

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«CRM-система для управління роботи поліклініки»

Виконала студентка групи ПЗ-22-3
ЯЦЮК Ліна Вячеславівна

Керівник:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ЧИЖМОТРА Олена Геннадіївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЯЦЮК Ліна Вячеславівна**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«CRM-система для управління роботи поліклініки»**, затверджена наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с.**

Вихідні дані для роботи: Об'єктом дослідження є процес побудови вебдодатку прикладного спрямування. Предметом дослідження є моделі, алгоритми, технології та засоби розробки вебдодатку для управління роботи поліклініки.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	02.03.2026р.	02.03.2026р.
2	Лобанчикова Н.М.	03.04.2026р.	03.04.2026р.
3	Лобанчикова Н.М.	04.05.2026р.	04.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з пояснювальної записки та програмного додатку. Пояснювальна записка обсягом 64 сторінки містить 49 ілюстрацій, 1 таблицю та 4 додатки, детально описує процес розробки, архітектуру, технічні рішення та особливості реалізації створеного програмного продукту.

Метою роботи є розробка CRM-системи для управління роботи поліклінікою, що підвищить ефективність роботи працівників поліклініки, адже подібна система збільшує швидкість та зручність роботи з записами пацієнтів на прийоми до лікарів.

Для реалізації системи було обрано клієнт серверну архітектуру.

Було використано такі технології як – Node.js, Express.js, React, Material UI та Supabase.

КЛЮЧОВІ СЛОВА: ВЕБ ДОДАТОК, HTML, REACT, MUI, SUPABASE, SSP, UI, UX, NODE, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	CRM-система для управління роботи поліклініки Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		Яцюк Л.В.						
Керівник		Лобанчикова Н.М.					3	76
Рецензент		Чижмоторя О.Г.				ФІКТ, гр.ІПЗ-22-3		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis consists of an explanatory note and a software application. The 64-page explanatory note contains 49 illustrations, 1 table, and 4 appendices. It provides a detailed description of the development process, architecture, technical solutions, and implementation features of the created software product.

The aim of the work is to develop a CRM system for managing a clinic's operations. This will improve the efficiency of the clinic's staff, as such a system increases the speed and convenience of handling patient appointments with doctors.

A client-server architecture was chosen to implement the system.

The following technologies were used: Node.js, Express.js, React, Material UI, and Supabase.

KEYWORDS: WEB APPLICATION, HTML, REACT, MUI, SUPABASE, SSP, UI, UX, NODE, SOFTWARE.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ЗМІСТ	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ	9
1.1 Аналіз проекту та постановка задачі	9
1.2 Аналіз існуючих аналогів	12
1.3. Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення	15
Висновки до першого розділу	18
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ	20
2.1. Визначення варіантів використання системи	20
2.2. Структура бази даних, взаємодія компонентів системи.....	26
2.3. Розробка функціоналу додатку	30
Висновки до другого розділу.....	41
РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ.....	42
3.1. Порядок встановлення та налаштування параметрів системи.....	42
3.2. Структура інтерфейсу системи для управління роботи поліклініки.....	46
3.3. Тестування роботи системи	55
Висновки до третього розділу	61
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	64
ДОДАТКИ.....	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

SPA – Single Page Application

JS – JavaScript

MVC – Model-view-controller

JWT – JSON Web Token

JSX – JavaScript XML

UI – User Interface

UX – User Experience

UX – Create, Read, Update, Delete

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Ефективна організація робочих процесів є запорукою успішного функціонування будь якого медичного закладу. Сучасні технології дозволяють пришвидшити рутинні завдання, мінімізувати вплив людського фактора та покращити загальну якість надання послуг. Розробка спеціалізованої CRM-системи для управління роботою поліклініки спрямована на створення зручного цифрового середовища, яке об'єднає всі ключові аспекти взаємодії між реєстратурою, лікарями в єдиному інтерфейсі.

Актуальність теми полягає у тому, що попри загальний розвиток технологій, значна частина невеликих медичних закладів досі не має спеціалізованого програмного забезпечення для ведення внутрішнього обліку. Багато поліклінік продовжують використовувати розрізнені інструменти загального призначення наприклад, текстові редактори чи електронні таблиці, а іноді й паперові журнали для планування розкладу та ведення історії пацієнтів. За таких умов розробка веб додатка є надзвичайно актуальною, оскільки вона здатна кардинально вирішити проблему неоптимізованого документообігу та значно підвищити загальну ефективність роботи медичного персоналу.

Метою роботи є розробка та програмна реалізація CRM-системи для управління роботою поліклініки, щоб надати працівникам медичного закладу зручну та функціональну платформу для пришвидшення щоденних завдань, яка забезпечить доступ до розкладу прийомів, спростить процес записів та оптимізує управління робочим часом лікарів. Дослідження та практичне застосування сучасних веб-технологій для побудови надійного і швидкого, зрозумілого інтерфейсу користувача.

Для досягнення поставленої мети було визначено комплекс завдань, що підлягають виконанню, а саме проведення аналізу предметної області та формування функціональних вимог до програмного продукту; проектування масштабованої архітектури та нормалізованої бази даних; розробка серверної логіки з імплементацією механізмів безпеки та рольової моделі доступу; створення

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

інтуїтивно зрозумілого клієнтського інтерфейсу, що включає візуальний розклад, інструменти швидкого пошуку та модулі ведення історії відвідувань; а також проведення фінального тестування і налагодження всіх компонентів для гарантування високої швидкодії та безперебійної роботи системи.

Об'єктом дослідження є процеси потреб медичного персоналу у цифровій платформі, визначення на цій основі архітектурних вимог та критеріїв для створення спеціалізованої CRM-системи для управління поліклінікою, а також аналіз ефективних методів веб розробки.

Предметом дослідження є розробка, впровадження та оптимізація клієнт-серверної системи управління поліклінікою, вивчення підходів до проектування зручних користувацьких інтерфейсів та безпечного обміну даними в сучасних медичних веб-додатках.

Структура та обсяг роботи становить пояснювальну записку кваліфікаційної роботи, яка складається зі вступу, трьох розділів, висновків, списку використаних джерел із 31 найменування, 4 додатки, 1 таблицю та 49 ілюстрацій. Текстова частина роботи викладена на 66 сторінках. Загальний обсяг роботи 76 сторінки.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

1.1 Аналіз проекту та постановка задачі

Основна задача - розробити CRM-систему для управління роботою поліклініки, яка об'єднає в собі функціонал для записів на прийом та візуального планування розкладу лікарів. Система повинна забезпечувати зручну взаємодію між реєстратурою та медичним персоналом, надаючи швидкий доступ до актуальної інформації про пацієнтів, графіки роботи та історію відвідувань.

Медична сфера сьогодні вимагає оперативності та точності. Якщо раніше для ведення обліку було достатньо паперових журналів або простих таблиць, то сучасні реалії диктують потребу у комплексному підході. Кількість пацієнтів зростає, структура клінік стає складнішою. Веб-технології дозволяють створити платформу, доступну з будь якого робочого місця або встановити на спільний сервер поліклініки, яка миттєво оновлює дані в реальному часі, що є критично важливим для уникнення накладок у розкладі лікарів.

З цього випливає, що розробка CRM-системи є необхідним кроком для модернізації медичного закладу. Система має бути ретельно продумана, щоб забезпечити безперебійну роботу персоналу.

Процес створення такої програмної системи є комплексним і охоплює кілька ключових етапів розробки - проектування архітектури та бази даних, розробка серверної логіки, створення інтерактивного клієнтського інтерфейсу, а також фінальне тестування і налагодження всіх модулів. Розглянемо основні вимоги до системи:

Розробка інтуїтивно зрозумілого інтерфейсу, що враховує дизайн, який має бути мінімалістичним та функціональним, щоб медичний персонал не витрачав зайвий час на навчання, а міг одразу приступити до роботи.

Візуальний розклад та структурованість модулів передбачає головну сторінку розкладу, що має надавати чітку часову сітку з відображенням вільних місць та існуючих записів з відповідним статусом, що дозволяє працівнику на

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

реєстратурі миттєво знайти потрібний час для пацієнта. Система повинна мати чітке розділення на розділи - пацієнти, лікарі, відділення, робочі години тощо, забезпечуючи швидку навігацію між ними.

Історія відвідувань пацієнта та швидкий пошук - кожен пацієнт повинен мати індивідуальний профіль, де зберігається вся хронологія візитів, діагнозів та призначень та рекомендацій. Наявність зручних фільтрів та інструментів пошуку за ПІБ чи контактними даними для миттєвої ідентифікації пацієнта в базі.

Швидкодія системи та надійність функціоналу є високим пріоритетом у системі, цього можна досягнути завдяки оптимізації запитів до бази даних, які повинні виключати затримки при завантаженні розкладу або оновленні статусів прийомів. Усі операції - від додавання нового лікаря до створення запису на прийом мають виконуватися коректно, забезпечуючи цілісність даних у БД.

Окрім загальних функціональних вимог, критично важливим аспектом розробки CRM системи для управління поліклінікою є чітке розподілення прав доступу. Оскільки з системою взаємодіятимуть різні категорії персоналу, система повинна підтримувати кілька типів користувачів з чітким доступом дозволів. Зокрема, адміністратор системи наділяється найвищим рівнем доступу, відповідаючи за управління поліклінікою, створення облікових записів працівників, управління відділеннями та додаванням лікарів тощо. Своєю чергою, працівник реєстратури забезпечується інструментами для оперативного запису пацієнтів, управління розкладом лікарів, зміни статусів прийому та базової обробки контактних даних. Окремою функціональною ланкою є лікар, який отримує зручний доступ до власного розкладу та робочих годин. Головна вимога до цієї ролі полягає у можливості переглядати медичну історію візитів пацієнтів закріплених за ним, вносити результати огляду, ставити діагнози та створювати медичні записи.

Безпека медичних даних є ще однією фундаментальною вимогою до постановки задачі. Враховуючи високу конфіденційність інформації про стан здоров'я пацієнтів CRM-система повинна реалізовувати надійні механізми автентифікації та авторизації. Усі мережеві запити між клієнтським додатком та

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

сервером мають передаватися виключно захищеними каналами, а доступ до бази даних повинен бути ізольований на рівні серверної логіки, має бути перевірка прав на додавання та редагування даних. Необхідно виключити будь-яку можливість доступу до медичних карток з боку осіб, які не беруть безпосередньої участі в лікувальному процесі.

Не менш важливою задачею є забезпечення масштабованості та адаптивності архітектури програмного продукту. Сучасна поліклініка - це динамічна структура, яка може розширюватися, можуть з'являтися нові відділення, збільшуватися кількість лікарів або зростати потік пацієнтів. Обраний стек технологій та спроектована база даних повинні бути готові до збільшення навантаження без втрати швидкодії. Крім того, модульна архітектура як клієнтської, так і серверної частини має дозволяти легко додавати новий функціонал у майбутньому без необхідності переписування існуючого основного функціоналу, наприклад, для подальшої інтеграції модуля аналітики, автоматичної генерації звітів тощо.

Ще одним важливим фактором, який зумовлює необхідність розробки даної CRM-системи, є потреба в оптимізації клієнтського досвіду та розв'язанні проблеми неконтрольованих черг. Традиційна організація прийому часто супроводжується тривалим очікуванням під кабінетами, що створює дискомфорт для відвідувачів. Завдяки впровадженню централізованого електронного розкладу та можливості точного планування часу візиту, система дозволяє рівномірно розподіляти потік пацієнтів протягом робочого дня, конкретний час прийому, зміна його статусу дозволить це зробити.

Підсумовуючи постановку задачі, успішне впровадження розробленої CRM-системи має комплексно вирішити низку поточних проблем медичного закладу. Очікується, що пришвидшення рутинних процесів дозволить суттєво знизити адміністративне навантаження на реєстратуру, повністю ліквідувати помилки, пов'язані з людським фактором, наприклад, подвійні записи на один і той самий час та підвищити загальну якість і швидкість обслуговування пацієнтів.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

1.2 Аналіз існуючих аналогів

Перед початком проектування власної системи доцільно провести детальний аналіз існуючих на ринку програмних рішень. Це дозволить виявити їхні переваги та недоліки. На сьогоднішній день існує низка вітчизняних цифрових платформ серед яких для детального аналізу було обрано Helsi.me та Salesforce Health Cloud.

Helsi.me - це українська цифрова платформа охорони здоров'я, розроблена з метою спрощення взаємодії між пацієнтами та медичними закладами [1]. Сервіс надає можливість онлайн-запису до лікарів державних та приватних клінік, перегляду електронних медичних записів, отримання електронних рецептів та направлень. Платформа інтегрована з реєстром eHealth Міністерства охорони здоров'я України, що забезпечує відповідність вимогам національного законодавства у сфері цифрової медицини.

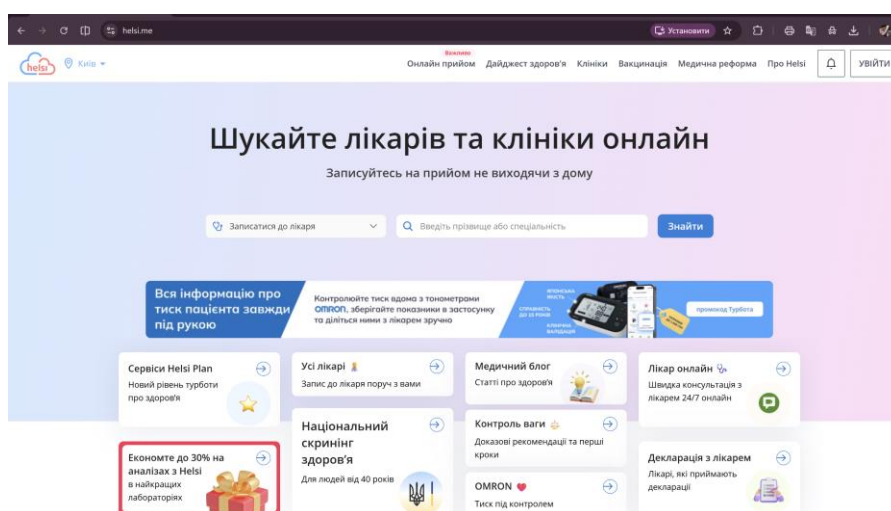


Рис.1.1. Інтерфейс продукту дозволяє здійснити запис до лікаря в кілька кроків через просту форму на сайті.

Головною особливістю Helsi.me є орієнтація на взаємодію пацієнта з медичною системою, такі як пошук лікаря, запис на прийом, доступ до власної медичної історії. Проте з точки зору управління поліклінікою платформа має суттєві обмеження - вона не охоплює внутрішні процеси закладу, такі як управління персоналом, контроль завантаженості лікарів, фінансовий облік та

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

формування звітності. Таким чином, Helsi.me виступає переважно як сервіс для пацієнтів, а не як інструмент управління медичним закладом.

Аналізуючи медичну систему Helsi.me, виділимо її ключові переваги, серед яких глибока інтеграція з державною електронною системою охорони здоров'я eHealth МОЗ України, можливість зручного цілодобового онлайн-запису на прийом без необхідності телефонних дзвінків, а також наявність великої бази підключених клінік по всій території країни. Проте, попри ці сильні сторони, платформа має і недоліки. Зокрема, у системі відсутні ефективні інструменти для автоматизованого контролю завантаженості лікарів, а функціональні можливості адміністратора поліклініки щодо управління внутрішніми процесами є досить обмеженими. Крім того, жорстка залежність платформи від державних реєстрів створює додаткові технічні труднощі для її повноцінної та швидкої інтеграції в роботу приватних медичних закладів.

Другим аналогом є Salesforce Health Cloud - це хмарна CRM-платформа корпоративного рівня, розроблена компанією Salesforce для управління взаємодією з пацієнтами у великих медичних організаціях [2]. Система надає широкий спектр інструментів - ведення профілів пацієнтів, координація медичної допомоги, аналітика, автоматизація бізнес-процесів та інтеграція з медичними пристроями і сторонніми системами. Платформа орієнтована на великі лікарняні мережі, страхові компанії та фармацевтичні організації, що забезпечує їй широке розповсюдження на міжнародному ринку.

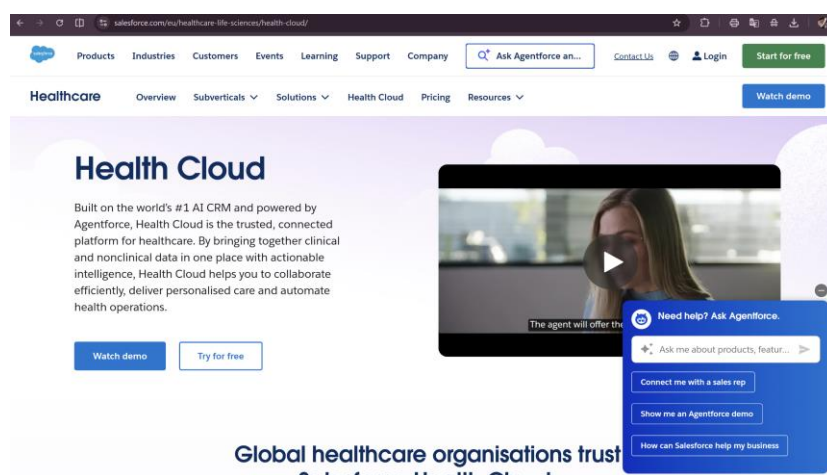


Рис.1.2. Інтерфейс сайту

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Проте для звичайної поліклініки Salesforce Health Cloud має суттєві обмеження. Вартість ліцензії є надмірно високою для невеликих медичних закладів, а впровадження системи потребує залучення окремих технічних спеціалістів та тривалого налаштування. Інтерфейс платформи є перевантаженим і складним у освоєнні для звичайного медичного персоналу. Окрім того, система не має локалізації під українське законодавство та стандарти охорони здоров'я, що робить її практично непридатною для використання у вітчизняних медичних закладах.

Розглядаючи платформу Salesforce Health Cloud, відзначимо її переваги, до яких належать широкі можливості інтеграції, зручна хмарна архітектура, що забезпечує доступ із будь-якого пристрою, а також висока масштабованість та продуманий користувацький інтерфейс. Водночас система має недоліки, що обмежують або ускладнюють її використання. До головних мінусів належать відсутність української локалізації та надмірна складність функціоналу для задоволення базових потреб стандартної поліклініки. Крім того, розгортання та подальша експлуатація цього комерційного рішення вимагають обов'язкового залучення вузькопрофільних технічних спеціалістів для постійної підтримки та налаштування платформи.

Таким чином, Helsi.me та Salesforce Health Cloud фактично закривають лише частину потреб медичного закладу. Helsi.me орієнтована виключно на взаємодію з пацієнтом і не надає інструментів для внутрішнього управління поліклінікою, тоді як Salesforce Health Cloud є надмірно масштабним і фінансово недоступним рішенням для вітчизняних медичних закладів.

Розроблювана система покликана об'єднати обидва напрямки - зручний запис та взаємодію з пацієнтами і повноцінні інструменти управління поліклінікою - в єдиному продукті, адаптованому до українських реалій. Зручний візуальний розклад прийомів, які пов'яже пацієнтів із поліклінікою, дозволить працівникам записувати їх на прийому до лікарів, зручно це робити, завдяки повному розумінню

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

за допомогою інтерфейсу планування роботи лікарів. Порівняння аналогів, для наочності результати аналізу наведено у порівняльній таблиці 1.1.

Таблиця 1.1.

Порівняння аналогічних систем

Характеристика/Функціонал	Helsi.me	Salesforce
Онлайн запис до лікаря	+	+
Управління персоналом	-	+
Електронні медкартки	+	+
Розклад лікарів	+	+
Доступність для малих клінік	+	-
Внутрішній документообіг	-	+
Українська локалізація	+	-
Відповідність МОЗ	+	-

Таблиця 1.1. показує порівняння аналогічних системи, ми порівнюємо певні характеристики та функціональні можливості систем. Таблиця показує які переваги та недоліки має кожна з систем.

1.3. Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Під час проектування сучасних веб додатків важливо визначити архітектуру проекту. Найчастіше можна зустріти монолітну, мікросервісну та клієнт-серверну.

Монолітна архітектура розгортає весь функціонал як єдине ціле [3]. Мікросервісна розбиває систему на більш дрібні незалежні одне від одного сервіси, де кожен з них має відповідати своїм функціональним вимогам та критеріям [4]. Клієнт серверна передбачає розподіл серверної та клієнтської частини [5]. Такий підхід добре масштабується та є складним в підтриманні та розумінні, зручно підходить для роботи декількох команд або однієї з огляду на специфіку проекту.

З огляду на специфіку та вимоги до системи керування поліклініки, найбільш оптимальним є вибір клієнт серверної архітектури. Цей підхід забезпечує чіткий розподіл між візуальним інтерфейсом користувача, тобто клієнтською частиною та серверною логікою обробки даних, тобто серверним додатком, що робить проект структурованим, зручним для підтримки та подальшого масштабування.

Для взаємодії між клієнтською та серверною частинами необхідно обрати архітектурний підхід до побудови API, де найпопулярнішими є REST API та GraphQL. GraphQL- це мова запитів, яка дозволяє клієнту гнучко вказувати, які саме дані йому потрібні, що допомагає уникнути зайвого навантаження на мережу [6]. Однак REST API є більш поширеним, надійним та інтуїтивно зрозумілим стандартом. Він використовує HTTP методи для обробки даних, які адресуються через чіткі кінцеві точки [7]. Зважаючи на структуру сутностей медичної системи, таких як лікарі, пацієнти та прийоми, архітектура REST API є найкращим вибором завдяки своїй зрозумілості та простоті використання. В такій системі немає потреби в побудові складних запитів для витягання та обробки даних, тож REST API найкращий вибір.

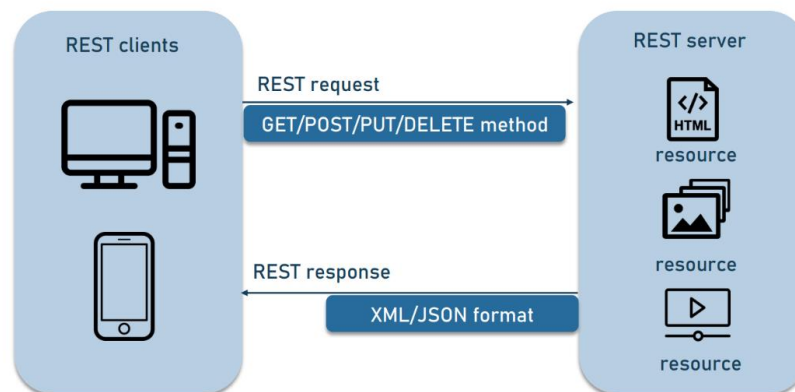


Рис.1.3. Приклад роботи REST API

Вибір технологій розробки доцільно почати з бази даних. Для збереження та управління даними поліклініки було обрано Supabase, яка використовує реляційну базу даних PostgreSQL [25]. На відміну від NoSQL рішень, реляційна структура

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

ідеально підходить для медичних систем, де важливе збереження цілісності даних та підтримка складних зв'язків між таблицями, наприклад, між графіком роботи лікаря, його відпустками та записами пацієнтів. Також завдяки тому, що Supabase є хмарним рішенням, інтеграція бази даних у систему стає простішою, адже платформа бере на себе всі аспекти адміністрування, безпеки та масштабування даних у хмарі, також зручність налаштування, не потрібно розгортати додатково локально БД, достатньо мати певні доступи до неї.



Рис.1.4. Supabase DB

Оскільки був обраний підхід REST API, для реалізації серверної частини ідеальним рішенням є використання середовища Node.js у поєднанні з фреймворком Express.js. Node.js компілює JavaScript код на сервері та відрізняється високою продуктивністю завдяки асинхронній моделі обробки запитів [13]. Express.js є мінімалістичним фреймворком, який надає зручний і ефективний інструментарій для побудови серверної маршрутизації та обробки HTTP запитів, дозволяючи легко розширювати функціонал за допомогою проміжних обробників, тобто додавати певну валідацію, налаштувати авторизацію тощо [14].

Для побудови інтерфейсу клієнтської частини було обрано бібліотеку React. Головна перевага полягає в компонентному підході та використанні віртуального DOM, що гарантує швидке оновлення інтерфейсу без повного перезавантаження сторінки [15]. React має велику екосистему та кількість готових бібліотек, що значно прискорює процес розробки. Для їх встановлення та зручного управління залежностями в межах проекту використовується менеджер пакетів npm. Node Package Manager - це інструмент середовища виконання Node.js та найбільший у

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

світі реєстр програмного забезпечення, який дозволяє автоматизовано завантажувати, оновлювати та інтегрувати сторонні модулі у свої додатки [26].

Для створення та налаштування базової структури клієнтського React-дodatка розглядалися два популярні інструменти - класичний Create React App та сучасний Vite. Create React App довгий час залишався стандартним рішенням в екосистемі React, використовуючи для збирання модулів [29]. Однак, для реалізації даної системи було обрано саме Vite, адже цей інструмент більш новий, вирішує проблему повільного запуску великих проєктів. Замість тривалого попереднього збирання всього коду, Vite делегує обробку модулів самому браузеру через стандарт ES Modules, завдяки чому локальне середовище розробки ініціалізується практично миттєво [30].

Для створення зручного та дизайну використовується бібліотека Material UI (MUI) [16]. Вона надає широкий набір готових, добре стилізованих компонентів, що дозволяє брати за основи готові компоненти та швидко їх стилізувати, але при цьому не створюючи все з нуля.

Щодо вимог до апаратного забезпечення, то серверна частина не є надмірно вимогливою до ресурсів завдяки ефективній асинхронній моделі Node.js. Для розгортання системи достатньо віртуального сервера з двоядерним процесором, 4 ГБ оперативної пам'яті та SSD-накопичувачем.

Таким чином, обраний набір інструментів є збалансованим, сучасним та повністю відповідає поставленим завданням, забезпечуючи створення надійної, швидкої та зручної платформи психологічної допомоги. Поєднання React з Node.js на бекенді та реляційної бази даних PostgreSQL через Supabase створює надійну та масштабовану платформу.

Висновки до першого розділу

У ході виконання першого розділу було поставлено задачу на розробку CRM системи роботи поліклініки та визначено основні вимоги до програмного продукту. Було досліджено та проаналізовано існуючі аналоги медичних систем за

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

визначеними критеріями. Це дозволило сформулювати основні цілі розробки, окреслити мінімально необхідний функціонал, такий як управління розкладом, лікарями й пацієнтами, а також визначити принципи побудови користувацького інтерфейсу для ефективної роботи медичного персоналу.

На основі сформованих вимог та бачення системи було обрано клієнт-серверну архітектуру на базі REST API. Також було визначено та обґрунтовано вибір стека технологій для реалізації поставлених завдань, до якого увійшли React для клієнтської частини, Node.js з Express.js для серверної логіки, а також хмарна платформа Supabase на базі PostgreSQL для управління базою даних.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ

2.1. Визначення варіантів використання системи

Для забезпечення ефективної та безпечної роботи медичної системи передбачено розподіл функціоналу між трьома основними ролями користувачів - адміністратором, лікарем та працівником реєстратури. Кожна роль має власний набір доступів та варіантів використання, що чітко відповідають посадовим обов'язкам персоналу клініки.

Роль головного адміністратора наділена повним доступом до системи та відповідає за її базове налаштування і технічний супровід. Головними варіантами використання для цієї ролі є управління обліковими записами всіх співробітників, що включає створення та редагування профілів лікарів, додавання акаунтів до системи. Крім того, адміністратор формує структуру клініки в системі, додаючи відділення, кабінети та встановлюючи стандартну тривалість прийому, а також має доступ до загального моніторингу роботи додатка, має можливість вносити дані про вихідні чи лікарняні дні до лікаря, коли він не працює, може змінювати записи на прийом за потреби.

Працівник реєстратури виступає першою ланкою взаємодії між пацієнтом та клінікою, тому його функціонал зосереджений на управлінні розкладом. Ця роль дозволяє реєструвати нових пацієнтів в базі даних, оновлювати їхню персональну інформацію та здійснювати пошук. Основний варіант використання полягає у роботі з візуальною сіткою розкладу - реєстратор аналізує графіки роботи лікарів, знаходить вільні місця та безпосередньо створює, переносить або скасовує записи на прийом, забезпечуючи безперервний потік клієнтів. Також може додавати перерву для лікаря, за потреби.

Роль лікаря орієнтована на медичну практику та ізольована від адміністративних налаштувань. Лікар отримує доступ до власної сторінки, де взаємодіє зі своїм розкладом. Головним варіантом використання є ведення медичної історії пацієнта. Під час візиту лікар відкриває профіль пацієнта, аналізує

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

попередні записи, додає інформацію про поточні симптоми чи скарги, фіксує діагноз та формує призначення чи рекомендації. Для моделювання поведінки системи, зокрема побудови діаграм варіантів використання та діяльності, було застосовано веб сервіс PlantText. Це спеціалізований онлайн-інструмент, який дозволяє автоматично генерувати графічні UML-діаграми на основі текстового коду, написаного за допомогою мови розмітки PlantUML [17].

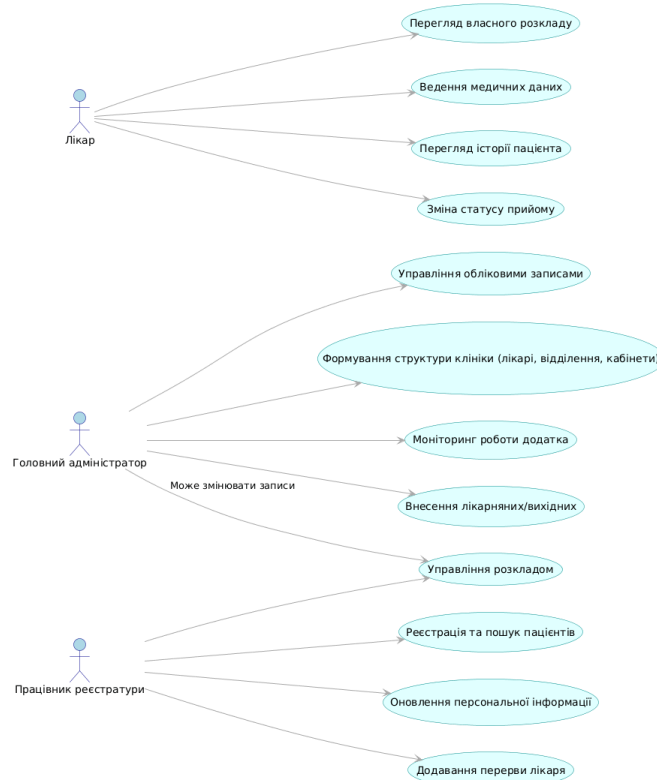


Рис.2.1. Діаграма варіантів використання

Опишемо діаграму діяльності загальної роботи система для управління поліклінікою. На рисунку 2.2. наведено діаграму діяльності, яка детально ілюструє алгоритм обслуговування пацієнта - від моменту запису до завершення візиту та формування медичного висновку. Цей бізнес-процес є центральним для функціонування поліклініки.

Процес складається з наступних логічних етапів, починається з ініціації та планування. Процес розпочинається із запиту на прийом. Користувач (працівник реєстрації) обирає необхідне медичне відділення, конкретного лікаря, а також дату та час візиту, яка потрібна пацієнту.

Далі це системна валідація - після введення даних система перевіряє доступність обраного часового слота у розкладі лікаря. Якщо слот уже зайнятий, алгоритм повертає процес на етап вибору параметрів прийому. У разі успішної перевірки система створює новий запис на прийом і встановлює для нього початковий статус «Заплановано».

Наступним буде проведення прийому, коли пацієнт фактично прибуває до кабінету, лікар переводить статус прийому в стан «В процесі». Під час медичного огляду лікар послідовно фіксує в системі скарги та симптоми, встановлює діагноз і додає рекомендації щодо подальшого лікування.

В кінці йде завершення та збереження даних, на фінальному етапі дані про медичні записи зберігаються. Цей документ автоматично пов'язується з поточним прийомом для збереження реляційної цілісності бази даних. Після цього статус прийому остаточно змінюється на «Завершено».



Рис.2.2. Діаграма діяльності загальної роботи системи

Такий лінійний та контрольований потік даних мінімізує ризик людської помилки. Також гарантує коректне ведення розкладу та забезпечує надійне збереження історії хвороби кожного пацієнта.

Для побудови діаграм діяльності використовуємо платформу Mermaid [22]. Цей інструмент дозволяє швидко та автоматизовано генерувати діаграми на основі їхнього текстового опису, що значно спрощує процес проектування для нас.

На рисунку 2.3. наведено діаграму діяльності, що ілюструє алгоритм створення або редагування даних лікаря з боку адміністратора системи. Процес починається переходом користувача на сторінку управління лікарями, після чого відкривається відповідне модальне вікно з формою. Користувач послідовно заповнює базову інформацію, такі як персональні дані (прізвище, ім'я, по батькові), закріплює фахівця за певним відділенням та призначає йому робочий кабінет. Важливим етапом є встановлення тривалості одного прийому (у хвилинах) та налаштування графіка робочих годин за днями тижня. На основі цих двох параметрів система виконує автоматичний динамічний розрахунок кількості доступних можливих прийомів для запису на кожен робочий день, що миттєво відображається в інтерфейсі та також на основі тривалості та початку робочого дня підлаштовує час прийомів, що є фундаментальним для правильної побудови таблиці розкладу.

Після завершення введення даних користувач приймає рішення про збереження або скасування змін. При натисканні кнопки «Відхилити» або закриття модального вікна (передбачається, що модальне вікно можна буде закрити декількома способами, але на діаграмі ця обробка події узагальнена) процес переривається, а вікно закривається без внесення змін до бази. У разі ініціації збереження система виконує валідацію введеної інформації (наприклад, перевірку обов'язкових полів чи коректності часових інтервалів). Якщо виявлено помилки, алгоритм повертає користувача до етапу редагування. При успішній валідації дані безпечно зберігаються в базі даних, дані на UI оновлюються, а користувач бачить актуалізований перелік лікарів.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				23
Змн.	Арк.	№ докум.	Підпис	Дата		

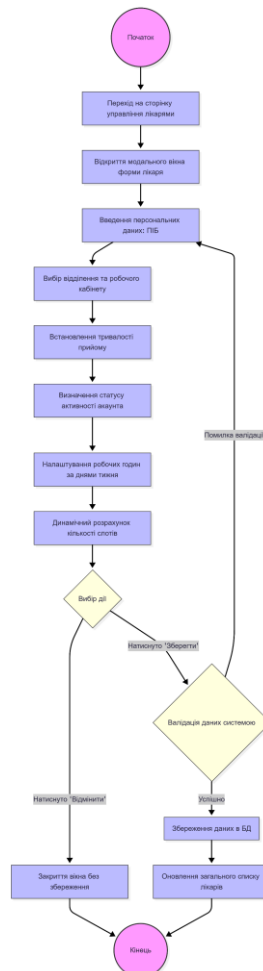


Рис.2.3. Діаграма діяльності додавання або редагування даних лікаря

На рисунку 2.4. наведено діаграму діяльності, що деталізує логіку автентифікації користувача та подальшої перевірки його прав доступу на основі прав доступу згідно до ролі. Цей алгоритм гарантує, що конфіденційна медична інформація доступна лише уповноваженому персоналу, певній ролі.

Процес розпочинається з введення користувачем облікових даних (логіну та пароля) на формі входу. Система перевіряє наявність та статус активності відповідного облікового запису в базі даних, а також проводить валідацію введеного пароля. У разі невідповідності даних процес переривається, а користувачу відображається повідомлення про помилку входу.

За умови успішної автентифікації система ідентифікує користувача, визначає його роль та зберігає активний сеанс на стороні клієнтського інтерфейсу. Під час кожної наступної спроби користувача звернутися до певного модуля або закритої

сторінки, система виконує два етапи контролю. Спочатку перевіряється дійсність поточного сеансу (якщо він закінчився, вийшов час дії - користувача перенаправляють на сторінку входу). Далі алгоритм перевіряє, чи має роль поточного користувача необхідні дозволи для перегляду запитуваного модуля. Якщо прав недостатньо, інтерфейс блокує дію та інформує про обмежений доступ. У разі успішної перевірки система надає доступ і відображає відповідні дані.

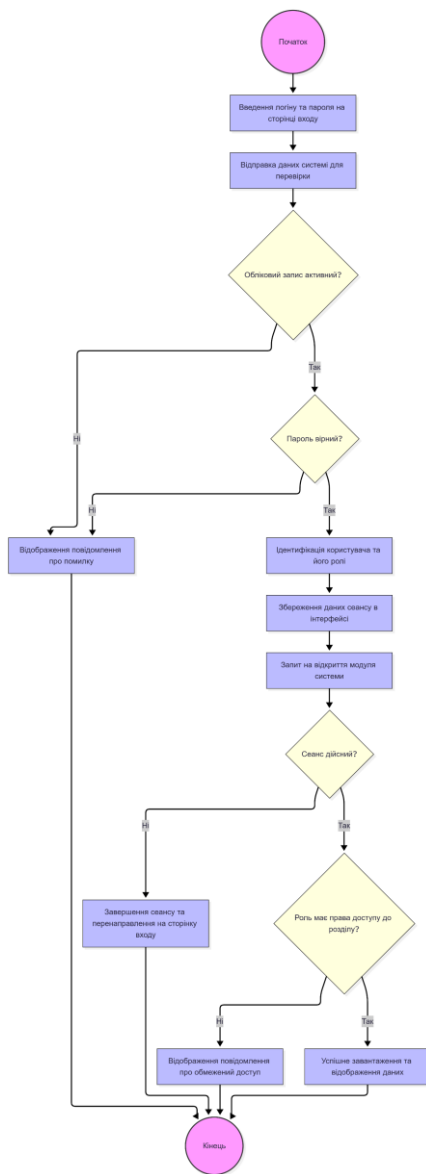


Рис.2.4. Діаграма діяльності авторизації та доступу

Таким чином ми розглянули основні варіанти використання системи та проілюстрували на діаграмах діяльності як може використовуватись система. Основні продумані алгоритми мають дати основу для подальшої розробки.

2.2. Структура бази даних, взаємодія компонентів системи

До початку реалізації серверної частини проекту слід почати з проектування бази даних. Зробити правильно на логічну структуру БД є важливою частиною, адже на цьому етапі можна продумати всі сутності і забезпечити коректну роботу між сутностями даних, полегшити роботу з даними, оптимізувати збереження.

В нашій системі було обрано Supabase – відкрита Backend-as-a-Service (BaaS) платформа, яка базується на надійній реляційній системі управління базами даних PostgreSQL [18]. Supabase має хорошу доступність, адже розташована в хмарі, що позбавляє нас необхідності самостійно налаштовувати сервер для БД.

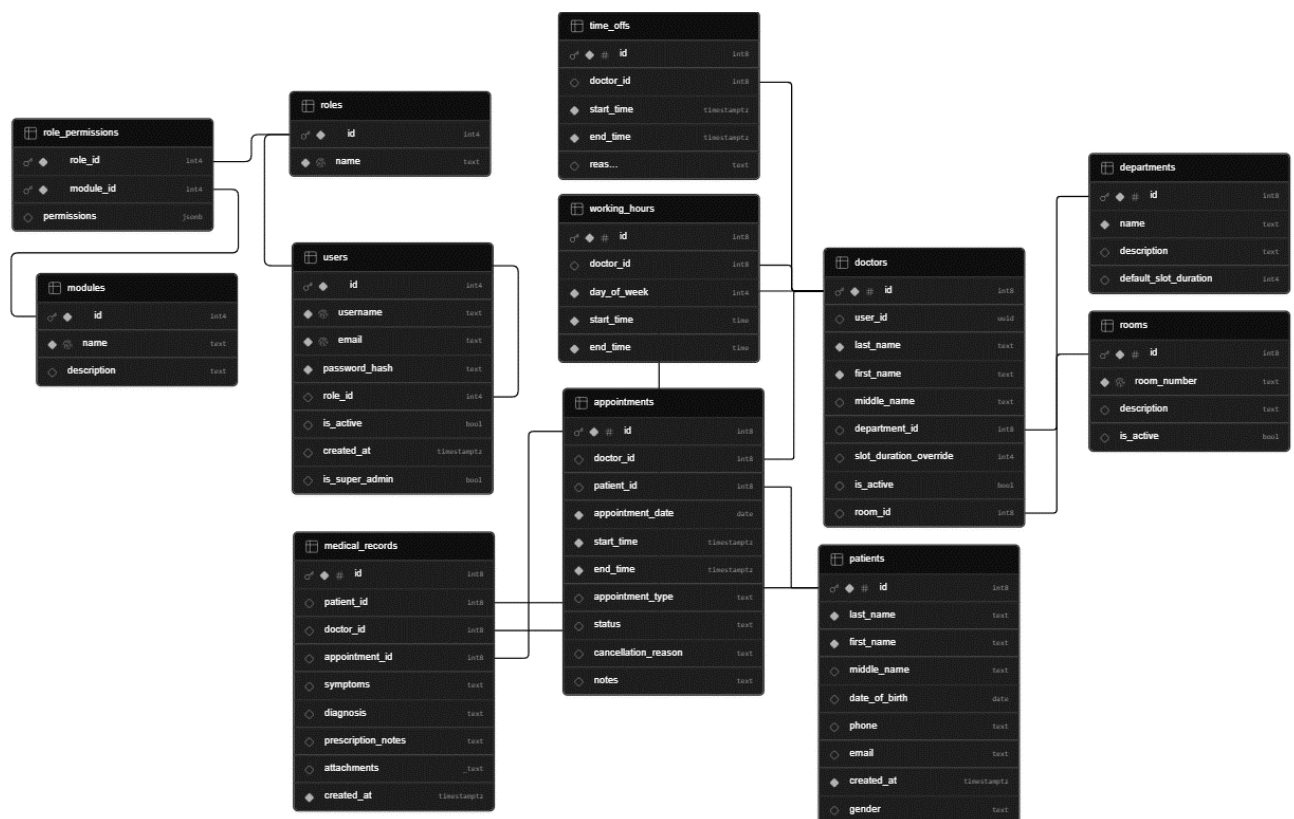


Рис.2.5. Діаграма бази даних

Опишемо кожну таблицю в БД яка використовується в системі роботи поліклініки. Надалі текст буде складатися з короткого опису таблицю на полів які вона містить, також вказанням типу даних, який використовується у таблиці, також опис буде з вказуванням основних зв'язків таблиці між собою.

Таблиця patients (Пацієнти) зберігає персональну та контактну інформацію пацієнтів. Вона потрібна для перегляду та пошуку даних лікарем, а також для реєстрації пацієнтів та створення прийомів працівником реєстратури. Структура таблиці включає унікальний ідентифікатор пацієнта (поле id типу int8), прізвище (last_name типу text), ім'я (first_name типу text) та по батькові (middle_name типу text). Крім того, зберігаються дата народження (date_of_birth типу date), контактний номер телефону (phone типу text) та електронна пошта для зв'язку (email типу text). Для системного обліку передбачено поля дати та часу додавання пацієнта в систему (created_at типу timestamp) і стать (gender типу text).

Таблиця departments (Відділення) містить дані про відділення поліклініки. Вона потрібна для логічного групування лікарів, структурування та фільтрування візуального інтерфейсу. Записи у цій таблиці складаються з унікального ідентифікатора відділення (id типу int8), назви відділення (name типу text), опису напрямку роботи відділення (description типу text) та стандартної тривалості прийому для цього відділення (default_slot_duration типу int4).

Таблиця rooms (Кабінети) зберігає дані про робочі кабінети для лікарів у поліклініці. Також по суті вона виступає як таблиця для нормалізації даних, щоб не збільшувати та не перевантажувати таблицю лікарів. Вона містить унікальний ідентифікатор кабінету (id типу int8), номер кабінету (room_number типу text), опис, що включає призначення чи загальні нотатки (description типу text), а також статус використання кабінету (is_active типу text).

Таблиця doctors (Лікарі) зберігає персональні та робочі дані лікарів. Вона пов'язує лікаря з акаунтом системи, відділенням та робочим кабінетом. Таблиця містить унікальний ідентифікатор лікаря (id типу int8) та посилання на акаунт у таблиці користувачів (user_id типу uuid), що потрібно для авторизації та перегляду даних відповідно до ролі. Також зберігаються прізвище (last_name типу text), ім'я (first_name типу text) та по батькові (middle_name типу text). Для зв'язків використовуються ідентифікатор відділення, до якого прив'язаний лікар (department_id типу int8), та ідентифікатор кабінету (room_id типу int8). Додатково

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

передбачено поля для індивідуальної тривалості прийому лікаря, яка йде поверх тривалості відділення (slot_duration_override типу int4), та статус лікаря, що вказує, чи він працює, чи перебуває у відпустці або має інші причини недоступності (is_active типу bool).

Таблиця appointments (Прийоми) є центральною таблицею системи та потрібна для керування розкладом. Вона зберігає заплановані, скасовані та завершені візити пацієнтів до лікарів на певну дату та час. Структура включає унікальний ідентифікатор прийому (id типу int8), ідентифікатор лікаря як посилання на відповідну таблицю (doctor_id типу int8) та ідентифікатор пацієнта (patient_id типу int8). Планування забезпечується полями дати прийому (appointment_date типу date), часу початку (start_time типу timestampz) та часу закінчення прийому (end_time типу timestampz). Також фіксуються тип прийому, наприклад, консультація чи огляд (appointment_type типу text), поточний статус для відстеження процесу (status типу text), причина скасування, яка заповнюється у разі відміни за потреби (cancellation_reason типу text), та додаткові нотатки до прийому, де можуть бути скарги чи інша додаткова інформація, яку реєстратор вважатиме за потрібне знати лікарю (notes типу text).

Таблиця medical_records (Медичні записи) зберігає дані результатів прийому, що дозволяє мати історію хвороб пацієнтів. До неї входять унікальний ідентифікатор медичного запису (id типу int8), ідентифікатор пацієнта (patient_id типу int8), ідентифікатор лікаря (doctor_id типу int8) та посилання на конкретний прийом, за підсумками якого створюється запис (appointment_id типу int8). Медична інформація зберігається у полях для скарг та симптомів (symptoms типу text), встановленого діагнозу або результату прийому (diagnosis типу text) та призначень із рекомендаціями (prescription_notes типу text). Також передбачено можливість додавання посилань на прикріплені файли (attachments типу масив text[]) та фіксується дата і час створення запису (created_at типу timestampz).

Таблиця working_hours (Робочі години) зберігає дані графіка роботи лікаря. Вона використовується для точної побудови графіка розкладу на клієнтському

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерфейсі та загального налаштування робочого графіка лікаря. Таблиця містить унікальний ідентифікатор запису (id типу int8), ідентифікатор лікаря (doctor_id типу int8), день тижня (day_of_week типу int4), а також час початку (start_time типу time) і час кінця робочої зміни лікаря (end_time типу time).

Таблиця time_offs (Відпустки\вихідні) зберігає інформацію про час, коли лікар недоступний для прийому, що може бути відпусткою, персональним вихідним або лікарняним. Ці дані також використовуються для побудови графіка розкладу на UI. Структура складається з унікального ідентифікатора (id типу int8), ідентифікатора лікаря (doctor_id типу int8), дати та часу початку (start_time типу timestamptz) і кінця періоду відсутності лікаря (end_time типу timestamptz), а також містить причину цієї відсутності (reason типу text).

Таблиця users (Користувачі) зберігає дані для авторизації в системі працівників поліклініки, по суті користувачів системи. Вона містить унікальний ідентифікатор користувача (id типу int8), логін або спеціальне ім'я в системі (username типу text), контактну електронну пошту (email типу text) та зашифрований пароль для безпечного зберігання (password_hash типу text). Додатково вказується ідентифікатор ролі користувача в системі (role_id типу int4), статус облікового запису (is_active типу bool), дата створення акаунта (created_at типу timestamptz) та спеціальне поле для встановлення необмежених прав доступу в системі (is_super_admin типу bool).

Таблиця roles (Ролі) зберігає дані ролей користувачів у системі для лікарів, працівників реєстратури та адміністраторів. Вона складається з унікального ідентифікатора ролі (id типу int4) та її назви (name типу text).

Таблиця modules (Модулі системи) зберігає інформацію про модулі системи, по суті сторінки. Вона додана для нормалізації даних та для надання доступу згідно з ролями в системі. Таблиця містить унікальний ідентифікатор модуля (id типу int4), його назву (name типу text) та опис модуля (description типу text).

Таблиця role_permissions (Дозволи ролей) зберігає дані ролей та їхнього доступу, по суті таблиця реалізує рольову модель. Вона визначає, які дії доступні

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

певній ролі в межах конкретного модуля. Таблиця містить ідентифікатор ролі (role_id типу int4), ідентифікатор модуля (module_id типу int4) та поле формату jsonb, що є спеціальним форматом бази даних Postgres. Це поле зберігає конкретні дозволи на читання, додавання, редагування та видалення даних у вигляді JSON-об'єкта, наприклад {"read": 1, "create": 0, "delete": 0, "update": 1}, де 0 виступає як заборона, а 1 відповідно як дозвіл.

База даних системи для управління роботи поліклініки побудована на чітких реляційних зв'язках [27]. Організаційна структура забезпечується зв'язком «багато до одного», де кожен лікар обов'язково закріплений за певним відділенням, може мати робочий кабінет та робочі години. Для керування розкладом використовується відношення «один до багатьох» між сутністю лікаря, його робочими годинами та днями відпусток або вихідних, що дозволяє системі визначати актуальні вільні місця на прийом. Центальною ланкою взаємодії є таблиця прийомів, яка об'єднує конкретного пацієнта та лікаря на певний час з прикріпленням статусом, що дозволяє відстежувати процес прийому. Після завершення візиту до цього прийому може створитись медичний запис з відношенням «один до одного», що гарантує збереження історії медичних записів без дублювання даних.

Побудована структура бази даних відповідає вимогам третьої нормальної форми (3NF). Згідно з класичною теорією проектування реляційних баз даних [19], приведення до 3NF дозволяє усунути надмірність даних, запобігти їхньому дублюванню та уникнути аномалій при оновленні, додаванні чи видаленні записів. Наприклад, контактні дані пацієнтів, деталі відділень та інформація про лікарів зберігаються в єдиних екземплярах у відповідних таблицях і зв'язуються за допомогою зовнішніх ключів, що забезпечує цілісність системи.

2.3. Розробка функціоналу додатку

Опишемо основні моменти розробки функціоналу для клієнт-серверного додатку. Оскільки клієнтська частина повинна брати дані з сервера, тому більшість

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

роботи буде побудовано таким чином, що спочатку ми робимо певні дії на сервері, результат яких будемо повертати на клієнт і там вже з отриманими даними будувати інтерфейс.

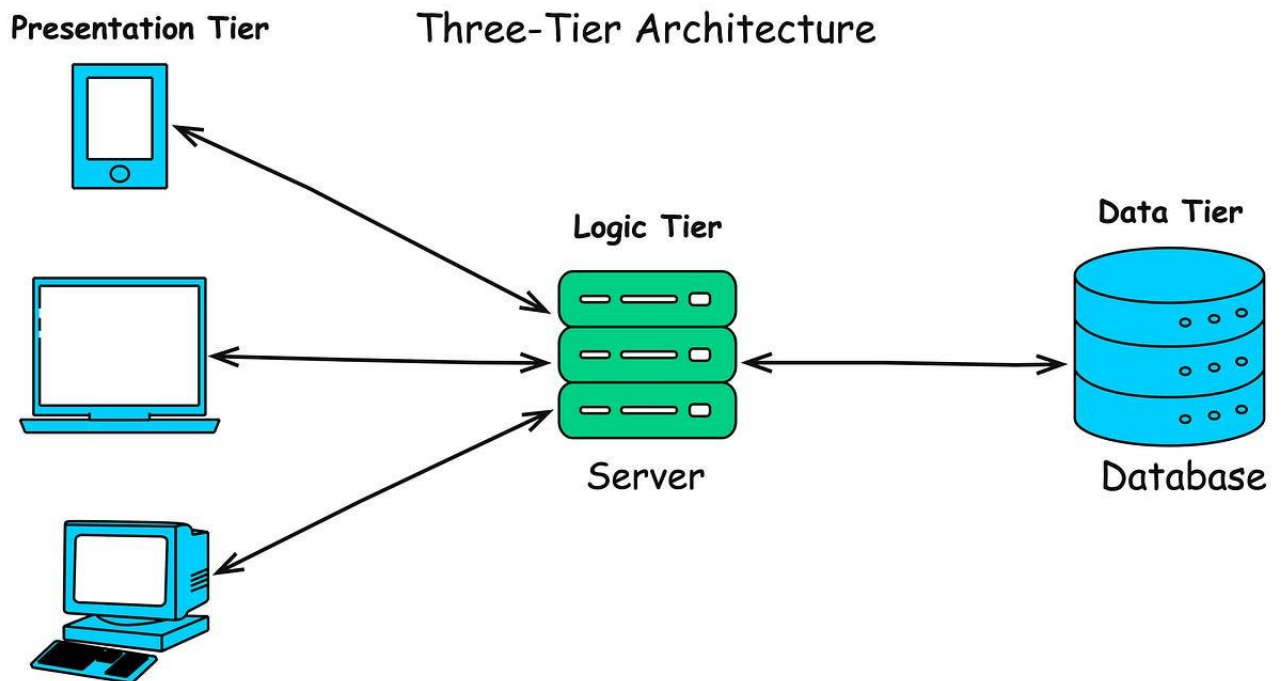


Рис.2.6. Огляд клієнт серверної архітектури

Клієнт-серверна модель у контексті системи роботи поліклініки передбачає чіткий поділ обов'язків між візуальним інтерфейсом та внутрішньою логікою програми [28]. Клієнтська частина, реалізовується на React, відповідає виключно за рендеринг сторінок, збір даних з форм включно з валідацією та обробку дій користувача. З іншого боку серверна частина на базі Node.js виконує роль центрального вузла - вона перевіряє права доступу, валідує вхідні дані та безпосередньо взаємодіє з базою даних Supabase. Такий розподіл гарантує безпеку даних поліклініки, оскільки клієнтський додаток не має прямого доступу до бази, а отримує лише ту інформацію, яку сервер дозволив йому побачити.

Взаємодія між клієнтом та сервером побудована на основі архітектурного стилю REST API, де обмін даними відбувається у форматі JSON [23]. Кожна ключова сутність системи, така як пацієнти, лікарі чи прийоми, має свою базову адресу - ендпоінт. Для управління цими ресурсами використовуються стандартні

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

HTTP методи, такі як GET для отримання списків або конкретних записів, POST для створення нових сутностей, PUT або PATCH для їх редагування та DELETE для видалення [24].

Для забезпечення чистоти коду та легкості підтримки, серверна частина API побудована з використанням адаптованої версії класичного патерну проєктування MVC (Model-View-Controller) [8]. У цій реалізації роль «Представлення» (View) повністю винесена на клієнтську частину, тоді як серверна частина API ділиться на шари маршрутизації, контролерів та логіки. Коли клієнт робить запит, він спочатку потрапляє на шар маршрутизації (Routes), який визначає, до якого функціоналу звертається користувач. Далі запит передається до контролера (Controller), завданням якого є прийняти вхідні параметри та сформувати фінальну відповідь клієнту. Контролер, у свою чергу, не містить складної логіки, а делегує її шару сервісів (Service). Саме в сервісах зосереджена вся бізнес-логіка системи (наприклад, перевірка наявності вільних місць у лікаря перед записом). Сервіси виконують безпосередні запити до бази даних, обробляють отриманий результат і повертають його контролеру для відправки на клієнтську частину.

Опишемо детальніше про процес обробки запитів. Першою ланкою в обробці запиту від клієнтського додатка є шар маршрутизації. Його єдина мета - перехопити HTTP запит за певним URL та перенаправити його до відповідного обробника. Наприклад, для отримання списку лікарів в Express.js створюється маршрут, який прослуховує GET-запити на ендпоінт /api/doctors і викликає відповідний контролер (додаток А).

Далі запит потрапляє до шару контролерів. Завдання контролера полягає виключно в прийомі вхідних даних (параметрів запиту, тіла HTTP-повідомлення), виклику відповідної бізнес-логіки та формуванні відповіді для клієнта. Контролер не повинен напряму звертатися до бази даних (додаток Б).

Вся ключова логіка зосереджена у шарі сервісів. Саме тут виконуються перевірки, форматування даних та безпосередня взаємодія з базою даних. Функцію шару доступу до даних виконує інтегрований клієнт Supabase. Сервіс формує запит

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

до таблиць PostgreSQL, використовуючи методи клієнта, і повертає готовий результат контролеру (додаток В).

Авторизація в системі реалізовується на основі JWT і поєднує перевірку користувача з рольовою моделлю доступу. JSON Web Token — це відкритий індустріальний стандарт (RFC 7519), який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами у вигляді JSON-об'єкта [9]. Дані в такому токени є захищеними та достовірними, оскільки вони мають криптографічний цифровий підпис.

Під час входу користувач передає логін (email або username) і пароль, на сервері виконується пошук облікового запису, перевірка активності акаунта та валідація пароля через його хеш (з використанням прм бібліотеки bcrypt). Після успішної перевірки сервер формує JWT-токен із ключовими даними користувача (ідентифікатор, роль, наявність прав super admin) зі строком дії 12 годин і додатково повертає набір прав доступу по модулях (create/read/update/delete), який зчитується з таблиць ролей і дозволів.

На рівні API всі маршрути (окрім маршруту авторизації) захищені проміжним програмним забезпеченням (middleware) для перевірки токена. Middleware - це компонент або функція, що вбудовується в етап обробки додатку і виконується між отриманням HTTP-запиту від клієнта та його передачею до контролера [10]. Головне завдання middleware полягає у можливості перехоплювати, аналізувати, відхиляти або модифікувати запити до того, як вони досягнуть бізнес-логіки.

У нашій системі управління поліклінікою middleware авторизації вилучає токен із заголовка Authorization (за схемою Bearer) та верифікує його, після чого декодовані дані користувача додаються до контексту поточного запиту. Після успішної перевірки токена застосовується другий рівень контролю доступу - перевірка конкретних прав до визначеного модуля або вимога наявності прав супер-адміністратора. На клієнтській частині після входу токен і дані користувача зберігаються в системі управління станом Redux та локальному сховищі. Локальне

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				33
Змн.	Арк.	№ докум.	Підпис	Дата		

сховище (localStorage) — це механізм Web Storage API, що забезпечує збереження даних у вигляді пар «ключ-значення» безпосередньо у браузері клієнта [11]. Особливістю цього сховища є відсутність часу закінчення терміну дії, інформація не видаляється при закритті вкладки чи браузера. Це дозволяє зберігати сеанс користувача активним і не вимагати повторної авторизації після перезавантаження сторінки додатка

Під час ініціалізації додатку виконується перевірка строку дії токена, і прострочені дані автоматично очищаються. Доступ до сторінок керується за допомогою приватних маршрутів, неавторизовані користувачі перенаправляються на форму входу, а за відсутності права на читання для певного модуля відображається повідомлення про обмежений доступ. Такий підхід забезпечує надійну та синхронізовану безпеку одночасно на сервері та в інтерфейсі користувача.

В системі також важливий момент розробки, це забезпечити оновлення даних на UI в режимі реального часу, а саме на сторінці розкладу. Тобто якщо один користувач змінить статус прийому чи створить новий, то інший має одразу побачити ці зміни без потреби в ручному перезавантаженні сторінки. Для реалізації такого функціоналу, в проєкті використовуємо server-sent events (SSE).

SSE у проєкті використовується як механізм односторонньої передачі подій від сервера до клієнта в режимі реального часу, що усуває необхідність постійного підтягування даних з API з боку клієнтської частини [12]. Технічно це реалізується шляхом відкриття довготривалого HTTP-з'єднання з MIME-типом text/event-stream. Сервер зберігає інформацію про підключених клієнтів і надсилає їм структуровані JSON-повідомлення у форматі подій (на рис 2.7. це можна побачити як «Event»). Для забезпечення стабільності з'єднання застосовується механізм відправки серверного службового повідомлення кожні 25 секунд. У разі закриття вкладки або обриву каналу зв'язку клієнт автоматично та коректно видаляється зі списку активних підписників.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

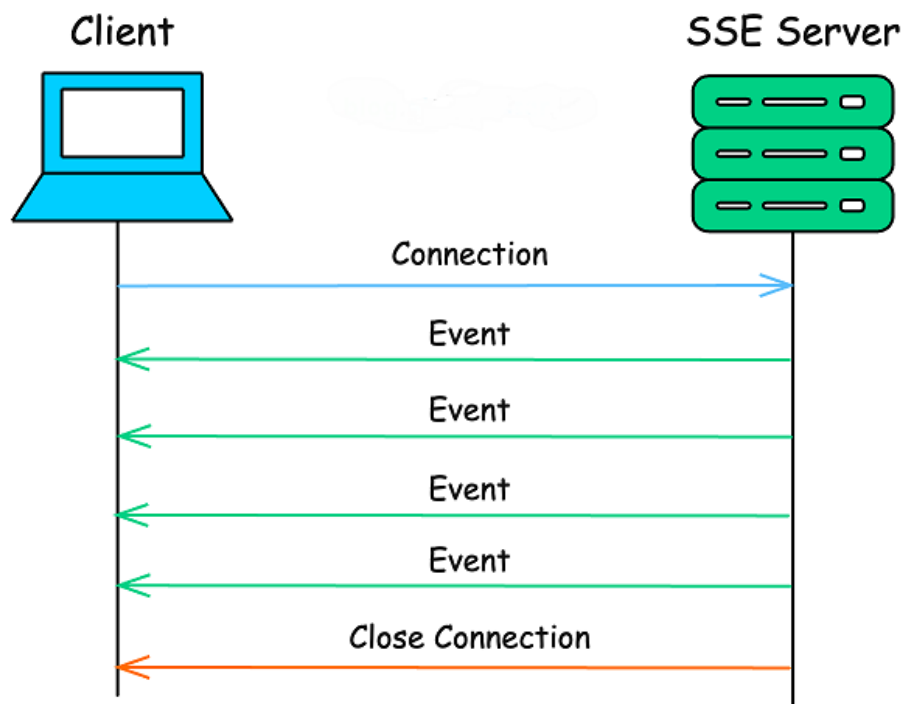


Рис.2.7. Приклад роботи SSE

У системі управління роботи поліклініки технологія SSE застосована в модулі управління розкладом прийомів. Клієнтська частина (сторінка Schedule) ініціює підключення через інтерфейс EventSource до кінцевої точки appointments/stream, передаючи токен авторизації. Фронтенд перехоплює та обробляє події типу CREATE (додавання нового прийому), UPDATE_STATUS (зміна статусу прийому) та TIME_OFF_CREATED (додавання вихідних лікарю), після чого миттєво оновлює глобальний стан Redux. На боці бекенду стрім подій відбувається безпосередньо в момент створення, зміни або скасування відповідного запису. Такий архітектурний підхід гарантує синхронізацію даних між усіма підключеними користувачами майже в реальному часі, суттєво зменшує навантаження на сервер за рахунок мінімізації зайвих GET-запитів та зберігає швидкість інтерфейсу системи.

Клієнтська частина системи побудована за модульним принципом із розподілом відповідальності між архітектурними рівнями, що забезпечує високу читабельність коду та зручність його масштабування. Рівень сторінок (Pages)

відповідає за реалізацію окремих бізнес-сценаріїв (розклад, управління лікарями та пацієнтами, адміністрування тощо) і забезпечує загальну композицію інтерфейсу.

Рівень компонентів містить багаторазові UI-елементи та незалежні локальні блоки, з яких конструюються сторінки та різні секції. Допоміжна обчислювальна логіка винесена в окремі утиліти та кастомні React-хуки, що дозволяє уникнути дублювання коду та централізувати правила обробки даних.

Для управління спільними даними застосовано концепцію глобального стану за допомогою бібліотеки Redux. Глобальний стан у контексті фронтенд-архітектури - це централізоване сховище даних, доступ до якого може отримати будь-який компонент застосунку незалежно від його рівня вкладеності в ієрархії інтерфейсу. Використання цього патерну вирішує проблему багаторівневої передачі властивостей (prop drilling) та надмірної зв'язності компонентів.

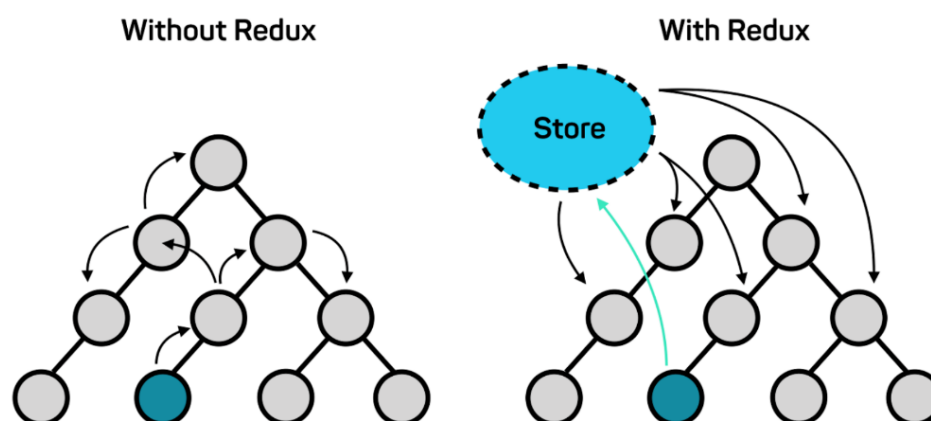


Рис.2.8. Приклад вирішення проблеми надмірної передачі властивостей (props drilling) завдяки Redux

Redux, у свою чергу, виступає як передбачуваний контейнер стану для JavaScript-додатків, зокрема з використанням сучасного стандарту Redux Toolkit [20]. Він забезпечує єдине джерело істини для всього клієнтського застосунку та гарантує, що будь-які зміни інформації відбуваються виключно за правилами через однонаправлений потік даних.

Принцип роботи цього механізму наочно проілюстровано на рисунку 2.4., де зображено базову архітектуру однонаправленого потоку даних у Redux. Життєвий

цикл оновлення стану складається з таких етапів як - інтерфейс користувача (UI), взаємодія користувача з візуальними елементами додатку (наприклад, натискання кнопки) ініціює спрацьовування обробника подій (Event Handler).

Далі це диспетчеризація (Dispatch) та Дія (Action) - обробник подій генерує та відправляє (dispatch) спеціальний об'єкт - подію (Action). Цей об'єкт обов'язково містить тип дії (type) та може включати корисне навантаження (payload) з новими даними.

Сховище (Store) та Редюсер (Reducer) описує згенеровану подію, яка потрапляє до центрального сховища (Store), де передається у редюсер (Reducer). Редюсер - це чиста функція, яка приймає поточний стан та отриману дію, обчислює необхідні зміни і повертає абсолютно новий об'єкт стану.

Останній йде оновлення стану (State), щойно стан у сховищі оновлюється, клієнтський інтерфейс (UI) автоматично отримує ці нові дані та оновлює відповідні компоненти, відображаючи актуальні дані та замикаючи цикл взаємодії.

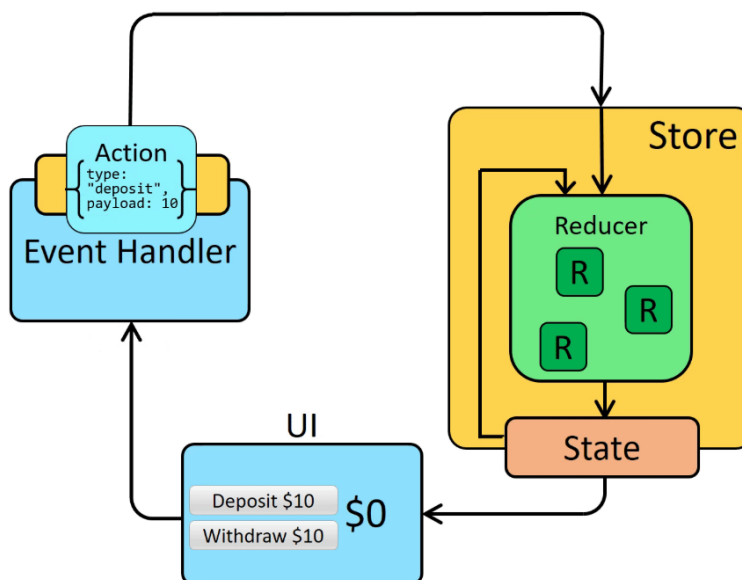


Рис.2.9. Приклад роботи Redux

Взаємодія з даними у нашій системі керування роботою поліклінки побудована наступним чином - дані з сервера завантажуються за допомогою асинхронних функцій (thunks), зберігається в окремих модулях (slices), а UI-

компоненти отримують доступ лише до необхідних їм фрагментів стану через функції-селектори. Така архітектурна структура робить інтерфейсу стабільним завдяки зрозумілому патерну доступу даних та спільному сховищу, полегшує процес тестування та дозволяє ітеративно нарощувати функціональність без ризику порушення цілісності інших частин системи.

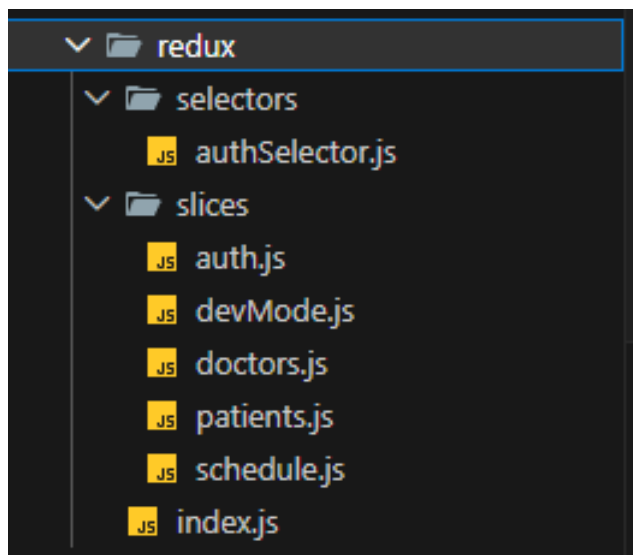


Рис.2.10. Структура redux у системі

Для реалізації інтерфейсу базових компонентів в клієнтській частині системи використовується MUI (Material UI) як бібліотеку UI-компонентів, що спирається на принципи системи Material Design. Це дозволяє швидко будувати консистентний дизайн із єдиною візуальною логікою, адаптивністю та великою вибіркою готових елементів (інпути, модальні вікна, таблиці, кнопки, індикатори завантаження тощо), суттєво зменшивши обсяг ручної верстки. Налаштування стилістики виконано через конфігурацію об'єкта theme та використання sx-властивостей (замість звичайного style у тегів), завдяки чому візуальне оформлення є централізованим, легко підтримується й може швидко модифікуватися чи оновлюватися без необхідності переписування самих компонентів. Також стосовно sx-властивостей, оскільки це не просто CSS стилі, було вирішено прописувати їх як JavaScript об'єкт, то підтримує зручне використання умовних параметрів (наприклад ситуація де треба задавати різні стилі в залежності від стану

компонента чи секції) та легке виокремлення стилів від логіки коду, що додає до читабельності коду.

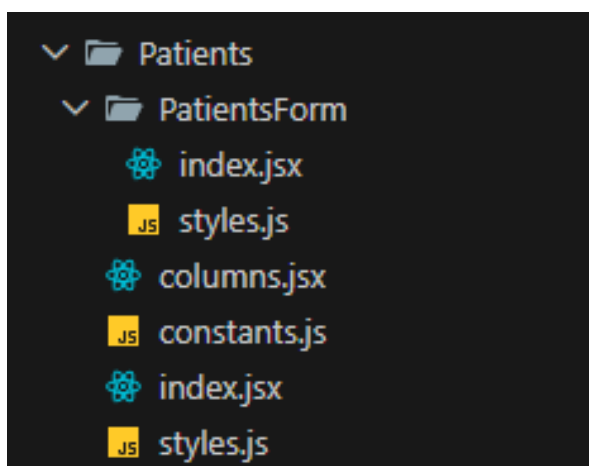


Рис.2.11. Приклад структури компонента, виокремлення стилів у styles.js

У підсумку, інтеграція MUI значно підвищила швидкість розробки, покращила користувацький досвід за рахунок стандартизованої та передбачуваної поведінки елементів. Також MUI спрощує масштабування інтерфейсу під час додавання нових функціональних модулів системи.



Рис.2.12. Material UI

Також у системі реалізовується багаторівневу валідацію даних, що дозволяє виявляти та запобігати помилкам ще до етапу виконання запитів до бази даних. На серверній частині контролери перевіряють наявність обов'язкових полів, коректність форматів вхідних параметрів, статус активності облікового запису, правильність пароля та наявність відповідних прав доступу. У разі невідповідності

даних встановленим вимогам, сервер генерує відповідний HTTP-статус (наприклад, 400 Bad Request або 403 Forbidden) із детальним повідомленням про помилку, що дозволяє клієнтській стороні коректно обробити таку ситуацію. Для тестування подібних запитів до виключно сторони API використовуємо додаток Postman. Postman - це спеціалізована програмна платформа та популярний графічний клієнт для розробки, тестування, документування та моніторингу API [21].

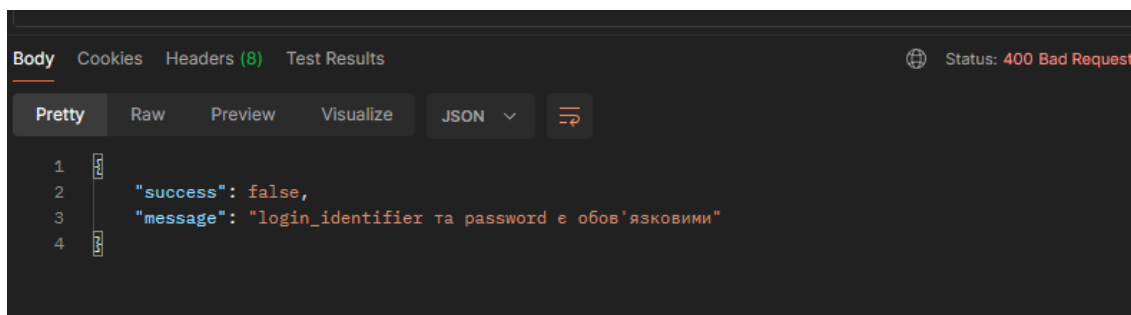


Рис.2.13. Спроба невдалого запиту до API

На клієнтській частині механізми валідації інтегровано безпосередньо у форми створення та редагування медичних записів, профілів пацієнтів і розкладу прийомів. Це на програмному рівні блокує можливість відправки запиту, якщо обов'язкові поля не заповнені або містять некоректні значення.

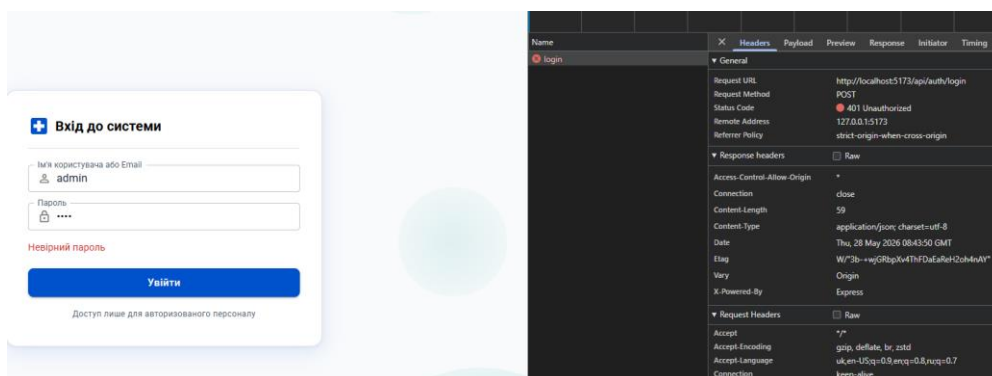


Рис.2.14. Спроба невдалого запиту зі сторони UI, вивід помилки на інтерфейсі

Такий двосторонній підхід суттєво підвищує надійність системи, знижує навантаження на сервер через зменшення кількості хибних запитів. Також покращує UX завдяки миттєвому інформуванню про помилки введення.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

Висновки до другого розділу

У ході виконання другого розділу було визначено архітектуру бази даних, використання та зв'язки між таблицями. Було визначено варіанти використання системи, можливі ролі для користувачів додатку, які будуть використовуватись в роботі системи, було побудовано та проілюстровано діаграму варіантів використання та діаграми діяльності до головних алгоритмів у системі.

Також були обґрунтовано та описано основні моменти налаштування та роботи серверного додатку, було показано основні технології, які використовуються в побудові CRM-системи для управління роботи поліклінікою.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

РОЗДІЛ 3 ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ

3.1. Порядок встановлення та налаштування параметрів системи

З метою візуалізації апаратної інфраструктури та розміщення виконуваних програмних модулів було побудовано діаграму розгортання (рис. 3.1). Ця модель детально ілюструє базову клієнт-серверну архітектуру, на якій ґрунтується розроблена система для управління поліклінікою.

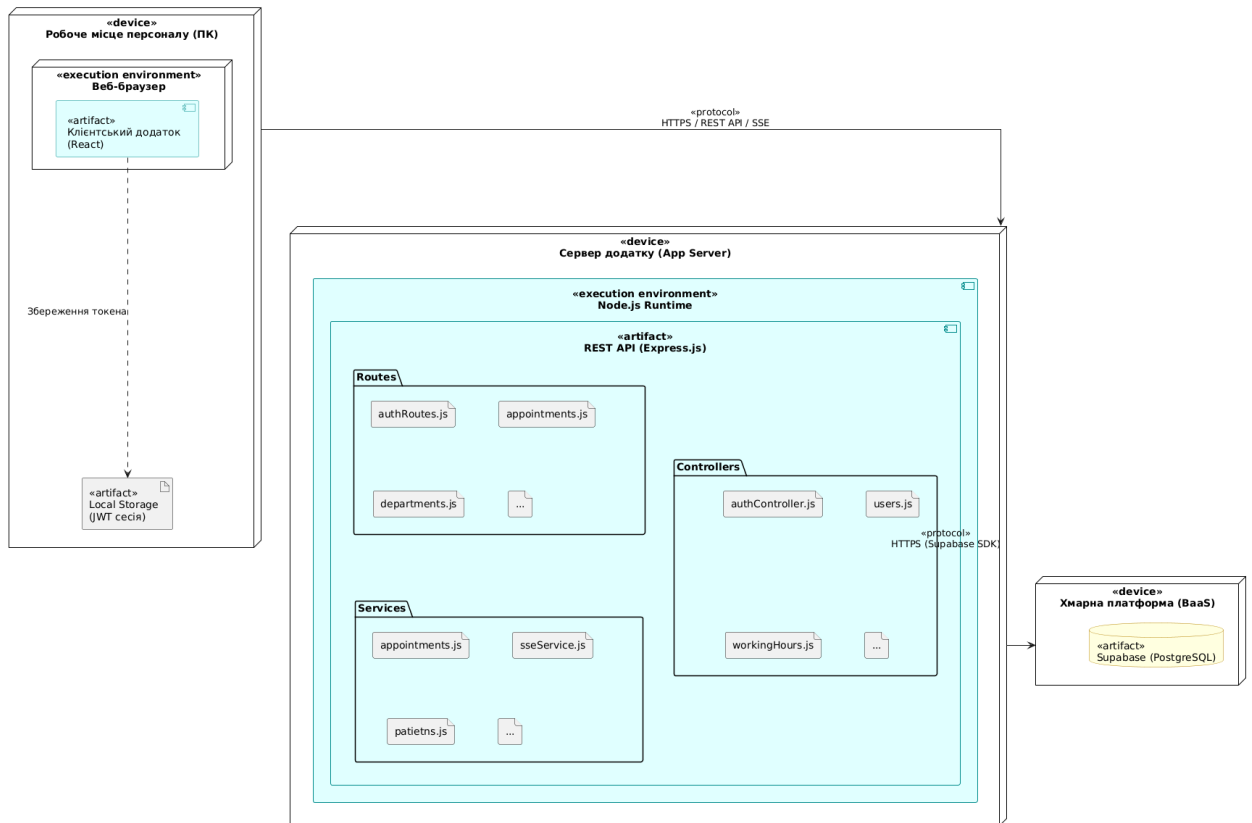


Рис.3.1. Діаграма розгортання

Розглянемо основні вузли системи та їхню взаємодію. Першим блоком є робоче місце персоналу (ПК), цей вузол репрезентує фізичний пристрій користувача (лікаря, реєстратора або адміністратора). Середовищем виконання виступає стандартний веб-браузер, у якому функціонує клієнтський додаток, розроблений за допомогою бібліотеки React. Для локального збереження даних сеансу (зокрема JWT-токенів для авторизації) використовується вбудований механізм браузера локального сховища (Local Storage).

Далі це сервер додатку (App Server), центральний обчислювальний вузол, на якому розгорнуто серверну частину системи. Програмним середовищем виконання є Node.js Runtime. На базі цього середовища функціонує REST API додаток, написаний з використанням фреймворку Express.js. Внутрішня архітектура бекенду розділена на три логічні шари, такі як Routes (Маршрутизатори) - відповідають за обробку HTTP-запитів та їхнє перенаправлення; Controllers (Контролери) - обробляють вхідні дані та керують потоком виконання запиту; Services (Сервіси) - містять безпосередню бізнес-логіку системи (робота з розкладом, авторизація тощо). Взаємодія між клієнтським ПК та сервером додатку здійснюється через захищений протокол HTTPS з використанням архітектури REST та технології Server-Sent Events (SSE) для трансляції оновлень.

Та останній це сервер бази даних - фізичний або віртуальний виділений сервер, що містить реляційну базу даних. Цей вузол відповідає за безпечне та надійне зберігання всієї медичної інформації, даних користувачів та розкладу. Взаємодія між сервером додатку та базою даних відбувається за допомогою офіційного Supabase SDK через протокол HTTPS. Такий чіткий розподіл на незалежні рівні забезпечує високу масштабованість системи, простоту її розгортання на хостингу та полегшує подальший технічний супровід і тестування кожного компонента окремо.

Для успішного розгортання та коректної роботи програмного комплексу необхідно підготувати робоче середовище та виконати ряд послідовних кроків з налаштування. Оскільки розроблена CRM-система для управління поліклінікою базується на сучасних веб-технологіях, першочерговим етапом є встановлення базового програмного забезпечення на сервер або локальний комп'ютер розробника. Головною вимогою є наявність встановленого середовища виконання Node.js, яке забезпечує роботу як серверної частини, так і інструментів збирання для клієнтського додатка. Разом із середовищем Node.js автоматично встановлюється пакетний менеджер npm, який є важливим для завантаження та управління всіма бібліотеками проекту.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Деталізуючи процес підготовки середовища, варто зазначити, що робота з пакетним менеджером npm повністю автоматизує процес збирання залежностей завдяки конфігураційному файлу `package.json`. Коли розробник запускає команду встановлення, система самостійно аналізує цей маніфест, створює в корені проекту локальну директорію `node_modules` та завантажує туди всі необхідні сторонні бібліотеки. Такий підхід гарантує, що на будь-якому комп'ютері персоналу або локальному сервері поліклініки буде розгорнуто абсолютно ідентичні версії програмних модулів, що дозволяє уникнути конфліктів сумісності та забезпечує стабільну роботу як клієнтської, так і серверної частин системи.

Після підготовки базового середовища процес розгортання починається з налаштування серверної частини системи. Необхідно відкрити термінал, перейти до кореневої папки API та ініціювати процес завантаження залежностей за допомогою відповідної команди пакетного менеджера. Наступним і найважливішим кроком є конфігурація змінних оточення. Для цього створюється спеціальний файл конфігурації, у якому обов'язково вказуються секретні ключі доступу до хмарної бази даних Supabase, унікальний рядок для генерації та перевірки токенів безпеки, а також порт локального сервера. Без правильного налаштування цих параметрів серверна частина не зможе встановити зв'язок з базою даних та автентифікувати користувачів.

Особливу увагу при налаштуванні параметрів слід приділити саме конфігураційним файлам оточення з розширенням `env`, які створюються окремо для бекенду та фронтенду. Використання таких файлів є важливим стандартом безпеки у сучасній веброзробці, оскільки вони дозволяють повністю відокремити конфіденційні дані та системні налаштування від основного вихідного коду програми.

Після збереження конфігурації сервер запускається спеціальною командою, і в терміналі з'являється повідомлення про успішне з'єднання. Команди для запуску API в режимі розробника або команди для збирання та запуску проекту

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

розташовані у файлі package.json, в якому по суті є вся конфігурація проекту для бекенду та фронтенду відповідно.

Налаштування клієнтської частини, яка реалізована за допомогою бібліотеки React, відбувається за аналогічним сценарієм. У терміналі необхідно перейти до папки фронтенду та виконати команду встановлення всіх необхідних пакетів для візуального інтерфейсу. Клієнтський додаток також потребує свого файлу змінних оточення, де прописується базова адреса раніше запущеного локального сервера, щоб фронтенд розумів куди саме відправляти мережеві запити. Що стосується бази даних, оскільки використовується хмарне рішення Supabase, відпадає необхідність локального встановлення та налаштування системи управління базами даних PostgreSQL. Замість цього розробнику достатньо зайти в панель управління платформою та виконати підготовлені SQL скрипти для створення необхідних таблиць, налаштування політик безпеки та внесення початкових тестових даних. Після виконання всіх цих кроків система управління поліклінікою повністю готова до використання, і проект можна відкрити у браузері за локальною адресою для перевірки працездатності інтерфейсу та його коректної взаємодії із серверною логікою.

Завершальний етап налаштування параметрів передбачає первинну перевірку взаємодії всіх вузлів через інтерфейс браузера. Після одночасного запуску серверної та клієнтської частин, розробник відкриває локальну адресу в браузері, де автоматично завантажується стартова сторінка CRM-системи, яка є формою для автентифікації. На цьому етапі додаток здійснює перший автоматичний запит до сервера для перевірки працездатності маршрутів, а сервер, у свою чергу, перевіряє таблиці бази даних. Якщо всі змінні оточення та SQL-скрипти було застосовано без помилок, система успішно обробить авторизацію користувача після введення правильних даних для цього та після відобразить головний екран із розкладом лікарів, що свідчить про повну готовність системи до подальшого наповнення реальними даними поліклініки.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2. Структура інтерфейсу системи для управління роботи поліклініки

Інтерфейс системи є ключовий фактором функціонування системи, адже саме інтерфейс відіграє головну роль, як саме буде керуватись робота поліклініки. Проектування бази даних, передбачення ролей саме тут розкриється для ефективної роботи працівників поліклініки, що забезпечує ефективну роботу.

Отже, почнемо огляд системи з авторизації працівника. На дані сторінці маємо невеличку форму входу, де працівник згідно до свого акаунта може увійти в систему (рис. 3.2.). В огляді роботи систем увійдемо під роллю головного адміністратора з повним доступом, щоб побачити всі головні сторінки.

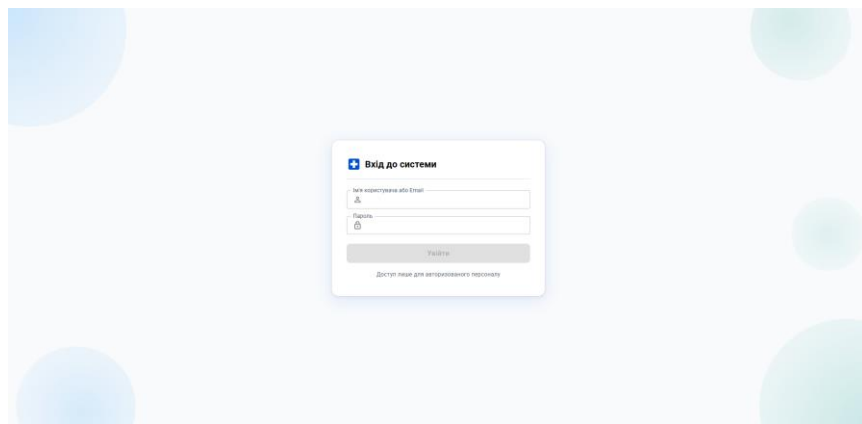


Рис.3.2. Сторінка входу до системи

Після авторизації ми бачимо інтерфейс системи який доступний після авторизації, показано модулі згідно до доступу ролі авторизованого користувача. Інтерфейс побудований такий чином, що у нас завжди будуть присутні головні елементи на будь якій сторінці, а саме бокова навігація, меню яке можна зменшити та збільшити, та хедер зверху сторінки, для додаткового розуміння на якій сторінці знаходиться працівник системи, також можливість вийти з системи.

Загалом після успішної авторизації ми переходимо головну сторінку системи, а саме модуль розкладу (рис.3.3.). Сторінка розкладу це основне місце роботи працівника реєстратури, де він може записувати на прийомі пацієнтів на відповідний час до певного лікаря. На сторінці бачимо поточну дату та час, секція фільтрів для ефективної роботи з розкладом та головну таблицю розкладу.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

Секція фільтрів містить в собі вибір дати, сортування за відділенням, пошук та сортування розкладу за лікарем, пошук прийомів за пацієнтом. Таблиця розкладу побудована таким чином – маємо розбиття по відділенням, та рядки з можливими місцями прийомів лікаря, згідно робочого графіку. Таблиця містить всю потрібну інформацію, яка може знадобитись працівнику реєстратури для того, щоб надати пацієнту поточну інформацію можливих варіантів запису варіантів на прийом до лікарів які працюють у конкретний день.

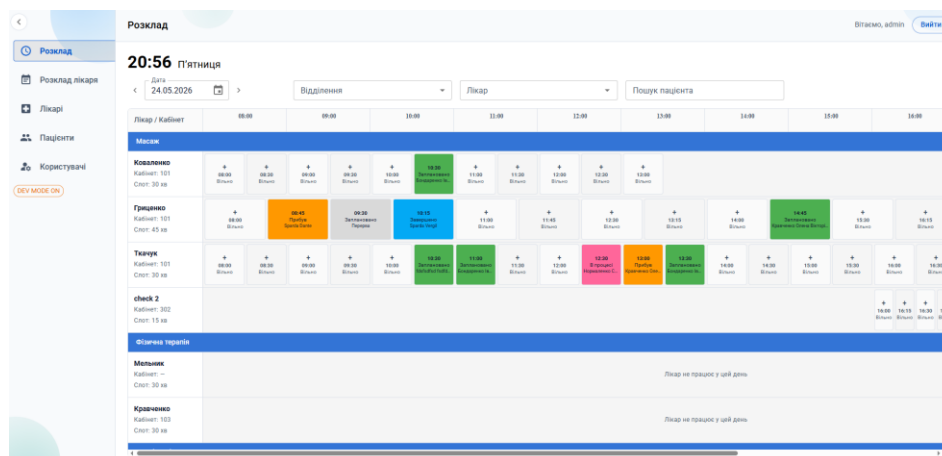


Рис.3.3. Розклад лікарів, сторінка записів на прийоми

Запис на прийом може відбутися після натиску на відповідний вільний слот прийому. У разі виконання такої дії, ми побачимо модальне вікно (рис.3.4.), яке містить форму запису пацієнта на прийом. Працівник реєстратури може обрати наявного пацієнта або додати нового, також може додати нотатки за необхідністю. Оскільки це форма створення запису, то статус прийому автоматично відзначається як запланований.

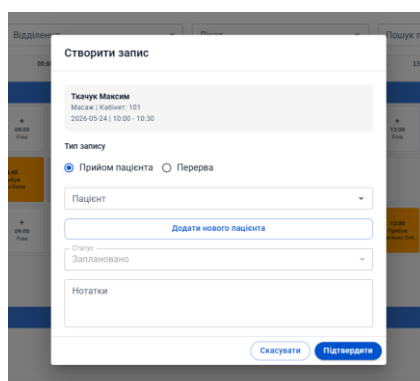


Рис.3.4. Модальне вікно запису на прийом

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

Також модальне вікно створення запису має опцію типу запису, в системі передбачена системи перерв для лікарів (рис.3.5.). Тобто лікар може за необхідністю мати перерву і це буде відповідно відомо для працівника реєстратури. У разі обрання опції перерви, інтерфейс сховає всі попередні поля для заповнення даних і залишить лише не обов’язкове поле для заповнення нотаток.

Рис.3.5. Модальне вікно створення запису, обрана опція перерви

У разі натиску на створений запис прийому, додається можливість змінити статус слоту (рис.3.6.). Для прикладу, працівник реєстратури бачить, що пацієнт прийшов в поліклініку та відповідно повідомив про це, в такому разі статус має вручну змінитись, для цього є відповідне меню у формі. Також у разі редагування запису, ми можемо загалом видалити запис у разі необхідності.

Рис.3.6. Модальне вікно редагування запису

Створені записи візуально відрізняються відповідно до статусу (рис.3.7.), що полегшує працівнику реєстратури пошук потрібного запису. Також слід зазначити, що таблиця розкладу адаптивно підлаштовується під розклад лікаря та його час прийомів, зазвичай час прийому лікаря 30 хвилин, але також можливі й інші варіанти і це коректно показує UI.

:00	10:00	11:00	12:00	13:00	
+ 09:30 Вільно	+ 10:00 Вільно	10:30 Заплановано Бондаренко Ів...	+ 11:00 Вільно	+ 11:30 Вільно	+ 12:00 Вільно
			+ 12:30 Вільно	+ 13:00 Вільно	
09:30 Заплановано Перерва	10:15 Завершено Sparda Vergil	+ 11:00 Вільно	+ 11:45 Вільно	+ 12:30 Вільно	+ 13:15 Вільно
+ 09:30 Вільно	+ 10:00 Вільно	10:30 Заплановано fdfsdfsd fsd...	11:00 Заплановано Бондаренко Ів...	+ 11:30 Вільно	+ 12:00 Вільно
			12:30 В процесі Нормаленко С...	13:00 Прибув Кравченко Оле...	13:30 Заплановано Бондаренко Ів...
					14:00 Вільно

Рис.3.7. Варіанти записів

Перейдемо до огляду інших сторінок. Сторінка лікарів системи (рис.3.8.), в даному модулі присутні такі основні елементи як секція фільтрів (пошук лікаря, фільтрування даних таблиці за відділенням та статусом), можливість додавання нового лікаря та таблиця за даними. Таблиця дає змогу передивитись всю основну інформацію про лікаря, яка може бути потрібна системі управління роботою поліклініки.

Розклад Розклад лікаря Лікарі Пациєнти Користувачі	Лікарі				
	Пошук за прізвищем, іменем або кабінетом...				Вітасмо, адмін Вийти
	Додати лікаря				
	Усі відділення Усі статуси				
	ПІБ	Відділення	Кабінет	Статус	Дії
	Коваленко Олександр Миколайович	Масаж	101	Активний	О Д В
	Гриченко Юлія Володимирівна	Масаж	101	Активний	О Д В
	Ткачук Максим Олександрович	Масаж	101	Активний	О Д В
	Мельник Тарас Ігорович	Фізична терапія	-	Неактивний	О Д В
	Кравченко Олена Сергіївна	Фізична терапія	102	Активний	О Д В
	Бойко Дмитро Андрійович	Кардіологія	201	Активний	О Д В
	Лисенко Наталя Вікторівна	Кардіологія	202	Активний	О Д В
	Поліщук Анатолій Степанович	Ортопедія	102	Активний	О Д В
	check 2 check check	Масаж	302	Активний	О Д В
	Розмір сторінки: 10 Сторінка 1 з 9				

Рис.3.8. Сторінка лікарів

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

В колонці «Дії» присутні 4 можливих варіантів взаємодії з даними. Перша це переглянути графік лікаря (рис.3.9.). При натиску відкривається бокова панель, яка містить список днів тижня з налаштований графіком роботи лікарів.

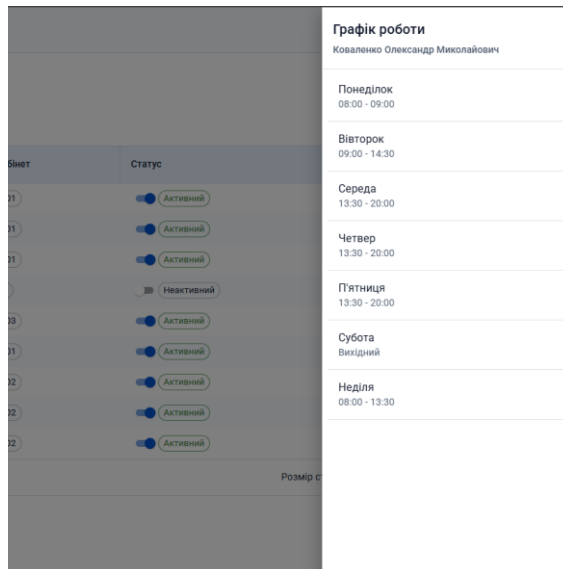


Рис.3.9. Бокова панель графіку лікаря

Далі йде можливість додати вихідний день або відпустку до лікаря (рис.3.10.). Оскільки в системі є можливість додавати дні коли лікар не працює, то така можливість на UI є корисною та зручною для адміністратора. Модальне вікно містить в собі дату початку та кінця днів відсутності лікаря, а також причина, яка буде відповідно показуватись на головній сторінці розкладу. Також присутня секція перевірки накладання вихідних днів на робочий графік лікаря, тобто присутня валідація, що не можна додати вихідний на день, де вже є активні записи.

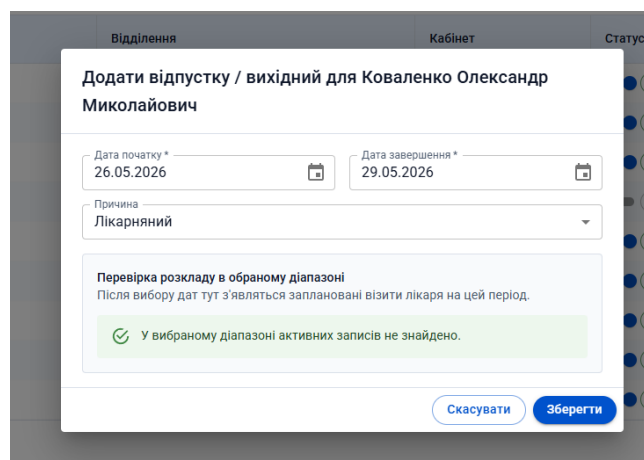


Рис.3.10. Модальне вікно додавання вихідних лікарю

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

Розглянемо модальне вікно додавання та редагування даних про лікаря. Дане вікно містить форму з персональним полями для лікаря та професійними (відділення, кабінет тощо, зображено на рис.3.11.). Також присутня секція робочих годин лікаря. Маємо список днів тижня, та опції вибору початку та кінця робочу дня, який підбирається під тривалість прийому лікаря та під початок робочого дня.

Редагувати лікаря

Прізвище

Коваленко

Ім'я

Олександр

По батькові

Миколайович

Відділення

Масаж

Кабінет

101 - Стандартний кабінет (Спільно з: Гриценко Ю., Ткачук М.)

Тривалість прийому (хв)

30

Активний

☒

Робочі години

Понеділок

08:00

09:00

Слотів: 2

Вівторок

09:00

14:30

Слотів: 11

Середа

13:30

20:00

Слотів: 13

Четвер

Слотів: 13

Скасувати

Зберегти

Рис.3.11. Модальне вікно редагування даних лікаря

Перейдемо до модуля пацієнтів, дана сторінка схожа на модуль пацієнтів (рис.3.12.), має схожу структури побудови UI. На сторінці пацієнтів інтерфейс показує пошук, кнопка додавання, яка відкриє модальне вікно з додаванням нових даних та таблицю з основними даними, що включає в себе контактну інформацію про пацієнта.

Пацієнти

Вітасмо, admin

Вийти

Додати

Пошук за прізвищем, іменем або телефоном...

ПІБ	Дата народження	Стать	Телефон	Пошта	Дії
Check Patient dasdasda	-	-	-	-	
Check Patient dasdasda	-	-	-	-	
fsdfsdfsdfsdfsdfsdfsdfs	06.11.2025		fsdfsdf	fsdfsdf	
Sparda Vergil	-		-	-	
Sparda Dante	-		-	-	
Бондаренко Іван Сергійович	15.04.1985		+380501234567	ivan.b@example.com	
Кравченко Олена Вікторівна	22.08.1992		+380679876543	olena.k@example.com	
Нормаленко Степан Чік	-	-	-	-	
Яцюк Ліна Вячеславівна	12.06.2026		4556757879	34435667	

Розмір сторінки: 10 Сторінка 1 з 9

Рис.3.12. Сторінка пацієнтів

Модальне вікно створення або редагування даних про пацієнтів (рис.3.13.) містить основні поля, які потрібні для додавання пацієнта до системи управління роботи поліклініки. Модальне вікно має декілька варіантів виходу – збереження даних, кнопка «Скасувати» або клік на місце поза модальним вікном.

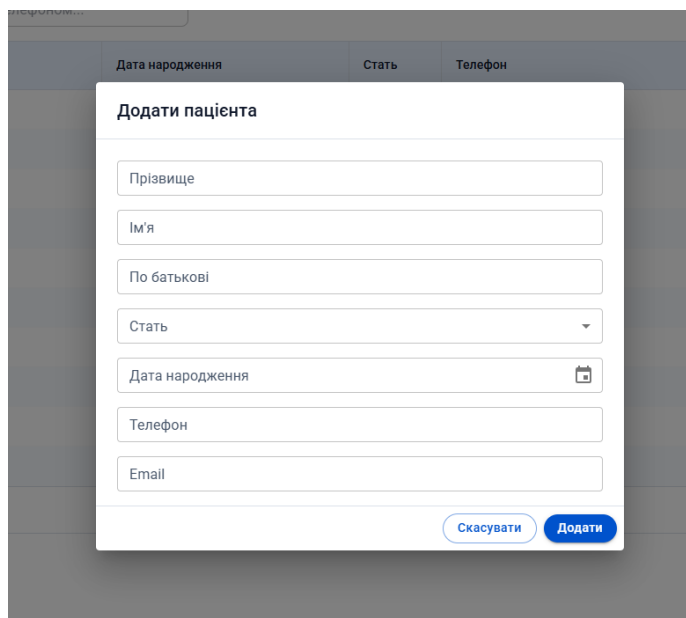
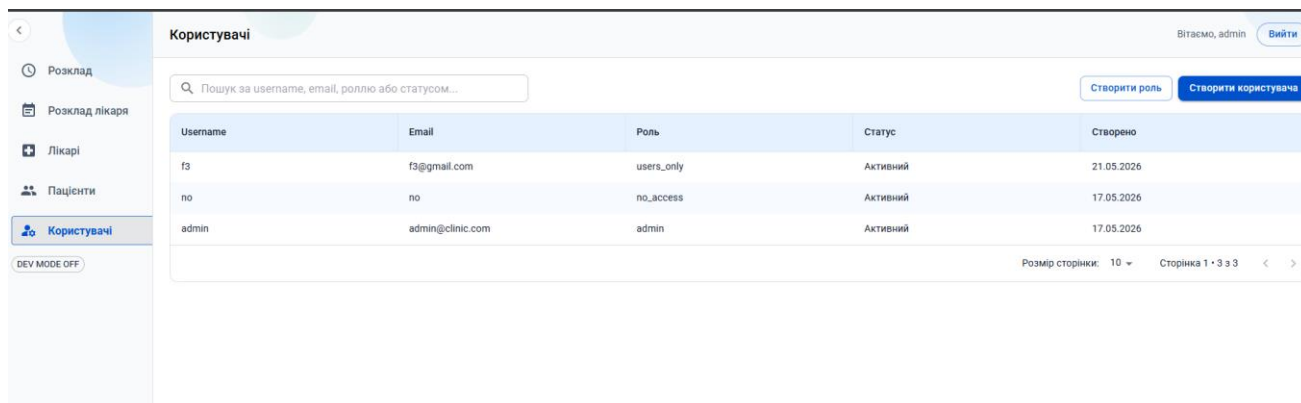


Рис.3.13. Модальне вікно додавання пацієнта

Сторінка користувачів (рис.3.14.) також побудована схожим чином, пошук та таблиця, та можливість додати нову роль для подальшого додавання її користувачу та відповідно кнопка додавання нового користувача до системи, який матиме можливість авторизуватись. Даний модуль доступний лише адміністратору, тут він може створити обліковий запис для працівника лікарня, додати потрібну роль до акаунту.



Username	Email	Роль	Статус	Створено
fs	fs@gmail.com	users_only	Активний	21.05.2026
no	no	no_access	Активний	17.05.2026
admin	admin@clinic.com	admin	Активний	17.05.2026

Рис.3.14. Сторінка користувачів системи

Після натиску на кнопку додавання нової ролі відкриється модальне вікно для створення нової ролі. Модальне вікно містить модулі, які використовуються в системі та відповідне налаштування доступу до цих модулів (рис.3.15.). Інтерфейс побудований таким чином, ми маємо модуль та відповідно чотири доступу до нього, які ми можемо обрати до цієї ролі.

Модуль	Створення	Читання	Редагування	Видалення
ПАЦІЄНТИ ТА ПРИЙОМ				
Пацієнти	☐	☐	☐	☐
Записи на прийом	☐	☐	☐	☐
Медичні записи	☐	☐	☐	☐
ЛІКАРІ				
Лікарі	☐	☐	☐	☐
Розклад лікарів	☐	☐	☐	☐
Адміністрування				
Користувачі	☐	☐	☐	☐

Рис.3.15. Модальне вікно створення ролі

В системі управління роботи поліклінікою також присутня сторінка з більш детальною інформацією про пацієнта (рис.3.16.). Даний модуль містить загальну інформацію (ту саму, що і таблиці на сторінці пацієнта), історію візитів та сторінку медичних даних, доступ до якої має лише лікар.

Дата	Час	Лікар	Статус	Тип
24.05.2026	14:45 - 15:30	Гриценко Юлія Володимирівна	Заплановано	Консультація
24.05.2026	13:00 - 13:30	Ткачук Максим Олегович	Прийнято	Консультація
21.05.2026	10:00 - 10:30	Ткачук Максим Олегович	Заплановано	Консультація
21.05.2026	09:00 - 09:30	Ткачук Максим Олегович	Закінчено	Консультація
17.05.2026	08:45 - 09:30	Гриценко Юлія Володимирівна	Прийнято	Консультація
01.05.2026	11:15 - 12:00	Гриценко Юлія Володимирівна	Прийнято	Консультація
23.04.2026	10:30 - 11:15	Гриценко Юлія Володимирівна	Заплановано	Консультація
19.04.2026	08:30 - 09:00	Коваленко Олександр Миколайович	Скасовано	Consultation
10.04.2026	09:00 - 09:30	Коваленко Олександр Миколайович	Закінчено	Первинна консультація (Масаж)

Рис.3.16. Модуль детальної інформації пацієнта, сторінка історії візитів

Інтерфейс медичних даних, такі як симптоми, діагноз або рекомендації, призначення лікування або приблизний план дій пацієнта доступний лише для лікаря (рис.3.17.). Інтерфейс показує список з історією медичних записів, де ми можемо переглянути попередні записи, а також блок з додавання нових даних відповідно до пацієнта та прийому. Також маємо панель, яка показує поточний системі, адже в процес роботи запланований таким чином, що лікар також маж можливість керувати статусом прийому, тобто змінити його з умовно «Прибув» до «В процесі».

Кравченко Олена Вікторівна

Телефон: +380679876543 | Email: olena.k@example.com | Дата народження: 22.08.1992

РЕЕСТРАТОР

ЛІКАР

Повернутися назад

ЗАГАЛЬНА ІНФОРМАЦІЯ

ІСТОРІЯ ВІЗИТІВ

МЕДИЧНА КАРТКА

Поточний візит: 24.05.2026 13:00 - 13:30

Статус

Прибув

Історія медичних записів

21.04.2026

Симптоми: Біль

Діагноз: Болить

Призначення: Потерпіти

10.04.2026

Симптоми: Скарги на гострий біль у попереку після фізичного навантаження. Біль віддає у праву ногу.

Діагноз: M54.5 Біль у нижній частині спини (Лямбаго). Остеохондроз поперекового відділу.

Призначення: 1. Уникати підняття важких предметів. 2. Курс лікувального масажу (10 сеансів). 3. Мазь Диклофенак локально 2 рази на день.

Новий запис

Симптоми

Діагноз

Призначення

Зберегти

Рис.3.17. Медична картка пацієнта

Також в інтерфейсі присутні всі інформаційні моменти, такі як завантаження, сповіщення про такі дії як створення, редагування, вивід помилок заповнення форм тощо. Додавання таких моментів суттєво покращує UX у системі, користувач системи після створення нової сутності, наприклад після додавання нового запису (рис.3.18.), буде бачити відповідне повідомлення про успішність чи помилку після своїх дій, що дає зрозуміти, що система коректно реагує на дії користувача.

✓

Запис успішно створено

Рис.3.18. Сповіщення про успішно створений запис на прийом

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

Отже система має зрозумілий UI, який коректно працює з даними з API, всі процеси правильно обробляються тощо. Користувацький досвід продуманий, всі потрібні інформативні показники присутні в системі.

3.3. Тестування роботи системи

Опишемо тестування системи управління роботи поліклінікою. Оскільки інтерфейс, фронтенд частина це місце з яким працюють працівники, перевіримо його на наявність ймовірних помилок. Загалом можливі помилки можуть бути з сторони заповнення форм, накладання даних, помилки з серверу.

В системі присутня валідація форм, почнемо перевірку з форми реєстрації (рис.3.19). При спробі ввести неправильні дані або неіснуючі дані, буде виведено помилку авторизації, тобто ніякого доступу до системи не буде надано.

Рис.3.19. Спроба авторизації з неправильним даними

Спроба запису на прийом на минулий день неможлива. Маємо враховувати механічні помилки працівник реєстратури, тож цей ймовірний сценарій пророблено. У разі спроби запису, UI показує відповідне допоміжне текстове повідомлення про неможливість дії (рис.3.20). Також візуально такий слот прийому має зменшену прозорість.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

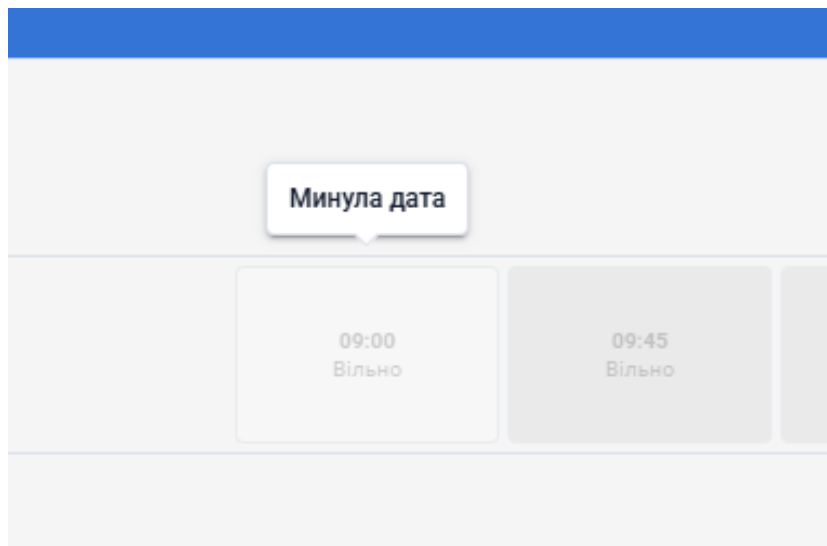


Рис.3.20. Спроба запису на минулу дату

Форма створення та редагування запису на прийом містить перевірку на обов'язкові поля. Якщо відкрити модальне вікно запису на прийом, та не обрати пацієнта (в даному випадку це єдине обов'язкове поле), і після цього натиснути на кнопку збереження, то маємо вивід допоміжного червоного тексту під полем вводу, відповідно до обов'язково поля (рис.3.21.). Отже в даному випадку, ми не можемо зберегти з UI пустий запис, та маємо відповідний візуальний показ стосовно цього, що вказує на конкретну причину помилки.

Рис.3.21. Спроба запису на прийом без пацієнта

Форма додавання лікаря у систему також містить обов'язкові поля, без яких створення лікаря не буде можливим. Після відкриття модального вікна для додавання записів нового лікаря, якщо нічого не заповнюючи, натиснути на кнопку збереження даних, то побачимо неможливість такої дії, та помилку у відповідних пропущених полях (рис.3.22.). Поля які є опціональними відповідно не включаються до валідації та не мають тексту з причиною помилки.

Рис.3.22. Спроба додавання лікаря без заповнених полів у формі

На сторінці пацієнтів, у формі додавання пацієнта до системи також оброблено подібний сценарій. Після відкриття модального вікна, яке містить форму для додавання даних пацієнта, у разі натиску на кнопку збереження даних, без заповнених полів, ми побачимо відповідні помилки (рис.3.23.). На такі поля як email та телефон діють додаткові правила валідації.

Рис.3.23. Спроба додавання пацієнта без заповнених полів у формі

У формі заповнення даних для нового пацієнта є поля телефону та пошти, вони не є обов’язковими, але все ж у разі додавання таких даних, на ці поля діє валідація. У разі невірнього формату введення ми побачимо відповідну помилку у модальному вікні додавання нових даних про пацієнта (рис.3.24.). Це є додаткова валідація в формі пацієнта, таким чином користувач не може ввести неможливий номер телефону або пошту.

The screenshot shows a modal window for adding new patient data. At the top, there is a date field with the value '04.11.1997'. Below it are two input fields: 'Телефон' (Phone) containing 'adsad' and 'Email' containing 'dsadas'. Both fields are outlined in red, indicating validation errors. Below the phone field, the text 'Вкажіть коректний номер телефону' (Specify a correct phone number) is displayed. Below the email field, the text 'Вкажіть коректний email' (Specify a correct email) is displayed. At the bottom right of the form are two buttons: 'Скасувати' (Cancel) and 'Додати' (Add).

Рис.3.24. Спроба збереження даних пацієнта з неправильно заповненими даними

Модуль користувачів, який використовується роллю адміністратора також має такий сценарій обробки форми. Модальне вікно, яке містить поля для заповнення даними для створення нового користувача, у разі спроби збереження незаповнених полів, отримаємо помилку, що свідчить про неможливість збереження пустих даних (рис.3.25.).

The screenshot shows a modal window titled 'Створення нового користувача' (Create new user). It contains four input fields: 'Username' with the placeholder 'Введіть username', 'Email' with the placeholder 'Введіть email', 'Пароль' (Password) with the placeholder 'Введіть пароль', and 'Роль' (Role) which is a dropdown menu with the placeholder 'Оберіть роль'. At the bottom left, there is a toggle switch labeled 'Активний' (Active) which is currently turned on. At the bottom right are two buttons: 'Скасувати' (Cancel) and 'Створити' (Create).

Рис.3.25. Спроба додавання нового користувача без заповнених полів у формі

Форма додавання вихідних днів лікарю присутня перевірка розкладу в обраному діапазоні (рис.3.26.). Якщо обрати дні відпустки у яких уже є актуальні записи на прийом і спробувати зберегти дані, побачимо відповідне сповіщення про неможливість такої дії.

Додати відпустку / вихідний для Ткачук Максим Олегович

Дата початку *
24.05.2026

Дата завершення *
26.05.2026

Причина
Відпустка

Перевірка розкладу в обраному діапазоні
Після вибору дат тут з'являться заплановані візити лікаря на цей період.

⚠ На вибрані дати знайдено 5 активних записів. Перед збереженням їх потрібно скасувати або перенести.

24.05.2026 • 5 записів

10:30 - 11:00

fdfsfdsd fdfdsfs fdfdsfs

11:00 - 11:30

Бондаренко Іван Сергійович

12:30 - 13:00

Нормаленко Степан Чік

13:00 - 13:30

Кравченко Олена Вікторівна

13:30 - 14:00

Бондаренко Іван Сергійович

Скасувати

Зберегти

Рис.3.26. Спроба додавання вихідних у період активних записів на прийом

Також в даній формі додавання вихідних днів лікарю, присутня валідація вибору дати початку та кінця вихідних днів. Така перевірка унеможливорює механічну помилку додавання неправильних дат. Дата початку не може бути раніше в минулому, дата кінця не може бути раніше дати початку. У разі спроби вручну ввести, для прикладу не той рік, то побачимо помилку про це, де текст містить причину даної помилки (рис.3.27).

ВідділенняКабінетСтіл

Додати відпустку / вихідний для Ткачук Максим Олегович

Дата початку *
21.05.2026

Дата завершення *
04.06.2024

Дата початку не може бути в минулому

Дата завершення не може бути раніше дати початку

Повинно бути

Рис.3.27. Перевірка правильності періоду дат

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

Також є перевірка, що дата завершення не може бути більше ніж 2 тижні від дати початку. Така перевірка зазначає ліміт відпустки лікаря. У модальному вікні додавання вихідних, у разі введене, для прикладу дати з наступного року, побачимо відповідну помилку (рис.3.28), що не дає можливості користувачу системи додати лікарю відпустку на рік.

Рис.3.28. Перевірка правильності періоду дат, кінцева дата перебільшує початкову

Загалом подібні помилки у модальному вікні додавання відпусток чи вихідних, можна додати лише ручним вводом з клавіатури, адже сам UI компонент вибору дати блокує вибір невалідної дати. Дати більше максимальної заблоковані для вибору, так само і з нижчими за мінімальну доступну, такі дні не мають можливість для натиску на них, та відмічені іншим кольором візуально (рис.3.29).

Рис.3.29. Доступні дати для вибору

Також у системі у разі фільтрування пошуком за недійсними запитами, будемо бачити відповідне повідомлення. Такий інформаційний блок дає зрозуміти користувачу, що система працює і реагує на його дії. Для прикладу, на сторінці розкладу спробуємо ввести в поле пошуку пацієнта дані про пацієнта, якого точно

не існує в системі, в такому випадку бачимо відповідний інформаційний блок (рис.3.30).

The screenshot shows a scheduling interface titled 'Розклад' (Schedule). At the top, it displays '22:32 Неділя' (Sunday). Below this, there are filters for 'Дата' (Date) set to '24.05.2026', 'Відділення' (Department), 'Лікар' (Doctor), and a search field for 'Пошук пацієнта' (Search patient) with the text 'ти хто'. A table below the filters shows time slots from 08:00 to 14:00. The table is currently empty, and a message at the bottom states: 'Немає даних для відображення за поточними фільтрами.' (No data for display with current filters.)

Рис.3.30. Перевірка неіснуючих даних

На сторінці розкладу у секції фільтрів в селекті лікарів, у разі пошуку хибного імені, таблиця розкладу не ламається. Ми бачимо відповідне повідомлення, що лікаря не знайдено (рис.3.31). Тобто вже на моменті вибору лікаря, ми не маємо можливості відсортувати таблицю розкладу за хибними даними.

The screenshot shows a dropdown menu for selecting a doctor. The dropdown is open, showing a search field with the text '!@#@' and a message 'Лікарів не знайдено' (Doctors not found). The dropdown is highlighted with a blue border.

Рис.3.31. Спроба пошуку лікаря з хибними даними

Система стабільно реагує на можливі помилки або варіанти хибного використання з боку користувача системи. Отже маємо стабільний інтерфейс, який готовий до використання.

Висновки до третього розділу

В результаті виконання третього розділу кваліфікаційної роботи, було описано порядок розгортання системи, інфраструктуру та деталізовано базові компоненти програмного продукту, а також описано покроковий порядок налаштування робочого середовища. Завдяки використанню багаторівневої

архітектури, що базується на патерні контролерів, сервісів та маршрутизаторів, вдалося забезпечити чіткий розподіл бізнес-логіки.

Окрім технічного розгортання, було детально описано порядок роботи з системою, показано функціонал системи, приклади використання від процесу авторизації працівника до прикладу управління розкладом та додавання записів пацієнтів. Продемонстрований користувацький інтерфейс підтверджує повну відповідність розробленого продукту початковим вимогам.

Також було перевірено стабільність та стійкість системи до ймовірних помилок валідації. Було проведено тестування стабільності системи, зокрема реагування на введення некоректних даних та нетипових дій користувача. Результати тестування підтвердили високий рівень стійкості додатка, впроваджені механізми багаторівневої валідації успішно перехоплюють виняткові ситуації, запобігаючи порушенню цілісності бази даних. Система показала, що працює без проблем та відповідно коректно реагує на ймовірні помилки користувача.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розроблено CRM-систему для управління роботи поліклініки, що включало розробку серверного додатку з підключення до хмарної бази даних та розробки користувацького інтерфейсу. Створена система повною мірою забезпечує медичних заклад зручним цифровим середовищем, яке об'єднує ключові аспекти взаємодії між реєстратурою та лікарями.

Розроблена CRM-система успішно вирішує проблему оптимізації документообігу, це є можливим завдяки створенню інтуїтивно зрозумілого інтерфейсу, який дозволяє медичному персоналу розпочати роботу без витрат часу на додаткове навчання. Реалізовано розклад, що дозволяє працівникам реєстратури знаходити вільні місця на прийом, уникаючи накладом під час запису пацієнтів. Для забезпечення швидкої навігації дані було чітко структуровано за логічними розділами - пацієнти, лікарі, відділення та робочі години. На інтерфейсі було додано зручні фільтри, включно з швидким пошуком за ключовими даними. Також створено індивідуальні профілі з детальною хронологією візитів, встановлених діагнозів, медичних призначень та рекомендацій. Ефективна взаємодія з базою даних та правильно побудований інтерфейс, дозволили усунути затримки під час формування розкладу лікарів, забезпечуючи безперебійну обробку всіх даних та загальну швидкодію системи.

Для забезпечення конфіденційності медичної інформації, у систему інтегровано рольову модель доступу, яка розмежовує повноваження. Адміністратор керує конфігурацією закладу, реєстратура працює з розкладом, а лікар має ізольований доступ до медичних записів своїх пацієнтів. Усі мережеві запити надійно захищені зі сторони API, а модульна архітектура системи гарантує її масштабованість, розроблена система дозволяє легко розширювати функціонал.

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Медична система Helsi.me. URL: <https://helsi.me/> (дата звернення: 08.06.2026).
2. Медична система Salesforce Health Cloud. URL: <https://www.salesforce.com/healthcare/cloud/> (дата звернення: 08.06.2026).
3. Монолітна архітектура ПЗ. URL: <https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/> (дата звернення: 08.06.2026).
4. Мікросервісна архітектура ПЗ. URL: <https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/> (дата звернення: 08.06.2026).
5. QA TestLab training center клієнт-серверна архітектура. URL: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/> (дата звернення: 08.06.2026).
6. GraphQL | The query language for modern APIs. URL: <https://graphql.org/> (дата звернення: 15.06.2026).
7. What is a RESTful API? AWS Documentation. URL: <https://aws.amazon.com/what-is/restful-api/> (дата звернення: 08.06.2026).
8. Патерни проектування об'єктно-орієнтованого програмного забезпечення / Е. Гамма, Р. Хелм, Р. Джонсон, Д. Вліссідес. Київ : Діалектика, 2020. 368 с.
9. Introduction to JSON Web Tokens. URL: <https://jwt.io/introduction> (дата звернення: 08.06.2026).
10. Using middleware // Express.js Documentation. URL: <https://expressjs.com/en/guide/using-middleware.html> (дата звернення: 08.06.2026).
11. Window: localStorage property // MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage> (дата звернення: 08.06.2026).
12. Server-sent events // MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 08.06.2026).
13. Node.js v26.3.0 documentation. URL: <https://nodejs.org/en/docs/> (дата звернення: 08.06.2026).

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

14. Express.js - Fast, unopinionated, minimalist web framework for Node.js. URL: <https://expressjs.com/> (дата звернення: 08.06.2026).
15. React The library for web and native user interfaces. URL: <https://react.dev/> (дата звернення: 15.06.2026).
16. MUI: The React component library you've always wanted. URL: <https://mui.com/> (дата звернення: 15.06.2026).
17. Інструмент для генерації UML-діаграм PlantText. URL: <https://www.planttext.com/> (дата звернення: 08.06.2026).
18. Supabase Documentation. URL: <https://supabase.com/docs> (дата звернення: 15.06.2026).
19. Опис основ нормалізації баз даних. Документація Microsoft Learn. URL: <https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description> (дата звернення: 12.06.2026).
20. Redux Toolkit The official, opinionated, batteries-included toolset for efficient Redux development. URL: <https://redux-toolkit.js.org/> (дата звернення: 08.06.2026).
21. Get started in Postman | Postman. URL: <https://learning.postman.com/docs/> (дата звернення: 15.06.2026).
22. Платформа для автоматизованої генерації діаграм Mermaid AI. URL: <https://mermaid.ai/> (дата звернення: 09.06.2026).
23. Проєктування RESTful web API. Документація Microsoft Learn. URL: <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design> (дата звернення: 09.06.2026).
24. Посібник з HTTP-методів в архітектурі REST. Інформаційний ресурс RestfulAPI.net. URL: <https://restfulapi.net/http-methods/> (дата звернення: 09.06.2026).
25. PostgreSQL: Documentation. URL: <https://www.postgresql.org/docs/> (дата звернення: 09.06.2026).
26. npm Docs. URL: <https://docs.npmjs.com/> (дата звернення: 09.06.2026).

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

27. Проектування реляційних баз даних та зв'язки між таблицями. Документація Microsoft Learn. URL: <https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints> (дата звернення: 09.06.2026).

28. Що таке клієнт-серверна архітектура? Cloudflare Learning Center. URL: <https://www.cloudflare.com/learning/serverless/glossary/client-server-model/> (дата звернення: 09.06.2026).

29. Getting Started | Create React App. URL: <https://create-react-app.dev/docs/getting-started/> (дата звернення: 09.06.2026).

30. Vite | Next Generation Frontend Tooling. URL: <https://vitejs.dev/> (дата звернення: 09.06.2026).

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

ДОДАТКИ

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

Лістинг коду, фрагмент маршрутизації на API:

```

const router = Router();

router.get("/", requirePermission("doctors", "read"),
DoctorsController.getAllDoctors);
router.get("/time-offs", requirePermission("doctors", "read"),
DoctorsController.getTimeOffs);
router.get(
  "/:id/time-off-conflicts",
  requirePermission("doctors", "read"),
  DoctorsController.getDoctorTimeOffConflicts,
);
router.get("/:id", requirePermission("doctors", "read"),
DoctorsController.getDoctorById);
router.post("/", requirePermission("doctors", "create"),
DoctorsController.createDoctor);
router.post(
  "/:id/time-offs",
  requirePermission("doctors", "update"),
  DoctorsController.createDoctorTimeOff,
);
router.put("/:id", requirePermission("doctors", "update"),
DoctorsController.updateDoctor);
router.delete("/:id", requirePermission("doctors", "delete"),
DoctorsController.deleteDoctor);

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Лістинг коду, фрагмент контролера на API:

```

export class DoctorsController {

  static async updateDoctor(req, res) {
    try {
      const { id } = req.params;
      const {
        user_id,
        last_name,
        first_name,
        middle_name,
        department_id,
        room_id,
        slot_duration_override,
        is_active,
      } = req.body;

      const payload = {
        user_id,
        last_name,
        first_name,
        middle_name,
        department_id,
        room_id,
        slot_duration_override,
        is_active,
      };

      const dataToUpdate = Object.fromEntries(
        Object.entries(payload).filter(([ , value ]) => value !== undefined),
      );

      if (Object.keys(dataToUpdate).length === 0) {
        logger.warn("No valid fields provided for doctor update", { id });
        return res.status(400).json({
          success: false,
          message:
            "Provide at least one field to update: user_id, last_name, first_name,
middle_name, department_id, room_id, slot_duration_override, is_active",
        });
      }

      const updatedDoctor = await DoctorsService.updateDoctor(Number(id),
dataToUpdate);

      if (!updatedDoctor) {
        logger.warn("Doctor not found for update", { id });
      }
    }
  }
}

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        return res.status(404).json({ success: false, message: "Doctor not found" });
    }

    logger.info("Doctor updated", { id, last_name: updatedDoctor.last_name });
    res.json({ success: true, data: updatedDoctor });
} catch (error) {
    logger.error("Error updating doctor", { id: req.params.id, message:
error.message });
    res.status(500).json({ success: false, message: error.message });
}
}

static async getTimeOffs(req, res) {
    try {
        const { date, doctorIds } = req.query;
        const timeOffs = await DoctorsService.getTimeOffs({ date, doctorIds });

        logger.info("Fetched doctor time offs", {
            date,
            count: timeOffs.length,
        });

        res.status(200).json({ success: true, data: timeOffs });
    } catch (error) {
        logger.error("Error fetching doctor time offs", {
            date: req.query.date,
            message: error.message,
        });

        if (error instanceof DoctorTimeOffValidationError) {
            return res.status(400).json({ success: false, message: error.message });
        }

        res.status(500).json({ success: false, message: error.message });
    }
}
}

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

Лістинг коду, фрагмент сервісу на API:

```

static async getTimeOffs({ date, doctorIds } = {}) {
  const normalizedDate = this.normalizeIsoDate(date);

  if (!this.isValidIsoDate(normalizedDate)) {
    throw new DoctorTimeOffValidationError("date is required");
  }

  const range = this.validateTimeOffRange(normalizedDate, normalizedDate);
  const normalizedDoctorIds = Array.isArray(doctorIds)
    ? doctorIds.map(Number).filter(Boolean)
    : this.parseDoctorIds(doctorIds);

  let query = supabase
    .from(TIME_OFFS_TABLE)
    .select("id, doctor_id, start_time, end_time, reason")
    .lt("start_time", range.endTime)
    .gt("end_time", range.startTime)
    .order("start_time", { ascending: true });

  if (normalizedDoctorIds.length > 0) {
    query = query.in("doctor_id", normalizedDoctorIds);
  }

  const { data: timeOffs, error } = await query;

  if (error) {
    throw new Error(error.message);
  }

  return timeOffs || [];
}

static async getDoctorTimeOffConflicts(doctorId, startDate, endDate) {
  const normalizedDoctorId = Number(doctorId);

  if (!normalizedDoctorId) {
    throw new DoctorTimeOffValidationError("doctor_id is required");
  }

  const range = this.validateTimeOffRange(startDate, endDate);

  const { data: appointments, error } = await supabase
    .from(APPOINTMENTS_TABLE)
    .select(
      "id, appointment_date, start_time, end_time, status, patients(id, last_name, first_name, middle_name)",
    )

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    )
    .eq("doctor_id", normalizedDoctorId)
    .gte("appointment_date", range.startDate)
    .lte("appointment_date", range.endDate)
    .neq("status", "Скасовано")
    .neq("status", "Завершено")
    .order("appointment_date", { ascending: true })
    .order("start_time", { ascending: true });

    if (error) {
        throw new Error(error.message);
    }

    return {
        count: (appointments || []).length,
        appointments: appointments || [],
        ...range,
    };
}

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Лістинг коду, фрагмент redux slice:

```

import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { showErrorToast, showSuccessToast } from "../../utils/toast";
import { apiFetch } from "../../utils/api";

import { API_BASE_URL } from "../../utils/config";

const pickPatientPayload = (patient = {}) => ({
  last_name: patient.last_name,
  first_name: patient.first_name,
  middle_name: patient.middle_name,
  gender: patient.gender,
  date_of_birth: patient.date_of_birth,
  phone: patient.phone,
  email: patient.email,
});

const normalizePatient = (patient = {}) => ({
  id: patient.id ?? null,
  last_name: patient.last_name ?? "",
  first_name: patient.first_name ?? "",
  middle_name: patient.middle_name ?? "",
  gender: patient.gender ?? "",
  date_of_birth: patient.date_of_birth ?? null,
  phone: patient.phone ?? "",
  email: patient.email ?? "",
  created_at: patient.created_at ?? null,
});

export const fetchPatients = createAsyncThunk(
  "patients/fetchPatients",
  async (_, { rejectWithValue }) => {
    try {
      const response = await apiFetch(`${API_BASE_URL}/patients`);
      if (!response.ok) throw new Error("Failed to fetch patients");
      const data = await response.json();
      return (data.data || []).map(normalizePatient);
    } catch (error) {
      return rejectWithValue(error.message);
    }
  },
);

export const createPatient = createAsyncThunk(
  "patients/createPatient",
  async (patientData, { rejectWithValue }) => {
    try {

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const payload = pickPatientPayload(patientData);
const response = await apiFetch(`${API_BASE_URL}/patients`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(payload),
});
if (!response.ok) throw new Error("Failed to create patient");
const data = await response.json();
return normalizePatient(data.data);
} catch (error) {
  return rejectWithValue(error.message);
}
},
);

export const updatePatient = createAsyncThunk(
  "patients/updatePatient",
  async ({ id, ...patientData }, { rejectWithValue }) => {
    try {
      const payload = Object.fromEntries(
        Object.entries(pickPatientPayload(patientData)).filter(([, value]) => value
        !== undefined),
      );
      const response = await apiFetch(`${API_BASE_URL}/patients/${id}`, {
        method: "PUT",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(payload),
      });
      if (!response.ok) throw new Error("Failed to update patient");
      const data = await response.json();
      return normalizePatient(data.data);
    } catch (error) {
      return rejectWithValue(error.message);
    }
  },
);

export const deletePatient = createAsyncThunk(
  "patients/deletePatient",
  async (id, { rejectWithValue }) => {
    try {
      const response = await apiFetch(`${API_BASE_URL}/patients/${id}`, {
        method: "DELETE",
      });
      if (!response.ok) throw new Error("Failed to delete patient");
      const data = await response.json();
      return data.data;
    } catch (error) {
      return rejectWithValue(error.message);
    }
  },
);

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
  },
);

const initialState = {
  patients: [],
  isLoading: false,
  error: null,
};

const patientsSlice = createSlice({
  name: "patients",
  initialState,
  reducers: {
    clearError: (state) => {
      state.error = null;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchPatients.pending, (state) => {
        state.isLoading = true;
        state.error = null;
      })
      .addCase(fetchPatients.fulfilled, (state, action) => {
        state.isLoading = false;
        state.error = null;
        state.patients = action.payload;
      })
      .addCase(fetchPatients.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload;
        showErrorToast(action.payload || "Failed to fetch patients");
      })
      .addCase(createPatient.pending, (state) => {
        state.isLoading = true;
        state.error = null;
      })
      .addCase(createPatient.fulfilled, (state, action) => {
        state.isLoading = false;
        state.error = null;
        state.patients.push(action.payload);
        showSuccessToast("Patient created successfully");
      })
      .addCase(createPatient.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload;
        showErrorToast(action.payload || "Failed to create patient");
      })
  }
});

```

		Яцюк Л.В.			ІПЗ.КР.Б-121-26-ПЗ	Арк.
		Лобанчикова Н.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

```

    .addCase(updatePatient.pending, (state) => {
      state.isLoading = true;
      state.error = null;
    })
    .addCase(updatePatient.fulfilled, (state, action) => {
      state.isLoading = false;
      state.error = null;
      const index = state.patients.findIndex((p) => p.id === action.payload.id);
      if (index !== -1) {
        state.patients[index] = action.payload;
      }
      showSuccessToast("Patient updated successfully");
    })
    .addCase(updatePatient.rejected, (state, action) => {
      state.isLoading = false;
      state.error = action.payload;
      showErrorToast(action.payload || "Failed to update patient");
    })
    .addCase(deletePatient.pending, (state) => {
      state.isLoading = true;
      state.error = null;
    })
    .addCase(deletePatient.fulfilled, (state, action) => {
      state.isLoading = false;
      state.error = null;
      state.patients = state.patients.filter((p) => p.id !== action.payload.id);
      showSuccessToast("Patient deleted successfully");
    })
    .addCase(deletePatient.rejected, (state, action) => {
      state.isLoading = false;
      state.error = action.payload;
      showErrorToast(action.payload || "Failed to delete patient");
    });
  },
});

export const { clearError } = patientsSlice.actions;
export default patientsSlice.reducer;

```