ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«АВТОМАТИЗОВАНА СИСТЕМА ПРОЦЕСІВ АДМІНІСТРУВАННЯ У МЕДИЧНОМУ ЦЕНТРІ»

на здобуття освітнього ступеня магістр
за спеціальності F6 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

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

_____ Микита БІЛОШИЦЬКИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Микита БІЛОШИЦЬКИЙ
(ім'я, ПРІЗВИЩЕ)
Керівник Валентина ДАНИЛЬЧЕНКО
PhD(доктор філософії). *(ім'я, ПРІЗВИЩЕ)*
Рецензент:
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Білошицькому Микиті Павловичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Автоматизована система процесів адміністрування у медичному центрі
керівник кваліфікаційної роботи: Валентина ДАНИЛЬЧЕНКО PhD(доктор філософії)
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “___ ” жовтня 2025 р. № ____

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Науково-технічна література з проєктування інформаційних систем у медицині.
2. Нормативні документи та стандарти у сфері охорони здоров’я.
3. Технічна документація сучасних веб-технологій (React, Node.js, PostgreSQL).
4. Матеріали з архітектури клієнт-серверних систем.
5. Методичні рекомендації щодо розробки та впровадження медичних інформаційних систем.

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

1. Аналіз предметної області, існуючих підходів та інформаційних систем адміністрування медичних центрів.

2. Проєктування архітектури автоматизованої системи адміністрування медичного центру та обґрунтування вибору програмних технологій.
3. Розробка, реалізація та тестування веб-орієнтованого прототипу системи, аналіз отриманих результатів.

5.Перелік ілюстраційного матеріалу: *презентація*

6. Дата видачі завдання «30» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз науково-технічної літератури та предметної області	30.10 – 05.11.25	
2.	Аналіз існуючих інформаційних систем адміністрування медичних центрів	06.11 – 10.11.25	
3.	Формування вимог до автоматизованої системи	11.11 – 15.11.25	
4.	Проєктування архітектури та структури бази даних системи	16.11 – 22.11.25	
5.	Розробка та реалізація основних функціональних модулів системи	23.11 – 06.12.25	
6.	Інтеграція модулів та тестування працездатності прототипу	07.12 – 13.12.25	
7.	Оформлення пояснювальної записки, висновків	14.12 – 20.12.25	
8.	Підготовка презентації та демонстраційних матеріалів	21.12 – 23.12.25	

Здобувач вищої освіти _____ Микита БІЛОШИЦЬКИЙ
(підпис) *(ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Валентина ДАНИЛЬЧЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття ступня магістр: 91 стор., 17 рис., 3 табл., 30 джерел.

Мета роботи – розробити та обґрунтувати автоматизовану систему процесів адміністрування у медичному центрі, що підвищує ефективність управління записом пацієнтів, графіками лікарів, використанням ресурсів та формуванням звітності.

Об'єкт дослідження – процеси адміністрування в медичному центрі.

Предмет дослідження – методи, моделі та інформаційні технології автоматизації процесів адміністрування в медичному центрі на основі веб-орієнтованих інформаційних систем, інтегрованих з медичними інформаційними системами та електронною системою охорони здоров'я.

Короткий зміст роботи. У роботі подано аналіз стану інформатизації медичних закладів, описано вибір засобів розробки, наведено проєктні рішення та реалізацію прототипу автоматизованої системи процесів адміністрування медичного центру, а також оцінено результати його впровадження. Окремий розділ присвячено питанням охорони праці та безпеки при експлуатації інформаційних систем у медичному центрі.

Ключові слова: МЕДИЧНИЙ ЦЕНТР, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, АДМІНІСТРУВАННЯ, ВЕБ-ОРІЄНТОВАНА СИСТЕМА, АДМІНІСТРАТИВНА ПАНЕЛЬ, БІЗНЕС-ПРОЦЕСИ.

ABSTRACT

The text part of the qualifying work for obtaining a bachelor's degree: 91 pp., 15 fig., 3 tables, 30 sources.

The purpose of the work is to develop and substantiate an automated system of administrative processes in a medical center, which increases the efficiency of managing patient records, doctors' schedules, resource use and reporting.

The object of the study is administrative processes in a medical center.

The subject of the study is methods, models and information technologies for automating administrative processes in a medical center based on web-based information systems integrated with medical information systems and an electronic health care system.

Summary of the work. The work presents an analysis of the state of informatization of medical institutions, describes the choice of development tools, provides design solutions and implementation of a prototype of an automated system of administrative processes in a medical center, and also evaluates the results of its implementation. A separate section is devoted to issues of occupational health and safety during the operation of information systems in a medical center.

Keywords: MEDICAL CENTER, MEDICAL INFORMATION SYSTEM, PROCESS AUTOMATION, ADMINISTRATION, WEB-ORIENTED SYSTEM, ADMINISTRATIVE PANEL, BUSINESS PROCESSES.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний розвиток медичних інформаційних систем та електронної системи охорони здоров'я зумовлює перехід медичних закладів до цифрового формату ведення медичної документації, обробки даних та організації взаємодії з пацієнтами [1–4]. Медичні інформаційні системи розглядаються як ключовий інструмент підтримки клінічних і адміністративних процесів – від реєстрації пацієнтів до формування звітності для органів управління охороною здоров'я [1–3].

Огляд наявних в Україні та за кордоном рішень (Doctor Eleks, Helsi.me, ЕМСІМЕД та інші) показує, що вони добре підтримують базові функції реєстрації пацієнтів, ведення електронних медичних карток та взаємодії з ЕСОЗ, однак питання організації внутрішніх процесів адміністрування в конкретному медичному центрі (управління записом, графіками, ресурсами, внутрішньою звітністю) часто залишаються недостатньо опрацьованими [5–10].

Дослідження, присвячені впровадженню госпітальних інформаційних систем, підтверджують, що наявність якісної інформаційної інфраструктури позитивно впливає на ефективність роботи персоналу, проте недосконаліть бізнес-процесів та відсутність зручних інструментів для адміністратора закладу призводять до дублювання операцій, перевантаження реєстрації та складнощів із контролем завантаженості ресурсів [11–18].

Паралельно зростає роль технологій штучного інтелекту, інтелектуального аналізу даних, а також стандартизованих підходів до обміну медичною інформацією (зокрема HL7 FHIR), що відкриває можливості для побудови гнучких веб-орієнтованих рішень, здатних інтегруватися з існуючими МІС та національною електронною системою охорони здоров'я [19–25]. В цих умовах актуальним є створення автоматизованої системи, яка б забезпечувала підтримку процесів адміністрування в медичному центрі та могла бути інтегрована в уже наявну інформаційну інфраструктуру

Аналіз стану досліджень. У наукових працях з медичної інформатики висвітлюються загальні принципи побудови медичних інформаційних систем, моделі подання медичних даних, функціональна структура МІС та особливості їх впровадження в лікувально-профілактичних закладах [1–3, 6]. Ряд публікацій присвячено огляду сучасних медичних інформаційних систем, аналізу їхніх модулів та оцінці ефективності застосування в клінічній практиці [5–8, 11–18].

Окремі роботи розглядають автоматизацію робочих процесів у медичних закладах, зазначаючи необхідність чіткої регламентації бізнес-процесів, усунення дублювання операцій та зменшення залежності від ручного введення даних адміністративним персоналом [16–18]. Разом з тим, питання побудови саме адміністративних веб-орієнтованих панелей для керування записом пацієнтів, графіками лікарів, ресурсами та оперативною звітністю досліджені недостатньо, особливо в контексті інтеграції з національною ЕСОЗ.

Мета і завдання дослідження. Метою роботи є розробка та обґрунтування автоматизованої системи процесів адміністрування у медичному центрі, яка забезпечує підвищення ефективності управління записом пацієнтів, графіками лікарів, ресурсами та звітністю на основі сучасних веб-технологій та інтеграції з медичною інформаційною інфраструктурою.

Для досягнення поставленої мети необхідно:

– проаналізувати предметну область та сучасний стан інформатизації медичних закладів;

– дослідити існуючі програмні рішення для автоматизації роботи медичних центрів та визначити їхні переваги й недоліки щодо підтримки процесів адміністрування;

– сформулювати вимоги до автоматизованої системи процесів адміністрування в медичному центрі;

– виконати аналіз та вибір технологій, архітектури та програмних засобів для розробки системи;

– спроєктувати моделі бізнес-процесів, інформаційну та функціональну структуру системи;

– розробити прототип веб-орієнтованої адміністративної панелі медичного центру та провести його тестування.

Об’єкт дослідження. Процеси адміністрування в медичному центрі.

Предмет дослідження. Методи, моделі та інформаційні технології автоматизації цих процесів на основі веб-орієнтованих інформаційних систем, інтегрованих з медичними інформаційними системами та електронною системою охорони здоров’я.

Методи дослідження. У роботі використовуються методи системного та порівняльного аналізу, методи моделювання бізнес-процесів та інформаційних структур, засоби програмної інженерії для проєктування і реалізації прототипу системи, а також експериментальні методи для тестування та оцінки ефективності розробленого рішення.

Наукова новизна полягає в узагальненні моделі автоматизованої системи процесів адміністрування медичного центру, орієнтованої на інтеграцію з електронною системою охорони здоров’я, та в уточненні підходів до моделювання бізнес-процесів з акцентом на адміністративні функції й підтримку управлінських рішень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика діяльності медичного центру та задач адміністрування

Медичний центр у сучасних умовах розглядається як організаційна структура, що надає амбулаторно-поліклінічні, діагностичні, консультативні та профілактичні послуги населенню на основі використання медичних інформаційних технологій [1]. На відміну від класичної лікарні стаціонарного типу, медичний центр, як правило, має більш гнучку організаційну модель, комбінує різні види медичних послуг (консультації, діагностика, малі хірургічні втручання, лабораторні дослідження) та орієнтується на підвищення якості сервісу для пацієнтів [1],[2].

Згідно з підручниками з медичної інформатики, діяльність медичного закладу можна подати як сукупність взаємопов’язаних процесів: клінічних (обслуговування пацієнтів), діагностичних (лабораторні та інструментальні дослідження), адміністративно-управлінських (планування, облік, контроль), фінансово-економічних та інформаційно-аналітичних [1],[2]. Для медичного центру, що працює в амбулаторному режимі, особливо важливими є процеси управління потоком пацієнтів, розкладом лікарів, завантаженістю кабінетів, використанням обладнання та формуванням управлінської звітності.

Типова організаційна структура медичного центру включає:

- керівництво (директор, медичний директор, головний лікар);
- адміністративно-управлінський персонал (адміністратор медичного центру, реєстратори, менеджери з роботи з пацієнтами, працівники бухгалтерії);
- лікарів різних спеціальностей (терапевти, сімейні лікарі, вузькі спеціалісти);
- середній медичний персонал (медичні сестри, лаборанти, фельдшери);
- допоміжний персонал (ІТ-фахівці, працівники служби експлуатації приміщень, технічний персонал) [1],[2].

Діяльність медичного центру орієнтована на:

- надання медичних послуг відповідно до затверджених протоколів та стандартів;
- забезпечення безперервності спостереження за пацієнтами (динамічне спостереження, контроль виконання рекомендацій);
- організацію взаємодії з пацієнтами (запис, нагадування, консультації);
- дотримання вимог законодавства, стандартів звітності та правил роботи з медичною інформацією [2],[3].

При цьому інформаційні потоки в медичному центрі включають: дані про пацієнтів (персональні дані, історія звернень), дані про медичні послуги (призначення, результати обстежень), фінансові дані (рахунки, оплати, договори зі страховими компаніями), управлінську інформацію (показники завантаженості, статистична звітність тощо) [1]-[3]. Їхня узгоджена обробка неможлива без використання медичних інформаційних систем та інтеграції з електронною системою охорони здоров’я.

У структурі діяльності медичного центру адміністративні процеси відіграють роль “каркасу”, який забезпечує узгоджену роботу всіх підрозділів. З точки зору медичної інформатики та організації охорони здоров’я, до основних задач адміністрування медичного центру належать [1]-[3]:

1. Управління потоками пацієнтів та записом на прийом:
- реєстрація первинних та повторних звернень пацієнтів;
- організація запису на прийом через різні канали (телефон, сайт, мобільний застосунок, реєстратура);
- керування чергою в режимі реального часу (онлайн-черга, перенесення записів, відміна візитів);
- розподіл пацієнтів між лікарями з урахуванням спеціалізації, графіка роботи, тривалості прийому.

2. Управління розкладами лікарів та ресурсами кабінетів:
- планування робочого часу лікарів (зміни, прийоми, відпустки, підміни);
- резервування кабінетів, діагностичного обладнання та інших ресурсів;

- контроль завантаженості графіків та недопущення конфліктів (накладання записів, надмірна концентрація візитів у пікові години).

3. Ведення адміністративно-управлінської та статистичної звітності:

- формування внутрішньої управлінської звітності (кількість прийомів, структура пацієнтів за віком, видом послуг, джерелом оплати);
- підготовка статистичних звітів для органів державного управління, страхових компаній, партнерів;
- контроль ключових показників діяльності (KPI) медичного центру та окремих підрозділів.

4. Організація документообігу та роботи з медичною інформацією:

- реєстрація та облік медичної документації в електронному та/або паперовому вигляді;
- забезпечення коректності внесення даних у медичну інформаційну систему;
- контроль доступу до медичної інформації, дотримання вимог конфіденційності та захисту персональних даних [3],[4].

5. Взаємодія з пацієнтами та сервісна підтримка:

- надання довідкової інформації щодо послуг, підготовки до обстежень, графіків роботи;
- організація системи нагадувань (SMS, e-mail, пуш-сповіщення) про візити, результати обстежень, необхідність повторних консультацій;
- обробка звернень, скарг та пропозицій пацієнтів, моніторинг задоволеності сервісом.

6. Фінансово-економічні аспекти адміністрування:

- первинний облік наданих послуг (рахунки, акти, чеки);
- взаємодія зі страховими компаніями та партнерами;
- контроль оплати послуг, боргових зобов’язань та фінансових показників (виручка, середній чек, рентабельність послуг) [2],[3].

7. Підтримка управлінських рішень та планування розвитку:

- аналіз динаміки звернень пацієнтів, структури послуг, завантаженості лікарів та ресурсів;

- виявлення «вузьких місць» у роботі медичного центру (надмірні черги, нерівномірна завантаженість, затримки з оформленням документації);
- вироблення управлінських рішень щодо оптимізації графіків, розширення спектра послуг, зміни структури підрозділів.

Таблиця 1.1

Основні учасники адміністративних процесів медичного центру та їхні функції.

Учасник	**Основні функції**
Директор / керівник	Стратегічне управління центром, затвердження структури, тарифів, політик, контроль ключових показників діяльності
Медичний директор / головний лікар	Координація медичної діяльності, контроль якості медичних послуг, погодження графіків лікарів, участь у формуванні стандартів роботи
Адміністратор медичного центру	Загальна організація роботи зміни, контроль запису, завантаженості ресурсів, вирішення конфліктних ситуацій, оперативне управління персоналом
Реєстратори / оператори call-центру	Запис пацієнтів, перенесення/скасування візитів, консультування щодо графіків, первинне внесення даних у систему
Лікарі	Робота з пацієнтами за записом, внесення медичних даних у систему, участь у формуванні рекомендацій щодо оптимізації графіків
Медичні сестри, лаборанти	Підготовка пацієнтів до процедур, виконання маніпуляцій, робота з

	лабораторними замовленнями та результатами
Бухгалтерія / фінансовий відділ	Облік наданих послуг, взаєморозрахунки з пацієнтами та страховими компаніями, фінансова звітність
ІТ-підрозділ / спеціаліст	Підтримка медичної інформаційної системи, інтеграція з ЕСОЗ, забезпечення безпеки та безперервності роботи інформаційної інфраструктури

Така структуризація дозволяє чітко виділити, які саме функції та задачі адміністрування є критичними для роботи медичного центру та мають бути підтримані автоматизованою системою. Саме ці задачі надалі стануть основою для формування вимог до розроблюваної системи та моделювання бізнес-процесів у наступних підрозділах [3]-[5].

1.2 Аналіз процесів адміністрування в медичних інформаційних системах

Медичні інформаційні системи (МІС) створюються як комплексні програмні рішення, що підтримують основні види діяльності медичного закладу: клінічну, діагностичну, адміністративно-управлінську, фінансову та аналітичну [1],[2]. З позиції задач, розглянутих у п. 1.1, особливий інтерес становлять саме процеси адміністрування, які безпосередньо впливають на організацію роботи медичного центру, пропускну здатність та якість сервісу для пацієнтів.

У більшості сучасних МІС функції системи групуються в окремі підсистеми: клінічну (ведення електронних медичних записів, історій хвороби, результатів досліджень), адміністративну (реєстратура, запис, управління розкладами лікарів і ресурсами), фінансову (облік послуг, рахунки, взаєморозрахунки) та аналітичну

(звітність, моніторинг показників діяльності, підтримка управлінських рішень) [1]-[3]. У межах цієї роботи основна увага зосереджується на адміністративній складовій МІС, оскільки саме вона реалізує ключові процеси, пов’язані з управлінням потоками пацієнтів, розкладами, кабінетами та звітністю.

Типовий набір функцій МІС, що стосуються адміністрування, включає реєстрацію пацієнтів і ведення електронної картотеки, запис на прийом та управління чергою, планування розкладів лікарів і кабінетів, організацію документообігу та формування звітності, ведення фінансово-адміністративних операцій і комунікацію з пацієнтами [2],[3]. На рівні реєстратури система забезпечує створення електронної картки пацієнта, фіксацію первинних і повторних звернень, пошук пацієнтів за різними критеріями. Модуль запису на прийом підтримує формування розкладу лікарів, запис через різні канали (телефон, сайт, інтернет-реєстратура), перенесення та скасування візитів, ведення списку очікування, відображення поточної черги.

Модулі управління розкладами лікарів та кабінетів призначені для планування робочого часу співробітників, врахування відпусток і замін, прив’язки лікарів до кабінетів та видів послуг, а також контролю конфліктів, пов’язаних із накладанням записів або подвійним бронюванням ресурсів [2]. Підсистема документообігу і звітності забезпечує формування внутрішньої управлінської документації (довідки, виписки, акти), підготовку статистичних звітів, а також експорт даних у встановлених форматах для подальшої обробки [3],[4]. Фінансово-адміністративні модулі підтримують реєстрацію фактів оплати послуг, роботу з тарифами, формування рахунків для пацієнтів і страхових компаній, ведення програм лояльності. Інструменти комунікації з пацієнтами відповідають за автоматичні нагадування про візити, інформування про готовність результатів та передачу організаційної інформації (графіки, підготовка до процедур).

Зручним є подання відповідності між задачами адміністрування (п. 1.1) та функціональними модулями МІС у вигляді таблиці (табл. 1.2).

Таблиця 1.2

Відображення задач адміністрування на функціональні модулі МІС

Задача адміністрування	**Відповідні модулі МІС**
Управління записом та чергою	Реєстратура, модуль розкладів, модуль онлайн-запису
Управління розкладами лікарів та кабінетів	Планувальник розкладів, календар ресурсів
Ведення картотеки пацієнтів	Модуль пацієнтів (EHR/EMR), реєстр пацієнтів
Формування управлінської та статистичної звітності	Звітний модуль, аналітичний модуль
Фінансовий облік послуг	Білінговий модуль, модуль взаєморозрахунків
Організація документообігу	Модуль документообігу, генератор шаблонів документів
Комунікація з пацієнтами	Модуль сповіщень, інтеграція з SMS/e-mail сервісами, пацієнтський портал

Адміністративні процеси в медичному центрі реалізуються через обмін інформацією між різними ролями користувачів МІС: адміністратором, реєстратором, лікарями, середнім медичним персоналом, бухгалтерією, керівництвом [2], [3]. Типовими є інформаційні потоки “пацієнт - реєстратура - МІС” (реєстрація звернень, запис, актуалізація даних), “адміністратор - МІС” (налаштування розкладів, керування ресурсами, блокування та відмітки службового часу), “лікар - МІС” (медичні записи, корекція розкладу, статус прийому), “МІС - керівництво” (агреговані звіти, показники діяльності), “МІС - пацієнт” (нагадування, оповіщення, довідкова інформація).

У цій структурі адміністратор медичного центру виступає центральною фігурою адміністративної підсистеми: він координує роботу реєстратури, контролює актуальність розкладів і завантаженість ресурсів, стежить за коректністю організаційної структури та налаштувань системи, а також відповідає за оперативне реагування на зміни в роботі закладу [2].

Попри наявність широкого функціоналу, реальна робота адміністративних модулів МІС нерідко супроводжується низкою обмежень і проблем [3]-[5]. До найтиповіших належать фрагментованість інтерфейсу та дублювання дій (коли операції запису, перенесення візиту, зміни розкладу та оформлення рахунку виконуються в різних розділах системи), недостатня гнучкість роботи з розкладами (складність оперативного коригування при зміні графіків), обмежені можливості аналітики для адміністратора (орієнтація стандартних звітів здебільшого на статистику або фінанси), низький рівень автоматизації рутинних операцій (нагадування, підтвердження візитів, повторні звернення), недостатня інтеграція з електронною системою охорони здоров’я та сторонніми сервісами, а також обмежені можливості кастомізації інтерфейсу й логіки роботи під конкретний медичний центр.

Унаслідок цього значна частина часу адміністратора та реєстратури витрачається на технічне “обслуговування” інформаційної системи (пошук потрібних форм, ручне внесення дубльованих даних, виконання рутинних операцій), а не на підвищення якості сервісу для пацієнтів та підтримку управлінських рішень. Це обґрунтовує потребу в розробці спеціалізованої автоматизованої системи процесів адміністрування медичного центру, яка б враховувала специфіку конкретного закладу, надавала зручний, орієнтований на адміністратора інтерфейс, інтегрувалася з наявною МІС та ЕСОЗ і забезпечувала розширені можливості аналітики.

Саме ці висновки стають основою для подальшого огляду конкретних програмних продуктів для автоматизації роботи медичних центрів (Doctor Eleks, Helsi, ЕМСІМЕД та ін.) та подальшого формування вимог до системи, що розробляється, у наступних підпунктах першого розділу.

1.3 Огляд існуючих програмних рішень для автоматизації медичних центрів

На ринку України та за кордоном представлено значну кількість медичних інформаційних систем, що позиціонуються як інструменти для автоматизації діяльності медичних закладів різного профілю. Частина з них орієнтована переважно на стаціонарні лікарні, інші – на амбулаторно-поліклінічні заклади та приватні медичні центри. Для цілей даної роботи інтерес становлять системи, які забезпечують підтримку адміністративних процесів: управління записом пацієнтів, розкладами лікарів, кабінетами та ресурсами, фінансовим обліком послуг, а також взаємодією з електронною системою охорони здоров'я [5], [6].

Узагальнено існуючі програмні рішення можна поділити на дві основні групи:

- комплексні медичні інформаційні системи (MIC), що охоплюють клінічний, адміністративний, фінансовий та аналітичний рівні;
- спеціалізовані сервіси для запису та взаємодії з пацієнтами, які інтегруються з MIC або працюють як окремі модулі.

До перших належать, зокрема, вітчизняні системи Doctor Eleks, ЕМСІМЕД, Медстар та інші програмні продукти, які тривалий час використовуються в українських медичних закладах [7]-[9]. До другої групи можна віднести сервіси типу Helsi, що поєднують функції онлайн-запису, електронної реєстратури та інтеграції з національною електронною системою охорони здоров'я, а також різноманітні CRM/онлайн-платформи для медичного бізнесу [8],[10].

Система Doctor Eleks позиціонується як комплексна MIC для медичних закладів різних рівнів. Вона має розвинений модуль клінічної роботи (електронні медичні картки, протоколи лікування, лабораторні та діагностичні підсистеми) та адміністративно-фінансові модулі (розклади лікарів, реєстратура, облік послуг, взаємодія зі страховими компаніями) [7],[9]. Для медичного центру важливими є можливості управління записом пацієнтів, планування розкладів, формування звітності по візитах і послугах, ведення електронної картотеки пацієнтів. Водночас,

у типових впровадженнях основний інтерфейс Doctor Eleks орієнтований на лікаря та медичний персонал, а адміністративні сценарії (робота адміністратора, реєстратури, керівництва) реалізовані в рамках загальної логіки МІС, що не завжди враховує специфіку роботи саме окремого медичного центру.

Медичні інформаційні системи сімейства ЕМСІМЕД / EMCI орієнтовані на комплексну підтримку діяльності медичних закладів з фокусом на інтеграцію з національною електронною системою охорони здоров'я, електронні рецепти, реімбурсацію та електронні направлення [8]. Система підтримує ведення електронних медичних карток, формування звітів для НСЗУ, а також базові функції адміністративної роботи: запис, розклади, облік послуг. Однак інтерфейси адміністрування часто проектуються з урахуванням регуляторних вимог і суттєвого обсягу форм звітності, що може призводити до перевантаження робочого місця адміністратора великою кількістю форм і реквізитів, не завжди необхідних для оперативної роботи конкретного медичного центру.

Сервіс Helsi у першу чергу відомий як онлайн-платформа для запису до лікаря, яка забезпечує пацієнтам доступ до розкладів лікарів, можливість обрати медичний заклад і спеціаліста, а також інтегрується з ЕСОЗ для фіксації декларацій, електронних направлень та інших документів [8],[10]. Для адміністратора медичного центру цей сервіс корисний тим, що дозволяє автоматизувати значну частину функцій реєстратури, зменшуючи кількість телефонних звернень, а також дає можливість оперативно коригувати розклади в публічному просторі. Водночас, Helsi фокусується здебільшого на зовнішньому контурі взаємодії з пацієнтом і не завжди закриває внутрішні задачі адміністрування: поєднання запису з управлінням кабінетами, внутрішньою аналітикою, фінансовими показниками, деталізованими управлінськими звітами.

Окрім згаданих систем, на ринку присутні й інші рішення, які позиціонуються як «кабінети лікаря», «онлайн-реєстратури», CRM-системи для приватної медицини. Вони зазвичай пропонують:

- електронну картку пацієнта з базовою історією звернень;
- онлайн-запис через веб-інтерфейс або мобільний застосунок;

- модуль нагадувань (SMS/e-mail);
- прості фінансові звіти (кількість візитів, суми оплат).

Такі системи можуть бути зручними для невеликих приватних кабінетів або вузькоспеціалізованих клінік, але для багатопрофільного медичного центру з кількома лікарями, кабінетами та напрямами діяльності їх функціональності часто недостатньо.

З точки зору задач, огляд існуючих рішень дозволяє виділити низку сильних сторін:

- наявність відпрацьованих механізмів запису на прийом та управління розкладами;
- підтримка електронних медичних карток і зв’язку з ЕСОЗ;
- реалізація базової фінансової звітності та функцій обліку послуг;
- інтеграція з сервісами нагадувань і зовнішніми каналами комунікації з пацієнтами.

Водночас виявляються обмеження з позиції потреб адміністратора медичного центру:

- інтерфейси часто орієнтовані на лікаря або на формування звітності для державних органів, а не на щоденну роботу адміністратора та керівника центру;
- недостатньо гнучкі інструменти для оперативного аналізу завантаженості, керування потоками пацієнтів, виявлення «пікових» годин, причин скасування візитів;
- обмежені можливості кастомізації бізнес-процесів під конкретну організаційну структуру та сервісну модель медичного центру;
- потреба в додаткових зовнішніх інструментах (таблиці, окремі CRM/BI-системи) для повноцінної підтримки управлінських рішень.

Для наочного порівняння існуючих рішень з точки зору підтримки ключових задач адміністрування доцільно навести таблицю 1.3, у якій будуть перелічені основні системи та позначено, які групи функцій для адміністратора вони реалізують.

Таблиця 1.3

Порівняльна характеристика програмних рішень з точки зору задач адміністрування

Система / сервіс	Doctor Eleks	EMCIМЕД / EMCI	Helsi	Онлайн-реєстратури / CRM
Запис і черга	+	+	+	+
Розклади лікарів/кабінетів	+	+	+	частково
Картотека пацієнтів	+	+	частково	частково
Фінансовий облік	+	+	частково	частково
Аналітика для адміністратора	частково	частково	обмежено	обмежено
Інтеграція з ЕСОЗ	+	+	+	залежить від рішення

У зазначеній таблиці знак “+” означає наявність повноцінної підтримки відповідної групи функцій, “частково” - обмежену підтримку або необхідність використання додаткових модулів, тоді як формулювання «обмежено» вказує на мінімальні аналітичні можливості для адміністратора (простий перелік записів, базові звіти).

Таким чином, огляд існуючих програмних рішень показує, що на ринку вже присутні потужні інструменти для автоматизації діяльності медичних закладів, однак з точки зору автоматизації процесів адміністрування конкретного медичного центру залишається простір для вдосконалення. Це насамперед стосується гнучкої підтримки бізнес-процесів, розширеної аналітики для адміністратора, зручності інтерфейсу, можливостей кастомізації та інтеграції різних аспектів роботи (запис, розклади, ресурси, фінанси, звітність) в єдиній адміністративній панелі. Саме ці аспекти будуть враховані при формуванні вимог до автоматизованої системи

процесів адміністрування медичного центру в наступних підпунктах першого розділу.

1.4 Проблеми та недоліки існуючих рішень

Огляд сучасних медичних інформаційних систем та сервісів для автоматизації роботи медичних центрів показує, що, попри значний функціональний потенціал, вони не повною мірою задовольняють потреби адміністрування конкретного медичного закладу [5]-[10]. У систематичних оглядах, присвячених впровадженню госпітальних інформаційних систем, відзначається, що основними проблемами є невідповідність між можливостями системи та реальними бізнес-процесами, складність користування, недостатня гнучкість налаштувань і обмежені аналітичні можливості для управлінського персоналу [11]-[14].

Однією з ключових проблем є фрагментованість інтерфейсів і сценаріїв роботи. У багатьох МІС операції запису пацієнтів, перенесення візитів, зміни розкладів, оформлення фінансових документів, формування звітів реалізовані в різних модулях або вкладках. Це змушує адміністратора постійно переключатися між вікнами, виконувати кілька послідовних кроків для вирішення типових задач, дублювати введення одних і тих самих даних у різних формах [12], [13]. В умовах інтенсивного потоку пацієнтів така організація роботи суттєво збільшує навантаження на реєстратуру та знижує оперативність обслуговування.

Другою важливою проблемою є недостатня гнучкість управління розкладами та ресурсами. Типові моделі розкладів у МІС будуються за відносно статичними схемами (фіксовані дні прийому, типові тимчасові слоти). Оперативне коригування (додавання позапланових прийомів, зміна тривалості візиту, перерозподіл пацієнтів між лікарями) часто вимагає ручного втручання, що супроводжується ризиком виникнення конфліктів у розкладах і потребою в додатковій комунікації з пацієнтами [11], [15]. Для медичного центру, який прагне гнучко реагувати на попит, така негнучкість створює бар’єр для підвищення сервісу.

Наступний блок проблем пов’язаний з обмеженими аналітичними можливостями для адміністратора та керівництва. Хоча більшість МІС містять звітні модулі, вони часто орієнтовані на формування регламентованої статистичної звітності або базових фінансових показників [11]-[16]. Адміністратору ж потрібні інструменти оперативної аналітики: динаміка звернень у розрізі днів і годин, завантаженість конкретних лікарів і кабінетів, частота скасування та перенесення візитів, структура послуг за напрямами, середній час очікування, виконання планових показників тощо. За відсутності зручних дашбордів і гнучких фільтрів персоналу доводиться експортувати дані з МІС у табличні редактори та самостійно будувати зведені таблиці, що є трудомістким і схильним до помилок процесом [15], [16].

Суттєвою проблемою є також низький рівень автоматизації рутинних операцій адміністрування. У ряді систем нагадування пацієнтам про візит, підтвердження запису, розсилка інформаційних повідомлень реалізовані частково або потребують підключення зовнішніх сервісів, які не інтегровані з основними модулями МІС [17]. Унаслідок цього частина дій (передзвони, ручне надсилання повідомлень, повторний запис) виконується поза основною системою, що розриває єдиний інформаційний простір і збільшує навантаження на персонал.

Окремою групою проблем є інтеграційні обмеження. Багато МІС історично розроблялися як відносно закриті рішення для окремого закладу або групи закладів і лише згодом доповнювалися інтерфейсами для взаємодії з національними системами охорони здоров’я, лабораторними інформаційними системами, зовнішніми сервісами запису [11], [18]. У результаті інтеграція з електронною системою охорони здоров’я, лабораторіями, платіжними сервісами, CRM-платформами часто реалізується у вигляді набора «надбудов» і конекторів, які працюють не завжди стабільно. Для адміністратора це означає необхідність дублювання дій у кількох системах або контроль за коректністю синхронізації, що знову ж таки збільшує трудомісткість адміністрування.

Значна увага в дослідженнях приділяється людському фактору та зручності інтерфейсу. Зазначається, що перевантажений, нелогічний інтерфейс, орієнтований

передусім на розробника або вузького IT-фахівця, а не на кінцевого користувача (адміністратора, реєстратора), є одним із ключових бар'єрів ефективного використання МІС [12], [13], [19]. При цьому саме адміністратор є тим користувачем, який щоденно працює з найбільшою кількістю записів, візитів, дзвінків, змін у розкладах, і кожна додаткова дія або перехід між формами безпосередньо впливає на продуктивність роботи та якість сервісу для пацієнтів.

Ще одна важлива проблема – обмежені можливості адаптації бізнес-процесів під конкретний медичний центр. Типові конфігурації МІС зорієнтовані на «середньостатистичний» заклад і не завжди дозволяють гнучко налаштувати сценарії обслуговування пацієнтів, рівні доступу для різних ролей, варіанти маршрутизації пацієнтів між підрозділами, порядок погодження нестандартних ситуацій (наприклад, робота з VIP-пацієнтами, корпоративними клієнтами, пакетами послуг) [14], [18]. У результаті медичні центри вимушені або підлаштовуватися під логіку системи, або створювати локальні «обхідні» процедури (додаткові журнали, таблиці, неформальні домовленості), які не відображаються повноцінно в інформаційній системі.

Не менш суттєвою є і проблема прозорості та контролю якості даних. За умови інтенсивної роботи, відсутності достатніх механізмів валідації та контролю введення даних у МІС виникають помилки у реєстрації пацієнтів, записах на прийом, прив'язці послуг до візитів, фінансових операціях [16], [19]. Для їх виправлення адміністраторам доводиться витрачати додатковий час на звірку даних, пошук розбіжностей та корекцію записів, що знижує довіру до звітності й ускладнює прийняття управлінських рішень на основі інформації з системи.

У сукупності зазначені проблеми свідчать про те, що, незважаючи на наявність на ринку функціонально насичених медичних інформаційних систем, потреби адміністратора та керівництва конкретного медичного центру часто залишаються не до кінця задоволеними. Системи забезпечують базову автоматизацію обліку та взаємодії з пацієнтами, але не завжди надають зручні інструменти для гнучкого управління потоками пацієнтів, ресурсами, розкладами, аналітикою та якістю даних.

Це обґрунтовує доцільність розробки спеціалізованої автоматизованої системи процесів адміністрування медичного центру, орієнтованої на потреби адміністратора як ключового користувача, яка забезпечуватиме: цілісну картину поточного стану роботи центру, зручні сценарії щоденних операцій, розширені можливості аналітики та інтеграцію з існуючою медичною інформаційною інфраструктурою. У наступному підпункті на основі результатів аналізу буде сформульовано постановку задачі та вимоги до такої системи.

1.5 Постановка задачі та вимоги до автоматизованої системи процесів адміністрування

Виконаний аналіз діяльності медичного центру, процесів адміністрування та можливостей сучасних медичних інформаційних систем показав, що існуючі рішення забезпечують базову підтримку запису пацієнтів, ведення картотеки, розкладів та фінансового обліку, однак не повною мірою відповідають потребам адміністратора конкретного медичного центру. Виявлено фрагментованість інтерфейсів, недостатню гнучкість роботи з розкладами та ресурсами, обмежені можливості оперативної аналітики, низький рівень автоматизації рутинних операцій та обмежену адаптивність бізнес-процесів під специфіку окремого закладу.

З огляду на це формулюється постановка задачі магістерської роботи:

Необхідно розробити веб-орієнтовану автоматизовану систему процесів адміністрування медичного центру, яка забезпечує підтримку управління записом пацієнтів, розкладами лікарів і кабінетів, використанням ресурсів та формуванням управлінської звітності, орієнтовану на потреби адміністратора й інтегровану з існуючою медичною інформаційною інфраструктурою.

Для реалізації цієї загальної задачі потрібно сформувати чіткі функціональні та нефункціональні вимоги до системи, а також визначити основні ролі користувачів і типові сценарії їх взаємодії з системою.

Об’єктом автоматизації є адміністративні процеси медичного центру, пов’язані з:

- управлінням потоками пацієнтів (запис, перенесення, скасування візитів, черга);
- плануванням та актуалізацією розкладів лікарів і кабінетів;
- веденням базових адміністративних даних про пацієнтів (картотека, історія звернень у розрізі адміністрування);
- формуванням внутрішньої управлінської звітності й аналітичних показників для керівництва;
- організацією комунікації з пацієнтами (нагадування, повідомлення);
- взаємодією з медичними інформаційними системами та електронною системою охорони здоров’я на рівні обміну необхідними адміністративними даними.

Цільове призначення системи – надати адміністратору медичного центру єдиний зручний інструмент, який дозволяє:

- оперативно керувати записом і розкладами;
- бачити поточний стан завантаженості центру;
- отримувати потрібні звіти та показники;
- мінімізувати кількість рутинних ручних операцій та дублювання дій у різних системах.

У розроблюваній системі передбачається підтримка щонайменше таких основних ролей користувачів:

- Адміністратор медичного центру – ключовий користувач системи; відповідає за загальне управління записом, розкладами, ресурсами, моніторинг завантаженості, роботу з аналітикою та звітністю.
- Реєстратор / оператор – здійснює щоденну роботу із записом пацієнтів, перенесенням і скасуванням візитів, консультуванням щодо графіків, первинним внесенням і уточненням даних пацієнтів.

- Керівник медичного центру (директор, медичний директор) – використовує систему для перегляду агрегованих звітів, аналізу завантаженості, контролю ключових показників діяльності, прийняття управлінських рішень.
- Лікар – при необхідності має обмежений доступ до інформації про власний розклад, записаних пацієнтів, динаміку завантаженості (без доступу до конфіденційних адміністративних і фінансових даних).

За потреби можуть бути додані й інші ролі (наприклад, аналітик, IT-адміністратор системи), однак у базовій конфігурації акцент робиться саме на адміністратора й реєстратуру.

На основі аналізу предметної області та існуючих рішень формулюються такі основні функціональні вимоги до автоматизованої системи процесів адміністрування медичного центру.

1. Управління записом пацієнтів

Система повинна забезпечувати:

- реєстрацію первинних і повторних звернень пацієнтів;
- запис пацієнта на прийом до конкретного лікаря, на конкретну дату й час з урахуванням доступних слотів;
- можливість швидкого перенесення візиту (зміна дати, часу, лікаря) із контролем конфліктів;
- скасування візитів із фіксацією причини (за необхідності);
- ведення списку очікування та можливість оперативного запису пацієнта на звільнені слоти;
- перегляд поточного стану запису (черга, список візитів на день/тиждень, фільтрація за лікарем, кабінетом, видом послуги).

2. Управління розкладами лікарів і кабінетів

Система повинна:

- підтримувати створення та редагування типових розкладів лікарів (робочі дні, години прийому, тривалість слоту, перерви);
- враховувати відпустки, лікарняні, відрядження, позапланові зміни;

- дозволяти оперативно змінювати розклад (додавати/видаляти слоти, змінювати тривалість прийому) з автоматичним аналізом уже існуючих записів;
- забезпечувати прив’язку лікарів до кабінетів, типів послуг, ресурсів (обладнання, лабораторні потужності тощо);
- відображати завантаженість лікарів і кабінетів у зручній формі (календар, таблиця, діаграми).

3. Ведення адміністративної картотеки пацієнтів

Система повинна:

- зберігати базові дані пацієнтів, необхідні для адміністрування (ПІБ, контактні дані, дата народження, ідентифікаційні реквізити, джерело направлення тощо);
- відображати історію звернень у вигляді переліку візитів (дата, лікар, тип послуги, статус візиту);
- підтримувати пошук пацієнтів за різними параметрами (ПІБ, телефон, ідентифікатор);
- за потреби – взаємодіяти з МІС/ЕСОЗ для уточнення статусу пацієнта або синхронізації окремих реквізитів (без дублювання клінічних даних).

4. Формування управлінської та оперативної звітності

Система повинна забезпечувати:

- формування звітів щодо кількості прийомів у розрізі періодів, лікарів, кабінетів, видів послуг;
- аналіз завантаженості лікарів і кабінетів (у вигляді таблиць, графіків);
- отримання показників щодо скасованих і перенесених візитів, причин скасування;
- побудову звітів для керівництва (динаміка звернень, структура послуг, виконання планових показників);
- експорт звітів у поширені формати (наприклад, PDF, XLSX) для подальшої обробки чи надання зовнішнім органам.

5. Підтримка комунікації з пацієнтами

Система повинна:

- формувати нагадування пацієнтам про майбутні візити (через доступні канали – SMS, e-mail, месенджери, залежно від вибраної інтеграції);
- фіксувати факт відправлення нагадувань та, за можливості, реакцію (підтвердження/відмова);
- підтримувати шаблони повідомлень (нагадування, інформація про перенесення візиту, загальні повідомлення медичного центру).

6. Інтеграція з МІС та іншими системами

Система повинна:

- мати можливість інтеграції з наявною МІС медичного центру для обміну даними щодо пацієнтів, візитів, розкладів (через API, обмін файлами, стандартні протоколи тощо);
- підтримувати базовий обмін з електронною системою охорони здоров’я (у межах адміністративних задач, визначених політикою закладу);
- за можливості – інтегруватися з сервісами SMS/e-mail-розсилки, календарями, CRM або фінансовими системами для розширеної аналітики.

7. Управління доступом та ролями

Система повинна:

- забезпечувати авторизацію користувачів за логіном і паролем (та/або іншими механізмами);
- підтримувати розмежування прав доступу залежно від ролі (адміністратор, реєстратор, керівник, лікар);
- дозволяти конфігурування прав: наприклад, реєстратор має доступ до запису й картотеки, але не до управлінських звітів, керівник – до агрегованої аналітики тощо.

2 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ АВТОМАТИЗОВАНОЇ СИСТЕМИ

2.1 Аналіз технологій побудови веб-орієнтованих інформаційних систем

Автоматизована система процесів адміністрування медичного центру за своєю природою є веб-орієнтованою інформаційною системою, оскільки повинна забезпечувати доступ до функціоналу з різних робочих місць (адміністратор, реєстратура, керівництво, лікарі) без встановлення спеціалізованого клієнтського програмного забезпечення. Такий підхід відповідає сучасним тенденціям розробки інформаційних систем, що базуються на використанні браузера як універсального клієнта та моделі «клієнт–сервер» [1], [2].

Типова веб-орієнтована інформаційна система має багаторівневу архітектуру, у межах якої виділяють: рівень представлення (front-end), рівень прикладної логіки (back-end) та рівень даних (СУБД). На клієнтському рівні реалізується інтерфейс користувача, відображення даних і базова валідація введених значень. На серверному рівні зосереджені бізнес-логіка, правила обробки запитів, механізми аутентифікації та авторизації, інтеграція з іншими системами. Рівень даних відповідає за надійне зберігання інформації про користувачів, пацієнтів, лікарів, розклади, записи на прийом, журнали дій тощо [2], [3].

Взаємодія між клієнтом і сервером здійснюється, як правило, через протокол HTTPS, що забезпечує шифрування трафіку та захист конфіденційної інформації, що є критичним для медичної сфери. Обмін даними реалізується у форматах, зручних для машинної обробки, переважно JSON, що спрощує інтеграцію між різними компонентами системи та зовнішніми сервісами [3]. На рівні прикладного програмного інтерфейсу широко використовується стиль REST (Representational State Transfer), який передбачає представлення сутностей системи у вигляді ресурсів, доступних через HTTP-методи (GET, POST, PUT, DELETE тощо). Це

дозволяє побудувати єдине API, до якого можуть підключатися веб-клієнти, мобільні додатки та інші інформаційні системи.

З точки зору організації клієнтського інтерфейсу для веб-додатків можна виділити два основні підходи: багатосторінкові додатки (MPA, Multi-Page Application) та односторінкові додатки (SPA, Single-Page Application). У класичній багатосторінковій моделі кожна дія користувача призводить до завантаження нової HTML-сторінки з сервера. Такий підхід простіший у реалізації, однак менш зручний для складних інтерфейсів із великою кількістю динамічних елементів, оскільки кожне перезавантаження сторінки супроводжується повною перерисовкою вмісту та додатковими затримками [1], [4].

У моделі SPA основна HTML-сторінка завантажується один раз, а подальша робота відбувається за рахунок асинхронних запитів до сервера (AJAX, Fetch, Web API) і динамічного оновлення вмісту за допомогою JavaScript. Це дозволяє реалізувати інтерактивні адміністративні панелі з таблицями, фільтрами, календарями, дашбордами без постійних перезавантажень сторінки, що суттєво підвищує зручність роботи адміністратора медичного центру.

Для побудови сучасних SPA активно використовуються JavaScript-бібліотеки та фреймворки (React, Angular, Vue тощо), які забезпечують компонентний підхід до розробки інтерфейсу. Компонентна модель дає можливість розділити інтерфейс на незалежні логічні блоки (таблиця пацієнтів, форма запису, панель фільтрів, графік завантаженості лікарів), повторно використовувати їх у різних частинах системи, спрощувати супровід та розвиток коду [4]. У поєднанні зі станом додатка (state management) це дозволяє забезпечити узгодженість відображуваних даних на сторінці з їхнім фактичним станом у системі.

На серверному боці веб-орієнтованих систем широке застосування отримали високорівневі платформи та фреймворки, що автоматизують рутинні задачі розробника: обробку HTTP-запитів, маршрутизацію, роботу з базою даних, реалізацію механізмів аутентифікації та авторизації, логування, обробку помилок тощо. До таких платформ належать, зокрема, Node.js з фреймворками Express,

NestJS, середовище .NET з ASP.NET Core, фреймворки Django, Flask, FastAPI для мови Python, Laravel, Symfony для PHP, Spring Framework для Java та інші [3], [4].

Вибір конкретної серверної технології залежить від вимог до продуктивності та масштабованості, доступності бібліотек, зручності роботи з базами даних та зовнішніми сервісами, а також від кваліфікації розробників. Для системи адміністрування медичного центру важливими є: підтримка ролей і прав доступу, можливість побудови REST API, інтеграція з реляційними СУБД, засоби захисту даних і підтримка подальшого розширення функціональності.

На рівні зберігання даних найбільш поширеними є реляційні системи керування базами даних (СУБД) - PostgreSQL, MySQL/MariaDB, Microsoft SQL Server, Oracle Database тощо. Вони забезпечують підтримку транзакцій, цілісність та узгодженість даних, можливість побудови складних запитів і звітів, механізми резервного копіювання та відновлення. Для інформаційної системи медичного центру, де ведеться облік пацієнтів, лікарів, записів на прийом і пов’язаних з ними довідників, реляційна модель даних є природною, оскільки дозволяє чітко описати зв’язки між сутностями (пацієнт - записи, лікар - розклади, статуси візитів тощо) [2], [3].

Поряд із реляційними СУБД застосовуються й нереляційні (NoSQL) рішення, орієнтовані на зберігання слабо структурованих даних, журналів подій, кешів тощо. Вони можуть бути корисними для додаткових задач (наприклад, зберігання історії логів або проміжних результатів аналітики), однак базові дані про пацієнтів і записи доцільно зберігати саме в реляційній БД.

Важливим аспектом побудови веб-орієнтованих систем є інтеграція з зовнішніми сервісами та інформаційними системами. Для автоматизованої системи медичного центру актуальними є:

- інтеграція з медичною інформаційною системою закладу та/або електронною системою охорони здоров’я (для узгодження даних про пацієнтів, статуси візитів, направлення тощо);
- взаємодія з сервісами надсилання SMS та e-mail-повідомлень (нагадування пацієнтам про візити);

- можливість експорту даних у зовнішні аналітичні системи або офісні пакети для подальшої обробки управлінської звітності.

Реалізація інтеграційних можливостей зазвичай здійснюється через відкриті REST API, використання вебхуків, черг повідомлень або інших стандартних протоколів обміну даними. Це дозволяє уникнути “замкнутості” рішення та забезпечити його взаємодію з уже наявною інформаційною інфраструктурою медичного закладу.

Таким чином, аналіз сучасних технологій побудови веб-орієнтованих інформаційних систем показує доцільність використання багаторівневої клієнт–серверної архітектури з розділенням рівнів представлення, бізнес-логіки та даних, застосування SPA-підходу для розробки зручної адміністративної панелі, використання високорівневого серверного фреймворку з підтримкою REST API та реляційної СУБД для зберігання основних даних медичного центру. Конкретний вибір технологій і інструментальних засобів для реалізації автоматизованої системи процесів адміністрування буде обґрунтовано у наступних підпунктах розділу.

2.2 Аналіз інструментів проєктування інтерфейсів користувача (UI/UX, Figma тощо)

Якість користувацького інтерфейсу є одним із ключових факторів ефективності веб-орієнтованих інформаційних систем, особливо в умовах інтенсивної роботи адміністратора медичного центру. Інтерфейс повинен забезпечувати швидкий доступ до основних операцій (запис, перенесення візитів, перегляд розкладів, формування списків пацієнтів), бути інтуїтивно зрозумілим і не вимагати від користувача спеціальної підготовки в галузі інформаційних технологій [5]. Тому вибір технологій для розробки інтерфейсу є важливим етапом проєктування системи.

Сучасні підходи до розробки веб-інтерфейсів ґрунтуються на використанні JavaScript-фреймворків та бібліотек, які реалізують компонентну модель побудови інтерфейсу. Найбільш поширеними рішеннями є React, Angular та Vue.js. Вони дозволяють розбити інтерфейс на окремі компоненти (таблиця пацієнтів, панель фільтрів, форма створення запису, дашборд завантаженості лікарів), які можуть повторно використовуватися та незалежно модифікуватися [6]. Це спрощує супровід системи, дає можливість поступового нарощування функціональності та зменшує ризик помилок при внесенні змін.

Фреймворк Angular надає розробнику розширений набір засобів «з коробки» (модулі, служби, засоби маршрутизації, форми, валідація тощо), однак має досить високий поріг входу і суворішу структуру проєкту. Це виправдано для великих командних проєктів, але для розробки окремого модулю адміністрування може бути надмірно складним. Vue.js характеризується простотою освоєння та гнучкістю, однак має дещо меншу поширеність у порівнянні з React та Angular, що впливає на доступність навчальних матеріалів і готових компонентів для побудови адміністративних панелей [6], [7].

Бібліотека React позиціонується як засіб для побудови інтерфейсу, зосереджений саме на представленні даних і компонентній моделі. Вона широко використовується у промислових веб-проєктах, має велику екосистему супутніх бібліотек і готових рішень для створення таблиць, форм, дашбордів, систем маршрутизації, керування станом додатка тощо [7]. React забезпечує:

- декларативний підхід до опису інтерфейсу, коли стан компонента однозначно визначає його відображення;
- легку інтеграцію з REST API серверної частини;
- можливість побудови односторінкового додатка (SPA) з мінімальною кількістю перезавантажень сторінок;
- активну підтримку спільноти та наявність якісної документації.

З огляду на ці переваги для розробки інтерфейсу автоматизованої системи процесів адміністрування медичного центру обрано саме React. Для швидкого розгортання та налаштування проєкту використано інструмент Vite, який

забезпечує мінімалістичну конфігурацію, швидку збірку та зручне середовище розробки. Vite дозволяє зосередитися на реалізації логіки інтерфейсу, не витрачаючи значних зусиль на налаштування складного інструментарію збірки.

Окреме значення для побудови зручного інтерфейсу мають засоби стилізації та верстки. Серед поширених підходів можна виділити:

- використання класичних CSS-файлів, підключених до компонента або сторінки;
- застосування препроцесорів (Sass, Less) для спрощення організації стилів;
- використання CSS-фреймворків (Bootstrap, Bulma, Tailwind CSS), що надають готові класи оформлення;
- застосування підходів CSS-in-JS, коли стилі описуються безпосередньо в JavaScript-коді компонентів.

CSS-фреймворки типу Bootstrap значно прискорюють створення стандартних елементів інтерфейсу (кнопки, таблиці, модальні вікна), однак можуть ускладнювати гнучку кастомізацію зовнішнього вигляду і призводити до перевантаження сторінки зайвими стилями [5], [7]. Підхід CSS-in-JS дозволяє локалізувати стилі в межах конкретного компонента та уникнути конфліктів між класами, проте збільшує обсяг коду компонентів.

З урахуванням відносно невеликого обсягу інтерфейсу, характерного для адміністративної панелі, та необхідності продемонструвати зрозумілу структуру верстки у межах магістерської роботи, доцільним є використання комбінації простих CSS-стилів та локальних стилів компонентів. Такий підхід забезпечує достатню гнучкість оформлення, не вимагає підключення важких фреймворків та спрощує розуміння коду інтерфейсу.

Важливим етапом створення зручного інтерфейсу є проєктування та прототипування. Для цього широко використовується графічний редактор Figma, який дозволяє будувати інтерактивні прототипи, макети екранів, узгоджувати розташування елементів, розміри, відступи, типографіку та кольорові схеми [8]. Перевагами Figma є:

- робота безпосередньо у веб-браузері;

- можливість спільного редагування макету декількома користувачами;
- наявність бібліотек компонентів і шаблонів інтерфейсів;
- простий експорт графічних елементів для подальшого використання у веб-додатку.

Для адміністративної панелі медичного центру Figma використовується на етапі проєктування для створення макетів основних екранів: головної панелі з вкладками “Пацієнти”, “Лікарі”, “Записи на прийом”, таблиць даних, форм створення та редагування записів, а також дашбордів із ключовими показниками. Це дозволяє заздалегідь узгодити розташування елементів інтерфейсу, забезпечити візуальну цілісність системи та спростити подальшу реалізацію інтерфейсу в React.

Таким чином, аналіз засобів розробки користувацького інтерфейсу показав доцільність використання React як основної бібліотеки для побудови компонентного інтерфейсу односторінкового веб-додатка, Vite як інструмента швидкого розгортання та збірки проєкту. Обраний набір технологій забезпечує поєднання гнучкості, продуктивності та зручності розробки, що є важливим для реалізації автоматизованої системи процесів адміністрування медичного центру.

2.3 Порівняння мов програмування, фреймворків та СУБД для реалізації системи

Вибір відповідних технологій для реалізації автоматизованої системи управління адміністративними робочими процесами в медичному закладі є ключовим етапом проектування. Конкретна мова програмування, обрана структура та система управління базами даних мають прямий вплив, що виходить за рамки простої швидкості створення прототипів; вони суттєво впливають на поточне обслуговування, майбутню масштабованість, сумісність з існуючими медичними інформаційними системами та загальну надійність рішення. Отже, перед початком фактичної розробки програмного забезпечення було проведено порівняльне

дослідження, що охоплювало кілька поширених технологічних стеків, здатних стати основою веб-адміністративної панелі для медичного центру.

Спочатку важливо встановити основні контрольні показники, що використовувалися для оцінки. Ці контрольні показники охоплювали: дотримання архітектури “клієнт-сервер”, вбудовану підтримку як розгортання веб-додатків, так і створення RESTful API, існування добре налагодженої екосистеми в поєднанні з активною залученістю спільноти, простоту навчання та використання, можливість безперешкодного взаємодії з реляційними рішеннями для зберігання даних, проддуктивність та потенціал для зростання, кросплатформеність і відсутність жорсткої прив’язки до конкретної операційної сисетми. Крім того, особливу увагу було приділено наявності готових програмних модулів, придатних для створення інтуїтивно зрозумілого інтерфейсу адміністратора, що містить такі важливі компоненти, як сортовані таблиці, форми введення даних, інтерактивні модальні вікна та функціональний календар запису на прийом.

Розглядаючи технології бекенду, було оцінено кілька варіантів: C# та .NET, Java з використанням Spring framework, Python з використанням Django або Flask та JavaScript, що працює на Node.js з Express. Усі вони є поширеними інструментами професійної розробки, кожен з яких має свої переваги та недоліки. Екосистема C#/.NET має вагомий досвід у корпоративному середовищі, пропонує надійні інструменти та охоплює сучасні веб-стандарти через ASP.NET Core. Тим не менш, розгортання C#/.NET часто вимагає складнішої конфігурації інфраструктури, зазвичай на користь середовищ Windows або контейнеризації, що збільшує вимоги до серверних ресурсів. Для попередньої моделі навчання така вимоглива інфраструктура не є непотрібною.

Java у поєднанні зі Spring пропонує порівнянні переваги: відмінну пропускну здатність, надійність, сувору типізацію та велику колекцію бібліотек, що охоплюють безпеку, управління транзакціями, взаємодію з базами даних та різні інтеграції. Водночас Java відзначається вищим порогом входу, необхідністю конфігурації численних модулів, а готовий застосунок зазвичай займає більше

пам’яті та потребує більше ресурсів для запуску, ніж аналогічні рішення на динамічних мовах.

Python, що використовує Django або Flask, привабливий завдяки своєму простому синтаксису та доступу до багатої екосистеми, яка включає інструменти для обробки даних та штучного інтелекту. Django відзначається тим, що пропонує комплексні вбудовані функції, включаючи об'єктно-реляційний маппер (ORM), автоматизований адміністративний інтерфейс, керування користувачами та механізм шаблонів, які значно пришвидшують початкове налаштування проекту. Незважаючи на це, якщо інтерфейс користувача створюється як односторінковий додаток (SPA) за допомогою окремого фронтенд-фреймворку, деякі функції Django для створення шаблонів HTML на стороні сервера стають зайвими. Більше того, швидкість обробки Python у сценаріях з великим трафіком зазвичай перевершується Node.js або Java. Хоча це не є основною проблемою для навчального прототипу, це може стати обмеженням при масштабуванні до масштабного виробництва.

Ще одна категорія життєздатних варіантів включає платформи, побудовані на JavaScript та Node.js. Ця архітектура унікальним чином використовує однакову мову програмування як для клієнтської (фронтенд), так і для серверної (бекенд) сторони. Такі інструменти, як фреймворк Express, пропонують компактний, але адаптивний інструментарій, призначений для створення RESTful API, керування логікою маршрутизації, обробки вхідних запитів, інтеграції функцій проміжного програмного забезпечення, тощо. Асинхронний характер Node.js виявляється особливо вигідним для завдань, що характеризуються великим обсягом повторюваних запитів, спрямованих на бази даних або зовнішні служби. Поширений сценарій під час роботи з історіями пацієнтів, інформацією про лікарів та координацією розкладу. Крім того, стандартизація на одній мові, JavaScript, значно спрощує підтримку коду та зменшує технологічні накладні витрати, пов'язані з вивченням та підтримкою різних наборів інструментів.

Враховуючи ці фактори, було прийнято рішення впровадити Node.js, використовуючи фреймворк Express, для серверних компонентів системи. Цей

вибір дозволив створити просту, але достатньо адаптивну структуру бекенду: окремі кінцеві точки для керування даними пацієнтів, медичними записами, послугами та призначеннями, централізоване управління помилками та узгоджений формат JSON для обміну даними. Крім того, ця технологія плавно інтегрується практично з будь-якою реляційною системою керування базами даних і легко упаковується для контейнерного розгортання в хмарних середовищах.

Не менш важливим є вибір фронтенд-технології, оскільки інтерфейс адміністратора безпосередньо впливає на ефективність роботи персоналу медичного центру. Серед сучасних фреймворків для розробки односторінкових додатків (SPA) були оцінені Angular, Vue.js та React. Angular виділяється як комплексний фреймворк, що може похвалитися власною архітектурою для модулів, шаблонів, впровадження залежностей та реактивних форм. Хоча він дуже підходить для значних проектів, що вимагають чітко визначеної командної співпраці, він також вимагає дотримання порівняно жорсткої структури та має більш високий поріг входу. Для невеликого адміністративного інтерфейсу така складність фреймворку, ймовірно, перешкоджатиме розробці, не даючи відповідних переваг.

Vue.js характеризується як компактний фреймворк, розроблений для поступового впровадження. Він сприяє швидкій розробці інтерактивних інтерфейсів користувача завдяки простому синтаксису шаблонів. Тим не менш, його екосистема бібліотек та компонентів менш обширна, ніж у React, особливо щодо спеціалізованих компонентів, таких як складні панелі медичного адміністрування, календарі планування або попередньо створені адаптивні таблиці. Хоча цей недолік не є критичним, проте ініціювання реального застосунку з нуля може вимагати створення більшої кількості функцій внутрішньо.

На відміну від Angular, React працює більше як бібліотека зосереджена на побудові інтерфейсу користувача, ніж як повний, всеохоплюючий фреймворк. Його переваги випливають з його компонентної архітектури, використання віртуального DOM, використання декларативного програмування та величезного репозиторію легкодоступних інструментів: універсальні таблиці, що підтримують сортування та фільтрацію, різні поля введення форм, діалогові вікна, селектори

дати та часу та бібліотеки побудови графіків, і це лише деякі з них. React безперешкодно підключається до будь-якого бекенду, який пропонує RESTful API, а мережеві операції зазвичай керуються або за допомогою вбудованих можливостей `fetch`, або спеціалізованих зовнішніх пакетів.

У контексті даного проєкту React забезпечив можливість розробити цілісний інтерфейс адміністратора: бічна панель навігації з окремими розділами для «Прийомів», «Пацієнтів», «Лікарів» та «Послуг», верхня панель, що відображає поточного адміністратора, сітки даних з однаковим стилем для списків пацієнтів, лікарів та послуг, спливаючі діалогові вікна для редагування записів для пацієнтів та лікарів, та привабливий щомісячний календар, що ілюструє заплановані зустрічі.

Супровідний розділ огляду зосереджувався на виборі відповідної системи управління базами даних. Враховуючи, що галузь дослідження за своєю сутністю включає окремі сутності, такі як пацієнт, лікар, певна послуга, планування зустрічі та медичний карта, а також зв'язки, що їх пов'язують, логічний шлях вказує на використання реляційної СУБД, яка за своєю сутністю підтримує транзакційні операції, забезпечує правильність даних та використовує обмеження зовнішніх ключів. Серед оцінюваних претендентів були MySQL та його похідна MariaDB, PostgreSQL, SQLite та MongoDB, яка є представником родини NoSQL, орієнтованої на документи.

Як MySQL, так і MariaDB надзвичайно популярні в галузі розробки веб-додатків, пропонуючи достатню швидкість обробки разом зі складними методами реплікації та масштабування. Тим не менш, їх реалізація вимагає запуску окремого серверного процесу, що включає налаштування для користувачів, налаштування дозволів та розробку стратегій резервного копіювання. Для попередньої моделі, розгорнутої або на локальному комп'ютері, або на скромному сервері, цей рівень інфраструктури виявляється дещо надмірним та ускладнює початкове налаштування середовища.

З іншого боку, PostgreSQL пропонує ще більш повний набір функцій: здатність обробляти складні формати даних, чудову підтримку транзакцій, тригери, збережені процедури та різноманітні альтернативи індексації. Це робить його

чудовим вибором для систем виробничого рівня, створених для величезних обсягів даних та суворих стандартів надійності. Однак, під час розробки прототипу, де кількість записів обмежена кількома сотнями або, можливо, кількома тисячами, використання всього потенціалу PostgreSQL є надмірним, а заглиблення в його спеціалізовані функції додає зайвого обсягу проекту.

NoSQL-бази даних, такі як MongoDB, чудово справляються з обробкою даних, які мають слабку структуру, зазвичай у форматі документів, що робить їх ідеальними для середовищ, де схема даних є гнучкою або погано визначеною. Однак у медичному закладі організація даних, як правило, досить жорстка: пацієнт постійно має повне ім'я, контактну інформацію та дату народження, лікар має галузь спеціалізації, контактний номер та власний номер кабінету, зустрічі незмінно пов'язують пацієнта, лікаря, послугу, певну дату та час, а також статус. Отже, використання документоорієнтованої системи керування базами даних (СКБД) не дає суттєвих переваг, проте створює складнощі щодо підтримки цілісності даних та забезпечення транзакційної узгодженості.

З огляду на цей сценарій, PostgreSQL найкраще підходить для попереднього, розробницького прототипу. Він функціонує як реляційна СКБД, яка зберігає всі дані у одному файлі й не вимагає підняття окремого серверного процесу. Доступ до цієї бази даних ініціюється безпосередньо з середовища Node.js, що спрощує як процес розгортання, так і портативність проекту. Для запуску програми потрібен лише інтерпретатор Node.js та сам файл бази даних. Для масштабу операцій, типового для невеликої клініки, де одночасне навантаження користувачів не є головною проблемою, можливості швидкості PostgreSQL цілком адекватні. Крім того, дизайн бази даних відповідав стандартам SQL, свідомо уникаючи функцій, унікальних для будь-якої окремої СУБД. Ця передбачливість означає, що якщо система перейде на промислове використання, міграція програми на MySQL може бути здійснена без суттєвих змін основної бізнес-логіки.

Ще однією переконливою причиною на користь вибору Node.js, Express, React у поєднанні з PostgreSQL є те, що кожен компонент у цьому стеку є вільно доступним, з відкритим вихідним кодом та підкріплений великою історією

розробки. Така конфігурація мінімізує прив'язку до постачальника, дозволяє використовувати систему без ліцензійних зборів та забезпечує доступ до великої кількості навчальних ресурсів, зразків коду та попередньо створених компонентів.

Важливо підкреслити вплив вибору технології на дизайн окремих компонентів. Використання React значно спростило розробку складних функцій інтерфейсу користувача. Яскравим прикладом є інтерфейс запису на прийом, побудований як незалежний блок. Цей модуль отримує масив даних про записи у форматі JSON із сервера, автоматично сортує ці записи за часом, візуально позначає поточну дату та дозволяє користувачам переглядати повні щоденні розклади в модальному діалоговому вікні. Блоки відображення для специфікацій пацієнтів, інформації про лікарів та каталогів послуг створені з єдиного, багаторазового компонента. Цей компонент підтримує сортування на основі різних атрибутів, зберігає заголовки стовпців видимими під час вертикального переміщення та містить інтерактивні кнопки для відкриття історії хвороби пацієнта або професійного резюме лікаря. Реалізація цього підходу призвела б до значно більших труднощів, якби розробка покладалася на традиційний рендеринг сторінок, який повністю обробляється сервером.

Бекенд використовує фреймворк Express для створення окремої організації точок входу API. Ця сегментована структура має покращити читабельність коду та тісно дотримуєтись принципів REST. Запити типу GET, POST, PUT та DELETE - безпосередньо пов'язана з фундаментальними діями бази даних, такими як отримання, створення, оновлення та видалення ресурсів. Крім того, функції проміжного програмного забезпечення використовуються для керування обробкою базових помилок та реєстрації вхідних запитів, що значно спрощує усунення несправностей протягом усього етапу розробки.

Взаємодія з базою даних PostgreSQL керується за допомогою спеціально створеного рівня абстракції даних. Цей рівень централізує всі основні дії щодо збереження даних, включаючи додавання, зміну та видалення записів для клієнтів, спеціалістів, доступних послуг та бронювань, а також складні запити, які вимагають об'єднання даних з кількох таблиць. Цей специфічний вибір дизайну

надає системі гнучкості, цей рівень доступу пізніше може бути адаптований для зв'язку з іншою системою баз даних без необхідності змін у презентації фронтенду або визначеннях шляхів API бекенду.

Узагальнюючи результати порівняння, можна зробити висновок, що обраний стек технологій забезпечує необхідний баланс між простотою та актуальністю. Впровадження Node.js у поєднанні з React дозволяє розробити повністю адаптивну веб-панель адміністрування з привабливим візуальним дизайном, динамічним календарем та зручними формами введення. Крім того, використання PostgreSQL спрощує процес розгортання та перевірки початкової концепції, водночас забезпечуючи можливість переходу на більш традиційні системи управління базами даних у майбутньому. Отже, для втілення автоматизованої системи управління адміністративними завданнями в медичному закладі було обрано оптимальне поєднання JavaScript та реляційної бази даних PostgreSQL.

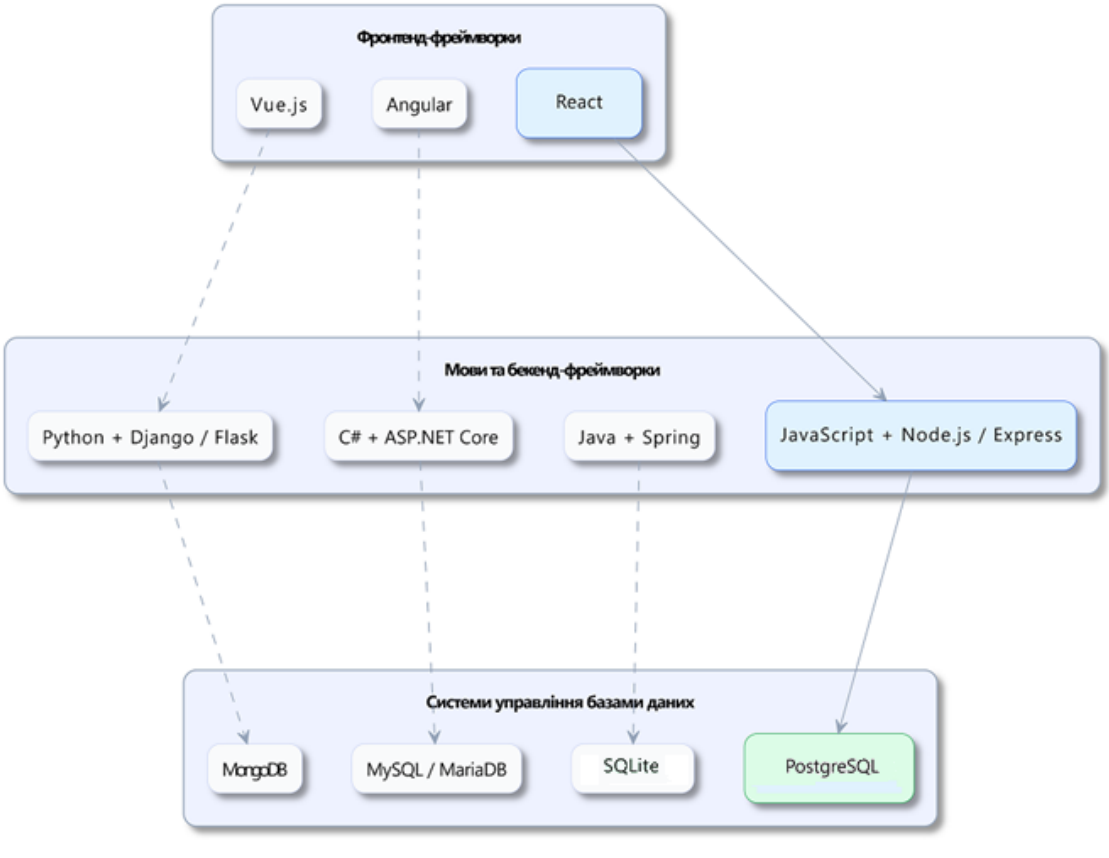


Рис. 2.1 Схема порівняння можливих стеків технологій та обраного варіанту для розробки системи адміністрування медичного центру.

На рис. 2.1 схематично показано порівняння основних стеків технологій, які можуть бути використані для реалізації адміністративної панелі медичного центру. На діаграмі виділено обраний варіант (React для фронтенду, JavaScript/Node.js/Express для бекенду та PostgreSQL для зберігання даних), а пунктирними стрілками позначено можливі альтернативи, розглянуті під час аналізу. Така візуалізація наочно показує логіку вибору технологій та їх взаємозв'язок у межах єдиної архітектури системи.

2.4 Обґрунтування вибору стеку технологій для розробки системи

Порівняльний аналіз мов програмування, фреймворків та систем управління базами даних, показав перевагу стеку технологій, який поєднує сучасні функції, мінімальні витрати на розгортання, просте обслуговування та можливість подальшого розширення під час розробки веб-панелі керування для медичного закладу. Враховуючи специфікації, а також конкретні операційні потреби, такі як обробка цифрових записів пацієнтів, контроль планування та управління послугами та лікарями, комбінація, реалізована в цьому проєкті, використовує React для інтерфейсу користувача, JavaScript/Node.js/Express для серверної частини та PostgreSQL як систему зберігання даних.

Це конкретне поєднання сприяє реалізації єдиної повноцінної логіки за допомогою JavaScript як на стороні користувача, так і в серверному середовищі. Використання єдиної мови мінімізує неточності, які часто виникають через перетворення структур даних між різними технологіями, спрощує адаптацію для нових програмістів та прискорює цикл розробки. Моделі сутностей (пацієнт, лікар, послуга, запис на прийом) описуються у схожому вигляді на бекенді й у фронтенді, що полегшує повторне використання валідаторів, форм і компонентів інтерфейсу. Ця методологія пропонує значні переваги, особливо для невеликих груп розробників, зосереджуючи зусилля на основній функціональності, а не борючись зі складнощами інтеграції кількох мов.

Рішення обрати React як основний фронтенд-фреймворк випливає з поєднання переваг. В першу чергу, React використовує компонентну архітектуру: кожна частина інтерфейсу користувача, така як форми для введення даних, таблиці списків пацієнтів, компоненти планування та спливаючі вікна медичних записів, побудована як незалежний, багаторазово використовуваний та комбінований блок. Це принципово спрощує як обслуговування, так і розширення функцій інтерфейсу. Введення нового сегмента до адміністративної панелі зазвичай передбачає інтеграцію існуючих компонентів, а не переробку значних частин кодової бази.

По-друге, React чудово справляється зі створенням односторінкових додатків (SPA), де контент оновлюється динамічно без необхідності повного перезавантаження сторінки. Ця можливість життєво важлива для адміністративної системи охорони здоров'я: персонал часто переміщується між різними представленнями - списками записей, медкартами пацієнтів, розкладами лікарів та каталогами послуг, застосовуючи фільтри та параметри сортування або відкриваючи детальні записи. Ефективний конвеєр рендерингу React гарантує, що оновлення, такі як зміна статусу запису з «заплановано» на «завершено», відображаються миттєво, забезпечуючи більш плавний та ефективний користувацький досвід для персоналу.

Крім того, React має переваги завдяки надійній екосистемі та значній базі користувачів. Попередньо створені бібліотеки компонентів для таких елементів, як таблиці, календарі, модальні вікна та поля введення форм, легкодоступні, що спрощує їхнє інтегрування в будь-який проект. Така велика кількість ресурсів дозволяє зосередити зусилля розробки на окремих функціях, необхідних медичному закладу, таких як архітектура записів пацієнтів, спеціалізовані поля послуг та логіка що керує щоденним та специфічним для лікаря відображенням зустрічей, замість того, щоб витрачати час на створення фундаментальних елементів інтерфейсу заново. Додаткова перевага полягає в безперешкодному поєднанні між компонентами React та дизайном: візуальну специфікацію можна безпосередньо перенести в робочий макет, забезпечуючи ідеальну відповідність між запланованим інтерфейсом користувача та його кінцевою реалізацією.

На серверній частині технологічний стек базувався на Node.js, використовуючи фреймворк Express. Відображаючи переваги фронтенду, цей вибір підтримує узгодженість JavaScript по всьому стеку, спрощуючи інтегроване моделювання як обробки даних, так і логіки програми. Node.js побудований на неблокуючій подієвій моделі введення-виведення, що робить його дуже ефективним для веб-застосунків, де основні завдання зосереджені на обробці HTTP-запитів, взаємодії з базами даних та передачі корисних даних JSON. Враховуючи очікуване операційне навантаження на адміністративну платформу медичного центру, що включає обмежену кількість адміністраторів, лікарів та реєстраторів, не було потреби в надто складних, високопродуктивних архітектурах. Отже, Node.js пропонує достатній запас продуктивності.

Express, у свою чергу, пропонує мінімалістичну, але адаптивну основу для побудови RESTful інтерфейсів. Кінцеві точки, призначені для пацієнтів, лікарів, документації та різних служб, структуровані в окремі колекції обробників запитів, що відображають організацію складових модулів системи. Така модульність спрощує процес документування API, спрощує інтеграцію заходів безпеки, функцій журналювання та протоколів управління помилками, а також по суті підтримує майбутнє масштабування, таке як впровадження мобільного додатку або взаємодія із зовнішніми системами медичних даних через ідентичні точки доступу REST. Крім того, ключовою перевагою Express є відсутність архітектурної жорсткості, що дозволяє безперешкодну інтеграцію зі специфічними архітектурними схемами та встановленими шаблонами проєктування.

Для постійного зберігання даних була обрана реляційна система управління базами даних PostgreSQL. На відміну від важких серверних рішень СУБД, таких як MySQL, PostgreSQL усуває необхідність запуску окремого серверного процесу, натомість весь набір даних знаходиться в одному файлі бази даних, до якого безпосередньо звертається серверна частина додатку. Для невеликого медичного центру це має низку переваг:

- спрощує початкове налаштування та розгортання програмного забезпечення (вимагаючи лише копіювання файлу бази даних разом із запуском серверного компонента).
- зменшуються витрати на адміністрування, оскільки не потрібні окремі інструменти для керування сервером БД.
- резервне копіювання виконується простим копіюванням файлу, що зручно для локальних розгортань або стендів.

PostgreSQL включає такі функції, як обробка транзакцій, гарантії узгодженості даних та дотримання принципів ACID - усі необхідні передумови для будь-якої програми, що працює з конфіденційними медичними даними. Структура бази даних, що включає таблиці для пацієнтів, лікарів, медичних записів та послуг, відповідає встановленим реляційним правилам, використовуючи первинні та зовнішні ключі. Такий вибір дизайну означає, що якщо виникне потреба перейти на більш надійну систему управління базами даних, основна бізнес-логіка програми повинна вимагати мінімальних, якщо взагалі вимагатиме, змін. Отже, PsotgreSQL досягає ідеального балансу між легкістю розгортання в невеликому локалізованому медичному закладі та збереженням потенціалу для майбутнього зростання.

Ще однією перевагою, що підтримує вибір цього технологічного стеку, є повна відсутність ліцензійних зборів та його тверда відданість відкритим галузевим специфікаціям. React, Node.js, Express та PstgreSQL - це програмні компоненти, що розповсюджуються за ліцензіями з відкритим кодом, що означає, що їх можна розгортати як в академічних умовах, так і в комерційних підприємствах без додаткових грошових витрат. Цей аспект виявляється особливо корисним для невеликих, незалежних медичних практик, яким часто бракує фінансових ресурсів для дорогих власних програмних пакетів, але які все ще потребують сучасних можливостей автоматизації.

Обраний стек ідеально відповідає практичним можливостям розробника, відточеним завдяки попереднім навчальним завданням та самостійній роботі. Це дає змогу зменшити ризики помилок на етапі реалізації та прискорити розробку

прототипу. Дійсно, існуюча, робоча версія інтерфейсу адміністратора служить доказом того, що використання React разом з Node.js/Express та PstgreSQL здатне задовольнити всі функціональні завдання, викладені в першому розділі: управління списками пацієнтів та лікарів, адміністрування послуг, ведення графіка прийому, забезпечення пошуку та уточнення даних, а також включення фундаментальних засобів контролю доступу для адміністраторів.

Таким чином, за результатами аналітичного огляду можливих технологій та з урахуванням функціональних і нефункціональних вимог до автоматизованої системи процесів адміністрування, рішення про використання конфігурації React + Node.js/Express + PostgreSQL є обґрунтованим. Ця конфігурація досягає оптимального балансу між сучасними стандартами та простотою впровадження. Це сприяє створенню інтуїтивно зрозумілого користувацького інтерфейсу для персоналу медичного закладу, гарантує цілісність збережених медичних записів та зберігає можливість майбутнього розширення та масштабування без необхідності фундаментального перегляду структури системи.

2.5 Вибір архітектури програмної системи та шаблонів проєктування

Вибір відповідного програмного забезпечення є вирішальним етапом. Це рішення має враховувати не лише виконання певних функціональних можливостей, але й забезпечення подальшого обслуговування, можливості розширення та надійної роботи автоматизованих робочих процесів у процесах адміністрування медичного центру. На цьому етапі важливо зіставити критерії відповідності, зі стеком технологій React - Node.js / Express - PostgreSQL. Крім того, це включає визначення логічної структури компонентів, уточнення того, як ці частини будуть взаємодіяти, та визначення точного обсягу відповідальності для кожного сегмента системи.

Враховуючи масштаб проекту, очікуваний обсяг користувачів та вимогу до простих процедур розгортання, використання традиційної трирівневої клієнт-

серверної архітектури є найрозумнішим шляхом. Ця структура чітко розділяє рівень представлення (фронтенд), рівень бізнес-логіки (бекенд) та рівень збереження даних (Система керування базами даних). Цей архітектурний шаблон є стандартним для сучасних веб-інформаційних систем та ідеально відповідає обраному набору технологій. Отже, система буде розроблена як єдиний монолітний додаток з чітко розмежованими шарами всередині свого середовища, що дозволить уникнути непотрібної структурної складності, пов'язаної з мікросервісами або сервісно-орієнтованими архітектурами, які були б непропорційними для потреб інформаційної системи, що обслуговує один медичний заклад.

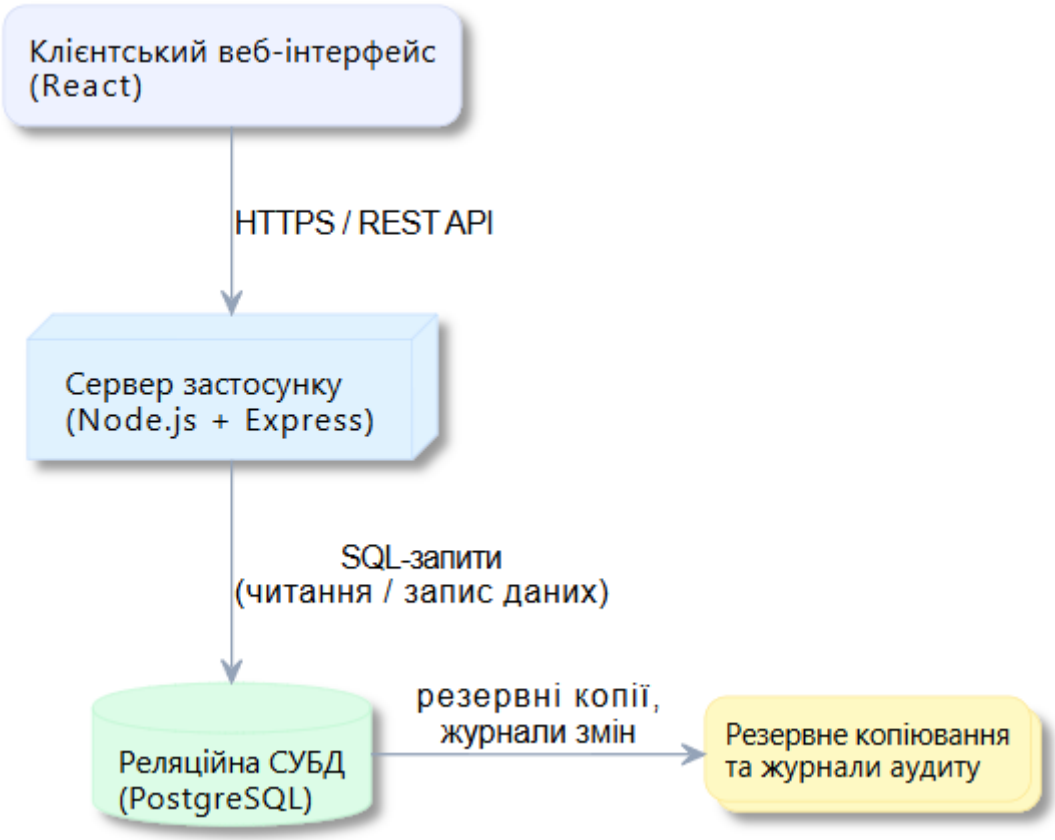


Рис. 2.2 Узагальнена трирівнева архітектура автоматизованої системи процесів адміністрування медичного центру.

На рисунку 2.2 у наочній формі показано трирівневу архітектуру розроблюваної системи. Клієнтський рівень представлений односторінковим веб-додатком на основі React, який взаємодіє з серверним застосунком Node.js/Express через REST-

інтерфейс поверх протоколу HTTP(S). Сервер, у свою чергу, працює з реляційною базою даних PostgreSQL через абстрактний шар доступу до даних, що відповідає обраному шаблону Repository.

Рівень інтерфейсу користувача використовує односторінковий додаток (SPA), розроблений за допомогою React. Структурно інтерфейс сегментовано на кілька функціональних областей: область “Графік прийому”, область “Керування пацієнтами”, область “Керування лікарями”, область “Каталог послуг” та область “Вхід адміністратора”. У кожній області інтерфейс побудовано з окремих компонентів React, призначених для певних візуальних елементів: сітки даних, форми введення даних, спливаючі вікна та механізми фільтрації та впорядкування даних. Компонентний підхід фактично реалізує шаблон “Component-based UI”, який можна розглядати як розвиток класичних шаблонів “Model–View–Controller” і “Model–View–Presenter” на боці клієнта.

Важливою рисою обраної архітектури є поділ компонентів інтерфейсу на так звані “розумні” (контейнерні) та “презентаційні”. Контейнерні компоненти керують взаємодією з сервером, включаючи отримання даних, реагування на дії користувача, такі як створення, модифікація або видалення записів, відправлення необхідних викликів REST-API та контроль внутрішнього стану. Потім вони передають отримані оброблені дані компонентам презентації. І навпаки, компоненти презентації займаються виключно відображенням візуального виводу та залишаються агностичними щодо походження даних. Ця диференціація безпосередньо відображається в архітектурному стилі “Container / Presentational”, тим самим підвищуючи можливість повторного використання компонентів та мінімізуючи надлишковий код відображення. Наприклад, таблиці даних пацієнтів, лікарів та послуг використовують спільні елементи презентації, налаштовані з різними наборами стовпців, тоді як відповідні їм компоненти-контейнери мають спеціалізовану логіку для пошуку та маніпулювання даними.

Адміністративний інтерфейс вирізняється значною залежністю від інтерактивних компонентів, таких як випадаючі меню з можливістю пошуку, механізми фільтрації, засоби вибору дати та модальні вікна для доступу до

медичних записів. Для досягнення цієї динамічної функціональності стратегія управління станом використовує React Hooks. Використання таких хуків useState та useEffect, можна розглядати як окремий шаблон дизайну - “функціональний компонент зі збереженням стану”, який замінює звичайні класи компонентів. Цей метод пропонує чіткіше визначення етапів життєвого циклу елементів інтерфейсу. Поточний стан інтерфейсу, що охоплює такі деталі, як вибраний фільтр, активна вкладка або переглянутий діапазон дат у календарі, підтримується внутрішньо в межах найвищих компонентів і передається вниз через props. Для системи, розробленої навколо одного типу користувача (адміністратора), цей метод виявляється достатнім і усуває необхідність у складних глобальних рішеннях для управління станом.

Основна логіка програми реалізована у вигляді серверного застосунку, побудованого за допомогою Node.js з використанням фреймворку Express. Цей рівень використовує архітектурну структуру, яка точно відображає канонічну багатошарову архітектуру, поділену на такі окремі логічні компоненти:

- шар маршрутизації і контролерів, який приймає HTTP-запити від клієнта, перевіряє їх коректність, викликає необхідні сервіси та формує HTTP-відповіді у форматі JSON;
- шар сервісів (бізнес-логіки), де реалізується прикладна логіка роботи з сутностями: правила створення запису на прийом, перевірка необхідних полів пацієнта, обробка зміни статусу запису, узгодження цін на послуги тощо;
- шар доступу до даних (Data Access Layer), який інкапсулює роботу з PostgreSQL та реалізує операції вибірки, вставки, оновлення і видалення записів.

Цей поділ відображає концепції архітектурного стилю Model-View-Controller (MVC), поширеного у веб-додатках: контролери Express узгоджуються з аспектом Controller, структури відповідей відображаються на View, а шар роботи з базою даних – з Model. Ключова відмінність від традиційних серверних фреймворків MVC полягає в тому, що фактичне відображення інтерфейсу користувача повністю

перекладається на сторону клієнта; єдиним обов'язком сервера стає обслуговування інтерфейсу прикладного програмування (API). Ця конфігурація схиляється до дизайну “Backend for Frontend” (BFF), де сервер спеціально адаптований до вимог клієнтської програми-споживача.

Щоб мінімізувати прив'язку основної логіки програми до певної системи керування базами даних (СУБД), використовується техніка, аналогічна шаблону Repository. Для кожної основної системної сутності, такої як пацієнт, лікар, послуга або запланований прийом, створюється окремий модуль доступу до даних. Кожен модуль забезпечує узгоджений інтерфейс операцій (наприклад, отримання за ідентифікатором, масове отримання, вставка, модифікація та видалення). Рівень бізнес-сервісів взаємодіє виключно з цією абстракцією репозиторію, свідомо уникаючи вбудованих команд SQL. Отже, перемикання базової СУБД в першу чергу вимагає заміни конкретної реалізації репозиторію. Ця методологія одночасно підвищує зрозумілість коду та спрощує процедури тестування, оскільки фактичний репозиторій можна замінити симульованою або “фіктивною” версією, яка працює зі структурами даних у пам'яті, коли це необхідно для цілей тестування.

Рекомендований вибір дизайну для серверної реалізації включає використання шаблону Singleton для обробки підключення до бази даних. Враховуючи, що PostgreSQL використовує єдиний файл бази даних та обходить необхідність окремого серверного процесу, найефективнішим підходом є встановлення єдиного спільного з'єднання, яке можуть використовувати всі сховища даних. Це знижує накладні витрати на відкриття та закриття з’єднань і запобігає появі конфліктів при одночасному доступі з різних частин застосунку.

Комунікація між фронтендом і бекендом організована у вигляді REST-орієнтованого API поверх протоколу HTTP. Для кожного окремого типу сутності, такого як пацієнти, медичний персонал, надані послуги або історії хвороби, встановлюється певний набір уніфікованих точок доступу (кінцевих точок). Ці кінцеві точки сприяють канонічним діям маніпулювання даними: запит GET для отримання даних, POST для створення нового запису, PUT або PATCH для модифікації та DELETE для видалення. Передача даних назад клієнту виконується

за допомогою форматування JSON, що забезпечує безперебійний парсинг у клієнтському середовищі JavaScript. У цьому контексті архітектурний стиль REST функціонує як основний протокол зв'язку, що з'єднує різні рівні системи: клієнт React взаємодіє з сервером виключно як з масивом керованих ресурсів, переходячи між ними, тоді як сервер залишається необтяженим будь-якою логікою, пов'язаною з візуальним рендерингом інтерфейсу.

Окрему увагу слід приділити вибору архітектури на рівні даних. Структура бази даних задумана реляційно, створюючи окрему таблицю, прив'язану первинним ключем, для кожного компонента системи: Patients, Doctors, Services, Appointments тощо. Зв'язки між цими компонентами реалізуються за допомогою зовнішніх ключів. Ця методологія дотримується принципів нормалізації, тим самим скорочуючи надлишкову інформацію, що є критичним фактором при управлінні конфіденційними персональними та клінічними даними. Концептуально це відповідає шаблону “Relational Data Model”, який є поширеним стандартом для операційних інформаційних систем у контексті охорони здоров'я.

Для оптимізації потоку інформації між рівнями додатків використовуються структуровані засоби обміну даними, що нагадують шаблон DTO (об'єкт передачі даних). Серверна логіка створює корисні навантаження JSON, що містять лише ті атрибути, необхідні для клієнтського інтерфейсу, наприклад, під час представлення списку запланованих зустрічей кожен запис негайно супроводжується повними іменами пацієнта та лікаря, назвою послуги, часом та поточним статусом. Ця стратегія усуває потребу в сторонніх клієнтських запитах та зменшує обчислювальне навантаження на компоненти React. Крім того, ці DTO служать для розмежування внутрішнього представлення сутностей у базі даних (включаючи технічні метадані та стани сервісів) від даних, явно дозволених для представлення в інтерфейсі користувача.

Безпека та обмеження доступу є ключовим нефункціональним елементом цієї архітектурної конструкції. Хоча ця реалізація наразі підтримує лише роль адміністратора, фреймворк за своєю суттю сприяє простому розширенню для врахування різноманітних ролей користувачів, таких як реєстратор, лікар або

керівник закладу. Для досягнення цієї мети модель “Role-Based Access Control” (RBAC) підходить для впровадження на рівні сервера: перевірки дозволів доступу інтегровані в кожну кінцеву точку, залежно від ролі користувача, як зазначено в його токені авторизації або активному сеансі. Ці процедури перевірки можна однаково керувати в межах проміжного програмного забезпечення Express, дотримуючись найкращої практики централізації змін політик в одному місці.

Враховуючи потенційні потреби масштабування, встановлена трирівнева структура, яка чітко розділяє клієнта, сервер і базу даних, забезпечує чіткий шлях для поступових удосконалень без необхідності фундаментального перегляду системи. Якщо виникне потреба, варіанти включають: розгортання рівня представлення та рівня бізнес-логіки на окремих серверах або в окремих контейнерних середовищах, заміну PostgreSQL більш надійною системою керування реляційними базами даних, або міграція спеціалізованих компонентів у незалежні мікросервіси, які взаємодіють з основним API через внутрішній REST-інтерфейс або асинхронні черги повідомлень. Завдяки включенню шаблонів проектування як репозиторію, так і об'єкта передачі даних (DTO), більшість майбутніх модифікацій будуть обмежені їхніми відповідними архітектурними рівнями, тим самим зберігаючи основну структуру програми.

Загалом, обрана трирівнева архітектура, що включає рівень презентації на основі React, багаторівневу серверну логіку з використанням Node.js/Express та реляційну структуру даних PostgreSQL, надає кілька ключових переваг:

- чіткий поділ відповідальності між рівнями системи та можливість їх незалежного розвитку;
- структуровану організацію коду завдяки використанню перевірених шаблонів проєктування (Component-based UI, Container / Presentational, MVC, Repository, Singleton, DTO, RBAC);
- спрощення процесів розробки, тестування та супроводу;
- можливість подальшого масштабування та інтеграції з іншими системами без кардинальної зміни архітектури.

Саме ця архітектурна основа буде основою для наступних етапів проектування та детального пояснення функціональної реалізації автоматизованої системи управління медичним центром.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПРОЦЕСІВ АДМІНІСТРУВАННЯ

3.1 Моделювання бізнес-процесів медичного центру

Перш ніж заглиблюватися в специфіку структури даних та дизайну інтерфейсу користувача, рекомендується скласти формальний опис операційного процесу медичного закладу, переведений у бізнес-процеси. Саме моделювання бізнес-процесів дозволяє відокремити “як має працювати заклад” від того, “як саме це буде реалізовано у коді”. Для будь-якої системи, що автоматизує адміністративні завдання, це формальне визначення є критично важливим, оскільки основна корисність системи полягає не лише у збереженні даних, а й у сприянні виконанню рутинних щоденних функцій: реєстрації пацієнтів, планування прийомів, управління розкладами лікарів, обробки каталогів послуг та моніторингу стану офіційних записів.

Для цілей цього завдання моделювання бізнес-процесів використовує стандарти нотації, спеціально спрямовані на ілюстрацію порядку дій, залучених сторін та сутностей даних. На початковому, концептуальному рівні пропонується використовувати ланцюги процесів, керовані подіями (EPC), оскільки вони яскраво зображують взаємодію між подіями та функціями. Згодом, на детальному рівні, нотація BPMN 2.0 буде використана для відображення робочих процесів, точок прийняття рішень, одночасних шляхів та взаємодії між зацікавленими сторонами. Ця дворівнева стратегія гарантує, що, з одного боку, керівництво підтримує високорівневе, легкозрозуміле уявлення про процеси, одночасно досягаючи необхідної точності для плавного переходу до фази технічної специфікації.

Базовим “кореневим” процесом, який розкладається на підпроцеси, є “Адміністрування діяльності медичного центру”. У рамках цієї основної процедури визначено окремі підпроцедури, які представляють собою самостійні набори завдань, що підтримуються власним програмним забезпеченням:

- реєстрація нового пацієнта та актуалізація його даних;
- запис пацієнта на прийом до лікаря;
- керування розкладом та статусами записів;
- ведення довідників лікарів і медичних послуг;
- авторизація адміністратора та контроль доступу до функцій системи.

Процес реєстрації нового пацієнта ініціюється тригерною подією: “Пацієнт звернувся до медичного центру вперше”. На рівні загального бізнес-потоку адміністратор виконує низку визначених кроків: перевірка того, чи особа вже внесена до каталогу даних, введення особистих даних (повне ім'я, номер телефону, дата народження та, де це доречно, паспортні дані та місце проживання), документування будь-яких відповідних спостережень та, якщо це можливо, присвоєння статусу преміум-клієнта (VIP). Результатом цієї послідовності є створення нового запису в реєстрі “Пацієнти”, який згодом слугує основою для наступних дій: планування прийомів, документування хронології візитів та підтвердження контактних даних. У конвенції моделювання EPC ця процедура зображується як ланцюг дій - “Перевірити наявність пацієнта”, “Зареєструвати пацієнта”, “Зберегти дані пацієнта” - пов'язаних проміжними етапами, такими як “Пацієнт не знайдено в базі даних”, “ Дані введено коректно” та подібні події.

Наступний вирішальний операційний процес зосереджений навколо “Забезпечення запису пацієнтів”. У щоденній роботі медичного закладу ця послідовність включає перевірку доступності лікаря на запитувану дату та проміжок часу, підтвердження необхідної консультації або процедури, реєстрацію запису в календарі, а потім повідомлення підтверджених даних пацієнту. У блок-схемі процесу помітні кроки охоплюють: ідентифікацію пацієнта (або пошук існуючого запису, або створення нового), вибір лікаря на основі зазначеної галузі спеціалізації, вибір доступної дати та проміжку часу шляхом посилання на вільні місця в розкладі та, нарешті, визначення відповідної медичної послуги. На системному рівні цього робочого процесу необхідно перевірити точність усієї вхідної інформації (підтвердження існування вибраного пацієнта, лікаря та послуги, а також правильності формату дати та часу) та запобігти будь-яким

конфліктам у розкладі. Завершення цього циклу позначається подією “Запис на прийом створено”, після чого запис стає видимим в інтерфейсі планування адміністративної команди. Деталізований бізнес-процес забезпечення запису пацієнта на прийом до лікаря подано на рис. 3.1.

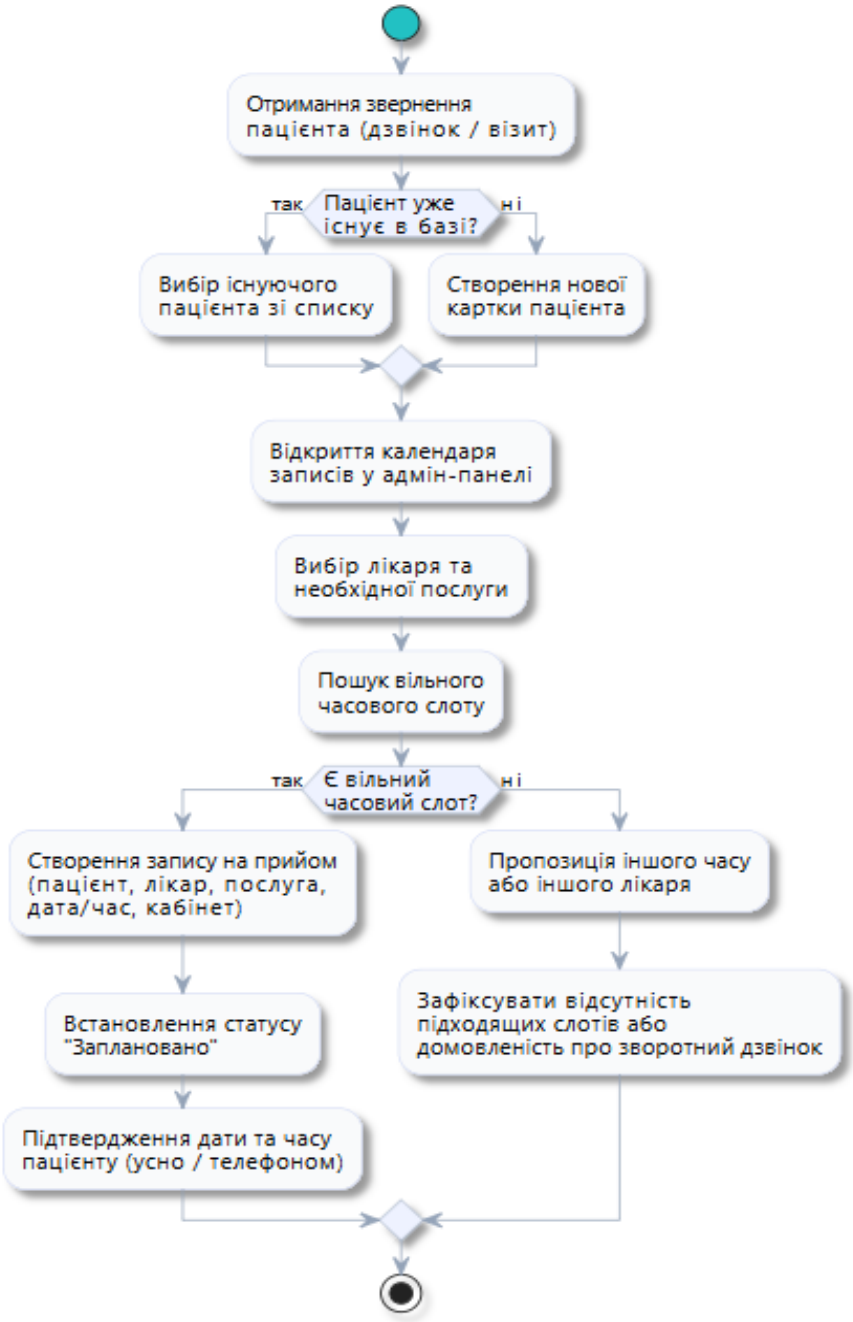


Рис. 3.1 Бізнес-процес запису пацієнта до лікаря.

Окремої уваги потребує бізнес-процес “Керування статусами записів на прийом”. На практиці, зустріч є динамічною, а не фіксованою, її статус змінюється,

щоб відобразити завершення, скасування, перенесення або залишення за початковим планом. У рамках бізнес-моделі ці зміни відображаються за допомогою конкретних подій та гілок рішень. Процес починається з ініціювання запису з позначкою “Заплановано”. Після фактичної консультації адміністратор або інший призначений співробітник оновлює статус на “Завершено”. Якщо пацієнт відмовляється від зустрічі або не приходить, запис переходить у статус “Скасовано” з можливістю задокументувати основну причину у супровідному текстовому полі.

У разі перенесення запису на прийом фактично створюється новий запис, що відрізняється чіткою датою та часовим інтервалом. Попередній запис, відповідно до встановлених правил організації, згодом матиме позначку, таку як “Скасовано” або “Перенесено”. Цей конкретний метод структурування даних полегшує представлення базового процесу, який потім перетворюється на зміни статусу запису через інтерфейс користувача, представлений у модальному календарному вікні.

Процес “Ведення довідника лікарів” описує операції, пов’язані з додаванням нових лікарів, оновленням їхніх контактних даних, спеціалізації та службової інформації (кабінет, примітки) та, за потреби, видаленням записів у разі припинення співпраці. З точки зору робочого процесу, визначення прав доступу має першочергове значення: базова структура передбачає, що адміністративний персонал виконує всі ці зміни. Тим не менш, майбутнє вдосконалення може включати більш детальний розподіл обов’язків (наприклад, надання керівникам відділів повноважень санкціонувати зміни в реєстрі персоналу). Точність довідника лікарів є абсолютно необхідною для належного виконання функцій запису та планування прийому, оскільки список доступних фахівців та можливість фільтрації записів у календарі безпосередньо залежать від цієї інформації.

Аналогічним чином моделюється бізнес-процес “Керування довідником медичних послуг”. У цій процедурі призначений адміністратор відповідає за створення нових записів, зміну існуючих або деактивацію записів про послуги для медичного закладу. Основні деталі для кожної послуги включають її назву,

короткий опис, стандартну плату та поточний операційний стан. З операційної точки зору, цей механізм гарантує, що дані, представлені в опублікованих цінових таблицях, точно відповідають послугам, які фактично доступні під час планування візиту пацієнта. Враховуючи, що оновлення каталогу послуг повинні негайно впливати на те, як плануються зустрічі та як розраховуються витрати на послуги, модель процесу повинна чітко окреслювати ці зміни та, де це доречно, піддавати їх встановленим протоколам підписання.

До групи підтримуючих процесів належить також “Авторизація адміністратора в системі”. Хоча ця процедура структурно менш складна, її бізнес-значення полягає в її критичній ролі у забезпеченні прав доступу. Схематичне зображення цього бізнес-процесу окреслює кроки, пов’язані з наданням необхідних облікових даних, перевіркою їх точності, наданням доступу основним компонентам системи, а потім переходом до основного інтерфейсу (вигляд планування зустрічей). Якщо автентифікація не вдається, система генерує сповіщення “Помилка входу” та запити на “Повторне введення облікових даних”, що підвищує прозорість сценаріїв взаємодії, які розробникам потрібно вбудувати в інтерфейс користувача.

Комплексне зображення розглянутих бізнес-процесів можна отримати за допомогою єдиної, уніфікованої діаграми. Це візуальне представлення служить для ілюстрації взаємозв'язків між цими процесами та відповідними інформаційними активами. У цій високорівневій схемі процес “Адміністрування медичного центру” розбитий на кілька взаємозалежних підпроцесів: “Реєстрація пацієнта”, “Запис на прийом”, “Керування статусами прийомів”, “Ведення довідників лікарів”, “Ведення довідника послуг”, “Авторизація адміністратора” (рис. 3.2). Кожен з цих підпроцесів взаємодіє з певними елементами даних (Пацієнт, Лікар, Послуга, Запис на прийом), які пізніше будуть відображені в майбутній моделі даних та макеті таблиць бази даних.

Рис. 3.2 Структура основних бізнес-процесів адміністрування медичного центру.

Ця методологія побудови бізнес-процесів встановлює чіткий концептуальний зв'язок між аналітичними висновками, представленими в перших розділах, та конкретною розробкою системи, детально описаною в заключному розділі. Описані моделі дозволяють:

- перевірити повноту функціональних вимог до автоматизованої системи.
- виявити потенційні вузькі місця (наприклад, дублювання операцій або надмірну залежність від ручних дій адміністратора).
- визначити точки інтеграції між окремими підпроцесами та модулями системи.
- сформувати основу для подальшого проєктування структури бази даних та інтерфейсу адміністративної панелі.

У подальшому на основі сформованих бізнес-процесів буде виконано деталізацію інформаційної моделі медичного центру, побудовано відповідні діаграми даних та розроблено макети інтерфейсів, що безпосередньо реалізують описані сценарії роботи адміністратора в розробленій автоматизованій системі.

3.2 Проєктування інформаційної структури (моделі даних, ER-діаграма)

Проектування інформаційної архітектури природно випливає з попереднього етапу складання схеми бізнес-процесів. Детально описані функціональні робочі

процеси, по суті встановлюють вимоги щодо зберігання системних даних: яка саме інформація має зберігатися, як ці фрагменти даних взаємопов'язані та які правила цілісності мають бути застосовані. Цей розділ має на меті формально виділити основні компоненти домену медичного закладу, вказати їхні характеристики та взаємозв'язки, а також побудувати концептуальну діаграму «сутність-зв'язок» (ER), яка буде основою фізичного розташування бази даних PostgreSQL.

Першим кроком є ідентифікація основних сутностей, що безпосередньо випливають з бізнес-процесів “Реєстрація пацієнта”, “Запис на прийом”, “Керування статусами записів”, “Ведення довідників лікарів та медичних послуг” і “Авторизація адміністратора”. У розробленій інформаційній структурі фундаментальними визначеними сутностями є Пацієнт, Лікар, Пропозиція послуг, Запланований прийом, а також допоміжна сутність AdminUser, яка обробляє автентифікацію та керування дозволами. Кожен інший елемент даних або служить для уточнення цих сутностей за допомогою властивостей, або являє собою реляційні зв'язки (наприклад, первинні та зовнішні ключі, перелічені значення для станів тощо).

Сутність “Пацієнт” є центральною з погляду збереження персональних медико-адміністративних даних. Ключові характеристики, визначені для цієї сутності, включають: окремий ідентифікатор пацієнта, повне ім'я особи, контактний номер телефону, дату народження, біологічну стать, адресу електронної пошти, адресу фактичного проживання, дані паспорта або інші документи, що ідентифікують особу, додаткове поле для текстових коментарів та позначку, що вказує на статус VIP. Крім того, спеціальний атрибут "позначки часу створення запису" фіксує точний час першої реєстрації запису пацієнта в системі, що дозволяє використовувати його для цілей аудиту або подальшої аналітичної роботи. Ця конфігурація допомагає адміністраторам у їхніх щоденних завданнях, таких як пошук осіб за іменем чи телефоном або швидке оновлення контактної інформації, а також створює умови для подальшого додавання функцій, таких як інтеграція автоматичних SMS-нагадувань або можливостей цифрової попередньої реєстрації.

Сутність “Лікар” служить для характеристики медичних працівників, яким призначені пацієнти. Обов'язковими атрибутами, що надаються для цього типу запису, є єдиний ідентифікаційний номер, повне ім'я фахівця, його медична спеціалізація, номер телефону для прямого контакту, адреса електронної пошти, номер призначеного кабінету для консультацій та розділ для додаткових описів. Також реєструється дата створення запису лікаря, що відображається в картці пацієнта. Така структура забезпечує однозначне пов'язування кожної консультації з призначеним їй лікарем та забезпечує плавну фільтрацію інтерфейсу на основі спеціалізації під час робочого процесу запису на прийом.

Сутність “Медична послуга” відображає номенклатуру процедур і консультацій, які надає медичний центр. Його основні точки даних включають унікальний ідентифікатор послуги, її офіційну назву, стислий опис, стандартну плату та логічний перемикач, що вказує на її поточний операційний статус («активний» або «неактивний»). Підтримка цього розмежування між активними та неактивними записами має вирішальне значення для збереження повного журналу змін у каталозі без вдавання до постійного видалення даних, що є життєво важливою функцією для точного фінансового аудиту та аналізу розвитку портфеля послуг у різні часові періоди.

Ключовою інтегруючою сутністю є “Запис на прийом”. Вона пов’язує між собою пацієнта, лікаря та послугу, відображаючи конкретний факт планування візиту. Для цієї сутності задаються такі атрибути: ідентифікатор запису, посилання на пацієнта (patient_id), посилання на лікаря (doctor_id), посилання на обрану послугу (service_id), дата й час прийому, статус (наприклад, “Заплановано”, “Завершено”, “Скасовано”), текстова примітка (коментар щодо причин скасування, уточнення по процедурі тощо), а також дата створення запису. Така конфігурація дозволяє відобразити життєвий цикл кожного прийому: від первинного внесення в календар до завершення консультації або скасування. Статус, збережений безпосередньо в сутності “Запис на прийом”, забезпечує швидке формування вибірок для інтерфейсу календаря та звітів, не вимагаючи складних обчислень під час виконання запитів.

Сутність “Користувач адміністративної панелі” має оптимізований набір характеристик: унікальний ідентифікатор користувача, ім'я доступу до облікового запису, представлення зашифрованого пароля, призначений рівень авторизації (наприклад, “адміністратор” або “реєстратор”) та, за бажанням, посилання на конкретного співробітника або організаційний підрозділ. Хоча в поточній конфігурації використовується єдине позначення адміністратора, підтримка окремої сутності користувача гарантує, що майбутні розширення системи можуть відбуватися безперебійно без необхідності змінювати основні структури таблиць транзакцій пацієнта, лікаря або консультації.

З цього вибраного набору сутностей потім визначаються шаблони зв'язку між ними. Існує зв'язок “один до багатьох”, що з'єднує сутності пацієнта та призначення: один пацієнт пов'язаний з можливістю численних узгоджених або завершених сеансів, проте кожен окремий запис сеансу пов'язаний виключно з одним пацієнтом. Аналогічний зв'язок “один до багатьох” встановлюється між лікарем та записом, оскільки один лікар керує кількома записами, тоді як кожен запис належить одному призначеному лікарю. Крім того, зв'язок “один до багатьох” пов'язує медичну послугу та медичний запис: дана послуга може бути відображена в кількох записах, але будь-який окремий запис вказує на конкретну послугу, заплановану для виконання під час цієї консультації. Ця структурна конфігурація невід'ємно реалізується шляхом включення зовнішніх ключів у таблицю Appointments, які служать вказівниками на первинні ідентифікатори ключів таблиць Patients, Doctors і Services.

З погляду нормалізації даних інформаційна модель орієнтується на досягнення принаймні третьої нормальної форми. Усі атрибути кожної сутності функціонально залежать від її первинного ключа і не містять транзитивних залежностей. Наприклад, адреса пацієнта не дублюється в таблиці записів, а зберігається лише в таблиці Patients, а назва послуги та її вартість не дублюються в кожному записі на прийом, оскільки вони належать сутності Service. Такий підхід зменшує обсяг зайвих даних, знижує ризик виникнення суперечливих значень (коли одна й та сама

інформація змінюється в одному місці, але залишається незмінною в іншому) та спрощує підтримку цілісності при оновленні.

Важливою складовою проєктування інформаційної структури є визначення обмежень цілісності. Для кожної сутності задаються первинні ключі, що гарантують унікальність записів, а для зовнішніх ключів у таблиці Appointments визначаються відповідні обмеження посилальної цілісності. У практичному вираженні це означає, що система не дозволяє створити запис на прийом для пацієнта, якого немає в реєстрі, або послатися на лікаря, запис якого був видалений чи деактивований. Частина обмежень реалізується на рівні СУБД (NOT NULL, FOREIGN KEY, CHECK), інша частина – у прикладній логіці Node.js/Express, де додатково перевіряються коректність дат і часових інтервалів, статусів та інших ділових правил.

Виняткова увага приділяється характеристикам даних, що регулюють конфіденційність та захист персональних даних. Враховуючи, що дані, що стосуються як пацієнтів, так і лікарів, містять персональні дані, практичне впровадження обмежує доступ до баз даних пацієнтів, лікарів та записів на прийом виключно через рівень API сервера, що підкріплений встановленими протоколами автентифікації. У структурі даних ця реальність відображається шляхом фізичного відокремлення реєстру облікових записів AdminUser від таблиць, що містять вміст медичних записів; крім того, будь-які елементи даних, які вважаються несуттєвими для адміністративних функцій, не копіюються та не представляються знову в інтерфейсі користувача.

Узагальнена структура сутностей та зв’язків між ними відображається на ER-діаграмі інформаційної моделі медичного центру (рис. 3.1). На діаграмі прямокутниками позначені сутності Patients, Doctors, Services, Appointments та AdminUser, у кожній із яких перераховано головні атрибути, що зберігаються у відповідній таблиці. Лінії зі знаками “1” і “∞” ілюструють зв’язки “один-до-багатьох” між пацієнтами та їх записами, лікарями та записами, послугами та записами. Окремі поля, такі як статус запису, ознака VIP-пацієнта або активність послуги, були винесені у вигляді атрибутів-перемикачів, що дозволяє ефективно

реалізувати фільтрацію та сортування в адміністративному інтерфейсі без ускладнення схеми додатковими таблицями-довідниками.

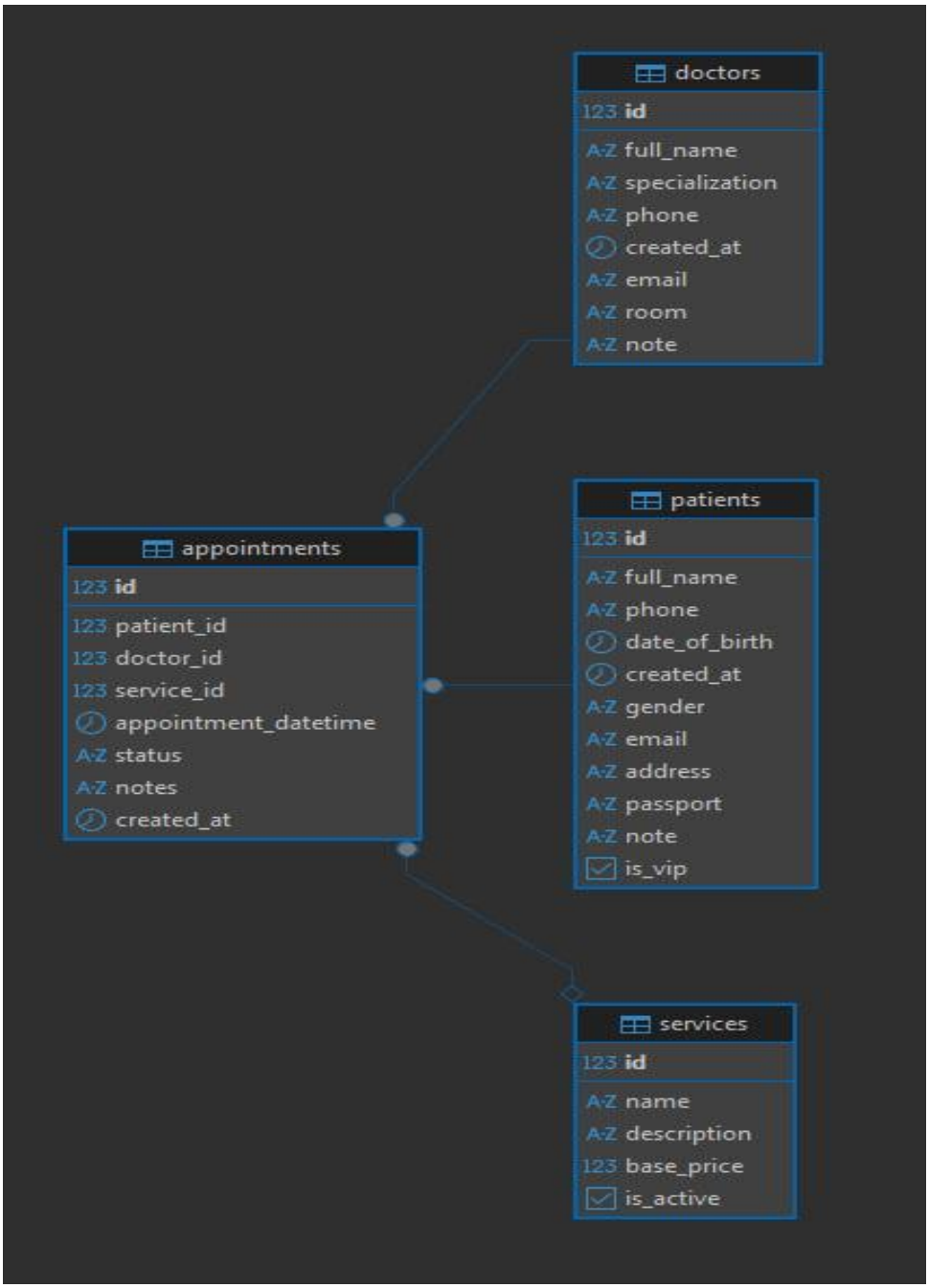


Рис. 3.3 ER-діаграма інформаційної системи моделі автоматизованої системи медичного центру.

Такий підхід до проєктування інформаційної структури забезпечує баланс між простотою фізичної реалізації в PostgreSQL та достатньою гнучкістю для подальшого розвитку системи. За потреби модель допускає природне розширення:

можна додати сутності “Візит” з деталізацією клінічних записів, “Платіж” для відображення фінансових операцій чи “Журнал дій адміністратора” для аудиту, не порушуючи вже сформовані зв’язки. При цьому основне ядро – зв’язка пацієнт–лікар–послуга–запис – залишається незмінним і продовжує підтримувати ключові адміністративні сценарії, описані в даному розділі.

Таким чином, у підрозділі виконано формалізацію інформаційної моделі автоматизованої системи медичного центру, визначено основні сутності, їх атрибути та взаємозв’язки, сформульовано вимоги до цілісності та нормалізації даних. Побудована ER-діаграма слугує базою для переходу до наступних етапів – конкретизації фізичної схеми бази даних PostgreSQL та опису способів її використання під час реалізації функціональних модулів адміністративної панелі.

3.3 Проєктування інтерфейсу адміністративної панелі медичного центру

Проєктування інтерфейсу користувача адміністративної панелі є ключовим етапом, оскільки саме через нього адміністратор взаємодіє з усіма функціональними можливостями системи. Саме тут дані, отримані в результаті аналізу бізнес-процесів та загальної інформаційної архітектури, перетворюються на відчутні екранні елементи: візуальні макети, інтерактивні елементи керування та визначені шляхи користувача. Головною метою цього дизайну інтерфейсу є сприяння швидкому виконанню рутинних завдань, таких як пошук людей, планування записів або оновлення списків лікарів та доступних процедур, що вимагає найменшої кількості кроків та мінімізації ймовірності помилок.

Вихідною точкою слугує концепція односторінкового веб-додатку (SPA), реалізованого засобами React. Це об'єднує майже всі основні можливості в один цілісний адміністративний робочий простір. У цьому просторі динамічно змінюється лише контент, що відображається в головній центральній області, тоді як загальні навігаційні елементи залишаються статичними. Цей архітектурний вибір зменшує розумові зусилля, необхідні від користувача: адміністратор

постійно розуміє, де він знаходиться на карті системи, і ніколи не втрачає з поля зору свою поточну область діяльності. Відповідно до цієї філософії, інтерфейс структуровано навколо чотирьох ключових операційних областей: “Записи на прийом”, “Пацієнти”, “Лікарі” та “Послуги”. Доступ до цих областей забезпечується через постійне бічне меню навігації, розташоване вздовж лівого краю кожного екрана, видимого в панелі адміністратора.

Бічна панель починається з емблеми системи медичного центру поруч із коротким заголовком “MedCenter. Адмін-панель”, що створює візуальний брендинг та чітко позначає адміністративну функцію. Далі йдуть навігаційні посилання, розташовані у вигляді вертикального масиву. Поточна обрана область візуально підкреслюється чітким відтінком фону та м’якими заокругленими краями, що робить поточний фокус користувача одразу помітним. У нижній частині панелі розміщено кнопку “Вихід”, яка забезпечує завершення сеансу адміністратора. Отже, вся основна глобальна навігація знаходиться в межах цього єдиного постійного компонента, звільняючи решту екранного простору виключно для представлення контенту, що стосується активного розділу. Загальний вигляд головного вікна адміністративної панелі медичного центра наведено на рис. 3.4.

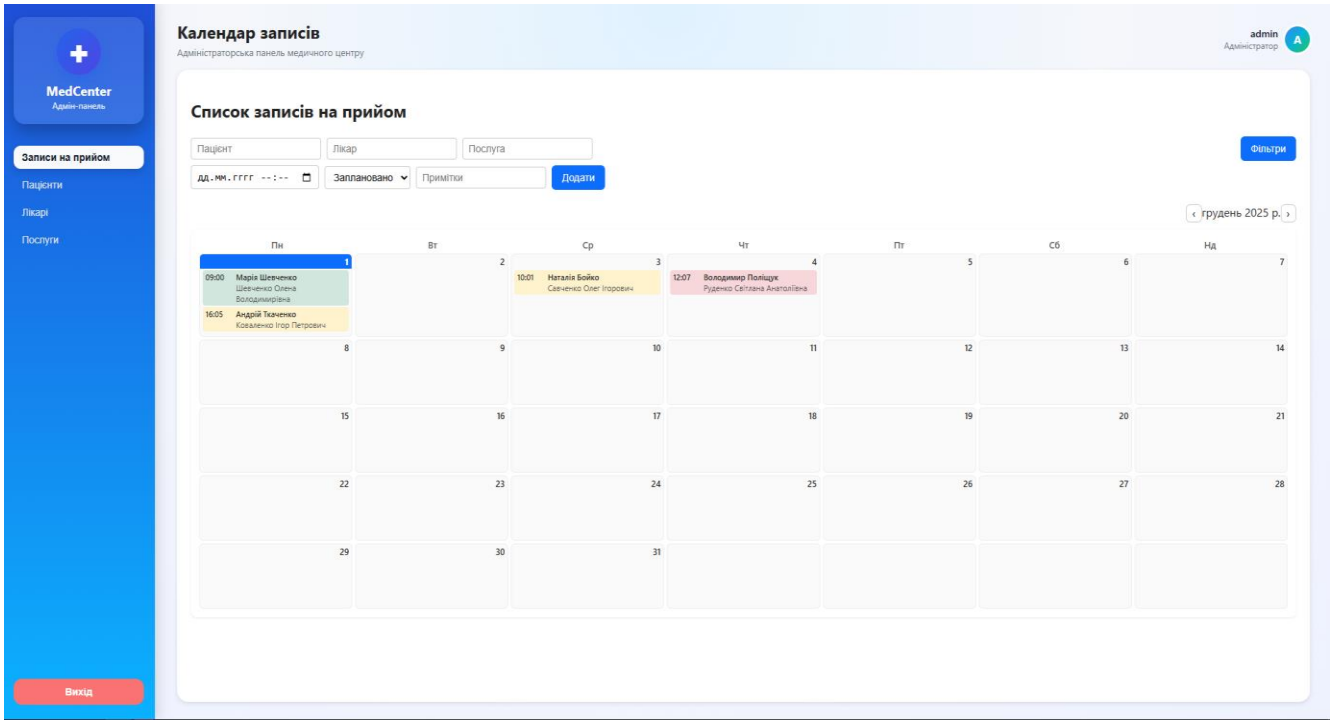


Рис. 3.4 Загальний інтерфейс адміністративної панелі медичного центру.

Основна операційна область інтерфейсу дотримується організаційної структури “панель-контент”. Заголовок містить назву конкретного модуля, який наразі відображається, наприклад, “Список пацієнтів”, “Доступні медичні послуги” або “Графік прийому”. Під цим заголовком знаходиться панель керування та уточнення, що містить поля введення для пошуку, випадаючі меню для вибору та кнопку дії для введення нових записів. Така структура відображає логічний робочий процес адміністратора: початкова орієнтація в розділі, подальше маніпулювання даними за допомогою механізмів фільтрації та, нарешті, ініціювання завдань створення або модифікації. Основна частина цього простору відведена для табличного відображення записів, яке ефективно відображає дані, що зберігаються у відповідній таблиці бази даних.

Структура представлення записів пацієнтів, лікарів, послуг та записів відповідає послідовній схемі: у верхній лівій частині кожної таблиці міститься послідовна нумерація. Далі йдуть основні характеристики запису (наприклад, повне ім'я, контактний номер та дата народження для пацієнтів; назва послуги та базова вартість послуг; або запланований час, лікар та клієнт для призначень, серед іншого). У крайньому правому куті кожного рядка міститься розділ “Дії” з кнопками для зміни та видалення. Використання цього стандартизованого макета таблиці в різних розділах спрощує навчання для користувачів та покращує очікувану функціональність інтерфейсу системи.

Значні зусилля були вкладені в розробку інтерфейсу для перегляду “Записи на прийом ”, оскільки це являє собою основний операційний центр адміністратора. Ця область містить кілька механізмів для звуження відображеної інформації: функцію пошуку на основі повного імені пацієнта або лікаря; вибір на основі календаря для визначення діапазону дат; та випадаючий список для фільтрації записів за їх поточним статусом. Цей набір елементів керування фільтрацією розташований безпосередньо перед основною таблицею зустрічей, що дозволяє швидко комбінувати параметри вибору для миттєвого виділення потрібного сегмента розкладу. Елемент керування для ініціювання нового запису “Додати” активує спливаюче модальне вікно, яке відображає всі необхідні поля для запису в

організованому макеті: включаючи поля для клієнта, лікаря, наданої послуги, дати, часу, номера кабінету та початкового статусу (Рис. 3.6). Таке розташування діалогового вікна мінімізує необхідність прокручування з боку в бік, розташовуючи поля у двох вертикальних стовпцях, логічно групуючи пов’язані точки даних.

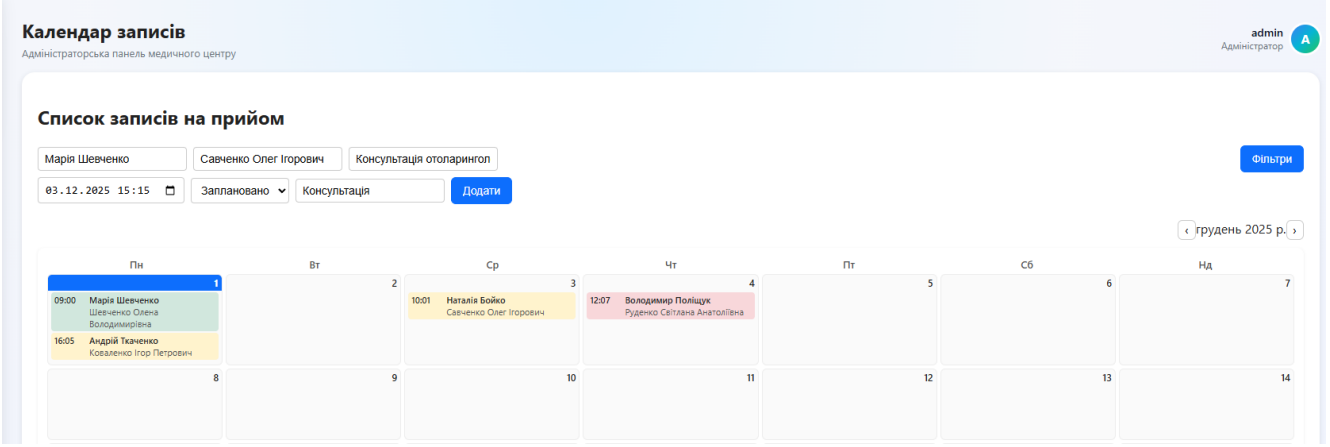


Рис. 3.5 Інтерфейс модуля “Записи на прийом” з фільтрацією та формою створення запису.

Фрагмент інтерфейсу модуля “Записи на прийом” із засобами фільтрації та формою створення нового запису показано на рисунку 3.5.

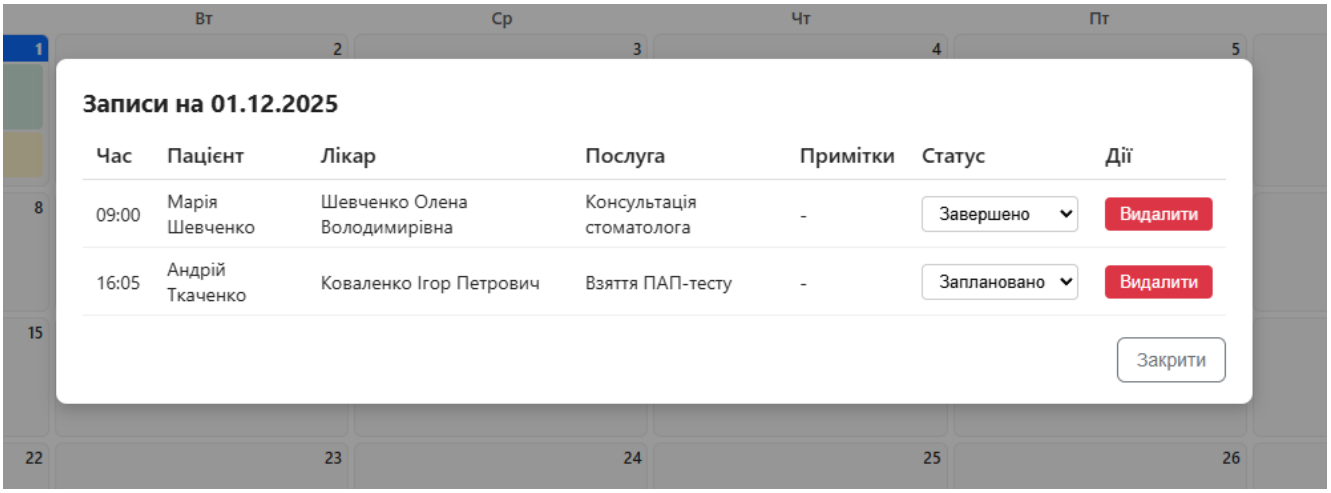


Рис. 3.6 Модальне вікно “Запису на прийом” з відображенням функціоналу.

Для операцій створення та редагування пацієнтів і лікарів використано єдиний шаблон форми. Основні поля даних, що включають повне ім'я, контактну інформацію, дату народження та професійну спеціалізацію, позначені як обов'язкові поля та мають чіткий візуальний акцент. Додаткові деталі, такі як внутрішні примітки, інформація про фізичну адресу або позначення VIP, розташовані біля завершення форми та чітко позначені як необов'язкові. Перевірки цілісності даних повністю здійснюються на стороні клієнта: неправильно відформатовані номери телефонів, незаповнені обов'язкові поля або недійсні записи дат супроводжуються підказками та підсвічуванням поля червоною рамкою. Такий механізм зменшує кількість помилок ще до відправлення даних на сервер і, відповідно, знижує навантаження на бекенд.

Дизайн інтерфейсу користувача для адміністративного розділу спочатку був прототипований у середовищі Figma. Це сприяло швидкій ітерації та тестуванню різних можливостей макета, не обмежених вимогами до кодування. Цей етап проектування був вирішальним для встановлення основної кольорової палітри та вибору шрифту. Фон основної робочої області має блідий відтінок, що покращує читабельність тексту та створює сильний візуальний контраст із вертикальною бічною панеллю навігації, стилізованою за використання ефекту синього градієнта. Таблиці й форми розміщені в окремих “картах” із легким тінюванням та закругленими кутами, що візуально відділяє їх від фону та допомагає користувачу концентруватися на поточній області роботи. Таке графічне розділення допомагає виділити інтерактивні елементи від загального фону, дозволяючи користувачам ефективніше зосереджуватися на поточному завданні. Для заголовків використовується збільшений кегль та напівжирний накреслення стандартного шрифту інтерфейсу (на кшталт Segoe UI або системного шрифту операційної системи), що зберігає сучасний вигляд інтерфейсу та забезпечує сумісність із браузерами.

Щодо того, як користувачі взаємодіють із платформою, інтегровано кілька додаткових компонентів для покращення загального користувацького досвіду. Для функціональних елементів керування застосовується окрема кольорова схема:

операції додавання або зберігання даних позначені яскравим синім відтінком, зміни позначені більш приглушеним тоном, а видалення сигналізуються червоним, що вимагає додаткового кроку для підтвердження користувача. Щоб запобігти випадковому вибору, рядки в таблицях даних отримують візуальний акцент шляхом підсвічування, коли на них наведено курсор миші, а при натисканні відбувається чітке перетворення фону, що сигналізує про те, що запланована функція була запущена. Найважливіше те, що сценарії, що передбачають потенційне знищення існуючих даних, такі як видалення запису або вихід з форми введення без фіксації останніх змін, спонукають систему видавати повідомлення підтвердження, надаючи користувачеві можливість перервати процес.

Врахування різних розмірів дисплея було центральним у процесі проектування. Хоча основним середовищем для адміністративних завдань є настільний комп'ютер або ноутбук, макет був розроблений для бездоганного відображення на дисплеях шириною 1280 пікселів або більше. Сітки даних оснащені можливістю бічного прокручування, якщо вони містять надлишок стовпців, що гарантує, що найважливіші поля відображаються одразу після перегляду. На менших екранах деякі стовпці даних автоматично зменшуються, а другорядні деталі акуратно ховаються в інформаційних спливаючих вікнах або доступні через спеціальне вікно, що відображає повну інформацію про запис.

Інтерфейс адміністративної панелі тісно пов’язаний із раніше сформованими бізнес-процесами. Наприклад, описаний раніше процес “Запис на прийом” безпосередньо проектується у послідовність дій у розділі “Записи на прийом”: вибір пацієнта зі списку, вибір лікаря та послуги, пошук вільного часовго слота та підтвердження запису. Зміна статусів запису, описана в моделі бізнес-процесів, реалізується через набір доступних значень у випадаючому списку статусу, а сценарії перенесення або скасування візиту - через відповідні дії з додатковим полем для введення примітки. Таким чином, кожний діагностований бізнес-процес має пряме відображення у вигляді конкретного екрану, форми чи елемента керування.

Крім того, зовнішній вигляд інтерфейсу відповідає вимогам щодо захисту даних та конфіденційності. Будь-який елемент, що містить конфіденційні персональні дані пацієнтів або медичного персоналу, видимий виключно в обмеженій адміністративній зоні, доступній лише після підтвердженого входу. Верхня панель навігації містить ідентифікаційні дані та позначення поточного користувача, що дозволяє керівному персоналу відстежувати облікові дані, під якими виконуються дії.

У майбутньому цей підхід забезпечує можливість масштабування моделі для кількох ролей та розгортання сегментованого доступу до окремих областей панелі, зберігаючи при цьому базову структуру інтерфейсу користувача.

Таким чином, у результаті виконаного етапу проєктування сформовано цілісну концепцію інтерфейсу адміністративної панелі медичного центру. Ця уніфікована концепція об'єднує єдину схему навігації, стандартизовані представлення таблиць даних та форм введення, інтуїтивно зрозумілі методи фільтрації та управління записами, а також включає необхідні міркування щодо ефективності робочого процесу адміністратора, надійності системи та надійної безпеки. Отримані макети екранів згодом слугуватимуть основою для розробки компонентів React та встановлення зв'язків з API бекенду.

3.4 Реалізація функціональних модулів системи

Реалізація автоматизованої системи процесів адміністрування медичного центру базується на попередньо обґрунтованій трирівневій архітектурі: клієнтський рівень (React-інтерфейс адміністративної панелі), серверний рівень (Node.js/Express) та рівень даних (PostgreSQL). На цьому етапі теоретично описані бізнес-процеси та інформаційна модель перетворюються на конкретні програмні модулі, кінцеві точки API, компоненти інтерфейсу та SQL-структури, які спільно реалізують повний цикл роботи адміністратора з пацієнтами, лікарями, послугами та записами на прийом.

Уся логіка взаємодії з даними на клієнтському боці зосереджена в окремих розділах адміністративної панелі: “Записи на прийом”, “Пацієнти”, “Лікарі” та “Послуги”. Кожен із цих розділів представлений окремим компонентом верхнього рівня, який відповідає за отримання даних через REST-API (рис. 3.7), обробку подій користувача та передачу даних у табличні або формові компоненти. Нижче розглянуто реалізацію основних функціональних модулів і спосіб їхньої інтеграції між собою.

```
69  // Функція завантаження даних з API
70  const fetchData = async (tab) => {
71    if (!isLoggedIn) return;
72
73    setLoading(true);
74    setError('');
75
76    try {
77      if (tab === 'patients') {
78        const res = await fetch(`${API_BASE_URL}/api/patients`);
79        const data = await res.json();
80        setPatients(data);
81      } else if (tab === 'doctors') {
82        const res = await fetch(`${API_BASE_URL}/api/doctors`);
83        const data = await res.json();
84        setDoctors(data);
85      } else if (tab === 'appointments') {
86        const [appsRes, patsRes, docsRes, servRes] = await Promise.all([
87          fetch(`${API_BASE_URL}/api/appointments`),
88          fetch(`${API_BASE_URL}/api/patients`),
89          fetch(`${API_BASE_URL}/api/doctors`),
90          fetch(`${API_BASE_URL}/api/services`),
91        ]);
92
93        const appsData = await appsRes.json();
94        const patsData = await patsRes.json();
95        const docsData = await docsRes.json();
96        const servData = await servRes.json();
97
98        setAppointments(appsData);
99        setPatients(patsData);
```

Рис. 3.7 Фрагмент коду головного компонента App із налаштуванням підключення до API та завантаженням данних довідників.

Першим кроком реалізації стало налаштування серверного середовища на базі Node.js з використанням фреймворку Express. Структура проєкту організована за шаровим принципом: в окремих каталогах зберігаються модулі маршрутизації, контролери, сервіси бізнес-логіки та репозиторії доступу до даних. Для підключення до PostgreSQL використано офіційний драйвер, а ініціалізація з'єднання винесена у спільний модуль, що реалізує підхід, аналогічний шаблону Singleton: створюється один пул підключень, який повторно використовується всіма репозиторіями. Параметри підключення (хост, порт, назва бази, логін та пароль) зберігаються у конфігураційному файлі або змінних оточення, що спрощує перенесення застосунку між середовищами розробки та розгортання (рис. 3.8).

```
// server.js
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const pool = require('./db'); // підключення до PostgreSQL

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

const ADMIN_USER = {
  username: 'admin',
  password: 'admin123',
};

// --- Базові маршрути ---

// Простий тестовий маршрут
app.get('/', (req, res) => {
  res.send('API медичного центру працює 🚑');
});

// Перевірка "здоров'я" API
app.get('/api/health', (req, res) => {
  res.json({ status: 'ok', message: 'Адмін-система жива' });
});
```

Рис. 3.8 Ініціалізація серверної частини Node.js/Express та підключення до бази данних PostgreSQL.

Реалізація функціоналу авторизації адміністратора ґрунтується на модулі, що обробляє вхід до системи. На стороні клієнта для цього передбачено окремий екран входу, де користувач вводить ім’я облікового запису та пароль. Цей екран реалізовано як компонент AdminLogin (рис. 3.9), який відправляє запит до кінцевої точки сервера (наприклад, POST /api/auth/login). На сервері відповідний контролер отримує облікові дані, звертається до репозиторію користувачів (таблиці, аналогічної admin_users у PostgreSQL), перевіряє наявність користувача та порівнює введений пароль із зашифрованим значенням у базі. У разі успіху формується відповідь з маркером авторизації або сеансовим ідентифікатором, який згодом використовується для захисту адміністративних маршрутів (рис. 3.10). Хоча поточна реалізація підтримує одну роль адміністратора, структура коду одразу враховує можливість розширення моделі доступу до багатьох ролей.

```
264  function LoginForm({ onLogin, error }) {
265    const [username, setUsername] = useState('');
266    const [password, setPassword] = useState('');
267    const [loading, setLoading] = useState(false);
268
269    const handleSubmit = async (e) => {
270      e.preventDefault();
271      setLoading(true);
272      await onLogin(username, password);
273      setLoading(false);
274    };
275
276    return (
277      <div style={styles.loginContainer}>
278        <div style={styles.loginCard}>
279          {/* Лого + назва */}
280          <div style={styles.loginBrand}>
281            <div style={styles.loginAvatar}>✚</div>
282            <div>
283              <div style={styles.loginBrandName}>MedCenter</div>
284              <div style={styles.loginBrandSubtitle}>Адміністративна панель</div>
285            </div>
286          </div>
287
288          <h2 style={styles.loginTitle}>Вхід</h2>
289
290          {error && <p style={styles.loginError}>{error}</p>}
291
292          <form onSubmit={handleSubmit} style={styles.loginForm}>
293            <div style={styles.loginFieldRow}>
294              <label
295                htmlFor="login-username"
296                style={styles.loginLabelInline}
```

Рис. 3.9 Реалізація екрана входу адміністратора в інтерфейсі React та перевірка стану авторизації на кліієнтській частині.

```
444  app.post('/api/login', (req, res) => {
445    const { username, password } = req.body;
446
447    if (username === ADMIN_USER.username && password === ADMIN_USER.password) {
448      return res.json({ success: true });
449    }
450
451    return res.status(401).json({ error: 'Невірний логін або пароль' });
452  });
453
```

Рис. 3.10 Сршрут авторизації адміністратора на сервері та перевірка облікових даних.

Функціональний модуль керування пацієнтами реалізує операції додавання, редагування, перегляду та видалення записів із таблиці patients. На клієнтському боці для цього створено сторінку, що відображає таблицю записів пацієнтів разом із панеллю інструментів для пошуку та створення нових записів. При завантаженні сторінки компонент верхнього рівня виконує виклик до кінцевої точки GET /api/patients, отримує масив об’єктів і зберігає його у внутрішньому стані (useState) (рис. 3.11).

```
366  // Додавання пацієнта
367  const handleAdd = async () => {···
404  };
405
406  // Видалення пацієнта
407  const handleDelete = async (id) => {···
431  };
432
433    const [sortField, setSortField] = useState('id');
434  const [sortDirection, setSortDirection] = useState('asc');
435
436  const handleSort = (field) => {···
447  };
448
449  const getFieldValue = (p, field) => {···
464  };
```

Рис. 3.11 Фрагмент компонента керування пацієнтами.

Для створення нового пацієнта використовується модальне вікно з формою, де реалізовано перевірку обов'язкових полів (повне ім'я, телефон, дата народження) (рис. 3.12).

```
480  // Відкрити медкарту
481  const openCard = async (id) => {
482    setCardError('');
483    setCardLoading(true);
484
485    try {
486      const res = await fetch(`${API_BASE_URL}/api/patients/${id}`);
487      const data = await res.json();
488
489      if (!res.ok) {
490        throw new Error(data.error || 'Помилка завантаження медичної карти');
491      }
492
493      setCardPatient({
494        ...data,
495        // для input type="date"
496        date_of_birth: data.date_of_birth
497          ? data.date_of_birth.slice(0, 10)
498          : '',
499      });
500      setCardOpen(true);
501    } catch (err) {
502      console.error(err);
503      setCardError(err.message);
504    } finally {
505      setCardLoading(false);
506    }
507  };
```

Рис. 3.12 Модальне вікно медичної карти пацієнта з можливістю редагування, додаткових полів.

Під час підтвердження форми формується запит POST /api/patients з JSON-представленням введених даних. На сервері відповідний контролер викликає сервісний метод, що перевіряє коректність даних, приводить їх до внутрішнього формату та передає в репозиторій, де формується параметризований SQL-запит до PostgreSQL. У випадку редагування запису використовується запит PUT або PATCH з передачою ідентифікатора пацієнта та оновлених полів, а для видалення

передбачено запит DELETE /api/patients/:id. На рівні бази даних налаштовано обмеження цілісності, що унеможливлюють видалення пацієнта, якщо з ним пов’язані записи на прийом, що дозволяє зберегти історичну інформацію.

Аналогічну структуру реалізовано для модуля керування лікарями. На окремій сторінці адміністратор може переглядати список лікарів, відфільтровувати їх за спеціалізацією або прізвищем, відкривати форму редагування та створювати нові записи (рис. 3.13).

```
850  function DoctorsTable({ doctors, onDoctorChanged }) {
851    const [newFullName, setNewFullName] = useState('');
852    const [newSpecialization, setNewSpecialization] = useState('');
853    const [newPhone, setNewPhone] = useState('');
854    const [saving, setSaving] = useState(false);
855    const [deletingId, setDeletingId] = useState(null);
856    const [localError, setLocalError] = useState('');
857
858    // Стан для картки лікаря
859    const [cardOpen, setCardOpen] = useState(false);
860    const [cardDoctor, setCardDoctor] = useState(null);
861    const [cardLoading, setCardLoading] = useState(false);
862    const [cardSaving, setCardSaving] = useState(false);
863    const [cardError, setCardError] = useState('');
864
865 >  const handleAdd = async () => {···
902    };
903
904 >  const handleDelete = async (id) => {···
928    };
929
930 >  const openCard = async (id) => {···
950    };
951
952 >  const closeCard = () => {···
956    };
957
```

Рис. 3.13 Фрагмент компонента довідника лікарів із формою додавання та модальногою карткою лікаря

Модальне вікно введення забезпечує введення повних імен, спеціалізації, контактних даних, номера кабінету та додаткових приміток. Валідація на клієнтському боці гарантує коректність формату електронної пошти та телефонів,

а також заповнення усіх необхідних полів, перш ніж дані буде відправлено через API. Відповідний серверний модуль працює з таблицею doctors, використовуючи репозиторій для вставки, вибірки та оновлення записів, при цьому весь SQL-код зосереджений у одному місці, що відповідає концепції шаблону Repository, сформульованій раніше.

Модуль керування медичними послугами реалізовано як окреме представлення каталогу доступних процедур. Таблиця послуг містить назву, опис, стандартну вартість та ознаку активності. У процесі роботи адміністратор може тимчасово деактивувати певні послуги, не вдаючись до фізичного видалення записів. На клієнтському боці це відображається у вигляді перемикача або відповідної колонки, що сигналізує про те, чи доступна ця послуга для нових записів. При спробі створення нового запису на прийом у формі відбору послуги відображаються лише активні позиції, що узгоджується з бізнес-вимогами щодо актуальності каталогу та прозорості для клієнтів.

Найбільш насиченим з точки зору логіки є модуль “Записи на прийом”, який об’єднує дані відразу з кількох таблиць: patients, doctors, services та appointments. На клієнтському боці він представлений у вигляді таблиці, де кожен рядок відображає конкретний запис: пацієнта, лікаря, назву послуги, дату і час прийому, номер кабінету та поточний статус (рис. 3.14).

```
1298    // --- стан форми додавання ---
1299    const [newPatientId, setNewPatientId] = useState('');
1300    const [newDoctorId, setNewDoctorId] = useState('');
1301    const [newServiceId, setNewServiceId] = useState('');
1302    const [newDateTime, setNewDateTime] = useState('');
1303    const [newStatus, setNewStatus] = useState('scheduled');
1304    const [newNotes, setNewNotes] = useState('');
1305
1306    // пошук у формах (пацієнт / лікар / послуга)
1307    const [patientQuery, setPatientQuery] = useState('');
1308    const [doctorQuery, setDoctorQuery] = useState('');
1309    const [serviceQuery, setServiceQuery] = useState('');
1310    const [patientOpen, setPatientOpen] = useState(false);
1311    const [doctorOpen, setDoctorOpen] = useState(false);
1312    const [serviceOpen, setServiceOpen] = useState(false);
1313
```

Рис. 3.14 Фрагменти компонента керування записами на прийом.

Над таблицею розташовані засоби фільтрації: поле пошуку за ПІБ пацієнта чи лікаря, селектор діапазону дат та випадаючий список для фільтрації за статусом (“Заплановано”, “Завершено”, “Скасовано”). Ці параметри використовуються для формування запиту до сервера (наприклад, GET /api/appointments?dateFrom=...&dateTo=...&status=...&q=...), де вже на рівні SQL реалізується відповідна вибірка даних з умовами WHERE та сортуванням за часом прийому (рис. 3.15).

```
415  app.get('/api/appointments', async (req, res) => {
416    try {
417      const result = await pool.query(
418  >     `SELECT a.id, ···
435        ORDER BY a.appointment_datetime`
436      );
437      res.json(result.rows);
438    } catch (error) {
439      console.error('Помилка при отриманні записів:', error);
440      res.status(500).json({ error: 'Помилка сервера' });
441    }
442  });
443
444  // Додати новий запис на прийом
445  app.post('/api/appointments', async (req, res) => {
446  >   try {···
479    } catch (error) {
480      console.error('Помилка при створенні запису:', error);
481      res.status(500).json({ error: 'Помилка сервера' });
482    }
483  });
```

Рис. 3.15 Маршрути роботи із записами на прийом.

Створення нового запису на прийом реалізовано через окрему модальну форму. Ця форма містить комбіновані поля вибору пацієнта, лікаря та послуги (із списків, завантажених з бази), вибір дати та часу, а також текстове поле примітки. При підтвердженні система виконує комплекс перевірок: наявність усіх обов’язкових полів, коректність формату дати та часу, а також відсутність конфліктів у розкладі лікаря для вказаного часовго слота. Остання умова реалізована на серверному рівні у вигляді окремого сервісного методу, який перед виконанням операції вставки

перевіряє базу даних на предмет існування записів зі статусами “Заплановано” або “Завершено” для того ж лікаря та діапазону часу. У разі виявлення конфлікту формується помилка з описовим повідомленням, яке інтерфейс показує адміністратору у вигляді повідомлення про неможливість створення запису на обраний час.

Зміна статусів записів на прийом реалізована через інтерфейс таблиці: у відповідній колонці відображається випадаючий список або блок керування, що дозволяє змінити статус із “Заплановано” на “Завершено” або “Скасовано”. При зміні статусу надсилається запит PATCH /api/appointments/:id, який оновлює відповідний запис у таблиці appointments. У випадку перенесення прийому фактично створюється новий запис з іншою датою та часом, а попередній отримує статус “Скасовано”. Така стратегія реалізації дозволяє зберегти повну історію записів без втрати інформації про початкові домовленості.

Для уніфікації взаємодії між клієнтською частиною та сервером у проєкті застосовано підхід, схожий на використання об’єктів передачі даних (DTO). Серверні контролери формують відповіді в узгодженому форматі: наприклад, при поверненні списку записів на прийом у кожному об’єкті одразу містяться розгорнуті дані про пацієнта, лікаря та послугу (їхні імена та назви), що усуває потребу в додаткових запитах із боку клієнта. Така структуризація відповідей зменшує затримку при відображенні таблиць та спрощує код компонентів React, які працюють з уже підготовленими даними.

З точки зору користувацької взаємодії, усі модулі адміністративної панелі реалізовані як функціональні компоненти React з використанням хуків useState та useEffect. Перший використовується для зберігання поточного набору даних, стану завантаження та параметрів фільтрації, другий - для виконання побічних ефектів, зокрема запитів до API при зміні фільтрів або відкритті розділу. Для повторно використовуваних елементів, таких як кнопки дій, модальні вікна або табличні заголовки, створено окремі допоміжні компоненти, що зменшує дублювання коду та покращує підтримуваність інтерфейсу.

Важливу роль у реалізації функціональних модулів відіграють механізми обробки помилок і валідації, які реалізовано як на клієнтському, так і на серверному рівні. На стороні клієнта форми введення даних обладнано перевірками на заповнення обов'язкових полів, коректність форматів дат та контактних даних. У разі некоректного введення користувач отримує наочні підказки й виділення проблемних полів. На стороні сервера усі критичні операції обгорнуто в блоки обробки помилок, а стандартизований формат відповіді з помилкою дозволяє інтерфейсу відобразити інформативне повідомлення (наприклад, про конфлікт у розкладі або порушення цілісності даних у базі).

У додаток до основної CRUD-функціональності, усі модулі системи враховують вимоги до конфіденційності даних. Доступ до адміністративних сторінок обмежується перевіркою стану авторизації; при спробі відкрити будь-який з розділів без авторизованого сеансу користувач перенаправляється на екран входу. На рівні API захищені маршрути перевіряють наявність та валідність маркера доступу, а дані, що повертаються, обмежуються лише тими полями, які необхідні для виконання адміністративних завдань.

Загалом реалізація функціональних модулів поєднує у собі чіткий поділ відповідальності між клієнтом, сервером і базою даних, використання перевірених шаблонів проєктування (Component-based UI, Repository, DTO, рольова модель доступу) та уважне ставлення до реальних сценаріїв роботи адміністратора медичного центру. У наступному підрозділі буде розглянуто процес тестування реалізованих модулів, перевірку їхньої відповідності бізнес-вимогам, а також оцінку ефективності впровадження системи з точки зору зниження навантаження на адміністративний персонал та підвищення прозорості операцій.

3.5 Тестування системи та оцінка ефективності впровадження

Метою етапу тестування є перевірка коректності реалізованих функціональних модулів адміністративної панелі медичного центру, відповідності їх бізнес-

вимогам, а також оцінка зручності роботи адміністратора з розробленим інтерфейсом. Тестування проводилося на локальному середовищі розробника: клієнтська частина запускалася у браузері Google Chrome, серверна частина – у середовищі Node.js з використанням Express, а рівень даних – у СУБД PostgreSQL.

У процесі перевірки було використано комбінований підхід, що включав:

- функціональне тестування основних CRUD-операцій для довідників «Пацієнти», «Лікарі», «Послуги» та модуля «Записи на прийом»;
- інтеграційне тестування взаємодії клієнтської частини з REST-API (маршрути /api/patients, /api/doctors, /api/services, /api/appointments);
- негативне тестування, спрямоване на перевірку обробки помилкових та неповних даних;
- оцінку зручності роботи інтерфейсу з точки зору типових сценаріїв адміністратора.

Для кожного основного модуля було сформовано набір тестових сценаріїв. Для модуля керування пацієнтами перевірялись операції створення нового запису, редагування даних медичної карти, відображення списку та спроба видалення пацієнта, для якого існують пов’язані записи на прийом. У першому випадку, при коректному заповненні обов’язкових полів, сервер повертав код успіху та новий ідентифікатор пацієнта, а таблиця даних на клієнтському боці автоматично оновлювалася. У другому – при спробі видалити пацієнта, прив’язаного до наявних прийомів, сервер повертав помилку з повідомленням про порушення цілісності, яке відображалося в інтерфейсі у вигляді попередження. Це підтвердило коректну роботу обмежень зовнішніх ключів та обробки помилок на рівні REST-API.

Аналогічні сценарії були прогнані для модуля керування лікарями та довідника послуг. Перевірялося додавання нового лікаря з обов’язковими полями (ПІБ, спеціалізація, телефон), оновлення електронної пошти та номера кабінету, а також робота модального вікна редагування. Для послуг тестувалися створення запису з назвою, описом і базовою вартістю, сортування за назвою та ціною, а також коректне відображення ознаки активності послуги при формуванні нового запису на прийом.

Окрему групу тестів присвячено модулю “Записи на прийом”, який об’єднує дані з кількох таблиць. Тут перевірялись такі сценарії:

- створення нового запису з вибором пацієнта, лікаря, послуги, дати й часу;
- фільтрація записів за діапазоном дат, статусом та конкретним пацієнтом/лікарем;
- зміна статусу запису з “Заплановано” на “Завершено” та “Скасовано” без перезавантаження сторінки;
- видалення запису та автоматичне оновлення календарного відображення.

У рамках негативного тестування перевірено спроби створити запис без вибору одного з обов’язкових параметрів (пацієнта, лікаря, дати/часу чи послуги). У цих випадках клієнтська частина блокувала відправлення запиту та виводила підказку про незаповнені поля, що підтвердило коректність валідації на стороні інтерфейсу. Додатково було змодельовано ситуації з некоректним форматом телефону та дати народження, помилки відображалися для користувача у вигляді зрозумілих повідомлень.

Під час інтеграційного тестування за допомогою браузера та інструментів налагодження перевірено правильність формату JSON-відповідей серверної частини. Було підтверджено, що маршрути для отримання записів на прийом повертають розгорнуту інформацію про пацієнта, лікаря та послугу в одному DTO-об’єкті, що дає змогу клієнтській частині відображати дані без додаткових запитів.

Результати тестування показали, що розроблена система в цілому відповідає сформульованим вимогам: основні бізнес-процеси додавання, редагування та пошуку інформації реалізовані коректно, помилки некоректного введення даних перехоплюються як на клієнтському, так і на серверному рівнях, а інтерфейс адміністративної панелі забезпечує швидкий доступ до ключових операцій. Це створює підґрунтя для подальшого розширення функціональності системи та її впровадження в реальних умовах роботи медичного центру.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено та досліджено автоматизовану систему процесів адміністрування медичного центру, яка орієнтована на підтримку щоденної роботи адміністратора зі записами на прийом, пацієнтами, лікарями та переліком медичних послуг. Актуальність теми зумовлена зростанням обсягів інформації у медичних закладах, підвищеними вимогами до оперативності обслуговування пацієнтів та необхідністю зменшення навантаження на адміністративний персонал за рахунок впровадження сучасних інформаційних технологій.

У вступі було сформульовано мету роботи – розробити прототип інформаційної системи адміністративної панелі медичного центру на основі сучасних веб-технологій, що забезпечує підтримку ключових бізнес-процесів закладу. Для досягнення цієї мети були визначені та розв’язані такі основні завдання: аналіз предметної області та існуючих рішень; опис та моделювання бізнес-процесів медичного центру; побудова інформаційної та програмної архітектури системи; проєктування інтерфейсу користувача адміністративної панелі; реалізація прототипу на базі стеку React + Node.js/Express + PostgreSQL; проведення тестування основних функціональних модулів.

У ході дослідження було проаналізовано особливості організації роботи медичних центрів, виявлено основні інформаційні потоки й ролі, що беруть участь у процесах адміністрування. На основі цього було сформовано й формально описано бізнес-процеси роботи адміністратора із записами пацієнтів, управлінням кадровим складом лікарів та каталогом медичних послуг. Отримані моделі стали основою для побудови логічної структури бази даних та функціонального наповнення адміністративної панелі.

На рівні архітектури системи було спроєктовано трирівневу модель, що включає клієнтський рівень (React-інтерфейс адміністратора), серверний рівень (Node.js/Express з REST-API) та рівень даних (PostgreSQL). Розроблено структуру таблиць для зберігання інформації про пацієнтів, лікарів, послуги та записи на

прийом, реалізовано зовнішні ключі й обмеження цілісності, які запобігають некоректним операціям (зокрема, видаленню пов’язаних записів). Такий підхід забезпечив узгодженість даних та підготовив основу для подальшого масштабування.

На базі проєктних рішень реалізовано прототип адміністративної панелі медичного центру. У клієнтській частині створено окремі функціональні модулі: керування пацієнтами, лікарями, медичними послугами та записами на прийом. Забезпечено можливості перегляду, додавання, редагування та видалення записів, а також фільтрації й сортування даних у табличних поданнях. Окрему увагу приділено модулю “Записи на прийом”, який об’єднує дані з кількох сутностей, підтримує роботу з календарем, зміну статусів записів та відображення лише актуальних послуг. Реалізовано екран авторизації адміністратора та базові механізми обмеження доступу до адміністративної частини системи.

Проведене тестування основних функціональних модулів показало коректну роботу CRUD-операцій, узгодженість дій інтерфейсу з бізнес-правилами та очікуваною поведінкою адміністратора медичного центру. Перевірено обробку типових помилок введення даних, реакцію системи на відсутність з’єднання з сервером, некоректні запити та порушення цілісності в базі. Отримані результати підтвердили відповідність реалізованого прототипу сформульованим функціональним вимогам.

Практичне значення виконаної роботи полягає в тому, що розроблений прототип може бути використаний як основа для впровадження реальної адміністративної системи в медичному центрі, а також як базова платформа для подальшого розширення: інтеграції з електронною медичною картою, сервісами онлайн-запису, SMS/e-mail-нагадуваннями для пацієнтів, системами аналітики та звітності для керівництва. Окремі архітектурні та програмні рішення можуть бути повторно використані в інформаційних системах споріднених доменів (клініки, діагностичні центри, приватні медичні кабінети).

Таким чином, поставлена у роботі мета досягнута, усі основні завдання виконані.

Результатом є цілісна концепція та прототип автоматизованої системи процесів адміністрування медичного центру, що поєднує сучасні технології веб-розробки, структуровану модель даних та орієнтованість на реальні потреби адміністративного персоналу. Перспективами подальшого розвитку є розширення рольової моделі доступу, інтеграція з зовнішніми медичними інформаційними системами, підключення модулів аналітики та прогнозування завантаженості лікарів на основі методів штучного інтелекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Булах І. Є. Медична інформатика. Тернопіль : Терноп. держ. мед. ун-т ім. І.Я. Горбачев., 2008. 309 с.
2. Момоток Л., Юшина Л., Рожнова О. Основи медичної інформатики. Київ, 2008. 232 с.
3. Що таке медичні інформаційні системи та які послуги вони надають. Електронна система охорони здоров'я eHealth. URL: https://ehealth.gov.ua/2024/05/07/shho-take-medychni-informatsijni-systemy-ta-yaki-poslugy-vony-nadayut/.
4. Електронна система охорони здоров'я (ЕСОЗ). Офіційний сайт МОЗ України та eHealth. URL: https://ehealth.gov.ua/.
5. Медичні інформаційні системи: завдання, можливості та перспективи розвитку. Evergreen. URL: https://evergreens.com.ua/ua/articles/medical-information-systems.html.
6. Лирчиков В. Огляд сучасних медичних інформаційних систем. Актуальні проблеми автоматизації та інформаційних технологій. 2021. Т. 25.
7. Хрустальова С. Автоматизована інформаційна система оптимізації діяльності медичного закладу. Виробництво & Мехатронні Системи. 2022. С. 118–121.
8. EMCI. Медичні інформаційні системи в Україні. Офіційний сайт EMCI/EMCIМЕД. URL: https://emci.ua/.
9. Медичні інформаційні системи та рішення для медичного бізнесу Doctor Eleks. URL: https://doctor.eleks.com/.
10. Helsi.me – медична інформаційна система для пацієнтів та лікарів. URL: https://helsi.me/.
11. Epizitone A., Moyane S. P., Agbehadji I. E. A Systematic Literature Review of Health Information Systems for Healthcare. Healthcare. 2023. Vol. 11, no. 7. P. 959.

12. Crisan E. L., Mihaila A. Health-care information systems adoption – a review of management practices. Vilakshan - XIMB Journal of Management. 2021. Ahead-of-print, ahead-of-print.

13. Rachid O., Hanan B., Fatima M. Ten years of Hospital Information Systems: A taxonomy attempt. Procedia Computer Science. 2024. Vol. 239. P. 1401–1408.

14. Improving the Performance of Hospital Information Systems Using Six Sigma for Kermanshah Province Hospitals / M. J. Jamshidi et al. Journal of Clinical Research in Paramedical Sciences. 2021. Vol. 10, no. 1.

15. Andwika I. B., Yundari E., Florenza Y. Hospital Management Information System in Increasing Efficiency. IC-ITECHS. 2024. Vol. 5, no. 1. P. 890–894.

16. Surya Utama, Balqis Nurmauli Damanik. Evaluating Hospital Information Systems: A Systematic Review of Effectiveness, Implementation, and Impact on Health Services Administration. Systematic Literature Review Journal. 2024. Vol. 1, no. 2. P. 12–24.

17. Joseph A. Information systems in healthcare: improving patient care and efficiency. Business Studies Journal. 2023. Vol. 15, no. 3. P. 1–3.

18. Zayas-Cabán T., Haque S. N., Kemper N. Identifying Opportunities for Workflow Automation in Health Care: Lessons Learned from Other Industries. Applied Clinical Informatics. 2021. Vol. 12, no. 03. P. 686–697.

19. Stone P. 10 Essential Healthcare Workflow Automation Strategies. Process automation. 2024.

20. Artificial-Intelligence-Based Clinical Decision Support Systems in Primary Care: A Scoping Review of Current Clinical Implementations / C. A. Gomez-Cabello et al. European Journal of Investigation in Health, Psychology and Education. 2024. Vol. 14, no. 3. P. 685–698.

21. Explainable AI for Clinical Decision Support Systems: Literature Review, Key Gaps, and Research Synthesis / M. Salimparsa et al. Informatics. 2025. Vol. 12, no. 4. P. 119.

22. Artificial intelligence-driven clinical decision support systems for early detection and precision therapy in oral cancer: a mini review / M. K. Karuppan Perumal et al. Frontiers in Oral Health. 2025. Vol. 6.

23. Health Level Seven International. HL7 FHIR Specification. URL: https://www.hl7.org/fhir/overview.html.

24. Савченко Л. Системний аналіз в охороні здоров'я. Львів : ЛНМУ, 2018.

25. Карпенко С. Г., Попов В. В., Тарнавський Ю. А., Шпортюк Г. А. Інформаційні системи і технології : навч. посіб. – К. : МАУП, 2004 / С. Карпенко та ін. Київ : МАУП, 2004.

26. du Boulay B. N.J. Nilsson, Artificial Intelligence: A New Synthesis T. Dean, J. Allen and Y. Aloimonos, Artificial Intelligence: Theory and Practice D. Poole, A. Mackworth and R. Goebel, Computational Intelligence: A Logical Approach S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Artificial Intelligence. 2001. Vol. 125, no. 1-2. P. 227–232.

27. Heaton J. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. Genetic Programming and Evolvable Machines. 2017. Vol. 19, no. 1-2. P. 305–307.

28. Sommerville I. Software Engineering. Pearson. 2015.

29. Pressman R., Maxim B. Software Engineering: A Practitioner's Approach, 8th Ed. McGraw-Hill, 2014.

30. Patterns of Enterprise Application Architecture / M. Fowler et al. Addison Wesley, 2002. 560 p.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

АВТОМАТИЗОВАНА СИСТЕМА ПРОЦЕСІВ АДМІНІСТРУВАННЯ МЕДИЧНОГО ЦЕНТРУ

на здобуття освітнього ступеня магістра зі спеціальності 126 Інформаційні системи та технології освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСДМ-61 Микита БІЛОШИЦЬКИЙ

Науковий керівник: доцент кафедри інформаційних систем та технологій, доктор філософії Валентина ДАНИЛЬЧЕНКО

Актуальність теми

Ця робота присвячена створенню веб-орієнтованої системи для оптимізації адміністративних процесів у медичних центрах, що вирішує проблеми зростання даних та фрагментованості систем.

Мета роботи

Розробити веб-орієнтовану автоматизовану систему для оптимізації адміністративних процесів та підвищення ефективності управління медичним центром.

Основні завдання

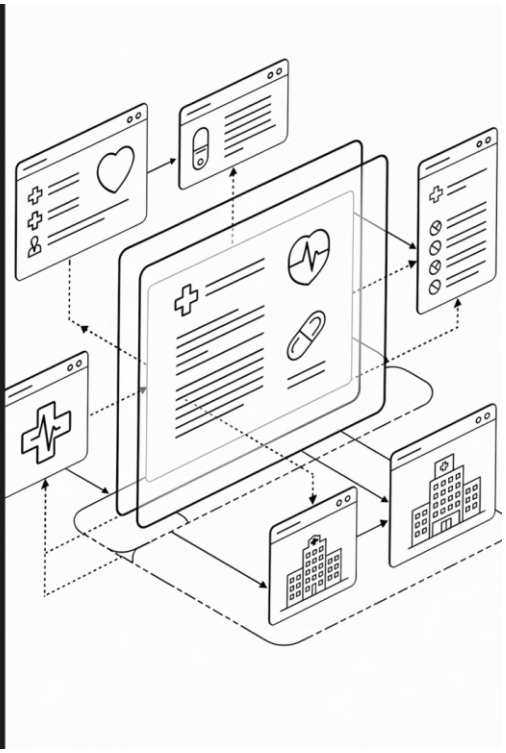
- Аналіз потреб та вимог до системи.
- Проєктування архітектури та реалізація прототипу.
- Тестування функціональних модулів.

Об'єкт дослідження

Адміністративні процеси медичного центру: реєстрація пацієнтів, планування прийомів, управління ресурсами та облік.

Предмет дослідження

Методи, алгоритми та програмні засоби для автоматизації адміністративних процесів у медичних закладах.



Задачі адміністрування медичного центру

1. **Управління пацієнтами та записами**
   Ефективна реєстрація пацієнтів та керування їхніми записами на прийом, зменшуючи час очікування та оптимізуючи потік.
2. **Планування розкладів лікарів та ресурсів**
   Оптимізація графіків роботи персоналу та використання обладнання, забезпечуючи максимальну продуктивність.
3. **Ведення картотеки пацієнтів**
   Централізоване зберігання та легкий доступ до медичних записів, історії хвороби та контактної інформації.
4. **Формування звітності**
   Автоматизоване створення управлінської та статистичної звітності для аналізу ефективності та прийняття рішень.

Задачі	Модулі MIC
Реєстрація пацієнтів	Модуль керування пацієнтами
Запис на прийом	Модуль запису на прийом
Керування розкладом	Модуль керування лікарями
Аналіз даних	Модуль звітності

Обґрунтування вибору технологій

Frontend: React

Використання **React** для створення **SPA**-архітектури забезпечує високу швидкість завантаження та інтерактивність користувацького інтерфейсу.

Backend: Node.js / Express

Node.js з фреймворком **Express** ідеально підходить для побудови швидких та масштабованих серверних додатків, що обробляють велику кількість запитів.

База даних: PostgreSQL

PostgreSQL обрано за його надійність, гнучкість та підтримку складних запитів, що є критично важливим для зберігання медичних даних.

Взаємодія: REST API

Взаємодія компонентів через **REST API** гарантує уніфікований та безпечний обмін даними між клієнтською та серверною частинами.

Цей стек технологій забезпечує високу **масштабованість**, **підтримуваність** та **продуктивність** системи, що дозволяє їй ефективно адаптуватися до зростаючих потреб медичного центру.

Архітектура системи

Клієнтський рівень

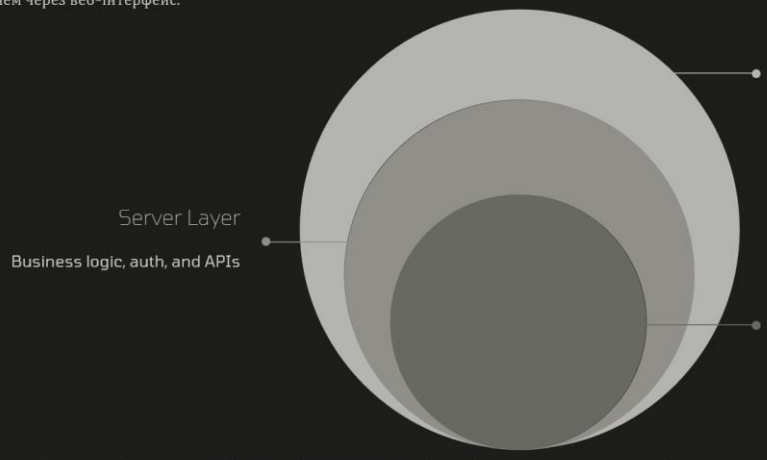
Відповідає за відображення даних та взаємодію з користувачем через веб-інтерфейс.

Серверний рівень

Обробляє бізнес-логіку, виконує авторизацію та автентифікацію, а також взаємодіє з базою даних.

Рівень бази даних

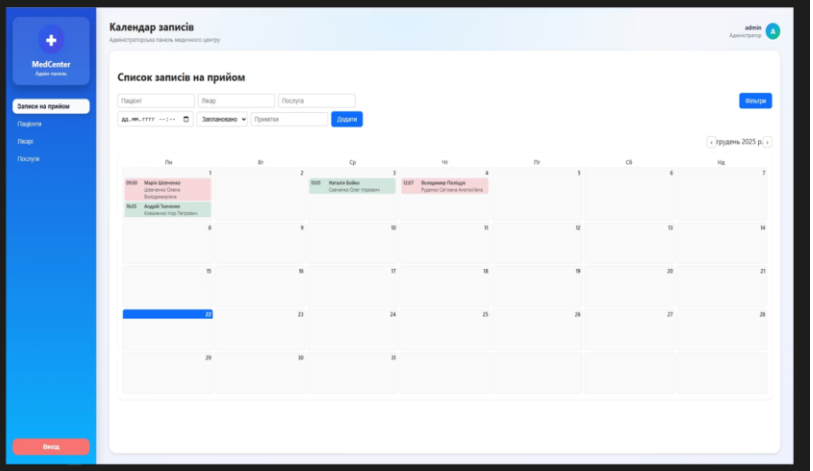
Зберігає всі медичні та адміністративні дані, забезпечуючи їхню цілісність та доступність.



Ця трирівнева архітектура забезпечує **чіткий поділ відповідальності** між рівнями, що спрощує розробку, тестування та подальше масштабування системи, а також підвищує її стійкість до відмов.

Інтерфейс адміністратора

- **Односторінковий веб-інтерфейс**
  Забезпечує безперебійну роботу та швидкий доступ до всіх функцій без перезавантаження сторінок.
- **Орієнтація на типові сценарії**
  Розроблено з урахуванням повсякденних завдань адміністратора для максимальної. зручності
- **Логічна сегментація**
  Інтерфейс розділений на функціональні області для інтуїтивного пошуку необхідних інструментів.
- **Фільтрація та сортування**
  Розширені можливості для швидкого пошуку та організації даних, що підвищує ефективність роботи.



Інтерфейс розроблено з фокусом на **зручність використання** та **ефективність**, дозволяючи адміністраторам швидко та легко керувати всіма процесами медичного центру.

Безпека та контроль доступу

Авторизація адміністратора

Впровадження надійних механізмів автентифікації та авторизації для забезпечення безпечного доступу до системи.

Обмеження доступу

Деталізований контроль над тим, які користувачі та групи мають доступ до певних адміністративних розділів.

Рольова модель доступу (RBAC)

Гнучка система розподілу прав доступу на основі визначених ролей (наприклад, адміністратор, лікар, медсестра).

Захист серверних маршрутів API

Забезпечення безпеки всіх кінцевих точок API від несанкціонованого доступу, ін'єкцій та інших атак.

Ці заходи гарантують **конфіденційність**, **цілісність** та **доступність** даних, захищаючи систему від несанкціонованого втручання та забезпечуючи її стабільну роботу.

Приклад програмної реалізації

- **Реалізація CRUD-операцій**

  Ефективне створення, читання, оновлення та видалення записів, що є основою управління даними.

- **Валідація даних на клієнті та сервері**

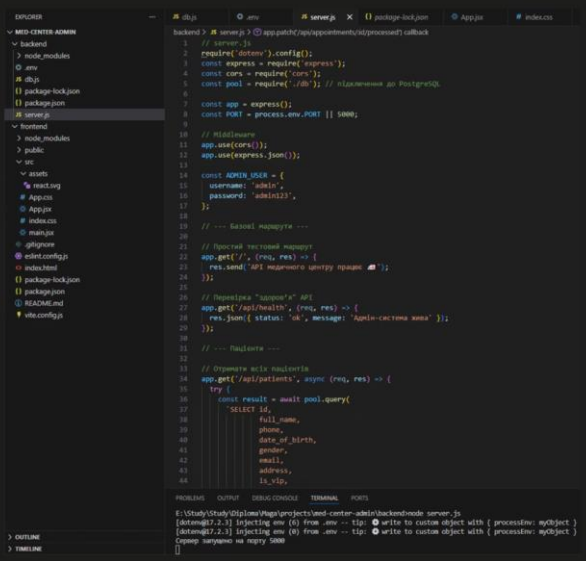
  Багаторівнева перевірка вхідних даних для забезпечення їх цілісності та запобігання помилкам.

- **Централізована обробка помилок**

  Єдиний механізм для перехоплення та логування помилок, що підвищує стабільність та легкість підтримки системи.

- **Узгодженість інтерфейсу з бізнес-правилами**

  Кожен елемент інтерфейсу відповідає специфічним вимогам та логіці роботи медичного центру.



Цей підхід забезпечує **надійність** та **ефективність** роботи системи, гарантуючи, що всі операції обробляються коректно та відповідно до встановлених стандартів.

Основні функціональні модулі

Керування пацієнтами

Реєстрація, редагування інформації, перегляд історії візитів та медичних даних пацієнтів.

Керування лікарями

Створення та редагування профілів лікарів, їхніх розкладів, спеціалізацій та доступності.

Керування медичними послугами

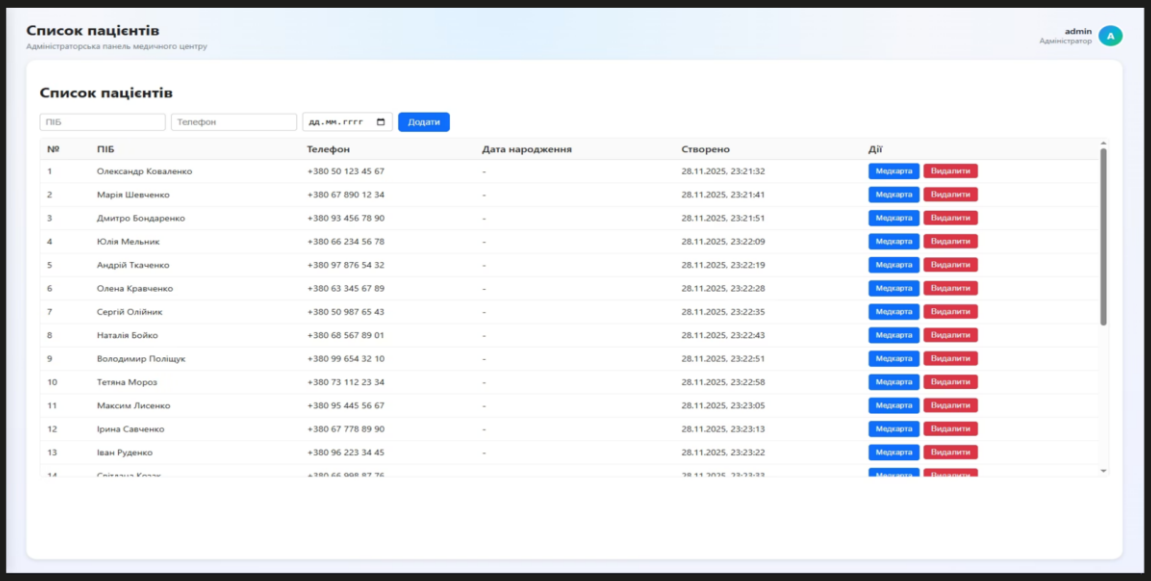
Додавання, редагування та каталогізація медичних послуг, їхньої вартості та описів.

Записи на прийом

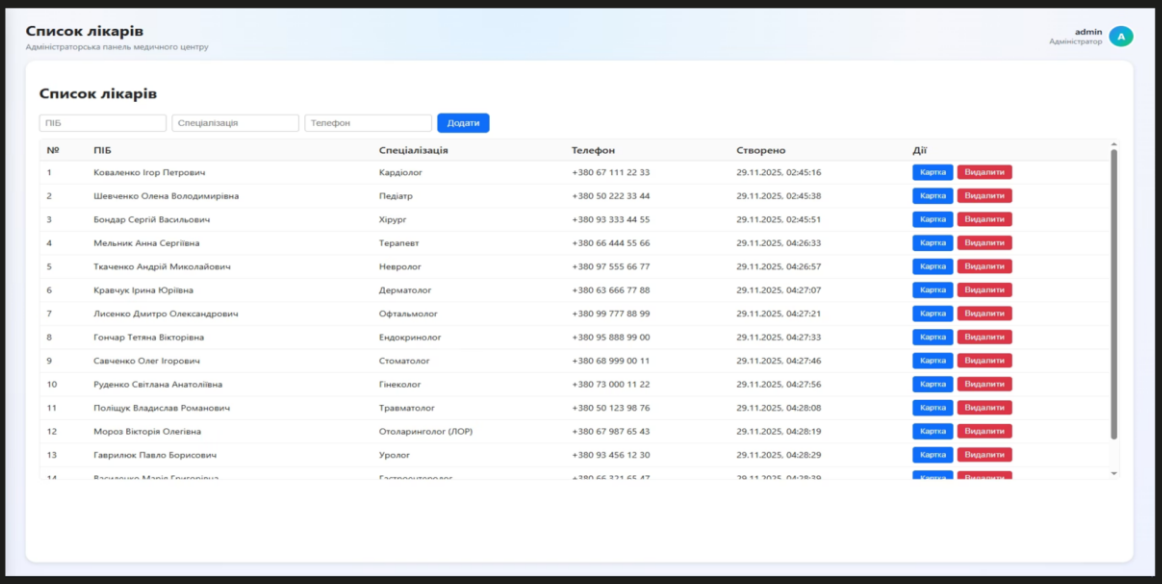
Система бронювання візитів, перегляд календаря завантаженості, зміна статусів записів.

Кожен модуль розроблений для **оптимізації конкретного аспекту** адміністративної роботи, що дозволяє медичному центру працювати злагоджено та ефективно.

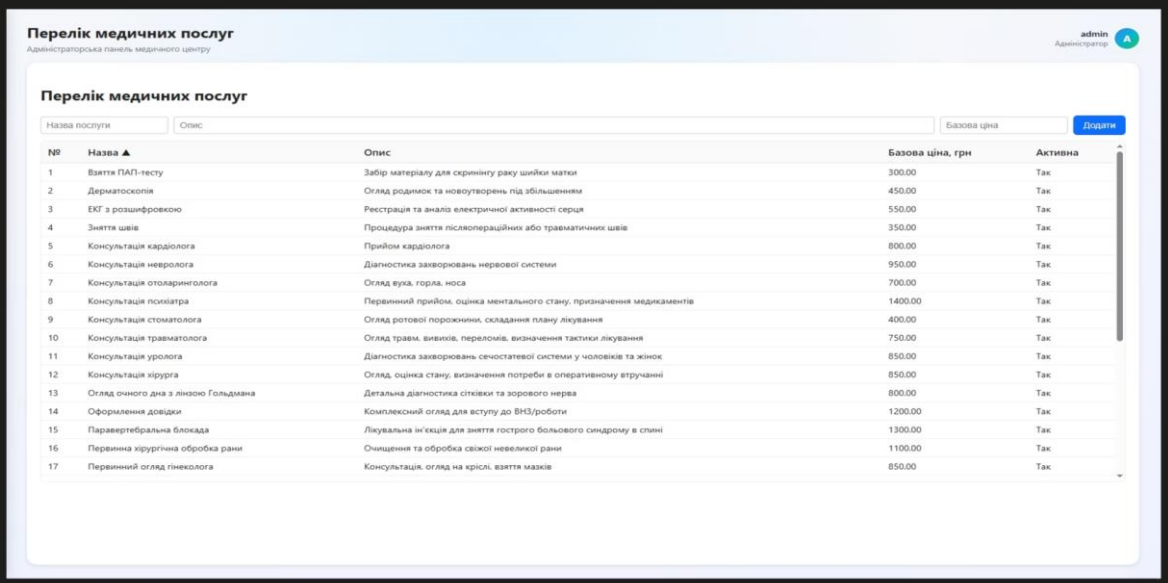
Модуль керування пацієнтами



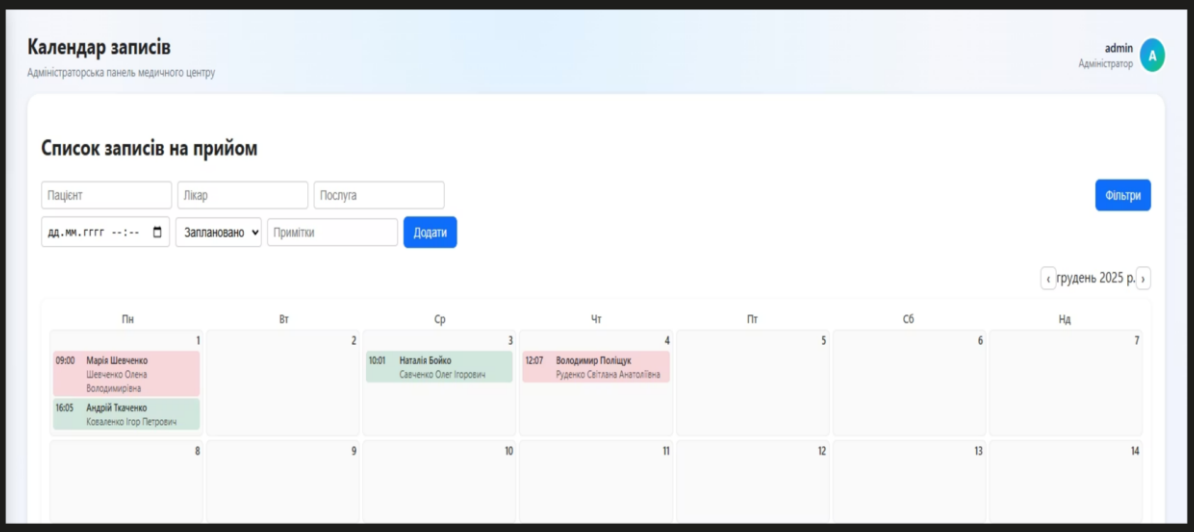
Модуль керування лікарями

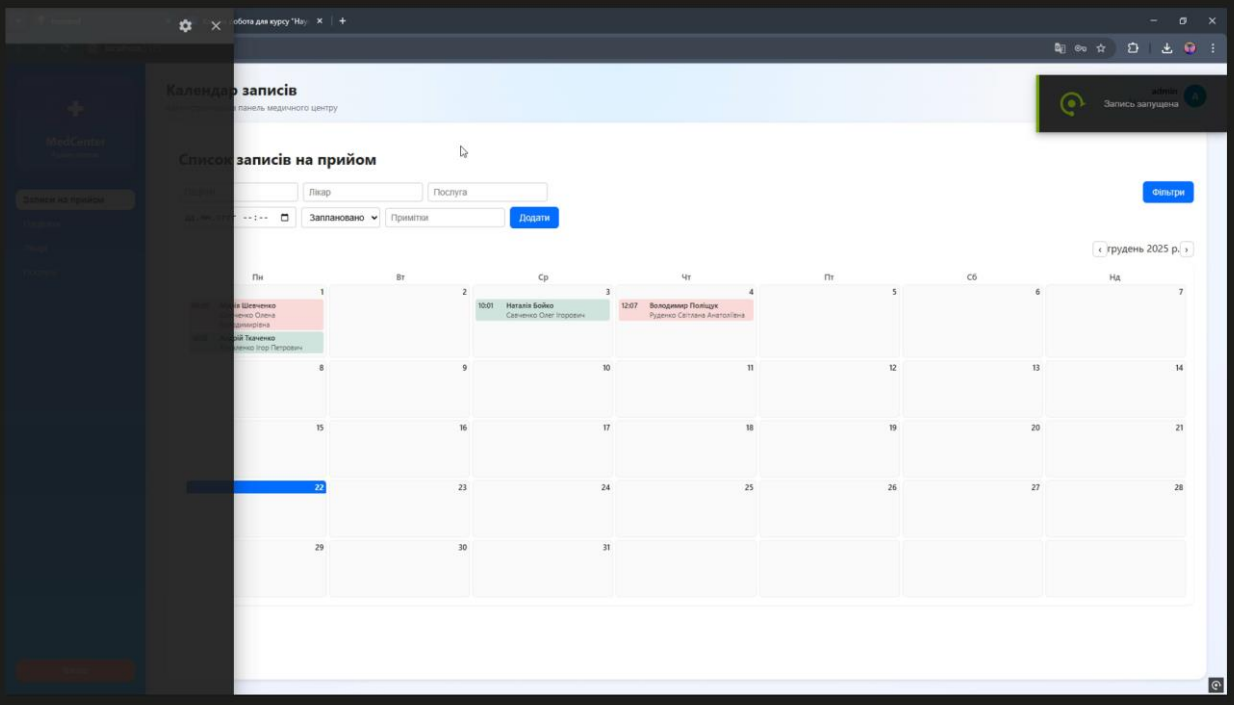


Модуль керування медичними послугами



Модуль запису на прийом





Тестування та практичне значення

Після ретельного етапу розробки, було проведено комплексне тестування системи для підтвердження її працездатності та ефективності.

Перевірка працездатності системи

Проведено перевірку коректності роботи основних функціональних модулів.

Перевірено сценарії створення, редагування та видалення даних.

Підтверджено стабільну роботу клієнтської та серверної частин.

Практичне значення

Прототип демонструє повний цикл адміністрування медичного центру.

Система може бути використана як основа для подальшого впровадження.

Автоматизація знижує навантаження на адміністративний персонал.

Ці результати підкреслюють **практичну цінність** розробленої системи для сучасних медичних закладів.

Висновки та перспективи

Досягнуті результати

- **Поставлену мету досягнуто:** успішно розроблено прототип системи для адміністрування медичного центру.
- **Усі завдання магістерської роботи виконано:** від аналізу вимог до тестування програмної реалізації.
- **Реалізовано цілісний прототип системи:** що охоплює основні функціональні модулі та забезпечує контроль доступу.

Подальші перспективи

Інтеграція з EHR-системами

Для обміну даними та створення єдиного інформаційного простору.

Розширена аналітика

Для глибшого розуміння показників роботи та прийняття обґрунтованих рішень.

Масштабування системи

Забезпечення можливості розширення функціоналу та підтримки зростаючої кількості користувачів.

Розроблена система є міцною основою для майбутнього розвитку та впровадження в реальні медичні заклади, сприяючи значній оптимізації їхньої діяльності.

Апробація

Білошицький М.П. “ШТУЧНИЙ ІНТЕЛЕКТ У МЕДИЦИНІ НА БАЗІ IOT: АНАЛІЗ ДАНИХ ВІД WEARABLE-ПРИСТРОЇВ”.

Теза у VI Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT» - м.Київ, 15 квітня 2025 року

Білошицький М.П., Білоус В.В., Данильченко В.М., Срібна І.М. “МОДЕЛІ МАШИННОГО НАВЧАННЯ ДЛЯ АНАЛІЗУ ТА ЗАХИСТУ ДАНИХ У СИСТЕМАХ МЕДИЧНОГО АДМІНІСТРУВАННЯ НА ОСНОВІ IOT”.

Стаття подана до друку у загальногалузевий науково-виробничий журнал “Зв'язок” - м.Київ

Дякую за увагу!