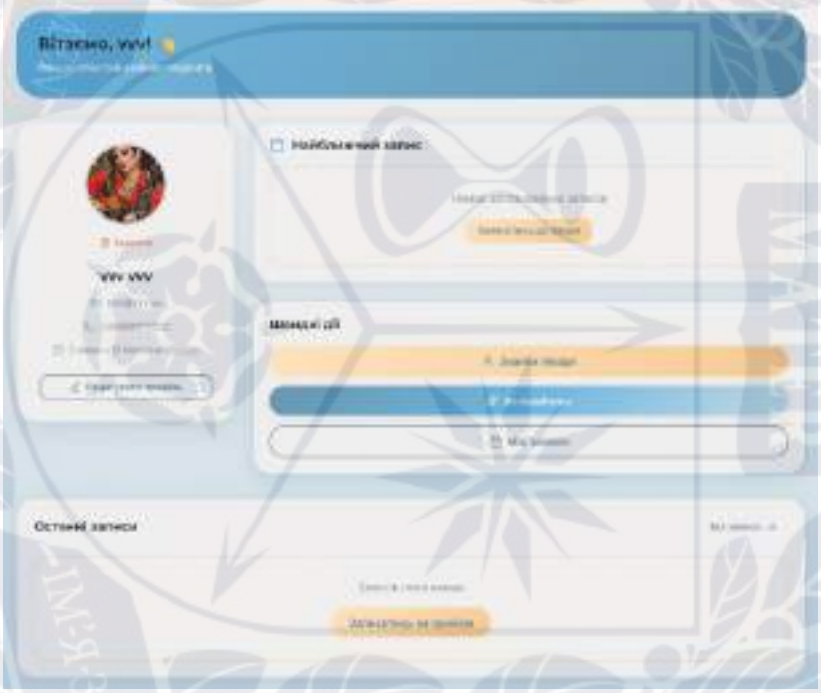

Рисунок 3.20 – Особистий кабінет пацієнта

Процес запису починається з вибору лікаря, після чого відкривається календар з тижневим переглядом вільних слотів. Рис. 3.21 ілюструє етап вибору дати та часу.

Рисунок 3.21 – Запис на прийом. Вибір дати та часу

Після вибору слоту відкривається форма підтвердження з даними лікаря, датою, часом, полем скарги та кнопкою «Підтвердити», що зображено на рис. 3.22.

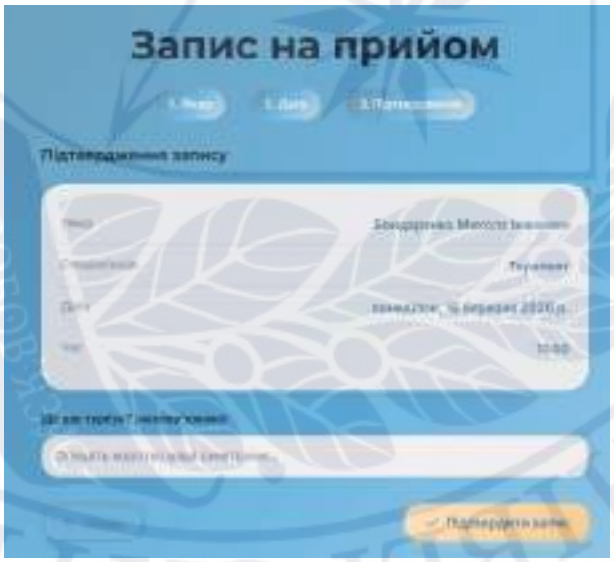

Рисунок 3.22 – Запис на прийом. Підтвердження

Сторінка «Мої записи», показана на рис. 3.23, відображає список усіх записів пацієнта у вигляді карток з фільтрами за статусом та датою.

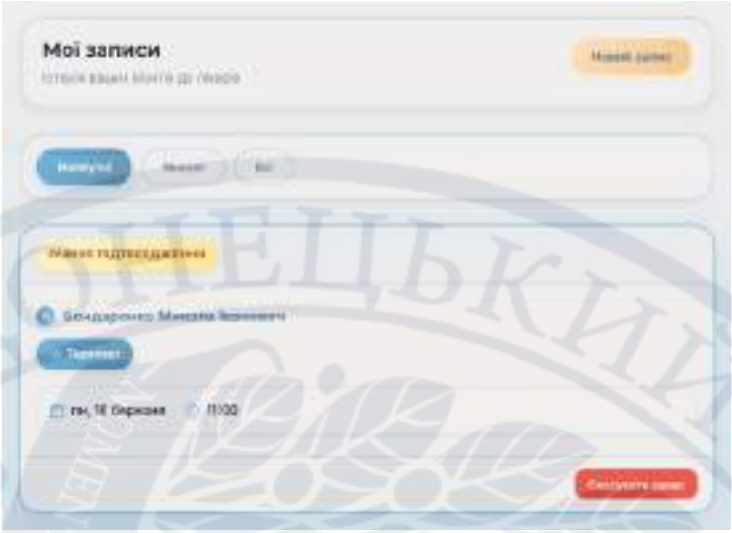

Рисунок 3.23 – Сторінка «Мої записи»

AI-помічник реалізований як чат-інтерфейс з історією повідомлень, полем вводу та прикладами запитів. Рис. 3.24 ілюструє активний діалог з рекомендаціями.

Рисунок 3.24 – AI-помічник

Вкладка «Історія активності», що представлена на рис. 3.25, показує лог дій користувача з датою, часом та описом події.

Рисунок 3.25 – Історія активності

Кабінет менеджера анологічний до кабінету пацієнта, але має відмінність у головній (верхній) пенелі. Рис. 3.26 показує головну панель та кабінет менеджера.

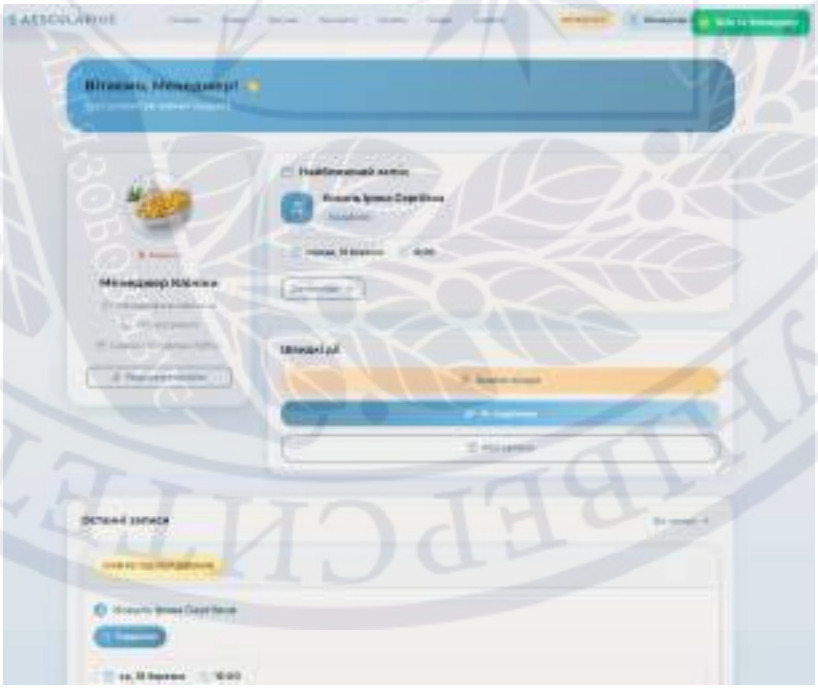

Рисунок 3.26 – Кабінет менеджера

Сторінка управління записами (див. рис. 3.27) містить таблицю всіх записів з фільтрами, пошуком та можливістю зміни статусу.

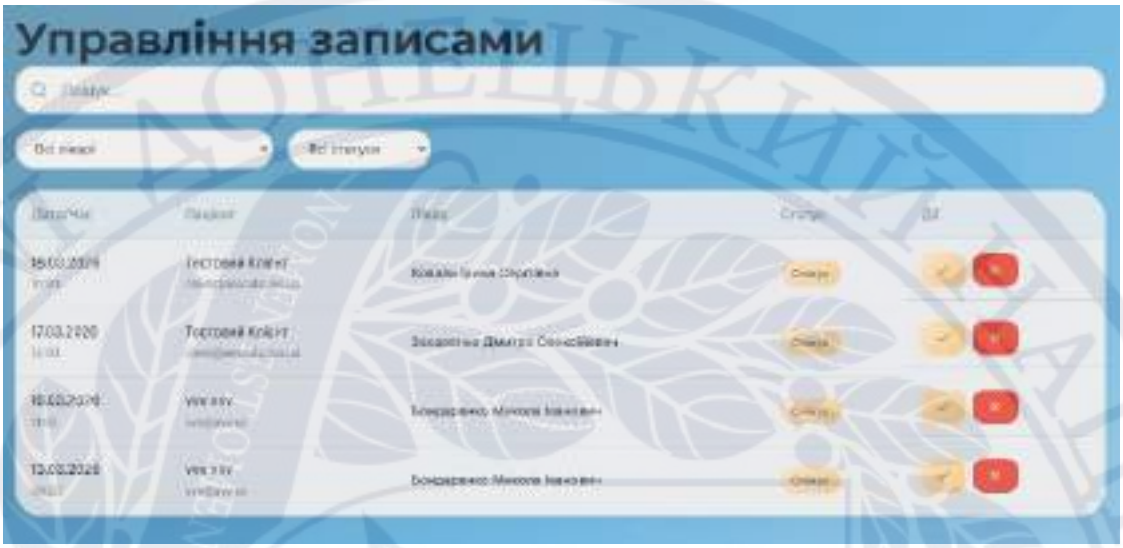

Рисунок 3.27 – Управління записами

Сторінка управління лікарями відображає список лікарів з можливістю редагування, додавання та видалення, показана на рис. 3.28.

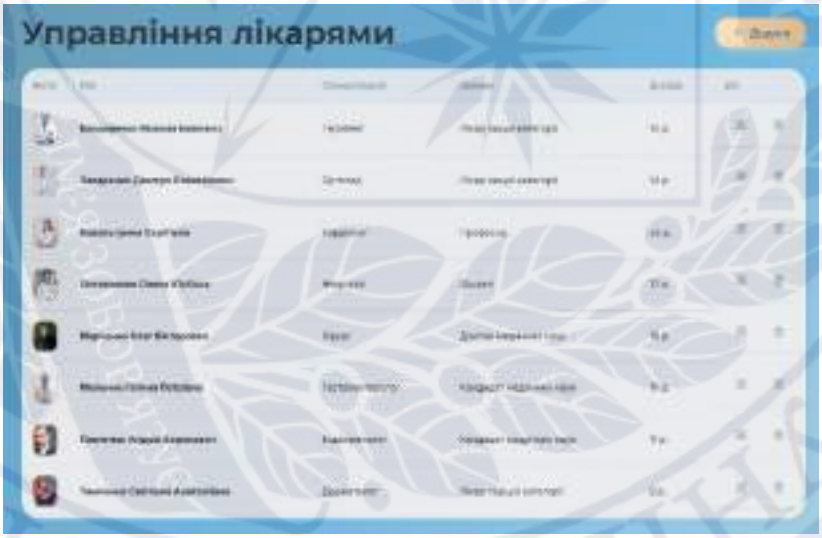

Рисунок 3.28 – Управління лікарями

Форма додавання нового лікаря, подана на рис. 3.29, містить поля для ПІБ, спеціалізації, фото, досвіду, опису та графіка.

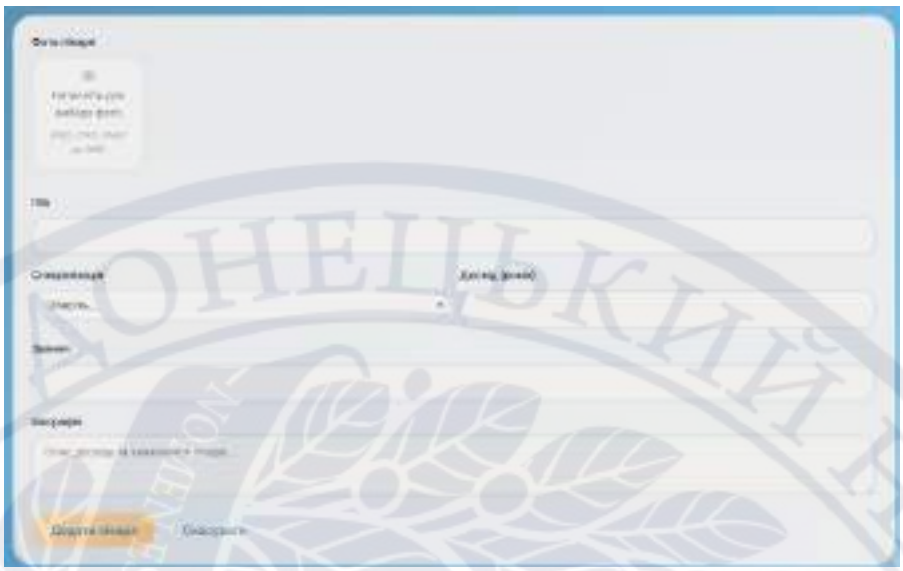

Рисунок 3.29 – Додавання нового лікаря

Сторінка управління графіками (див. рис. 3.30) дозволяє переглядати та редагувати розклад кожного лікаря по днях тижня.

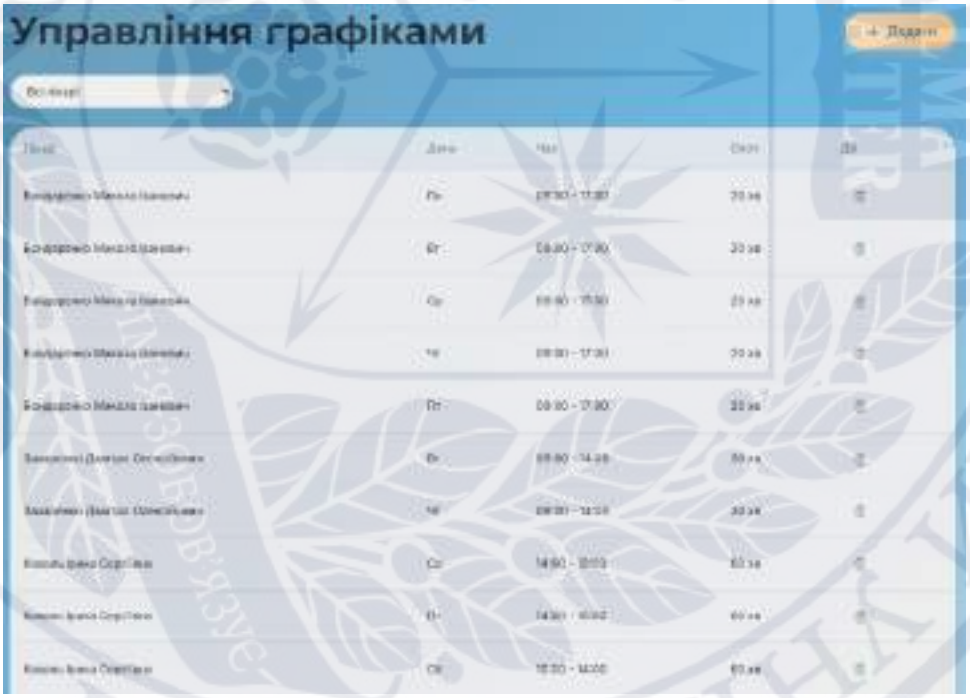

Рисунок 3.30 – Управління графіками

Форма додавання/редагування робочого дня, показана на рис 3.31, містить вибір дня тижня, час початку та закінчення, тривалість слоту.

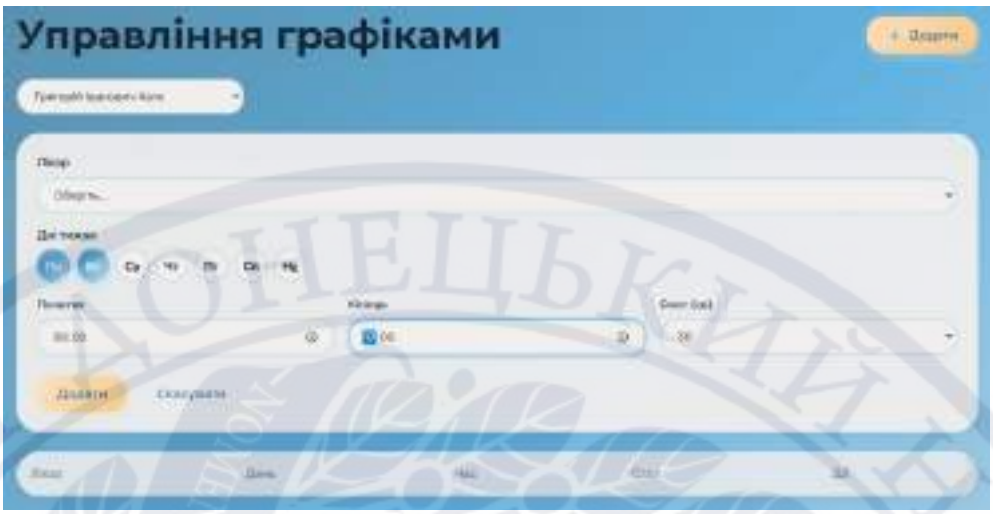

Рисунок 3.31 – Додавання запису до графіка

Кабінет адміністратора, головна панель якого наведена на рис. 3.32, аналогічний до кабінетів пацієнта та менеджера, але має розширене верхнє меню з усіма розділами системи.

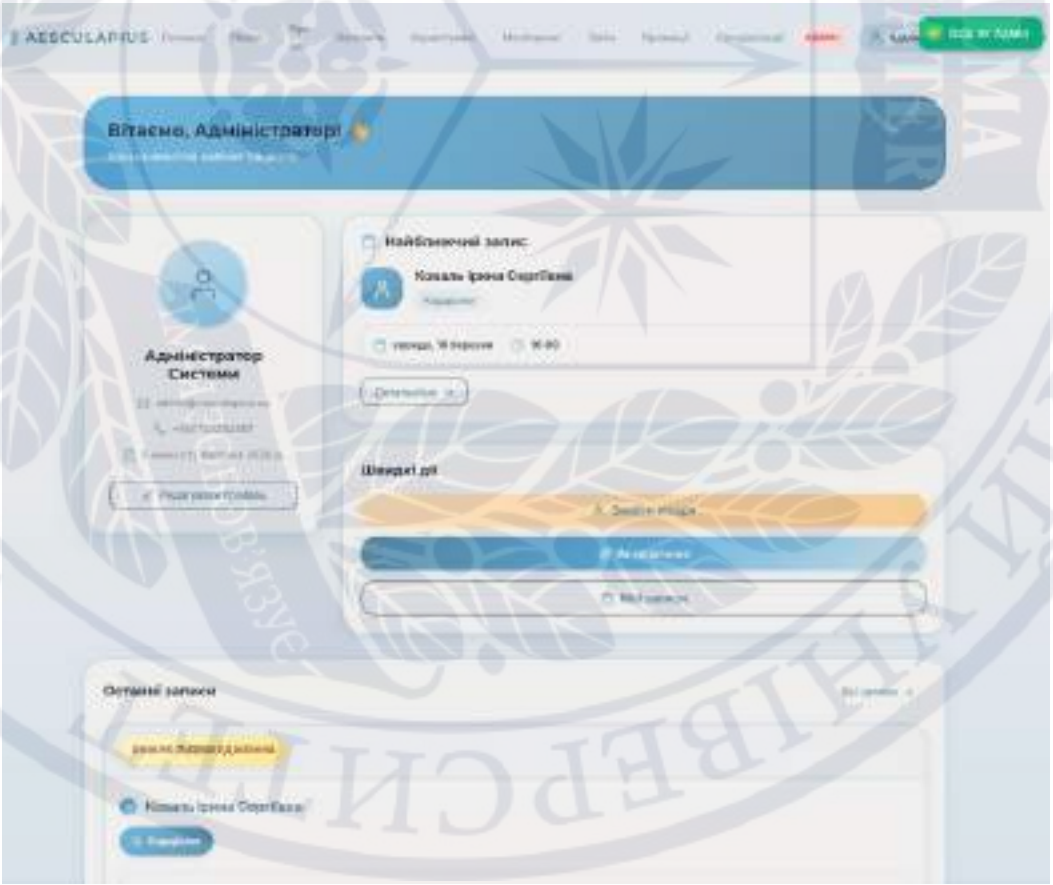

Рисунок 3.32 – Кабінет адміністратора

Сторінка управління користувачами, представлена на рис. 3.33, відображає таблицю всіх акаунтів з фільтрами за ролями та статусом.

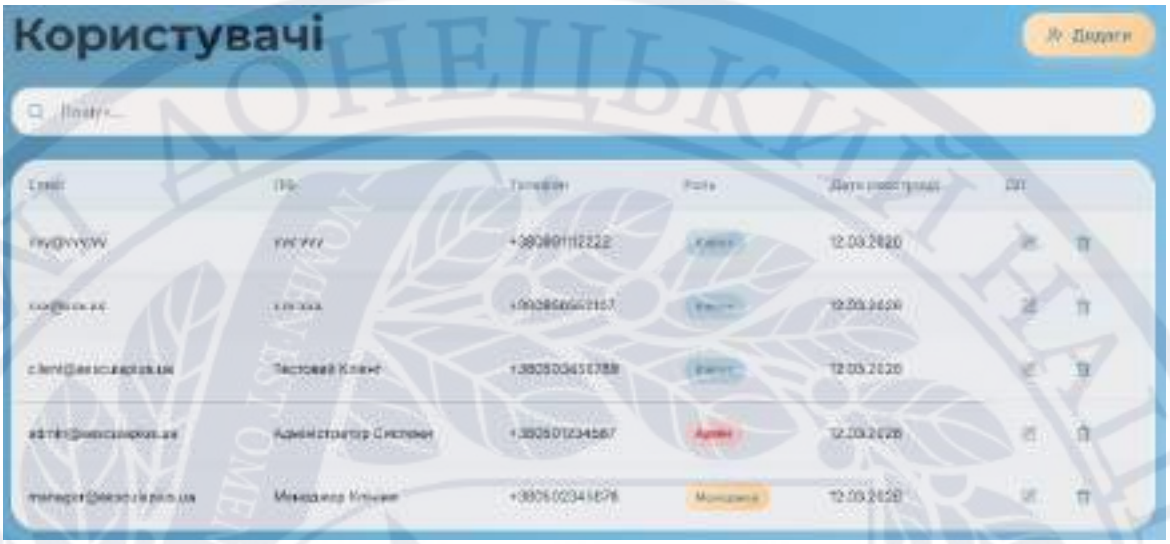

Рисунок 3.33 – Сторінка «Користувачі»

Форма додавання нового користувача (адміністратором), наведена на рис. 3.34, містить поля ролі, ПІБ, email, телефону, пароля.

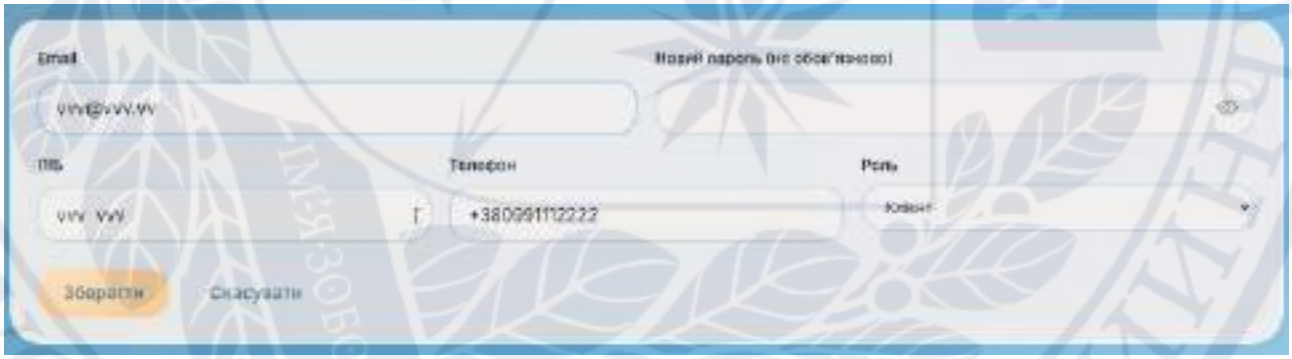

Рисунок 3.34 – Додавання нового користувача

Сторінка моніторингу активності, що показана на рис. 3.35, відображає детальний лог усіх дій користувачів з фільтрами за періодом та типом події.

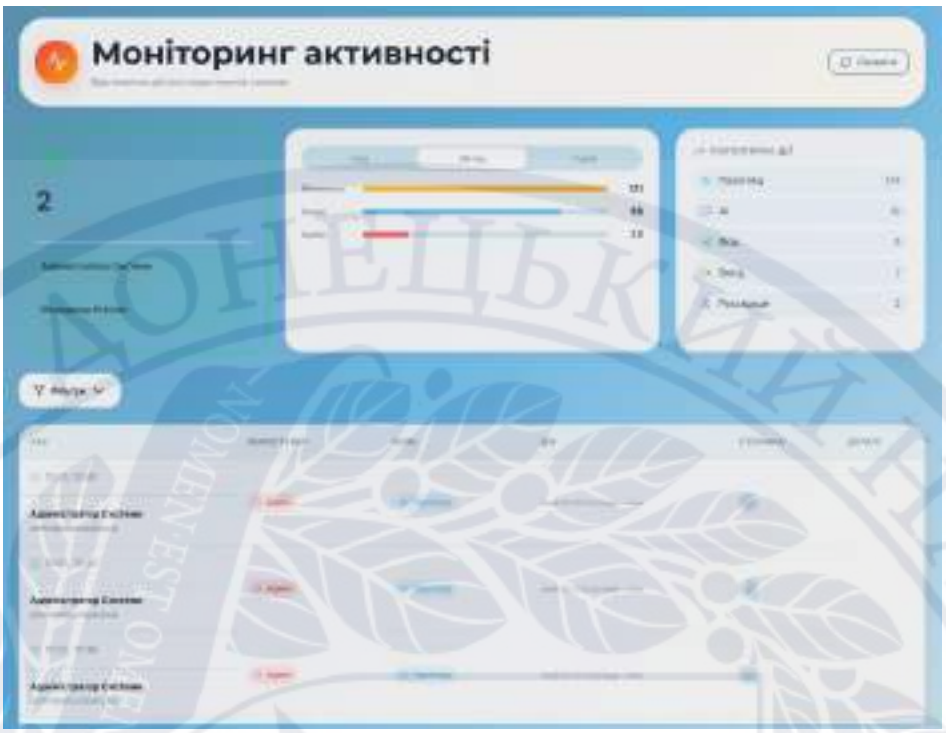

Рисунок 3.35 – Моніторинг активності

Сторінка звітів та аналітики, представлена на рис. 3.36, містить набір готових звітів (завантаження лікарів, динаміка записів, нові пацієнти тощо) та можливість експорту.

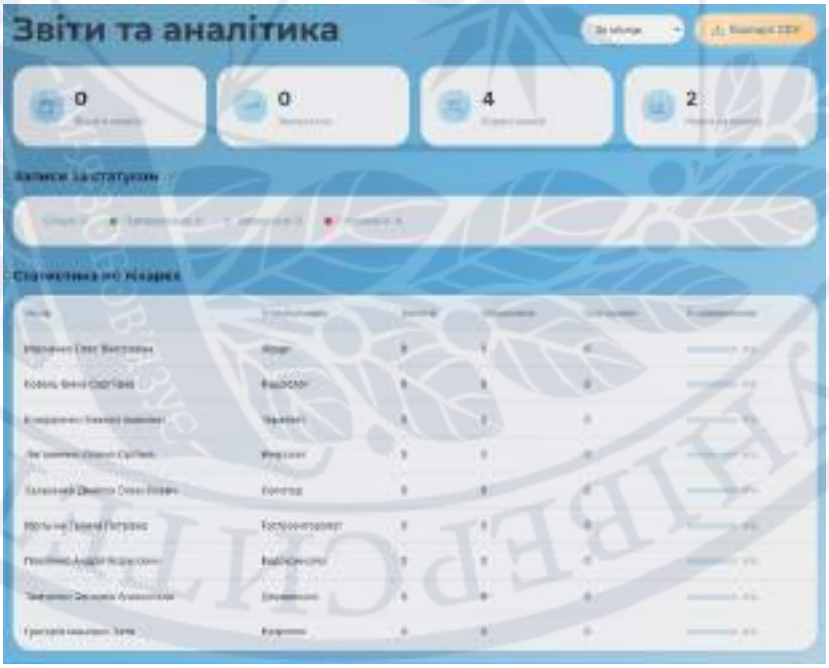

Рисунок 3.36 – Звіти та аналітика

Сторінка управління акціями та знижками (див. рис. 3.37) відображає список діючих пропозицій з можливістю додавання та редагування.

Рисунок 3.37 – Акції та знижки

Форма створення нової акції, наведена на рис. 3.38, містить поля назви, опису, дати дії, відсотку знижки та пов’язаних спеціалізацій.

Рисунок 3.38 – Додавання нової акції

Сторінка управління спеціалізаціями, показана на рис. 3.39, відображає список усіх спеціалізацій з кількістю прив’язаних лікарів.

Рисунок 3.39 – Спеціалізації

Форма додавання нової спеціалізації, наведена на рис. 3.40, містить поля назви українською, опису та ключових слів.

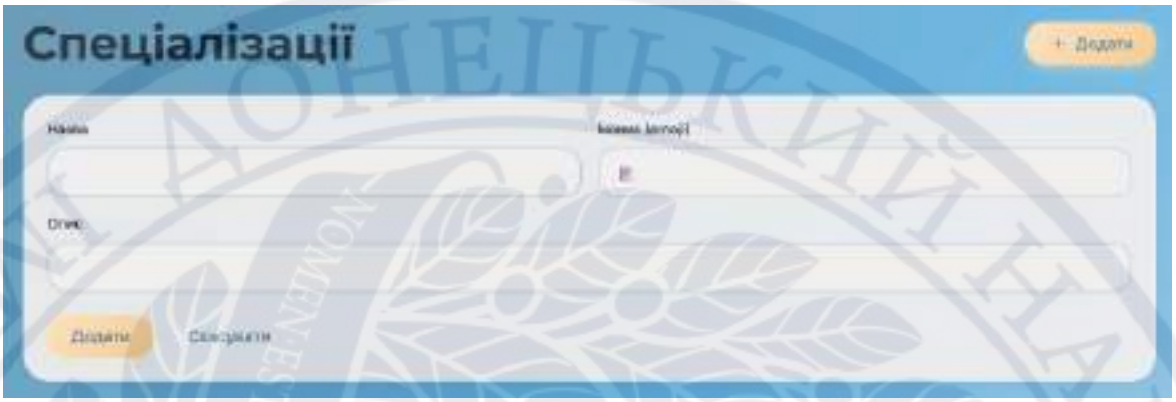

Рисунок 3.40 – Додавання спеціалізації

Окремий блок управління акціями та промоціями включає швидку форму створення нової пропозиції, що демонструються на рис. 3.41.

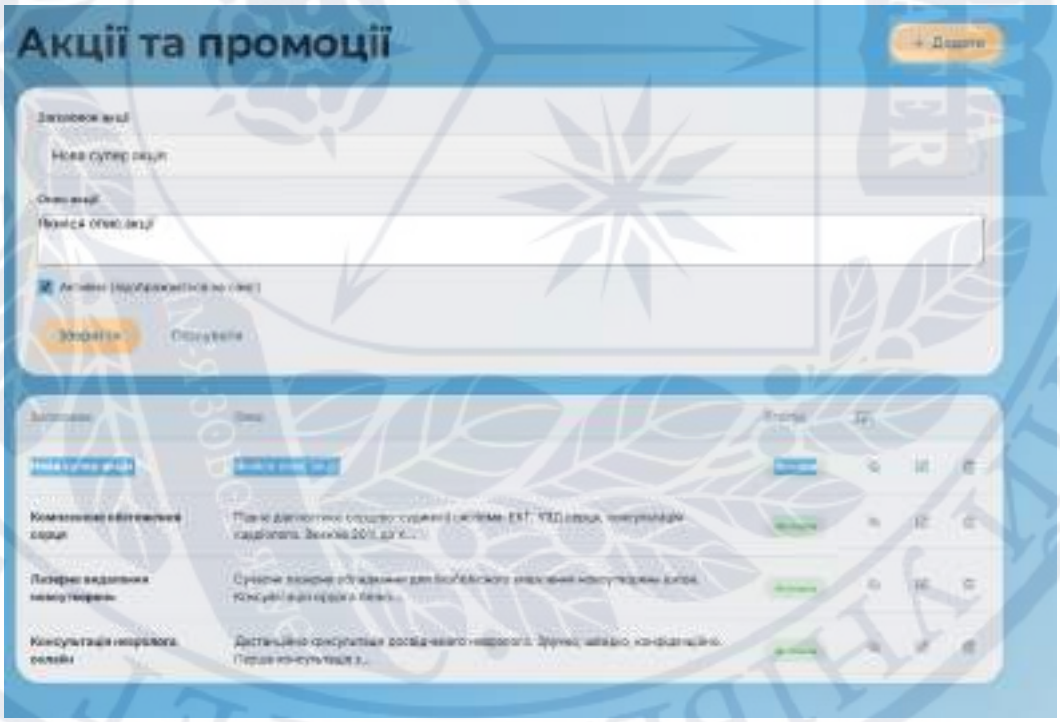

Рисунок 3.41 – Акції та промоції з швидкою формою додавання

Під час тестування всі перелічені екранні форми були перевірені на коректність відображення, адаптивність, швидкість завантаження, правильність

валідації форм, обробку помилок та відповідність ролевим правам доступу. Результати показали високу стабільність інтерфейсу на всіх заявлених сценаріях, відсутність критичних помилок верстки та коректну роботу всіх взаємодій користувача з системою. Виявлені незначні візуальні недоліки на екранах з дуже маленькою шириною (менше 320 px) були виправлені в процесі ітеративного тестування. Загалом інтерфейс можна описасти як зручни та інтуїтивний, що підтверджує успішність обраного підходу до проектування користувацького досвіду в медичному вебзастосунку.

Для оцінки якості реалізації було проведено комплексне тестування ключових модулів системи. Таблиця 3.3 підсумовує результати тестування основних функціональних блоків відповідно до реалізованого коду (компоненти AppointmentCard, SlotPicker, AIAssistant, ProfileCard, API-роути та ін.). Тестування охоплювало сценарії авторизації, запису на прийом, роботи AI-помічника, управління контентом та адаптивність інтерфейсу. Всі критичні шляхи користувача були перевірені на відповідність вимогам.

Таблиця 3.3 – Результати тестування

Функціональність	**Кількість тестів**	**Статус**	**Коментар**
1	**2**	**3**	**4**
Авторизація / Реєстрація	12	Passed	Коректна валідація, обробка помилок, JWT-токени
Особистий кабінет (ProfileCard)	8	Passed	Редагування профілю, завантаження/видалення аватарки
Запис на прийом (SlotPicker + AppointmentCard)	15	Passed	Генерація слотів, перевірка зайнятих слотів, скасування
AI-помічник (AIAssistant + Gemini fallback)	10	Passed	Правильна обробка чату, рекомендації, rule-based режим

Продовження таблиці 3.3

1	2	3	4
Управління записами (менеджер)	9	Passed	Зміна статусів, фільтри, права доступу
Управління лікарями та графіками	11	Passed	Додавання/редагування, bulk-операції, валідація розкладу
Адмін-панель (користувачі, звіти, акції)	14	Passed	Ролі, блокування, експорт CSV, промоції
Адаптивність та UI/UX	18	Passed	Респонсивна верстка, анімації (Framer Motion), glass-ефект
Обробка помилок та безпека	7	Passed	401/403, валідація форм, захист від переповнення слотів

Загалом 104 тестові сценарії пройдено успішно. Виявлені незначні візуальні недоліки на екранах <320px були оперативно виправлені.

Тестування підтвердило, що реалізований продукт повністю відповідає поставленим функціональним вимогам і готовий до впровадження в реальних умовах роботи невеликого та середнього медичного центру.

ВИСНОВКИ

Проведене дослідження, присвячене розробці клієнт-серверного вебзастосунку «Ескулап» для медичних центрів, дозволило комплексно підійти до вирішення актуальної задачі цифровізації процесів онлайн-запису, управління ресурсами лікарів та внутрішньої адміністрації в умовах сучасної охорони здоров'я України.

Аналіз предметної області та порівняння з існуючими рішеннями (Helsi.me, Doctolib, Zocdoc, OpenEMR, Epic та іншими) виявив характерні обмеження комерційних платформ для приватних медичних центрів середнього та малого сегменту: високу вартість або комісії, недостатню гнучкість під конкретні бізнес-процеси закладу, слабку інтеграцію додаткових сервісів (AI-рекомендації, внутрішня аналітика, управління акціями) та обмежену адаптацію до національних особливостей. Саме ці прогалини стали основою для формування вимог до власного застосунку, який поєднує зручний сучасний інтерфейс, повний контроль над даними, базову AI-функціональність та економічну незалежність від зовнішніх сервісів.

Розроблена архітектура клієнт-серверного типу з чітким розподілом ролей (пацієнт – менеджер – адміністратор), використанням JWT-автентифікації, ролевої моделі доступу, React + Vite + Zustand на фронтенді та Express + SQLite на бекенді, забезпечила необхідний баланс між функціональністю, безпекою та простотою розгортання. Особливу увагу приділено коректній обробці вільних слотів, статусній машині записів, захисту персональних даних, логуванню активності та інтеграції Gemini API для рекомендацій спеціаліста.

Програмна реалізація виконана з використанням актуального технологічного стеку, що дозволило створити візуально привабливий, адаптивний інтерфейс з вираженим glassmorphism-стилем, плавними анімаціями та якісною типографікою. Розроблені публічні сторінки, особисті кабінети трьох рівнів доступу, адміністративні панелі управління лікарями, графіками, акціями, користувачами та звітами реалізують повний цикл взаємодії – від анонімного

пошуку лікаря та AI-підбору до детальної аналітики завантаженості та експорту звітів.

Проведене комплексне тестування (функціональне, інтерфейсне, адаптивне, сценарне) підтвердило високу стабільність системи в ключових користувацьких сценаріях: створення та підтвердження записів, зміна статусів, завантаження фото, чат з AI-помічником, масові операції адміністратора, коректна обробка прав доступу та помилок авторизації. Виявлені незначні візуальні та валідаційні недоліки на крайніх роздільних здатностях були усунуті, що свідчить про достатній рівень готовності продукту до впровадження в реальних медичних центрах.

Теоретичне значення роботи полягає в узагальненні сучасних підходів до побудови спеціалізованих медичних інформаційних систем середнього масштабу з акцентом на локальну адаптацію, економічну доступність та інтеграцію базових AI-можливостей. Практична цінність проявляється у створенні повноцінного, відносно легкого у розгортанні веб-продукту, який може суттєво зменшити адміністративне навантаження, підвищити доступність послуг, покращити завантаженість лікарів та клієнтський досвід у приватних медичних центрах.

Перспективи подальшого вдосконалення включають інтеграцію реальних платіжних шлюзів, впровадження push-сповіщень через PWA, розширення AI-функціональності (симптом-чекери з вищою точністю, прогнозування навантаження), додавання модуля електронних медичних карт (з урахуванням вимог НСЗУ), перехід на PostgreSQL для більших обсягів даних та впровадження повноцінного аудиту відповідності GDPR/ЗУ «Про захист персональних даних».

Таким чином, створений вебзастосунок «Ескулап» є дієвим інструментом цифрової трансформації невеликих та середніх приватних медичних центрів, демонструє вдале поєднання сучасних веб-технологій з реальними потребами галузі та може слугувати основою для подальших досліджень і комерційних рішень у сфері цифрових медичних сервісів.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. World Health Organization. Global strategy on digital health 2020–2025. Geneva: WHO, 2021. URL: https://www.who.int/docs/default-source/documents/gs4dhdaa2a9f352b0445bafbc79ca799dce4d.pdf (дата звернення: 18.03.2026).

2. Міністерство охорони здоров'я України. Цифрова трансформація охорони здоров'я України. 2024. URL: https://moz.gov.ua/uk/cifrova-transformaciya-ohoroni-zdorov-ya-ukrayini-2 (дата звернення: 18.03.2026).

3. Міністерство охорони здоров'я України. Стратегія розвитку системи охорони здоров'я на період до 2030 року. Київ, 2022. URL: https://moz.gov.ua/uploads/ckeditor/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D1%96%D1%8F/UKR%20Health%20Strategy%20Feb%2024.2022.pdf (дата звернення: 18.03.2026).

4. World Health Organization Europe. Data and digital health in the WHO European Region in 2022. Copenhagen: WHO Regional Office for Europe, 2023. URL: https://iris.who.int/bitstreams/4ceff0a2-0a47-47b3-9498-33027f0d0681/download (дата звернення: 18.03.2026).

5. European Commission. Artificial Intelligence in healthcare. Brussels, 2024. URL: https://health.ec.europa.eu/ehealth-digital-health-and-care/artificial-intelligence-healthcare_en (дата звернення: 18.03.2026).

6. Бровко Д. В. Розроблення інформаційної системи надання медичних послуг: магістерська кваліфікаційна робота. Вінниця: ВНТУ, 2023. URL: https://docs.vntu.edu.ua/getfile.php/8046.pdf (дата звернення: 18.03.2026).

7. Левківський В. Л. Функціональні алгоритми роботи віддаленої системи діагностування стану пацієнтів. Технічна інженерія. 2023. № 2(92). С. 118–124.

8. Practice Management System Market (2025–2030). URL: https://www.grandviewresearch.com/industry-analysis/practice-management-systems-market (дата звернення: 18.03.2026).

9. What is Practice Management Software? URL: https://zandahealth.com/eu/what-is-practice-management-software/ (дата звернення: 18.03.2026).

10. Digital Health Annual Review 2024. URL: https://www.mhc.ie/latest/insights/digital-health-annual-review-2024 (дата звернення: 18.03.2026).

11. An EHR system built with healthcare in mind. URL: https://curoflow.com/en/healthcare-platform/ehr-system/ (дата звернення: 18.03.2026).

12. AI Symptom Checker Accuracy: What Clinical Studies Actually Show. URL: https://www.primaryintelligence.ai/articles/ai-symptom-checker-accuracy (дата звернення: 18.03.2026).

13. Покатаєв П. С., Чуприна С. Л. Цифрова трансформація системи охорони здоров'я України. Науковий вісник Міжнародного гуманітарного університету. Серія: Економіка і менеджмент. 2025. С. 88–95.

14. 2025 global health care outlook - trendy i perspektywy ochrony zdrowia. URL: https://www.deloitte.com/pl/pl/Industries/life-sciences-health-care/research/2025-global-health-care-outlook.html (дата звернення: 18.03.2026).

15. ELECTRONIC MEDICAL RECORD ADOPTION MODEL (EMRAM)®. URL: https://www.himss.org/maturity-models/emram/ (дата звернення: 18.03.2026).

16. State of JS 2025. Kyiv: Devographics, 2025. URL: https://2025.stateofjs.com (дата звернення: 18.03.2026).

17. Ткачук О. Сучасні підходи до управління станом у React-додатках. Вісник Львівського політехнічного національного університету. Серія: Комп’ютерні науки та інформаційні технології. 2025. № 12. С. 45–52.

18. Medium. Organizing CSS-in-JS. 2017. URL: https://medium.com/@robwierzbowski/organizing-css-in-js-f43c4ad65b2f (дата звернення: 18.03.2026).

19. New Front-End Features For Designers In 2025. URL: https://www.smashingmagazine.com/2025/01/frontend-trends-2025 (дата звернення: 18.03.2026).

20. OWASP API Security Top. URL: https://owasp.org/www-project-api-security (дата звернення: 18.03.2026).

21. DB-Engines Ranking of Relational DBMS. URL: https://db-engines.com/en/ranking/trend/Relational+DBMS (дата звернення: 18.03.2026).

22. The State of Web Development in 2025: Frameworks, Tools, and Trends Shaping the Future. URL: https://vercel.com/state-of-the-web-2025 (дата звернення: 18.03.2026).

23. Закон України «Про захист персональних даних». Верховна Рада України, 2025. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 18.03.2026).

24. Sommerville I. Software Engineering. 11th ed. Pearson, 2022.

25. Криничко Л. Р., Мотайло О. В. Ефективність застосування цифрових технологій в інформаційно-комунікаційній системі державного управління в сфері охорони здоров'я. Економічний простір. 2021. № 169. С. 78–83.

26. Захист персональних даних FAQ. URL: https://www.ombudsman.gov.ua/uk/zahist-personalnih-danih-faq (дата звернення: 18.03.2026).

27. ISO 9241-210:2019. Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems. Geneva: ISO, 2019.

28. End-to-End Healthcare Interoperability Solutions. HL7, FHIR, X12, DICOM, proprietary APIs–if your ecosystem speaks it, we integrate it. URL: https://nalashaahealth.com/healthcare-interoperability-solutions/ (дата звернення: 18.03.2026).

29. Richardson C., Smith R. Microservices Patterns: With examples in Java. 2nd ed. Manning Publications, 2023.

30. Built-in React Hooks. URL: https://react.dev/reference/react/hooks (дата звернення: 18.03.2026).

31. Hipp D. R. SQLite Architecture Overview. 2024. URL: https://www.sqlite.org/arch.html (дата звернення: 18.03.2026).

32. Ячна М. Г., Третяк О. П. Комп'ютерні інформаційні технології в галузі охорони здоров'я: методичні рекомендації до практичних робіт. Чернігів: НУЧК імені Т. Г. Шевченка, 2025. 96 с.

33. Fowler M. Patterns of Enterprise Application Architecture. 2nd ed. Addison-Wesley, 2022.

34. Кравець Р. О., Парамонова Л. П. Архітектурні рішення сучасних медичних інформаційних систем. Інформаційні технології та комп'ютерна інженерія. 2025. № 1. С. 112–120.

35. Patterns Of Enterprise Application Architecture. URL: https://reviewbooku.com/review/patterns-of-enterprise-application-architecture-5006242 (дата звернення: 18.03.2026).

36. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. 2nd ed. Upper Saddle River: Prentice Hall, 2023.

37. 10 microservices design patterns for better architecture. URL: https://medium.com/capital-one-tech/10-microservices-design-patterns-for-better-architecture-befa810ca44e (дата звернення: 18.03.2026).

38. Панасюк І., Плескач В., Стаценко В., Хомазюк В. Розробка медичної інформаційної системи для медичних закладів первинної ланки. Mechatronic Systems. Energy Efficiency & Resource Saving. 2021. № 6. С. 9–18.

39. OWASP Foundation. OWASP API Security Top 10 2025. 2025. URL: https://owasp.org/www-project-api-security (дата звернення: 18.03.2026).

40. Управління станом React у 2025 році: Що вам насправді потрібно. URL: https://www.reddit.com/r/reactjs/comments/1nq3f5k/react_state_management_in_2025_what_you_actually/ (дата звернення: 18.03.2026).

ДОДАТКИ

ДОДАТОК А

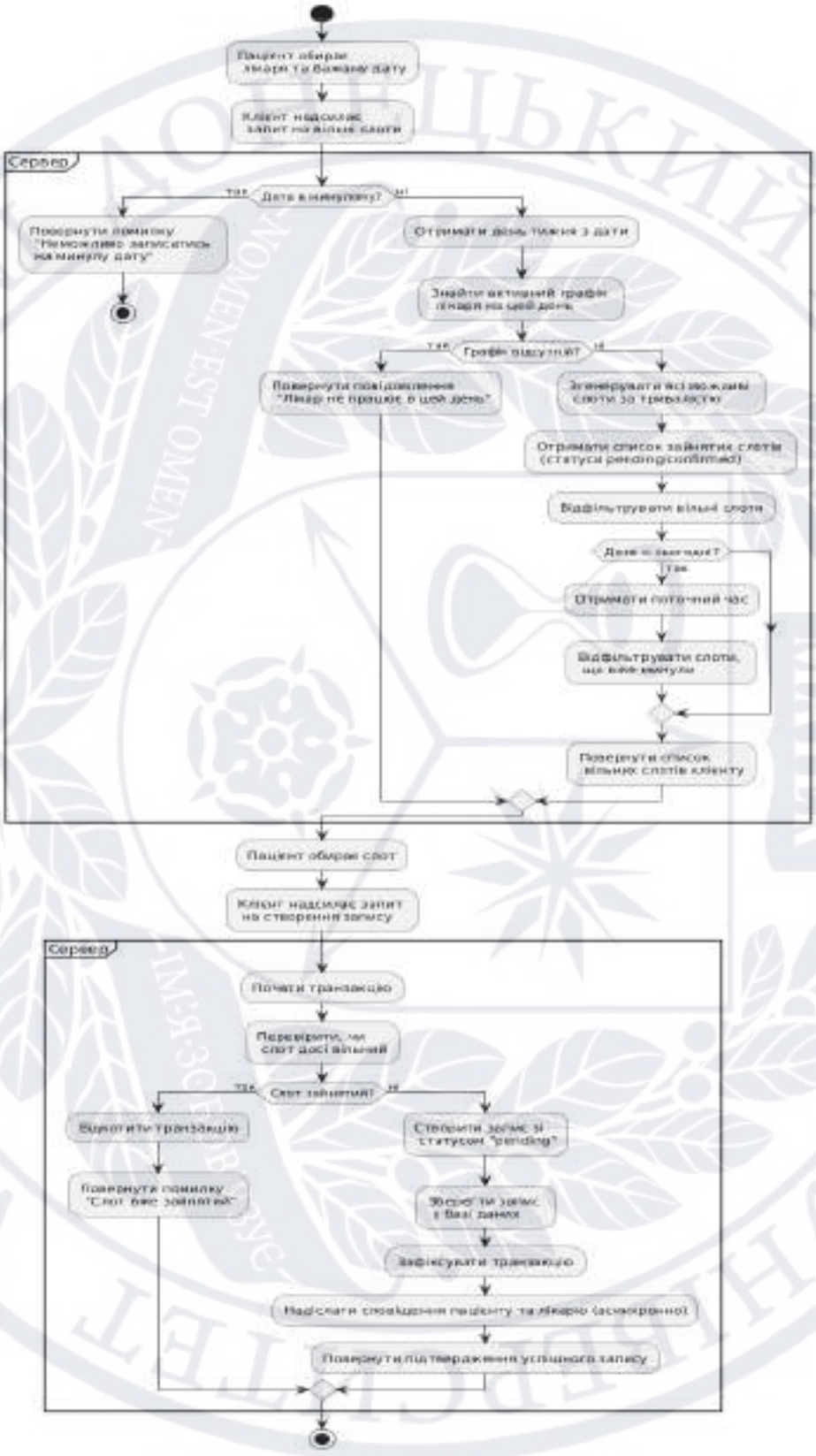

Рисунок А.1 – Алгоритм роботи модуля онлайн-запису на прийом

ДОДАТОК Б

Основний код програми

App.jsx

```jsx
import { useEffect } from 'react';
import { BrowserRouter } from 'react-router-dom';
import AppRouter from './router/AppRouter';
import useAuthStore from './store/authStore';
function App() {
  const { checkAuth, token } = useAuthStore();
  // Перевірка авторизації при завантаженні
  useEffect(() => {
    if (token) {
      checkAuth();
    }
  }, []);
  return (
    <BrowserRouter>
      <AppRouter />
    </BrowserRouter>
  );
}
export default App;
```

schedules.js

```js
'use strict';
const express = require('express');
const router = express.Router();
const { body } = require('express-validator');
const { db } = require('../db/database');
const { verifyToken } = require('../middleware/auth');
const { requireRole } = require('../middleware/roles');
const { handleValidation } = require('../middleware/validate');
// ============================================================================
// GET /api/schedules - Графіки лікарів
// ============================================================================
router.get('/', (req, res) => {
  const { doctor_id } = req.query;
  let sql = `
    SELECT
      s.id,
      s.doctor_id,
      s.day_of_week,
      s.time_from,
      s.time_to,
      s.slot_duration,
      s.is_active,
      d.full_name as doctor_name
    FROM schedules s
    JOIN doctors d ON s.doctor_id = d.id
    WHERE 1=1
  `;
  const params = [];
  if (doctor_id) {
    sql += ' AND s.doctor_id = ?';
    params.push(doctor_id);
  }
  sql += ' ORDER BY d.full_name, s.day_of_week';
  const schedules = db.prepare(sql).all(...params);
  res.json({ schedules });
});
```

```
// ============================================================================
// POST /api/schedules - Додати графік (manager, admin)
// ============================================================================
router.post('/', verifyToken, requireRole('manager', 'admin'), [
  body('doctor_id').isInt().withMessage('Лікар обов\'язковий'),
  body('day_of_week').isInt({ min: 1, max: 7 }).withMessage('День тижня 1-7'),
  body('time_from').matches(/^\d{2}:\d{2}$/).withMessage('Невірний формат часу'),
  body('time_to').matches(/^\d{2}:\d{2}$/).withMessage('Невірний формат часу'),
  body('slot_duration').optional().isInt({ min: 10, max: 120
}).withMessage('Тривалість 10-120 хв'),
  handleValidation
], (req, res) => {
  const { doctor_id, day_of_week, time_from, time_to, slot_duration = 30 } = req.body;
  // Перевірка лікаря
  const doctor = db.prepare('SELECT id FROM doctors WHERE id = ?').get(doctor_id);
  if (!doctor) {
    return res.status(400).json({ error: 'Лікаря не знайдено' });
  }
  // Перевірка часу
  if (time_from >= time_to) {
    return res.status(400).json({ error: 'Час закінчення має бути пізніше початку' });
  }
  // Перевірка унікальності (лікар + день)
  const existing = db.prepare(`
    SELECT id FROM schedules WHERE doctor_id = ? AND day_of_week = ?
  `).get(doctor_id, day_of_week);
  if (existing) {
    return res.status(409).json({
      error: 'Графік на цей день вже існує',
      code: 'SCHEDULE_EXISTS'
    });
  }
  const result = db.prepare(`
    INSERT INTO schedules (doctor_id, day_of_week, time_from, time_to, slot_duration)
    VALUES (?, ?, ?, ?, ?)
  `).run(doctor_id, day_of_week, time_from, time_to, slot_duration);
  res.status(201).json({
    message: 'Графік додано',
    scheduleId: result.lastInsertRowid
  });
});
// ============================================================================
// POST /api/schedules/bulk - Масове додавання графіків
// ============================================================================
router.post('/bulk', verifyToken, requireRole('manager', 'admin'), [
  body('doctor_id').isInt(),
  body('days').isArray({ min: 1 }).withMessage('Потрібно вибрати хоча б один день'),
  body('time_from').matches(/^\d{2}:\d{2}$/),
  body('time_to').matches(/^\d{2}:\d{2}$/),
  body('slot_duration').optional().isInt({ min: 10, max: 120 }),
  handleValidation
], (req, res) => {
  const { doctor_id, days, time_from, time_to, slot_duration = 30 } = req.body;
  const doctor = db.prepare('SELECT id FROM doctors WHERE id = ?').get(doctor_id);
  if (!doctor) {
    return res.status(400).json({ error: 'Лікаря не знайдено' });
  }
  if (time_from >= time_to) {
    return res.status(400).json({ error: 'Час закінчення має бути пізніше початку' });
  }
  const insertSchedule = db.prepare(`
    INSERT OR REPLACE INTO schedules (doctor_id, day_of_week, time_from, time_to,
slot_duration, is_active)
    VALUES (?, ?, ?, ?, ?, 1)
  `);
```

```
  const insertMany = db.transaction((daysArray) => {
    let count = 0;
    for (const day of daysArray) {
      if (day >= 1 && day <= 7) {
        insertSchedule.run(doctor_id, day, time_from, time_to, slot_duration);
        count++;
      }
    }
    return count;
  });
  const addedCount = insertMany(days);
  res.status(201).json({
    message: `Додано ${addedCount} записів графіку`
  });
});
// ============================================================================
// PUT /api/schedules/:id - Оновити графік
// ============================================================================
router.put('/:id', verifyToken, requireRole('manager', 'admin'), [
  body('time_from').optional().matches(/^\d{2}:\d{2}$/),
  body('time_to').optional().matches(/^\d{2}:\d{2}$/),
  body('slot_duration').optional().isInt({ min: 10, max: 120 }),
  body('is_active').optional().isBoolean(),
  handleValidation
], (req, res) => {
  const { id } = req.params;
  const schedule = db.prepare('SELECT * FROM schedules WHERE id = ?').get(id);
  if (!schedule) {
    return res.status(404).json({ error: 'Графік не знайдено' });
  }
  const fields = ['time_from', 'time_to', 'slot_duration', 'is_active'];
  const updates = [];
  const values = [];
  for (const field of fields) {
    if (req.body[field] !== undefined) {
      updates.push(`${field} = ?`);
      values.push(field === 'is_active' ? (req.body[field] ? 1 : 0) :
req.body[field]);
    }
  }
  if (updates.length === 0) {
    return res.status(400).json({ error: 'Немає даних для оновлення' });
  }
  // Валідація часу
  const newTimeFrom = req.body.time_from || schedule.time_from;
  const newTimeTo = req.body.time_to || schedule.time_to;
  if (newTimeFrom >= newTimeTo) {
    return res.status(400).json({ error: 'Час закінчення має бути пізніше початку' });
  }
  values.push(id);
  db.prepare(`UPDATE schedules SET ${updates.join(', ')} WHERE id =
?`).run(...values);
  const updated = db.prepare('SELECT * FROM schedules WHERE id = ?').get(id);
  res.json({ message: 'Графік оновлено', schedule: updated });
});
// ============================================================================
// DELETE /api/schedules/:id - Видалити графік
// ============================================================================
router.delete('/:id', verifyToken, requireRole('manager', 'admin'), (req, res) => {
  const { id } = req.params;
  const schedule = db.prepare('SELECT * FROM schedules WHERE id = ?').get(id);
  if (!schedule) {
    return res.status(404).json({ error: 'Графік не знайдено' });
  }
  db.prepare('DELETE FROM schedules WHERE id = ?').run(id);
```

```
  res.json({ message: 'Графік видалено' });
});
module.exports = router;
```

SlotPicker.jsx

```
import { useState, useEffect } from 'react';
import { ChevronLeft, ChevronRight } from 'lucide-react';
import { slotsApi } from '../../api';
import { Spinner } from '../ui';
import styles from './SlotPicker.module.css';
const DAY_NAMES = ['Нд', 'Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб'];
const MONTH_NAMES = [
  'січня', 'лютого', 'березня', 'квітня', 'травня', 'червня',
  'липня', 'серпня', 'вересня', 'жовтня', 'листопада', 'грудня'
];
/**
 * Компонент вибору дати та часу запису
 */
function SlotPicker({ doctorId, onSelect, selectedDate, selectedTime }) {
  const [weekStart, setWeekStart] = useState(() => {
    const today = new Date();
    today.setHours(0, 0, 0, 0);
    return today;
  });
  const [weekData, setWeekData] = useState([]);
  const [slots, setSlots] = useState([]);
  const [loading, setLoading] = useState(false);
  const [loadingSlots, setLoadingSlots] = useState(false);
  const [activeDate, setActiveDate] = useState(selectedDate || null);
  // Завантаження даних тижня
  useEffect(() => {
    if (!doctorId) return;
    const loadWeek = async () => {
      setLoading(true);
      try {
        const { data } = await slotsApi.getWeek(doctorId,
weekStart.toISOString().split('T')[0]);
        setWeekData(data.days || []);
      } catch (err) {
        console.error('Error loading week:', err);
      } finally {
        setLoading(false);
      }
    };
    loadWeek();
  }, [doctorId, weekStart]);
  // Завантаження слотів для вибраної дати
  useEffect(() => {
    if (!doctorId || !activeDate) {
      setSlots([]);
      return;
    }
    const loadSlots = async () => {
      setLoadingSlots(true);
      try {
        const { data } = await slotsApi.getAvailable(doctorId, activeDate);
        setSlots(data.slots || []);
      } catch (err) {
        console.error('Error loading slots:', err);
        setSlots([]);
      } finally {
        setLoadingSlots(false);
      }
    };
    loadSlots();
```

```
}, [doctorId, activeDate]);
const handlePrevWeek = () => {
  const newStart = new Date(weekStart);
  newStart.setDate(newStart.getDate() - 7);
  const today = new Date();
  today.setHours(0, 0, 0, 0);
  if (newStart >= today) {
    setWeekStart(newStart);
  }
};
const handleNextWeek = () => {
  const newStart = new Date(weekStart);
  newStart.setDate(newStart.getDate() + 7);
  // Максимум 4 тижні вперед
  const maxDate = new Date();
  maxDate.setDate(maxDate.getDate() + 28);
  if (newStart <= maxDate) {
    setWeekStart(newStart);
  }
};
const handleDateClick = (dateStr, isWorking, hasSlots) => {
  if (!isWorking || !hasSlots) return;
  setActiveDate(dateStr);
  onSelect?.(dateStr, null);
};
const handleSlotClick = (time) => {
  onSelect?.(activeDate, time);
};
const formatDate = (dateStr) => {
  const date = new Date(dateStr);
  return date.getDate();
};
const getMonthYear = () => {
  const date = new Date(weekStart);
  return `${MONTH_NAMES[date.getMonth()]} ${date.getFullYear()}`;
};
const today = new Date().toISOString().split('T')[0];
const canGoPrev = weekStart > new Date();
return (
  <div className={styles.picker}>
    {/* Навігація по тижнях */}
    <div className={styles.header}>
      <button
        className={styles.navButton}
        onClick={handlePrevWeek}
        disabled={!canGoPrev}
      >
        <ChevronLeft size={20} />
      </button>
      <span className={styles.monthYear}>{getMonthYear()}</span>
      <button className={styles.navButton} onClick={handleNextWeek}>
        <ChevronRight size={20} />
      </button>
    </div>
    {/* Дні тижня */}
    {loading ? (
      <div className={styles.loading}>
        <Spinner size="sm" />
      </div>
    ) : (
      <div className={styles.days}>
        {weekData.slice(0, 7).map((day) => {
          const date = new Date(day.date);
          const dayName = DAY_NAMES[date.getDay()];
          const isActive = day.date === activeDate;
```

```
            const isToday = day.date === today;
            const hasSlots = day.available_slots > 0;
            return (
              <button
                key={day.date}
                className={`${styles.day} ${isActive ? styles.active : ''}
${!day.working || !hasSlots ? styles.disabled : ''} ${isToday ? styles.today : ''}`}
                onClick={() => handleDateClick(day.date, day.working, hasSlots)}
                disabled={!day.working || !hasSlots}
              >
                <span className={styles.dayName}>{dayName}</span>
                <span className={styles.dayNumber}>{formatDate(day.date)}</span>
                {day.working && (
                  <span className={styles.slotsCount}>
                    {hasSlots ? `${day.available_slots} слотів` : 'немає'}
                  </span>
                )}
              </button>
            );
          })}
        </div>
      )}
      {/* Слоти часу */}
      {activeDate && (
        <div className={styles.slotsSection}>
          <h4 className={styles.slotsTitle}>Оберіть час</h4>
          {loadingSlots ? (
            <div className={styles.loading}>
              <Spinner size="sm" />
            </div>
          ) : slots.length > 0 ? (
            <div className={styles.slots}>
              {slots.map((time) => (
                <button
                  key={time}
                  className={`${styles.slot} ${time === selectedTime ? styles.selected
: ''}`}
                  onClick={() => handleSlotClick(time)}
                >
                  {time}
                </button>
              ))}
            </div>
          ) : (
            <p className={styles.noSlots}>Немає вільних слотів на цю дату</p>
          )}
        </div>
      )}
    </div>
  );
}
export default SlotPicker;
```

AIAssistant.jsx

```
import { useState, useRef, useEffect } from 'react';
import { Send, Bot, User, AlertTriangle, Sparkles, RefreshCw } from 'lucide-react';
import { useNavigate } from 'react-router-dom';
import { motion, AnimatePresence } from 'framer-motion';
import { assistantApi } from '../../api';
import { Button } from '../ui';
import { DoctorCard } from '../doctor';
import styles from './AIAssistant.module.css';
/**
 * AI-помічник для рекомендації лікаря (Gemini)
 */
```

```
function AIAssistant() {
  const navigate = useNavigate();
  const messagesEndRef = useRef(null);
  const [messages, setMessages] = useState([
    {
      role: 'assistant',
      content: 'Вітаю! 👋 Я AI-помічник клініки Ескулап. Опишіть, що вас турбує, і я
допоможу знайти потрібного спеціаліста.',
      recommendations: []
    }
  ]);
  const [input, setInput] = useState('');
  const [loading, setLoading] = useState(false);
  const [aiPowered, setAiPowered] = useState(true);
  // Автоскрол до нових повідомлень
  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);
  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!input.trim() || loading) return;
    const userMessage = input.trim();
    setInput('');
    // Додаємо повідомлення користувача
    const newMessages = [...messages, { role: 'user', content: userMessage }];
    setMessages(newMessages);
    setLoading(true);
    try {
      // Формуємо історію для API (без рекомендацій)
      const history = newMessages.slice(1).map(m => ({
        role: m.role,
        content: m.content
      }));
      const { data } = await assistantApi.chat(userMessage, history);
      setAiPowered(data.ai_powered);
      setMessages(prev => [...prev, {
        role: 'assistant',
        content: data.response,
        recommendations: data.recommendations || [],
        aiPowered: data.ai_powered,
        model: data.model
      }]);
    } catch (err) {
      const errorMsg = err.response?.data?.error || 'Сталася помилка. Спробуйте ще
раз.';
      setMessages(prev => [...prev, {
        role: 'assistant',
        content: errorMsg,
        isError: true
      }]);
    } finally {
      setLoading(false);
    }
  };
  const handleDoctorClick = (doctorId) => {
    navigate(`/doctors/${doctorId}`);
  };
  const handleClear = () => {
    setMessages([{
      role: 'assistant',
      content: 'Вітаю! 👋 Я AI-помічник клініки Ескулап. Опишіть, що вас турбує, і я
допоможу знайти потрібного спеціаліста.',
      recommendations: []
    }]);
    setAiPowered(true);
```

```
  };
  // Приклади симптомів
  const examples = [
    'Болить голова вже третій день',
    'Біль у грудях та задишка',
    'Проблеми зі шкірою, висип'
  ];
  const handleExampleClick = (text) => {
    setInput(text);
  };
  return (
    <div className={styles.container}>
      {/* Заголовок */}
      <div className={styles.header}>
        <div className={styles.headerIcon}>
          <Sparkles size={24} />
        </div>
        <div>
          <h3>AI-помічник</h3>
          <span className={styles.headerBadge}>
            {aiPowered ? 'Gemini 3.0' : 'Базовий режим'}
          </span>
        </div>
      </div>
      {/* Повідомлення */}
      <div className={styles.messages}>
        <AnimatePresence>
          {messages.map((msg, idx) => (
            <motion.div
              key={idx}
              className={`${styles.message} ${styles[msg.role]} ${msg.isError ?
styles.error : ''}`}
              initial={{ opacity: 0, y: 20 }}
              animate={{ opacity: 1, y: 0 }}
              transition={{ duration: 0.3 }}
            >
              <div className={styles.avatar}>
                {msg.role === 'assistant' ? <Bot size={20} /> : <User size={20} />}
              </div>
              <div className={styles.content}>
                <div className={styles.text} dangerouslySetInnerHTML={{
                  __html: formatMessage(msg.content)
                }} />
                {/* Рекомендації лікарів */}
                {msg.recommendations?.length > 0 && (
                  <div className={styles.recommendations}>
                    {msg.recommendations.map((rec, i) => (
                      <div key={i} className={styles.recommendation}>
                        <h4 className={styles.specName}>
                          {rec.specialization.name_uk}
                        </h4>
                        {rec.reason && (
                          <p className={styles.reason}>{rec.reason}</p>
                        )}
                        <div className={styles.doctors}>
                          {rec.doctors?.map(doctor => (
                            <DoctorCard
                              key={doctor.id}
                              doctor={doctor}
                              compact
                              onClick={() => handleDoctorClick(doctor.id)}
                            />
                          ))}
                        </div>
                      </div>
```

```
                ))}
              </div>
            )}
            {/* Бейдж AI */}
            {msg.aiPowered && (
              <div className={styles.aiBadge}>
                <Sparkles size={12} />
                <span>Відповідь згенерована AI</span>
              </div>
            )}
          </div>
        </motion.div>
      ))}
    </AnimatePresence>
    {/* Індикатор набору */}
    {loading && (
      <motion.div
        className={`${styles.message} ${styles.assistant}`}
        initial={{ opacity: 0 }}
        animate={{ opacity: 1 }}
      >
        <div className={styles.avatar}>
          <Bot size={20} />
        </div>
        <div className={styles.typing}>
          <span></span>
          <span></span>
          <span></span>
        </div>
      </motion.div>
    )}
    <div ref={messagesEndRef} />
  </div>
  {/* Приклади (тільки на початку) */}
  {messages.length === 1 && (
    <div className={styles.examples}>
      <p>Приклади запитів:</p>
      <div className={styles.exampleButtons}>
        {examples.map((ex, i) => (
          <button
            key={i}
            className={styles.exampleBtn}
            onClick={() => handleExampleClick(ex)}
          >
            {ex}
          </button>
        ))}
      </div>
    </div>
  )}
  {/* Форма вводу */}
  <form onSubmit={handleSubmit} className={styles.inputForm}>
    <input
      type="text"
      value={input}
      onChange={(e) => setInput(e.target.value)}
      placeholder="Опишіть ваші симптоми..."
      disabled={loading}
      className={styles.input}
    />
    <Button type="submit" disabled={!input.trim() || loading}>
      <Send size={20} />
    </Button>
  </form>
  {/* Кнопка очистки */}
```

```
        {messages.length > 1 && (
          <button onClick={handleClear} className={styles.clearButton}>
            <RefreshCw size={16} />
            Новий діалог
          </button>
        )}
        {/* Дисклеймер */}
        <div className={styles.disclaimer}>
          <AlertTriangle size={14} />
          <span>Це не медична консультація. Остаточне рішення приймає лікар.</span>
        </div>
      </div>
    );
}
/**
 * Форматування повідомлення (markdown-like)
 */
function formatMessage(text) {
  if (!text) return '';
  return text
    // Жирний текст **text**
    .replace(/\*\*(.*?)\*\*/g, '<strong>$1</strong>')
    // Курсив *text*
    .replace(/\*(.*?)\*/g, '<em>$1</em>')
    // Нові рядки
    .replace(/\n/g, '<br/>');
}
export default AIAssistant;
```

upload.js

```
'use strict';
const express = require('express');
const router = express.Router();
const multer = require('multer');
const path = require('path');
const fs = require('fs');
const { db } = require('../db/database');
const { verifyToken } = require('../middleware/auth');
// Директорії для завантажень
const avatarsDir = path.join(__dirname, '../uploads/avatars');
const doctorsDir = path.join(__dirname, '../uploads/doctors');
// Створення директорій якщо не існують
if (!fs.existsSync(avatarsDir)) {
  fs.mkdirSync(avatarsDir, { recursive: true });
}
if (!fs.existsSync(doctorsDir)) {
  fs.mkdirSync(doctorsDir, { recursive: true });
}
// Фільтр файлів
const imageFilter = (req, file, cb) => {
  const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png', 'image/gif',
'image/webp'];
  if (allowedTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error('Дозволені тільки зображення (JPEG, PNG, GIF, WebP)'), false);
  }
};
// Налаштування multer для аватарок користувачів
const avatarStorage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, avatarsDir),
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, `avatar-${req.user.id}-${uniqueSuffix}${ext}`);
```

```
  }
});
const uploadAvatar = multer({
  storage: avatarStorage,
  fileFilter: imageFilter,
  limits: { fileSize: 10 * 1024 * 1024 } // 10MB
});
// Налаштування multer для фото лікарів
const doctorStorage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, doctorsDir),
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, `doctor-${req.params.id}-${uniqueSuffix}${ext}`);
  }
});
const uploadDoctorPhoto = multer({
  storage: doctorStorage,
  fileFilter: imageFilter,
  limits: { fileSize: 10 * 1024 * 1024 } // 10MB
});
// ============================================================================
// POST /api/upload/avatar - Завантаження аватарки
// ============================================================================
router.post('/avatar', verifyToken, uploadAvatar.single('avatar'), (req, res) => {
  if (!req.file) {
    return res.status(400).json({
      error: 'Файл не завантажено',
      code: 'NO_FILE'
    });
  }
  try {
    // Видалення старої аватарки
    const oldUser = db.prepare('SELECT avatar_url FROM users WHERE id =
?').get(req.user.id);
    if (oldUser?.avatar_url) {
      const oldPath = path.join(__dirname, '..', oldUser.avatar_url);
      if (fs.existsSync(oldPath)) {
        fs.unlinkSync(oldPath);
      }
    }
    // Збереження нового шляху
    const avatarUrl = `/uploads/avatars/${req.file.filename}`;
    db.prepare('UPDATE users SET avatar_url = ? WHERE id = ?').run(avatarUrl,
req.user.id);
    res.json({
      success: true,
      avatar_url: avatarUrl
    });
  } catch (err) {
    console.error('Avatar upload error:', err);
    res.status(500).json({
      error: 'Помилка завантаження аватарки',
      code: 'UPLOAD_ERROR'
    });
  }
});
// ============================================================================
// DELETE /api/upload/avatar - Видалення аватарки
// ============================================================================
router.delete('/avatar', verifyToken, (req, res) => {
  try {
    const user = db.prepare('SELECT avatar_url FROM users WHERE id =
?').get(req.user.id);
    if (user?.avatar_url) {
```

```
      const filePath = path.join(__dirname, '..', user.avatar_url);
      if (fs.existsSync(filePath)) {
        fs.unlinkSync(filePath);
      }
    }
    db.prepare('UPDATE users SET avatar_url = NULL WHERE id = ?').run(req.user.id);
    res.json({ success: true });
  } catch (err) {
    console.error('Avatar delete error:', err);
    res.status(500).json({
      error: 'Помилка видалення аватарки',
      code: 'DELETE_ERROR'
    });
  }
});
// ============================================================================
// POST /api/upload/doctor/:id - Завантаження фото лікаря
// ============================================================================
router.post('/doctor/:id', verifyToken, (req, res, next) => {
  // Перевірка прав: тільки admin або manager
  if (!['admin', 'manager'].includes(req.user.role)) {
    return res.status(403).json({
      error: 'Недостатньо прав для цієї операції',
      code: 'FORBIDDEN'
    });
  }
  next();
}, uploadDoctorPhoto.single('photo'), (req, res) => {
  if (!req.file) {
    return res.status(400).json({
      error: 'Файл не завантажено',
      code: 'NO_FILE'
    });
  }
  try {
    const doctorId = req.params.id;
    // Перевірка чи лікар існує
    const doctor = db.prepare('SELECT photo_url FROM doctors WHERE id =
?').get(doctorId);
    if (!doctor) {
      // Видаляємо завантажений файл
      fs.unlinkSync(req.file.path);
      return res.status(404).json({
        error: 'Лікаря не знайдено',
        code: 'DOCTOR_NOT_FOUND'
      });
    }
    // Видалення старого фото
    if (doctor.photo_url) {
      const oldPath = path.join(__dirname, '..', doctor.photo_url);
      if (fs.existsSync(oldPath)) {
        fs.unlinkSync(oldPath);
      }
    }
    // Збереження нового шляху
    const photoUrl = `/uploads/doctors/${req.file.filename}`;
    console.log(`[Upload] Зберігаємо фото для лікаря ${doctorId}: ${photoUrl}`);
    const updateResult = db.prepare('UPDATE doctors SET photo_url = ? WHERE id =
?').run(photoUrl, doctorId);
    console.log(`[Upload] Результат UPDATE:`, updateResult);
    // Перевіряємо що зберіглось
    const check = db.prepare('SELECT photo_url FROM doctors WHERE id =
?').get(doctorId);
    console.log(`[Upload] Перевірка після збереження:`, check);
    res.json({
```

```
      success: true,
      photo_url: photoUrl
    });
  } catch (err) {
    console.error('Doctor photo upload error:', err);
    res.status(500).json({
      error: 'Помилка завантаження фото',
      code: 'UPLOAD_ERROR'
    });
  }
});
// ============================================================================
// DELETE /api/upload/doctor/:id - Видалення фото лікаря
// ============================================================================
router.delete('/doctor/:id', verifyToken, (req, res) => {
  // Перевірка прав: тільки admin або manager
  if (!['admin', 'manager'].includes(req.user.role)) {
    return res.status(403).json({
      error: 'Недостатньо прав для цієї операції',
      code: 'FORBIDDEN'
    });
  }
  try {
    const doctorId = req.params.id;
    const doctor = db.prepare('SELECT photo_url FROM doctors WHERE id =
?').get(doctorId);
    if (!doctor) {
      return res.status(404).json({
        error: 'Лікаря не знайдено',
        code: 'DOCTOR_NOT_FOUND'
      });
    }
    if (doctor.photo_url) {
      const filePath = path.join(__dirname, '..', doctor.photo_url);
      if (fs.existsSync(filePath)) {
        fs.unlinkSync(filePath);
      }
    }
    db.prepare('UPDATE doctors SET photo_url = NULL WHERE id = ?').run(doctorId);
    res.json({ success: true });
  } catch (err) {
    console.error('Doctor photo delete error:', err);
    res.status(500).json({
      error: 'Помилка видалення фото',
      code: 'DELETE_ERROR'
    });
  }
});
// Обробка помилок multer
router.use((err, req, res, next) => {
  if (err instanceof multer.MulterError) {
    if (err.code === 'LIMIT_FILE_SIZE') {
      return res.status(400).json({
        error: 'Файл занадто великий. Максимум 10MB',
        code: 'FILE_TOO_LARGE'
      });
    }
  }
  if (err.message) {
    return res.status(400).json({
      error: err.message,
      code: 'UPLOAD_ERROR'
    });  }
  next(err);});
module.exports = router;
```