МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВИСОКОНАДІЙНОЇ БАЗИ ДАНИХ ДЛЯ МЕДИЧНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ З ПІДТРИМКОЮ ІСТОРІЇ ЗМІН»**

Виконала: студентка 4 курсу, групи 6.1262
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та технології
(назва освітньої програми)
Д.Е.Греджева
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н., Г. А. Добровольський
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

« ____ « _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Греджевій Діані Едуардівній
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка високонадійної бази даних для медичної інформаційної системи з підтримкою історії змін

керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 « грудня 2025 року № 1741-с

2. Строк подання студентом роботи 08.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування медичної вебсистеми.
3. Розробка та тестування вебсистеми.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.01.2026	
2.	Збір вихідних даних.	16.01.2026	
3.	Обробка методичних та теоретичних	29.01.2026	
	джерел.		
4.	Розробка першого та другого розділу.	12.02.2026	
5.	Розробка третього розділу.	15.04.2026	
6.	Оформлення та нормоконтроль	08.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2026	

Студент _____ Д. Е. Греджева
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г. А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка високонадійної бази даних для медичної інформаційної системи з підтримкою історії змін»: 76 с., 50 рис., 5 табл., 13 джерел, 1 додаток.

БАЗА ДАНИХ, ЕЛЕКТРОННА МЕДИЧНА КАРТКА, ЖУРНАЛ АУДИТУ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, МІГРАЦІЇ, СКБД POSTGRESQL, ЦІЛІСНІСТЬ ДАНИХ, JWT-АВТЕНТИФІКАЦІЯ, MEDSYSTEM.

Об’єкт дослідження – процес збереження, обробки та забезпечення цілісності конфіденційних даних у медичних інформаційних системах.

Предмет дослідження − методи, алгоритми та архітектурні рішення для створення високонадійної структури бази даних медичного призначення із механізмами автоматичного логування та версіонування змін.

Мета роботи: проектування та практична реалізація відмовостійкої і надійної архітектури бази даних для медичної установи, яка забезпечує повний аудит дій користувачів, збереження хронології медичних записів та захист від втрати інформації.

Методи дослідження – системний аналіз предметної області, проектування реляційних баз даних, тригерне програмування SQL, архітектурне моделювання програмного забезпечення, методи криптографічного захисту (хешування bcrypt, токени JWT).

У ході роботи розроблено вебсистему «MedSystem», призначену для автоматизації медичного закладу та захищеного збереження даних. Спроектовано базу даних у СКБД PostgreSQL з використанням UUID та автоматичним тригерним версіонуванням змін у форматі JSONB. Реалізовано серверний API (Node.js) та клієнтську частину (React.js) із розмежуванням ролей доступу.

SUMMARY

Bachelor’s Qualifying Thesis «Development of a Highly Reliable Database for a Medical Information System with Support of Change History»: 76 pages, 50 figures, 5 tables, 13 sources, 1 supplement.

DATABASE, ELECTRONIC HEALTH RECORD, AUDIT LOG, CLIENT-SERVER ARCHITECTURE, MEDICAL INFORMATION SYSTEM, MIGRATIONS, POSTGRESQL DBMS, DATA INTEGRITY, JWT AUTHENTICATION, MEDSYSTEM.

The object of the study – the process of storing, processing, and ensuring the integrity of confidential data in medical information systems.

Subject of the study – methods, algorithms, and architectural solutions for creating a highly reliable medical database structure with mechanisms for automated logging and revision versioning.

Aim of the study: the design and practical implementation of a fault-tolerant and reliable database architecture for a medical facility that provides a full audit of user actions, preservation of medical records chronology, and protection against data loss.

Methods of research – system analysis of the subject domain, relational database design, SQL trigger programming, software architectural modeling, and cryptographic protection methods (bcrypt hashing, JSON Web Tokens).

In the course of the work, the web system "MedSystem" was developed, designed for the automation of a medical facility and secure data storage. Designed database in PostgreSQL database using UUID and automatic trigger versioning of changes in JSONB format. Implemented server API (Node.js) and client part (React.js) with delineation of access roles.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат…….. 4
Summary 4
Вступ 8
1 Аналіз предметної області та проектування системи 10
1.1 Огляд наукових праць та аналіз стану вирішення проблеми 10
1.2 Постановка завдань дослідження, призначення та область застосування системи 11
1.3 Аналіз можливих методів дослідження, варіантів технічної реалізації та порівняльний аналіз 12
1.4 Міжнародні стандарти надійності та безпеки медичних інформаційних систем 14
1.5 Висновки до першого розділу 15
2 Проєктування структури та архітектури медичної інформаційної системи 16
2.1 Опис діаграм прецедентів та рольової моделі доступу 16
2.2 Розробка структурної схеми медичної системи та архітектури бази даних 18
2.3 Технічні вимоги до апаратного та програмного забезпечення системи 20
2.4 Опис логіки взаємодії компонентів та діаграми послідовності аудиту змін 21
2.5 Висновки до другого розділу 24
3 Алгоритмічно-програмне забезпечення інформаційної системи «medsystem» 25
3.1 Розробка алгоритмічного забезпечення системи та логіки аудиту подій 25
3.2 Програмна реалізація структури бази даних та опис таблиць 26
3.2.1 Розробка er-діаграми бази даних 27
3.2.2 Опис структури та характеристика таблиць сховища даних 28

3.3 Програмна реалізація інтерфейсу користувача та взаємодія з базою даних 31
3.3.1 Вхід до системи 31
3.3.2 Пацієнт 33
3.3.3 Лікар 38
3.4 Фрагменти програмної реалізації системи 55
3.5 Висновки до третього розділу 55
Висновки 56
Перелік посилань 58
Додаток А Фрагменти програмної реалізації системи 59

ВСТУП

Сьогодні комп’ютерні програми активно витісняють паперові картки та журнали з наших лікарень і поліклінік. Ми всі бачимо це на прикладі великих платформ, таких як «Helsi», через які можна швидко записатися до лікаря чи подивитися свої рецепти [10]. Автоматизація медицини дійсно полегшує життя: лікарям простіше вести облік, а пацієнтам не потрібно стояти в довгих чергах за карткою. Але коли справа доходить до розробки таких систем, виникає серйозна інженерна проблема – як зробити базу даних максимально надійною і захищеною. Медична інформація є дуже чутливою. Якщо через збій у програмі зникне історія хвороби або хтось таємно змінить діагноз пацієнта, наслідки можуть бути дуже важкими, аж до загрози здоров’ю людини. Аналіз того, як працюють подібні системи, показує одну велику проблему: більшість простих баз даних зберігають лише ту інформацію, яка є на цей момент. Якщо користувач оновлює або видаляє запис, стара версія просто зникає назавжди. У медицині це неприпустимо, адже завжди треба знати, який лікар, коли і що саме виправив у картці.

Саме тому виникла необхідність створити нову медичну систему «MedSystem» з високонадійною базою даних, яка буде автоматично записувати та зберігати повну історію всіх змін.

Для реалізації цього проєкту були обрані перевірені та доступні технології. Як основу системи я використав безкоштовну СКБД PostgreSQL. Вона дуже надійна та швидка.

Для того, щоб зберігати історію виправлень і не гальмувати при цьому роботу всієї програми, в базі даних було створено спеціальну таблицю аудиту, яка записує всі дії у гнучкому форматі JSONB. Сама структура бази складається з 9 чітко пов’язаних таблиць. Щоб захистити систему від збоїв та атак, у коді налаштовано пул постійних з’єднань, додано автоматичне створення резервних копій, а замість звичайних порядкових номерів для записів використано зашифровані ID (UUID). Безпека користувачів тримається на перевірці паролів через bcrypt та захищених

токенах JWT [9], які ділять усіх, хто заходить на сайт, на лікарів та пацієнтів. База даних тут працює в парі з сервером на Node.js [7] / Express.js [8] та зрозумілим сайтом на React.js [6], який відкривається на комп’ютері.

Медичну систему «MedSystem» можна спокійно брати та впроваджувати у невеликих приватних кабінетах, лабораторіях чи локальних лікарнях, де потрібен надійний електронний облік пацієнтів та суворий контроль за тим, хто і які зміни вносить у медичні документи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЕКТУВАННЯ СИСТЕМИ

1.1 Огляд наукових праць та аналіз стану вирішення проблеми

Створення надійних медичних інформаційних систем (МІС) сьогодні є одним із головних завдань комп'ютерних наук. Оскільки лікарні повністю переходять на цифрові технології, виникає гостра потреба у правильному проєктуванні баз даних для цих систем. Аналіз сучасних вітчизняних та зарубіжних наукових праць показує, що питанням збереження даних та безпеки приділяється дуже велика увага.

У роботах українських дослідників, які вивчають автоматизацію медичних закладів, часто піднімається тема захисту конфіденційної інформації (лікарської таємниці). Автори наголошують, що медична картка пацієнта не може бути звичайною таблицею в базі даних, до якої має доступ будь-який програміст чи користувач [11]. Наприклад, у періодичних наукових виданнях з комп'ютерних наук часто розглядаються архітектурні патерни безпеки, такі як керування доступом на основі ролей (RBAC). Це дуже актуально для нашого проєкту, оскільки в «MedSystem» є чіткий поділ на лікарів та пацієнтів, і система повинна надійно блокувати спроби несанкціонованого перегляду чужих карток.

Зарубіжні автори у своїх публікаціях більше фокусуються на проблемах відмовостійкості баз даних та збереженні хронології (історії) змін. У багатьох дослідженнях доведено, що класичні реляційні бази даних без спеціальних модулів аудиту втрачають інформацію під час операцій оновлення (UPDATE) або видалення (DELETE) [1]. Якщо користувач змінює рядок у таблиці, СКБД просто перезаписує блоки пам'яті, і старий стан даних зникає назавжди. Науковці пропонують різні варіанти вирішення цієї проблеми, серед яких:

- створення дублюючих історичних таблиць для кожної сутності [2];
- використання часових (темпоральних) баз даних [3];

- впровадження систем журналювання на основі тригерів або логування подій на рівні сервера додатків [4].

Аналіз показує, що створення окремих історичних таблиць під кожну сутність сильно роздуває структуру бази даних і ускладнює написання SQL-запитів. Тому в нашому дослідженні обрано більш сучасний та гнучкий напрямок: використання об’єктно-реляційної СКБД PostgreSQL з реалізацією єдиної таблиці аудиту audit_log, де історичні дані зберігаються в безсхемному форматі JSONB. Це обґрунтовано тим, що ми можемо в один рядок записати повний зліпок зміненого об’єкта, не створюючи десятки додаткових таблиць. Таким чином, обраний напрямок дослідження дозволяє поєднати строгі реляційні зв’язки з гнучким логуванням.

1.2 Постановка завдань дослідження, призначення та область застосування системи

На основі проведеного аналізу літературних джерел та реальних потреб автоматизації локальних медичних установ, було сформульовано задачу на розробку інформаційної системи «MedSystem».

Призначення системи – забезпечення швидкої, зручної та безпечної взаємодії між лікарями та пацієнтами в межах медичного закладу через персональні комп’ютери. Програма покликана автоматизувати ведення розкладу лікарів, запис пацієнтів на прийоми, збереження результатів оглядів та ведення електронних карток.

Область застосування результатів роботи – приватні медичні кабінети, стоматологічні клініки, локальні амбулаторії, діагностичні лабораторії та невеликі медичні центри, які потребують надійного цифрового обліку, але не мають бюджетів на розгортання великих державних систем.

Для досягнення цієї мети у роботі поставлено такі завдання дослідження:

Проєктування реляційної моделі, яка без дублювання інформації описує користувачів, пацієнтів, лікарів, їхні графіки, прийоми та медичні записи (всього 9 сутностей відповідно до коду реалізації).

Розробка та реалізація механізму високої надійності бази даних за допомогою налаштування пулу з'єднань (pool.js), що дозволить програмі стабільно працювати під навантаженням.

Створення системи повного аудиту дій (таблиця audit_log), яка фіксуватиме тип операції (CREATE, UPDATE, DELETE), час зміни, ID запису та деталі у форматі JSONB.

Реалізація засобів захисту конфіденційності на рівні сервера додатків за допомогою JWT-автентифікації та перевірки прав доступу за ролями (auth.js), а також захисту медичної таємниці через перевірку зв'язків між конкретним лікарем і пацієнтом (assertDoctorPatient).

1.3 Аналіз можливих методів дослідження, варіантів технічної реалізації та порівняльний аналіз

Для створення високонадійної системи з підтримкою історії змін необхідно обрати правильні інструменти та методи реалізації. Розглянемо можливі варіанти для трьох головних компонентів архітектури: системи збереження даних (СКБД), методу логування історії та способу організації зв'язку сервера з базою.

Вибір системи керування базами даних (СКБД)

Варіант А (NoSQL, наприклад MongoDB): Добре підходить для збереження історії у вигляді документів, але погано справляється зі строгими зв'язками (наприклад, коли треба чітко пов'язати графік лікаря, запис пацієнта та медичну картку). Є ризик порушення цілісності даних.

Варіант Б (Реляційна СКБД PostgreSQL): Забезпечує строгу посилальну цілісність завдяки первинним та зовнішнім ключам (FOREIGN KEY) з

підтримкою каскадних правил видалення (ON DELETE CASCADE / SET NULL) [5]. Має вбудований тип JSONB для гнучкого логування.

Варіанти технічної реалізації історії змін (аудиту)

Варіант А (Логування на рівні сервера додатків): Запис у таблицю аудиту робиться через бекенд-контролери на Node.js під час обробки запитів від користувача. Плюс – простота реалізації та легкий доступ до ID авторизованого користувача через JWT. Мінус – якщо хтось змінить дані напряму в БД через консоль, цей крок не запишеться.

Варіант Б (Логування на рівні СКБД через тригери PL/pgSQL): База даних сама автоматично логує будь-які зміни, навіть якщо вони зроблені в обхід сервера. Плюс – абсолютна надійність. Мінус – важче передати ID користувача з вебдодатка всередину тригера СКБД.

Рішення: У нашому проєкті реалізовано комбінований підхід: структура таблиці audit_log закладена на рівні схеми СКБД, а запис фіксується через параметризовані SQL-запити на сервері, що дозволяє чітко зв’язати дію з конкретним user_id, отриманим з токена безпеки.

Варіанти організації підключення до бази даних

Варіант А (Класичне відкриття з’єднання для кожного запиту): Сервер відкриває новий коннект до PostgreSQL при кожному кліку пацієнта і закриває його після відповіді. Це швидко перевантажує процесор і пам’ять, система починає гальмувати.

Варіант Б (Використання пулу з’єднань pg.Pool): Сервер заздалегідь створює набір готових підключень, які використовуються повторно. Це забезпечує високу надійність і швидкість роботи під великим навантаженням.

Для наочності проведемо порівняльний аналіз архітектурних рішень за ключовими критеріями (швидкість, надійність, гнучкість, складність) (див. табл. 1.1).

Таким чином, на основі порівняльного аналізу було обґрунтовано архітектуру системи «MedSystem»: використання **СКБД PostgreSQL** із пулом постійних з’єднань **pg.Pool**, збереження реляційних зв’язків із використанням

типів даних **UUID** для захисту посилань, та організація єдиного журналу змін **audit_log** із використанням типу даних **JSONB**. Такий вибір забезпечує максимальну транзакційну надійність, високу швидкість обробки даних на комп'ютерах користувачів та гнучкість при веденні історії хвороби.

Таблиця 1.1 – Порівняння варіантів технічної реалізації сховища

Критерій порівняння	**Варіант 1** (NoSQL + Логування в коді)	**Варіант 2** (Реляційна БД + Окремі іст. таблиці)	**Варіант 3** (PostgreSQL + пул з'єднань + JSONB лог)
Транзакційна надійність	низька	висока	**Найвища (ACID)**
Гнучкість структури логу	висока	низьке	**висока**
Навантаження на сервер БД	низьке	високе	**Оптимальне (завдяки пулу)**
Складність підтримки схеми	мінімальна	дуже висока	**Середня (найбільш збалансована)**

1.4 Міжнародні стандарти надійності та безпеки медичних інформаційних систем

При проєктуванні та розробці медичних інформаційних систем критично важливим є дотримання міжнародних стандартів у сфері якості, надійності та безпеки програмного забезпечення [12]. Основними орієнтирами у цьому напрямку є сімейство стандартів ISO.

Зокрема, стандарт **ISO/IEC 25010** (Системна та програмна інженерія – Вимоги до якості систем і програмного забезпечення та їх оцінка) визначає модель якості програмних продуктів, розподіляючи її на такі характеристики, як надійність (reliability), захищеність (security), функціональна придатність та

сумісність [13]. У контексті збереження медичних даних надійність передбачає відмовостійкість СКБД та здатність системи відновлювати свій працездатний стан без втрати критичної інформації.

Питання захисту конфіденційних даних пацієнтів та забезпечення цілісності інформації регламентуються фундаментальним стандартом інформаційної безпеки **ISO/IEC 27001**, а також його спеціалізованим розширенням для галузі охорони здоров'я – **ISO 27799** (Медична інформатика – Управління безпекою інформації в охороні здоров'я на основі ISO/IEC 27002). Ці стандарти висувають жорсткі вимоги до контролю доступу, автентифікації користувачів, а також обов'язкового ведення детальних журналів аудиту (audit logs). Механізм логування історії змін, що розглядається в межах даного дипломного проєкту, є базовою технічною вимогою зазначених стандартів, оскільки він забезпечує невідмовність (non-repudiation), дозволяє відстежити будь-які несанкціоновані модифікації медичних карток та гарантує повну прозорість життєвого циклу даних у високонадійних інформаційних системах.

1.5 Висновки до першого розділу

Аналіз медичної сфери підтвердив, що автоматизація управління цифровими записами потребує підвищеної відмовостійкості. Для вирішення цієї проблеми розроблено інформаційну систему «MedSystem» з архітектурою сховища даних на базі СКБД PostgreSQL. Використання формату JSONB для логування транзакцій забезпечує строгий контроль цілісності даних та ведення історії змін. Сформульовані технічні завдання гарантують високу надійність системи та її повну відповідність міжнародним стандартам безпеки ISO/IEC 25010 і ISO 27799.

2 ПРОЄКТУВАННЯ СТРУКТУРИ ТА АРХІТЕКТУРИ МЕДИЧНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Опис діаграм прецедентів та рольової моделі доступу

На етапі проєктування архітектури медичної інформаційної системи «MedSystem» першочерговою задачею є побудова концептуальної моделі її функціонування.

Згідно проєкту, у системі повністю відсутня роль адміністратора, а вся бізнес-логіка розподілена між двома головними акторами: лікарем та пацієнтом. На основі цього було побудовано рольову модель, де кожен актор має строго обмежений набір прецедентів (див. рис. 2.1).

Актор у ролі «Пацієнт» взаємодіє з системою через такі основні прецеденти:

- реєстрація та авторизація: створення облікового запису та отримання захищеного токена доступу;
- заповнення персональної картки: внесення та редагування особистих даних (телефон, адреса, група крові, алергії та хронічні захворювання);
- запис на прийом: перегляд доступних робочих днів лікаря та бронювання конкретного часового слота;
- перегляд медичної картки: ознайомлення зі своїми діагнозами, симптомами та рекомендаціями щодо лікування, які вніс спеціаліст;
- отримання сповіщень: перегляд повідомлень про нові, скасовані чи перенесені візити.

Актор у ролі «Лікар» має ширші повноваження, орієнтовані на клінічну роботу:

- керування графіком роботи: налаштування активних днів тижня, часу початку та завершення зміни, а також тривалості одного прийому;
- перегляд профілів пацієнтів: доступ до інформації про хворих, які закріплені за ним або записані до нього на консультацію;
- проведення прийому: зміна статусів записів (з запланованого на підтверджений або завершений) та фіксація результатів огляду;
- ведення медичних записів: внесення скарг, діагнозів та схем лікування до електронної картки пацієнта;
- робота з персональними нотатками: збереження та оновлення особистих робочих записів на робочому столі (дашборді).

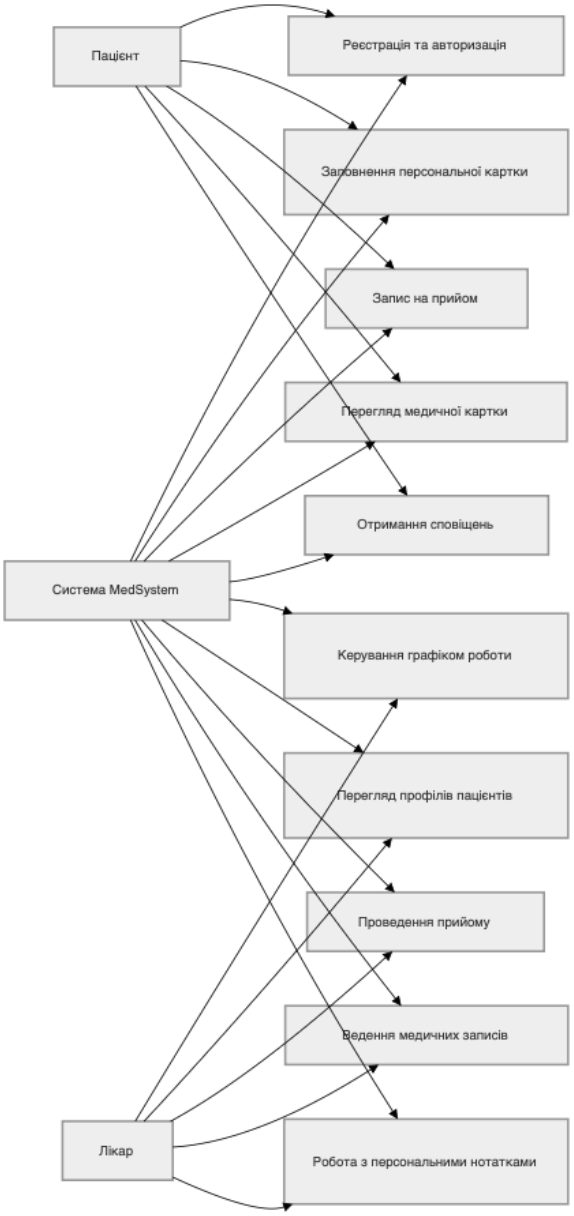


Рисунок 2.1 – Діаграма прецедентів

Всі ці варіанти використання безпосередньо впливають на структуру сховища даних. Кожна дія актора ініціює відповідні SQL-запити до таблиць СКБД PostgreSQL, а механізм безпеки на базі ролей контролює, щоб жоден прецедент пацієнта не міг отримати доступ до закритих функцій лікаря.

2.2 Розробка структурної схеми медичної системи та архітектури бази даних

Для деталізації взаємодії внутрішніх модулів системи «MedSystem» було розроблено її структурну схему. Вона базується на трьохланковій клієнт-серверній архітектурі, де кожен рівень виконує свої чітко визначені функції. Основними структурними блоками розробленої системи є:

- **графічний інтерфейс користувача (Frontend):** реалізований на базі бібліотеки React.js. Він відповідає за візуалізацію даних на персональних комп'ютерах користувачів, збір інформації через форми реєстрації/запису та передачу HTTP-запитів на сервер. Для збереження сесії авторизованого користувача та його ролі використовується глобальне управління станом через React Context API;
- **сервер додатків та безпеки (Backend):** побудований на платформі Node.js із використанням фреймворку Express.js. Цей блок обробляє запити від клієнта, реалізує бізнес-логіку програми та забезпечує захист даних. У ньому міститься модуль безпеки auth.js, який перевіряє JWT-токени користувачів та розмежовує права доступу. Також тут працює утиліта контролю доступу assertDoctorPatient, яка на рівні логіки сервера перевіряє, чи має право конкретний лікар переглядати дані пацієнта (див. додаток А, рис. А.5);
- **блок бази даних (Database):** ядром системи є СКБД PostgreSQL (р. 18). Зв'язок із сервером забезпечує модуль pool.js, що керує пулом з'єднань pg.Pool (див. додаток А, рис. А.2). Фізична структура сховища складається з 9 взаємопов'язаних таблиць, які відображають усі сутності системи.

Архітектура бази даних розроблена таким чином, що поточні бізнес-дані повністю відділені від історичних. Облікові записи користувачів зберігаються в users, а їхні розширені профілі – у таблицях patients та doctors. Зв'язок між ними реалізовано через зовнішні ключі (REFERENCES) з правилом каскадного видалення ON DELETE CASCADE. Розклад та записи на прийом фіксуються в doctor_schedules та appointments, а результати лікування – в medical_records. При

цьому для реалізації високої надійності та ведення історії змін спроєктовано ізольований блок логування подій – таблицю audit_log, куди автоматично записуються всі операції модифікації даних у форматі JSONB. Маршрутизацію операцій та кінцеві точки API для медичних карток наведено у додатку А, рис. А.3..

2.3 Технічні вимоги до апаратного та програмного забезпечення системи

Для забезпечення стабільної та відмовостійкої роботи медичної інформаційної системи «MedSystem» на персональних комп'ютерах закладу охорони здоров'я було сформульовано мінімальні та рекомендовані вимоги до технічного комплексу (див. табл. 2.2). Оскільки система використовує легковажний та асинхронний стек технологій (Node.js та React.js), вона не потребує дорогого серверного обладнання. Для клієнтських робочих місць лікарів достатньо базових ПК із підтримкою сучасних веббраузерів, що мінімізує витрати на модернізацію інфраструктури установи. Основне обчислювальне навантаження з обслуговування бази даних PostgreSQL та пулу з'єднань покладається на сервер, параметри якого масштабуються відповідно до обсягу медичних карток. Такий підхід гарантує високу швидкість обробки транзакцій та стабільний доступ до журналу аудиту навіть на техніці минулих поколінь.

Таблиця 2.2 – Мінімальні вимоги до апаратного забезпечення системи «MedSystem»

Найменування компонента	**Мінімальні параметри**	**Рекомендовані параметри**
Центральний процесор (CPU)	Intel Core i3 / AMD Ryzen 3 (2.0 ГГц)	Intel Core i5 / AMD Ryzen 5 (3.0 ГГц)
Оперативна пам'ять (RAM)	Не менше 4 Гб	8 Гб або більше

Накопичувач (HDD/SSD)	5 Гб вільного місця	20 Гб вільного місця (SSD)
Мережевий адаптер	100 Мбіт/с	1000 Мбіт/с (Гігабітний канал)
Периферійні пристрої	Клавіатура, миша, монітор (1280x1024)	Клавіатура, миша, монітор Full HD

Вимоги до програмного забезпечення та мережевих каналів зв'язку системи:

- **операційна система:** Windows 10/11, macOS або Linux (Ubuntu Server для розгортання бази даних та бекенду).
- **система керування базами даних:** СКБД PostgreSQL версії 13 або вище з активованим системним розширенням uuid-ossp для генерації унікальних ідентифікаторів.
- **середовище виконання сервера:** Node.js (LTS версія 16+) та пакетний менеджер npm.
- **пропускна здатність мережі:** Для клієнтських комп'ютерів – від 1 Мбіт/с; для сервера бази даних та сервера додатків – від 100 до 1000 Мбіт/с для швидкої обробки транзакцій та передачі великих обсягів медичних карток.

2.4 Опис логіки взаємодії компонентів та діаграми послідовності аудиту змін

Для демонстрації динаміки роботи системи та відстеження часових процесів було спроєктовано діаграму послідовності (sequence diagram) (див. рис. 2.3). Вона наочно відображає, як клієнтська частина, сервер Express.js та база даних PostgreSQL взаємодіють між собою під час виконання критично важливих операцій, а також як саме реалізовано підтримку історії змін.

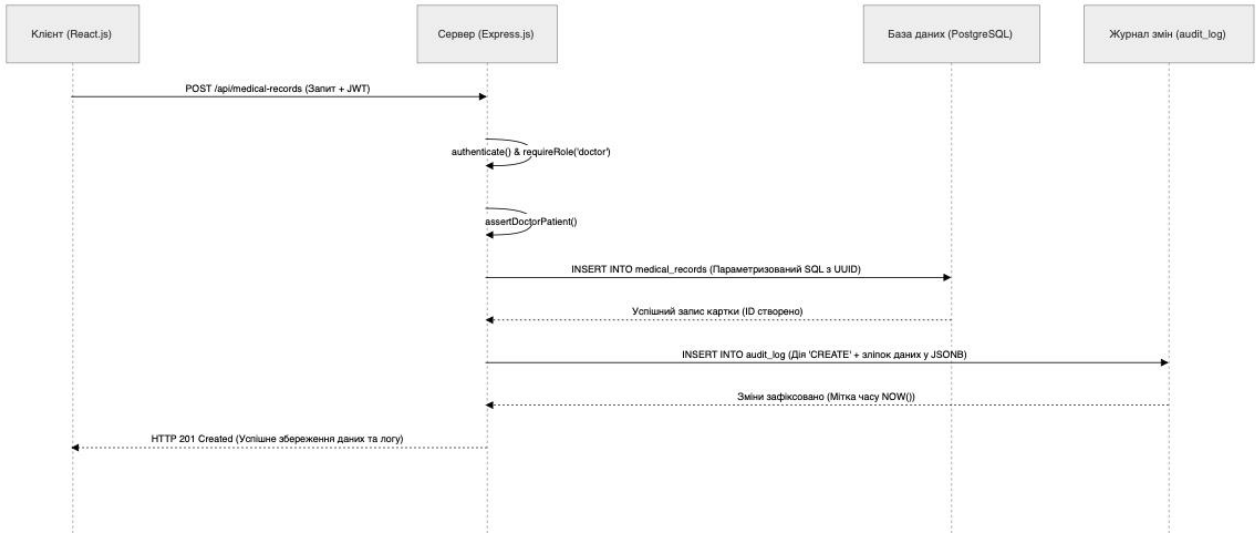


Рисунок 2.3 – Діаграма послідовності процесу внесення медичних записів та фіксації історії змін

Розглянемо логіку взаємодії компонентів на прикладі процесу створення лікарем нового запису в медичній картці пацієнта (medical_records):

- **ініціалізація запиту:** Лікар заповнює клієнтську форму (вводить симптоми, діагноз та лікування) і натискає кнопку «Зберегти». React-додаток формує HTTP POST-запит, додає у заголовки Authorization токен JWT і відправляє його на сервер Node.js.
- **перевірка безпеки (Middleware):** Сервер приймає запит і передає його у модуль auth.js. Функція authenticate перевіряє валідність JWT-токена за допомогою секретного ключа JWT_SECRET. Якщо токен правильний, з нього вилучаються дані користувача (його ID та роль). Далі функція requireRoleперевіряє, чи дійсно цей користувач є лікарем. Якщо перевірку пройдено, викликається функція assertDoctorPatient, яка робить швидкий запит до таблиці doctor_patients і підтверджує, що цей лікар дійсно має право вносити записи даному пацієнту.

- **робота з пулом з’єднань:** Сервер додатків звертається до модуля pool.js. Механізм pg.Poolмиттєво виділяє одне з вільних активних з’єднань із базою даних PostgreSQL для виконання транзакції.
- **запис поточних даних:** Через виділене з’єднання сервер виконує параметризований SQL-запит INSERT INTO medical_records .... СКБД PostgreSQL генерує криптографічно стійкий PRIMARY KEY у форматі UUID за допомогою розширення uuid-ossp, перевіряє обмеження цілісності та записує дані в таблицю.
- **фіксація історії змін (Аудит):** Одразу після успішного додавання медичного запису, сервер (або внутрішній тригерний механізм) формує другий SQL-запит до таблиці audit_log. У цей журнал передається user_id лікаря, тип дії (CREATE), назва зміненої таблиці (medical_records), ID нового запису, а також повний зліпок внесених даних (симптоми, діагноз, рекомендації), який упаковується в один рядок завдяки гнучкому типу даних JSONB. База даних фіксує поточний час через функцію NOW() і зберігає історичний лог.
- **відповідь системи:** Після успішного завершення обох записів СКБД повертає серверу статус виконання, пул pg.Pool маркує з’єднання як вільне для наступних завдань, а сервер Express.js відправляє на фронтенд підтвердження у форматі JSON. Інтерфейс лікаря оновлюється, відображаючи успішне збереження.

Така послідовність дій гарантує, що жодне оновлення чи видалення медичних даних не пройде повз систему аудиту, забезпечуючи стовідсоткову надійність та хронологічну точність ведення медичних документів.

2.5 Висновки до другого розділу

У другому розділі розроблено діаграму прецедентів та рольову модель для системи «MedSystem», що чітко розмежувало права «Лікаря» та «Пацієнта» при роботі з медичними даними. Спроєктовано триланкову клієнт-серверну архітектуру (React.js, Node.js/Express.js, PostgreSQL) із розділенням реляційних таблиць та ізольованого журналу аудиту. Також визначено апаратні й програмні вимоги для стабільної роботи бази даних та серверного середовища. Завдяки діаграмі послідовності деталізовано наскрізний процес обробки запитів: від JWT-валідації та перевірки зв’язків через assertDoctorPatient до транзакційного запису даних і логування подій у форматі JSON.

3 АЛГОРИТМІЧНО-ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ «MEDSYSTEM»

3.1 Розробка алгоритмічного забезпечення системи та логіки аудиту подій

Алгоритмічна структура медичної інформаційної системи розроблена за принципом асинхронної обробки подій та суворого контролю транзакцій на кожному етапі взаємодії користувача зі сховищем даних. Головною метою побудови алгоритмів є забезпечення транзакційної надійності, валідації прав доступу на кожному етапі та автоматичної фіксації повної історії змін безпосередньо у процесі модифікації інформації.

Загальний алгоритм функціонування системи починається з етапу ідентифікації та автентифікації користувача. При спробі входу до системи клієнтське середовище збирає введені облікові дані (электронну пошту та пароль) і передає їх на сервер додатків. Алгоритм обробки авторизації на бекенді виконує пошук відповідного запису в таблиці користувачів. Після знаходження користувача запускається криптографічна перевірка відповідності пароля за допомогою алгоритму bcrypt. У разі успішного збігу сервер генерує унікальний сесійний токен JWT, у тіло якого зашивається ідентифікатор користувача та його роль (лікар або пацієнт). Цей токен повертається на клієнт і зберігається в глобальному стані додатка через React Context API, стаючи обов’язковим ключем для всіх подальших запитів.

Особливе місце в алгоритмічному забезпеченні займає логіка виконання критично важливих операцій – створення медичної картки або зміна статусу запису на прийом. Блок-схема цього процесу включає такі послідовні кроки:

1) **прийом HTTP-запиту:** Клієнтська частина на React.js відправляє дані на сервер Express.js.

2) **перевірка автентифікації:** Перевіряється наявність та валідність токена в Express-middleware. Якщо токен прострочений, алгоритм повертає помилку.
3) **рольова валідація:** Функція перевірки ролі визначає, чи має користувач право виконувати цю дію (наприклад, пацієнту заблоковано доступ до редагування медичних карток).
4) **контроль доступу на рівні зв'язків:** Запускається допоміжний алгоритм, який перевіряє таблицю зв'язків лікарів та пацієнтів. Лікар отримує допуск до картки лише тоді, коли цей пацієнт офіційно закріплений за ним.
5) **транзакційний запис у БД:** Через пул з'єднань до СКБД PostgreSQL відправляється параметризований SQL-запит.
6) **системний аудит (Історія змін):** Після успішного виконання основного запиту алгоритм ініціює обов'язковий запис у таблицю журналу змін, де зберігає зліпок операції у форматі JSONB та мітку часу.

Така логічна послідовність алгоритмів унеможливлює виконання будь-яких операцій в обхід системи безпеки та гарантує, що кожна дія в базі даних буде зафіксована у хронологічному журналі історії виправлень. Програмну реалізацію контролера з логікою транзакцій ACID та записом до аудит-логу наведено у додатку А, рис. А.4.

3.2 Програмна реалізація структури бази даних та опис таблиць

Для фізичної реалізації сховища медичної системи було обрано об'єктно-реляційну СКБД PostgreSQL, яка забезпечує повну відповідність вимогам ACID та стабільну роботу під високим паралельним навантаженням. Зв'язок сервера з базою даних реалізовано за допомогою пулу постійних з'єднань, що дозволяє раціонально використовувати ресурси персонального комп'ютера.

3.2.1 Розробка ER-діаграми бази даних

На рисунку 3.1 представлено ER-діаграму (схему зв’язків сутностей) розробленої бази даних, що відображає архітектуру збереження інформації, логічні зв’язки між таблицями, а також розподіл потоків даних всередині медичної інформаційної системи.

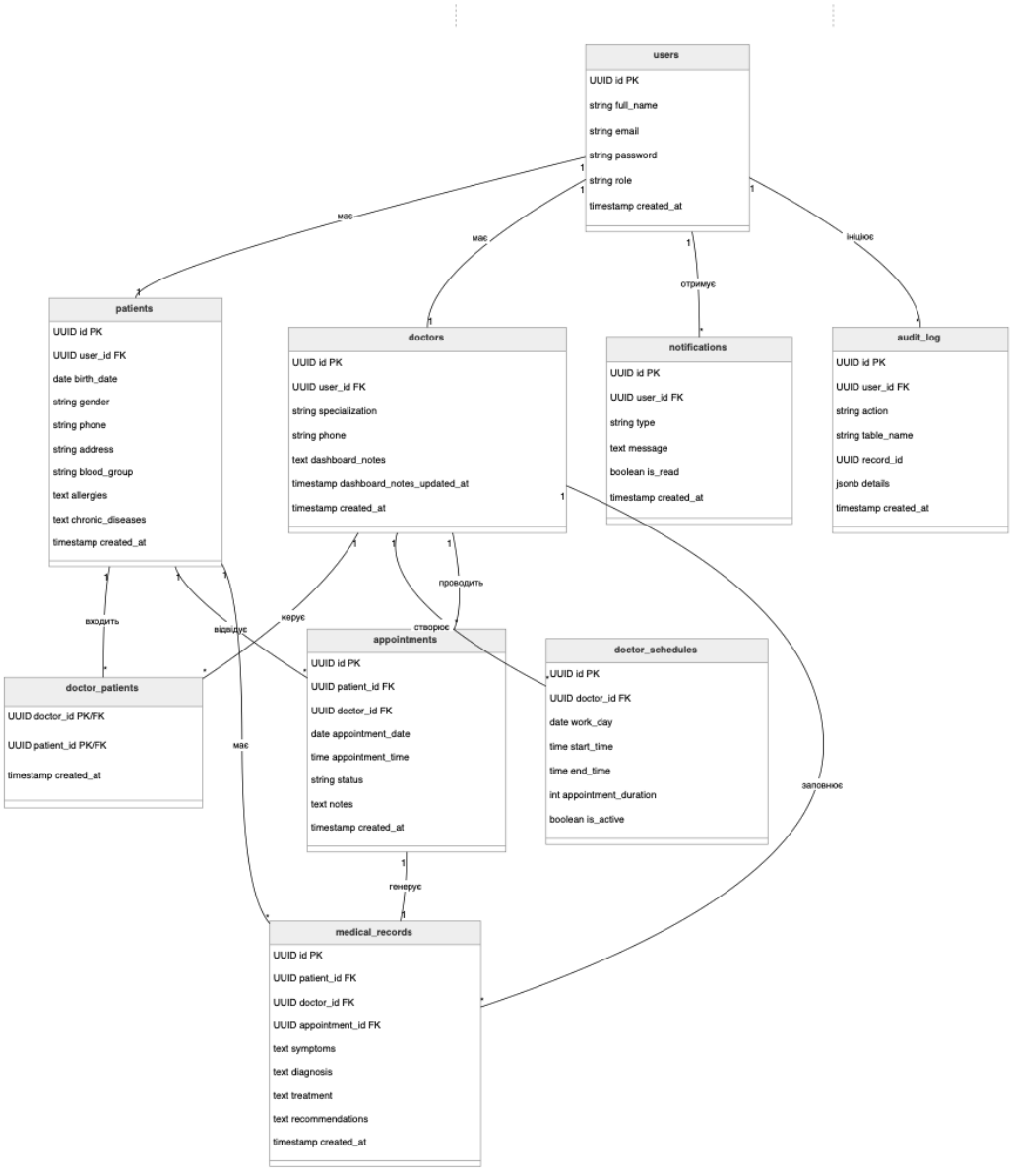


Рисунок 3.1 – ER-діаграма бази даних системи MedSystem

3.2.2 Опис структури та характеристика таблиць сховища даних

Спроєктована база даних містить 9 взаємопов’язаних таблиць (див. табл. 3.2, 3.3, 3.4, 3.5). Замість класичних числових лічильників (INT) для первинних ключів використано зашифровані ідентифікатори UUID, що підвищує надійність системи та унеможливлює перебір записів зловмисниками. Повний SQL-скрипт розгортання структури бази даних та журналу аудиту міститься у додатку А, рис. А.1.

Таблиця 3.2 – Структура таблиці користувачів users

Назва поля	Тип даних	Обмеження / Ключі	Опис поля
id	UUID	PRIMARY KEY, default v4	Унікальний ідентифікатор користувача
full_name	VARCHAR(255)	NOT NULL	Прізвище, ім'я та по батькові
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта для логіну
password	VARCHAR(255)	NOT NULL	Зашифрований bcrypt-хеш пароля
role	VARCHAR(20)	CHECK (in 'doctor', 'patient')	Рольовий статус користувача
created_at	TIMESTAMP	DEFAULT NOW()	Дата та час створення акаунту

Таблиця 3.3 – Структура таблиці пацієнтів patients

Назва поля	Тип даних	Обмеження / Ключі	Опис поля
id	UUID	PRIMARY KEY	Ідентифікатор картки пацієнта

user_id	UUID	FOREIGN KEY REFERENCES users(id)	Зв'язок із базовим акаунтом
birth_date	DATE	-	Дата народження пацієнта
gender	VARCHAR(10)	CHECK (in 'male', 'female')	Стать пацієнта
phone	VARCHAR(30)	-	Контактний номер телефону
blood_group	VARCHAR(40)	-	Група крові та резус-фактор
allergies	TEXT	-	Алергічні реакції пацієнта
chronic_diseases	TEXT	-	Хронічні захворювання

Таблиця 3.4 – Структура таблиці журналу змін audit_log

Назва поля	**Тип даних**	**Обмеження / Ключі**	**Опис поля**
id	UUID	PRIMARY KEY	Ідентифікатор запису аудиту
user_id	UUID	FOREIGN KEY REFERENCES users(id)	Автор виправлення/зміни
action	VARCHAR(100)	'CREATE', 'UPDATE', 'DELETE'	Тип транзакційної операції
table_name	VARCHAR(100)	NOT NULL	Назва модифікованої таблиці
record_id	UUID	-	ID рядка, який піддався зміні

details	JSONB	-	Історичний зліпок полів у JSON
created_at	TIMESTAMP	DEFAULT NOW()	Системний час фіксації зміни

Таблиця 3.5 – Структура таблиці медичних записів medical_records

Назва поля	Тип даних	Обмеження / Ключі	Опис поля
id	UUID	PRIMARY KEY	Ідентифікатор медичного запису
patient_id	UUID	FOREIGN KEY REFERENCES patients(id)	Пацієнт, якого оглядали
doctor_id	UUID	FOREIGN KEY REFERENCES doctors(id)	Лікар, який проводив огляд
appointment_id	UUID	REFERENCES appointments(id)	Посилання на прийом у розкладі
symptoms	TEXT	-	Скарги пацієнта та симптоми
diagnosis	VARCHAR(500)	-	Встановлений медичний діагноз
treatment	TEXT	-	Призначена схема лікування

Цілісність даних на рівні ядра СКБД реалізовано за допомогою каскадних правил безпеки. При видаленні облікового запису користувача з таблиці користувачів, його профіль у таблицях пацієнтів або лікарів видаляється автоматично завдяки правилу ON DELETE CASCADE. Водночас для медичних

карток налаштовано обмеження ON DELETE SET NULL. Це гарантує високу надійність зберігання архівів: навіть якщо сам запис прийому буде видалено з розкладу, медична картка пацієнта з його діагнозом та історією лікування залишиться недоторканою.

3.3 Програмна реалізація інтерфейсу користувача та взаємодія з базою даних

Після завершення розробки структури бази даних та програмної логіки було реалізовано користувацький інтерфейс медичної інформаційної системи. Інтерфейс вебзастосунку розроблено з урахуванням принципів зручності використання, швидкого доступу до основних функцій та забезпечення комфортної взаємодії користувачів із системою.

Система підтримує дві основні ролі користувачів: пацієнта та лікаря. Для кожної ролі передбачено окремий набір функціональних можливостей і відповідний особистий кабінет. Інтерфейс забезпечує виконання основних операцій, пов’язаних із реєстрацією користувачів, записом на прийом, переглядом медичних даних, веденням електронних медичних карток та управлінням графіком роботи лікарів.

Нижче наведено опис основних сторінок та елементів інтерфейсу розробленої медичної інформаційної системи.

3.3.1 Вхід до системи

Перше, що бачить відвідувач сайту – це текстові поля для введення логіну та паролю і кнопка ‘Увійти’ (див. рис. 3.7). Якщо логін та пароль введено правильно, то програма переходить на наступну сторінку.

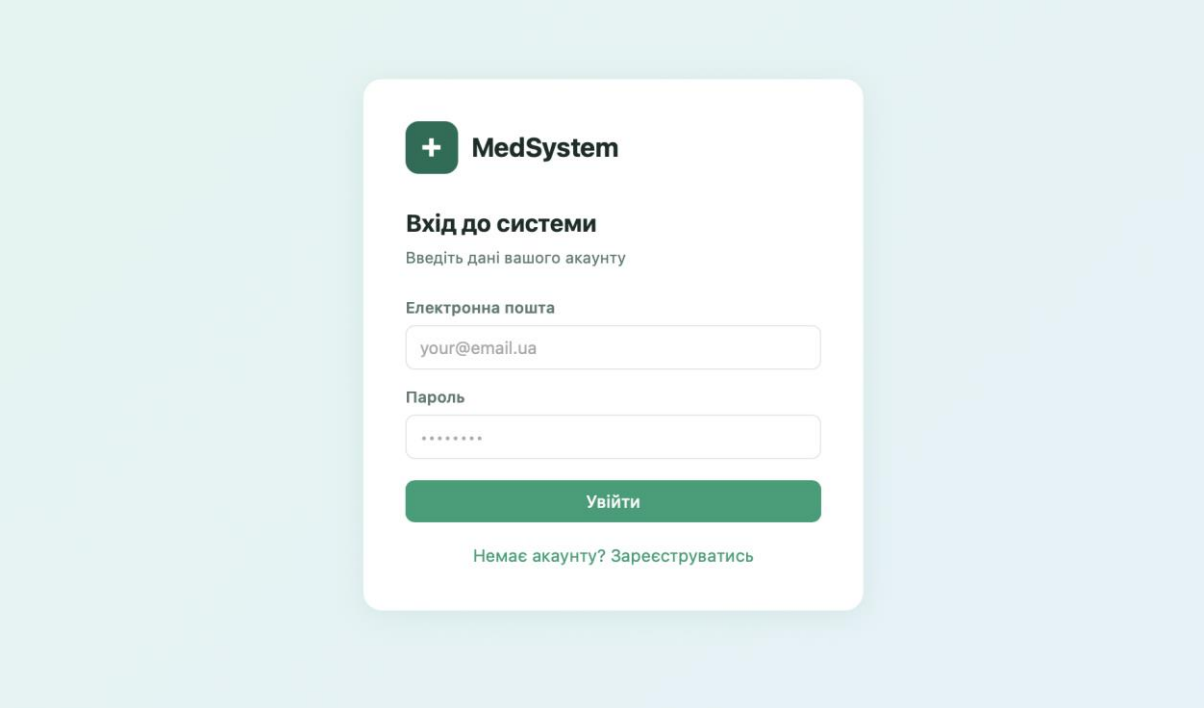


Рисунок 3.7 – Вікно входу до системи

Якщо пароль або логін не співпадає, то видається помилка.

Якщо відвідувач ще немає свого особистого облікового запису, то він може зареєструватися скориставшись кнопкою ‘Зареєструватись’ (див. рис. 3.8).

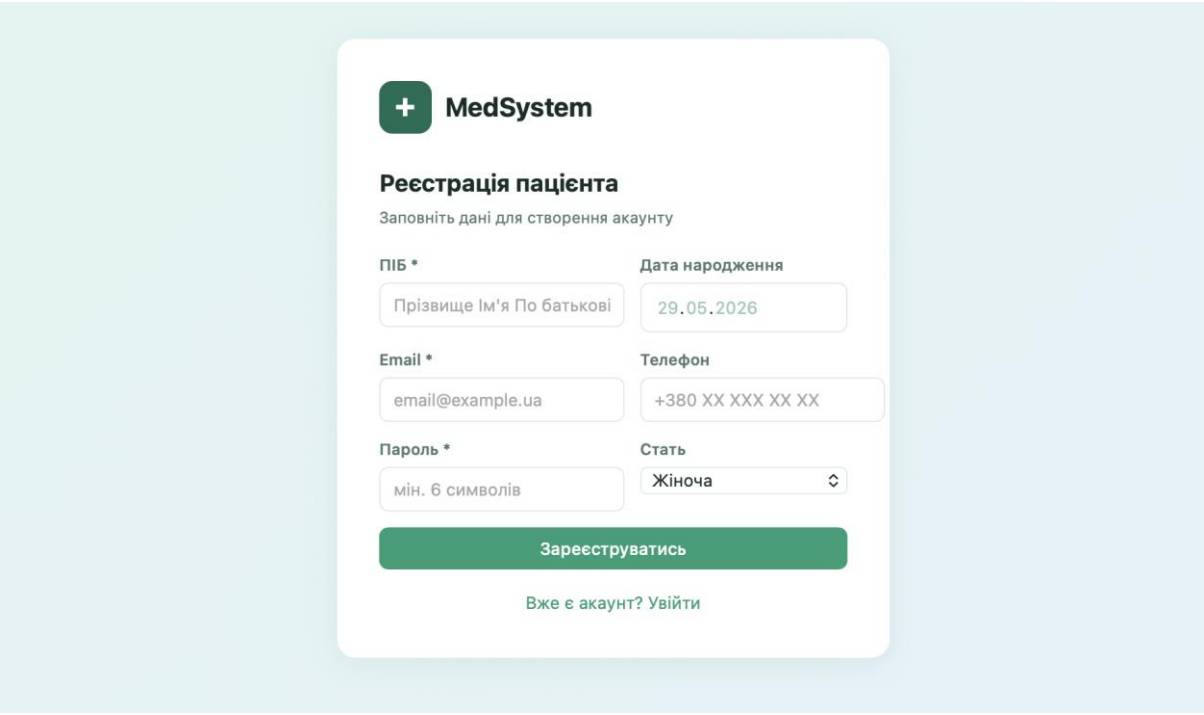


Рисунок 3.8 – Вікно реєстрації пацієнта

Тут необхідно ввести особисті дані для ідентифікації особи, а саме: ПІБ, дату народження, електронна пошта, номер телефону, пароль від аккаунта (бачить та знає лише пацієнт), стать. Фрагмент програмного коду компонента форми реєстрації представлено у додатку А, рис. А.6.

Пункти є обов’язковими до заповнення!

Знизу є кнопка для повернення до попереднього меню на випадок помилкового натискання кнопки реєстрації.

Логіни поділяються на 2 типа (лікар та пацієнт).

3.3.2 Пацієнт

Після вдалої реєстрації або входу пацієнт потрапляє до особистого кабінету (див. рис. 3.9).

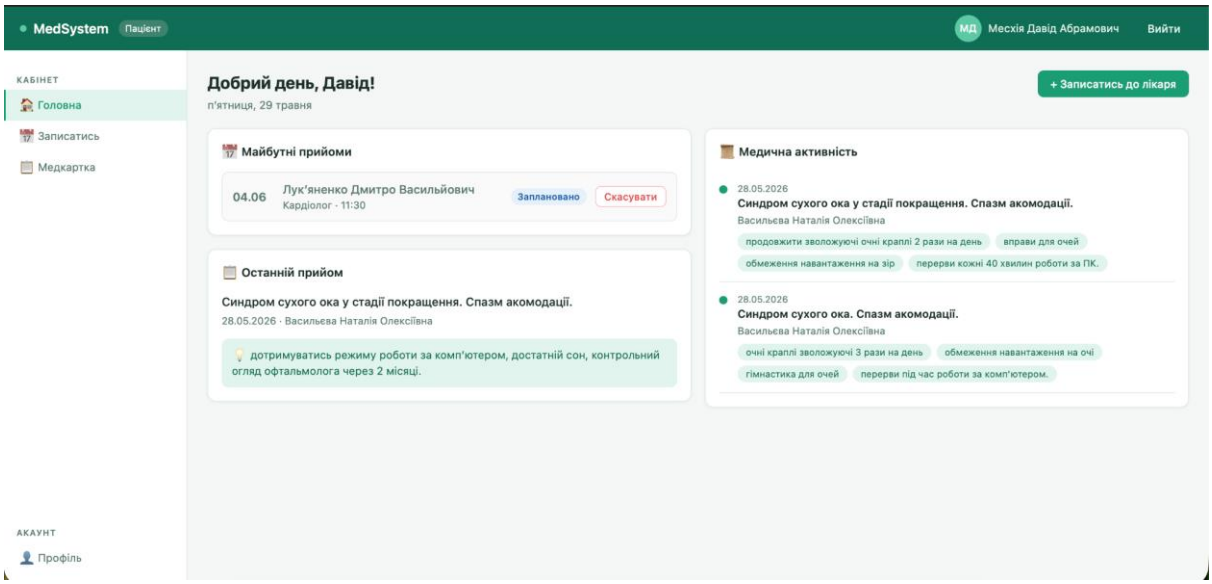


Рисунок 3.9 – Особистий кабінет пацієнта

На вкладці ‘Головна’ пацієнт має можливість переглянути:

- медичну активність – історію відвудувань;
- майбутні прийоми – дату, час та лікаря до кого запланований прийом (є можливість скасувати прийом та подивитись актуальний статус запису);

- останній прийом – інформація про прийом, що був останній раз та рекомендації лікаря.

В правому верхньому куті розташовано ім’я користувача та кнопку ‘Вийти’ яка переносить на вікно входу (див. рис. 3.7). Код реалізації головної сторінки пацієнта наведено у додатку А, рис. А.7.

Наступна вкладка ‘Записатись’ (див. рис. 3.10).

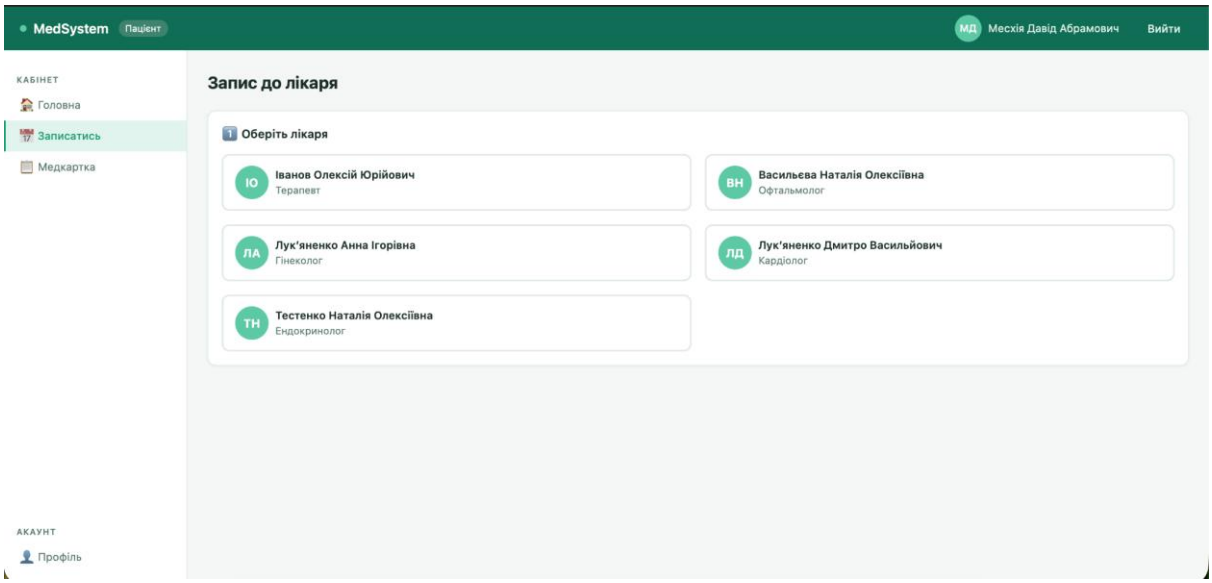


Рисунок 3.10 – Запис до лікаря

Тут маємо можливість обрати лікаря і обрати комфортний час для відвідування (див. рис. 3.11).

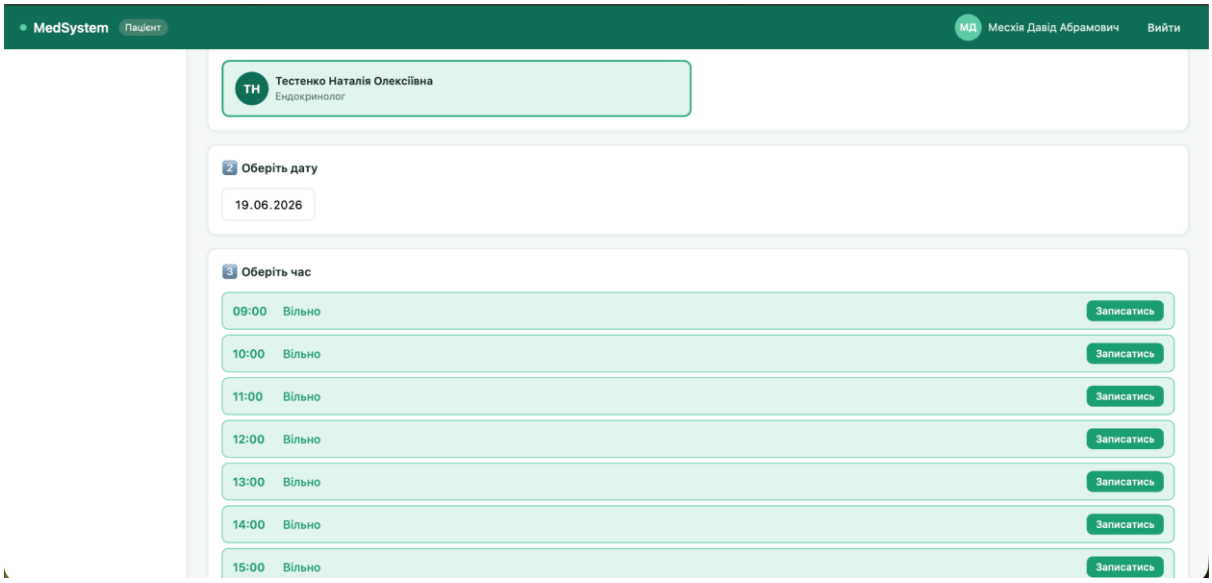


Рисунок 3.11 – Вікно вибору зручного часу для запису

Після успішного запису бачимо вікно, яке нам повідомляє про успішний запис пацієнта (див. рис. 3.12).

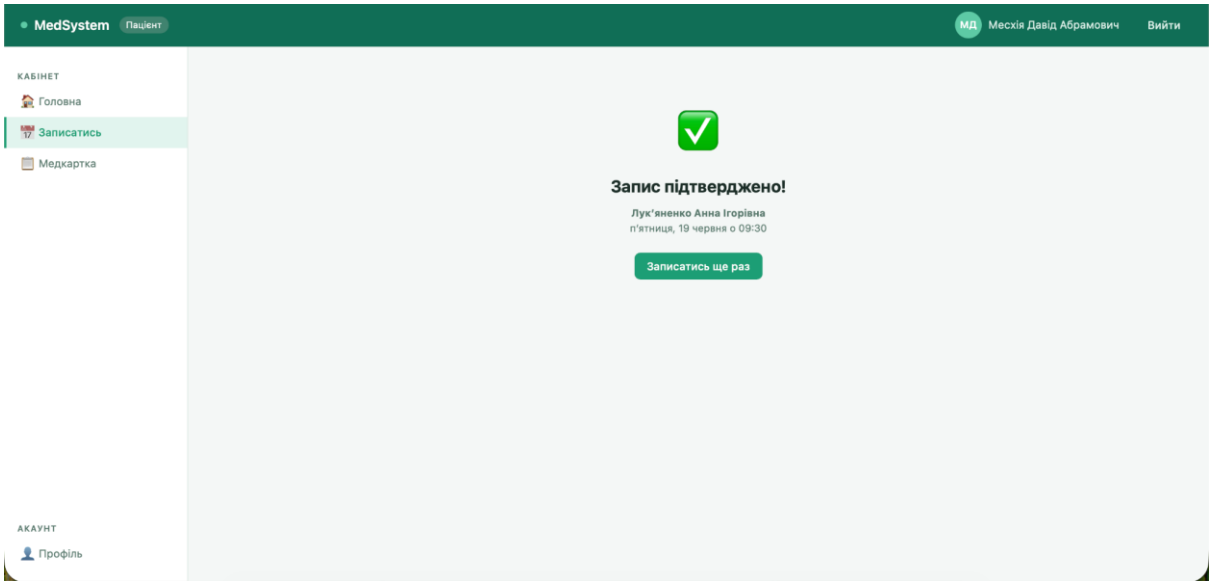


Рисунок 3.12 – Запис підтверджено

З цього вікна можна перейти на інші вкладки або натиснути ‘Записатись ще раз’. Програмний модуль покрокового інтерактивного запису пацієнта реалізовано у додатку А, рис. А.8.

Якщо лікар в обрану дату та час не приймає пацієнтів, то висвічується попередження (див. рис. 3.13) і пацієнту слід обрати інший день та/або час.

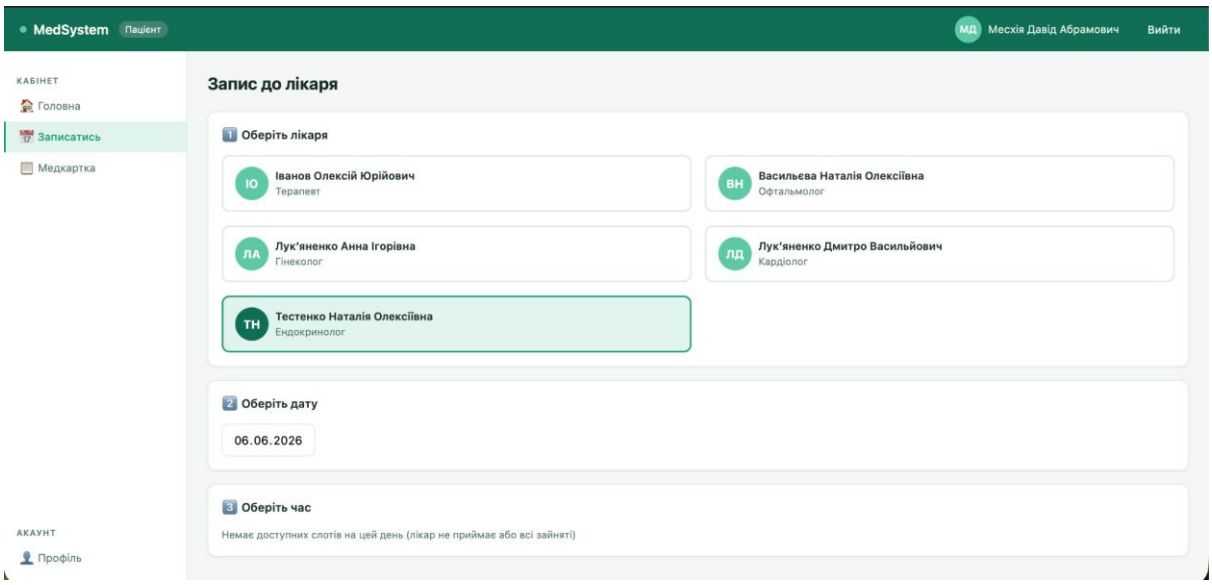


Рисунок 3.13 – Попередження ‘Немає вільних слотів’

В разі невірного запису або з якихось причин треба скасувати запис до лікаря, у пацієнта є кнопка ‘Скасувати запис’ (див. рис. 3.14) на головній сторінці в розділі ‘Майбутні прийоми’ (див. рис. 3.11).

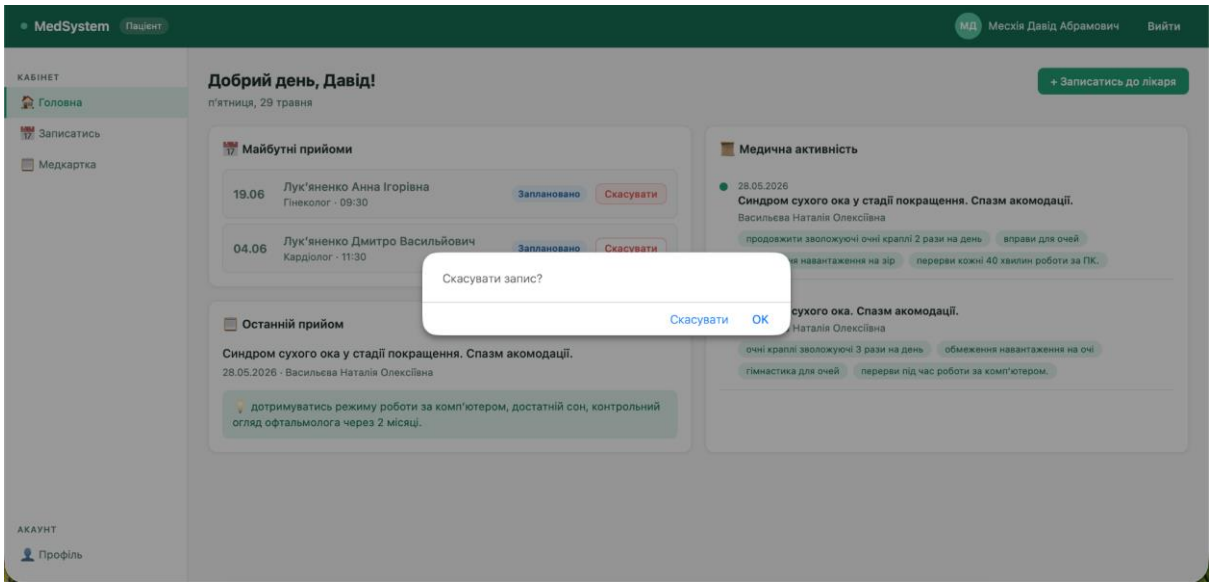


Рисунок 3.14 – Скасувати запис

Наступна вкладка ‘Медкарта’ в якій описано дані пацієнта та історію лікувань (див. рис. 3.15).

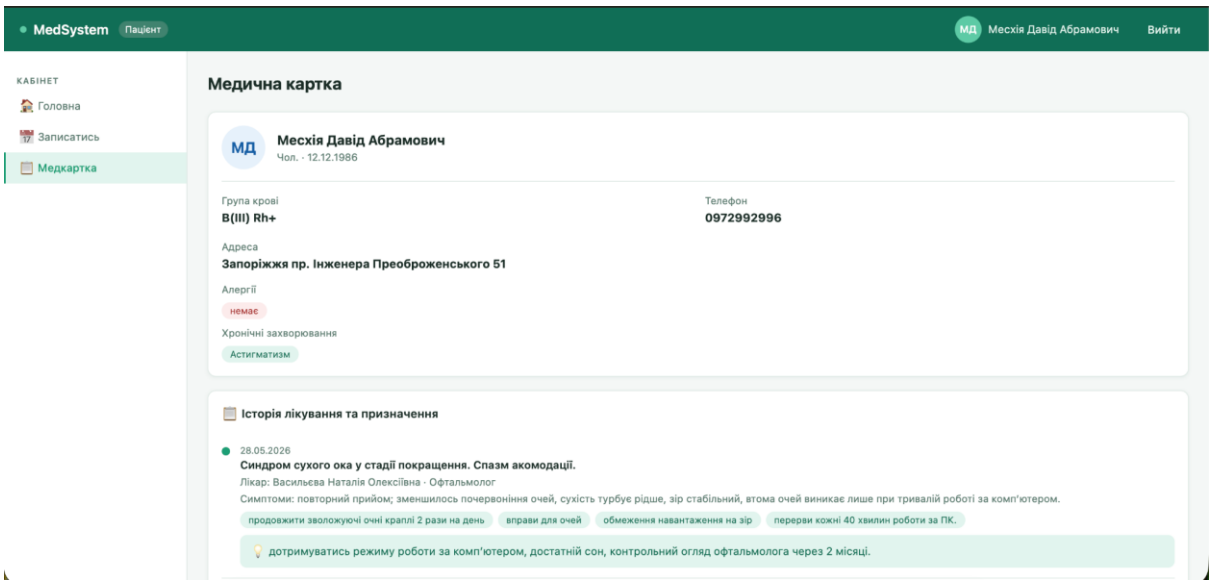


Рисунок 3.15 – Медична карта пацієнта

В лівому нижньому куті вкладка ‘Профіль’ (див. рис. 3.16) для редагування персональних даних.

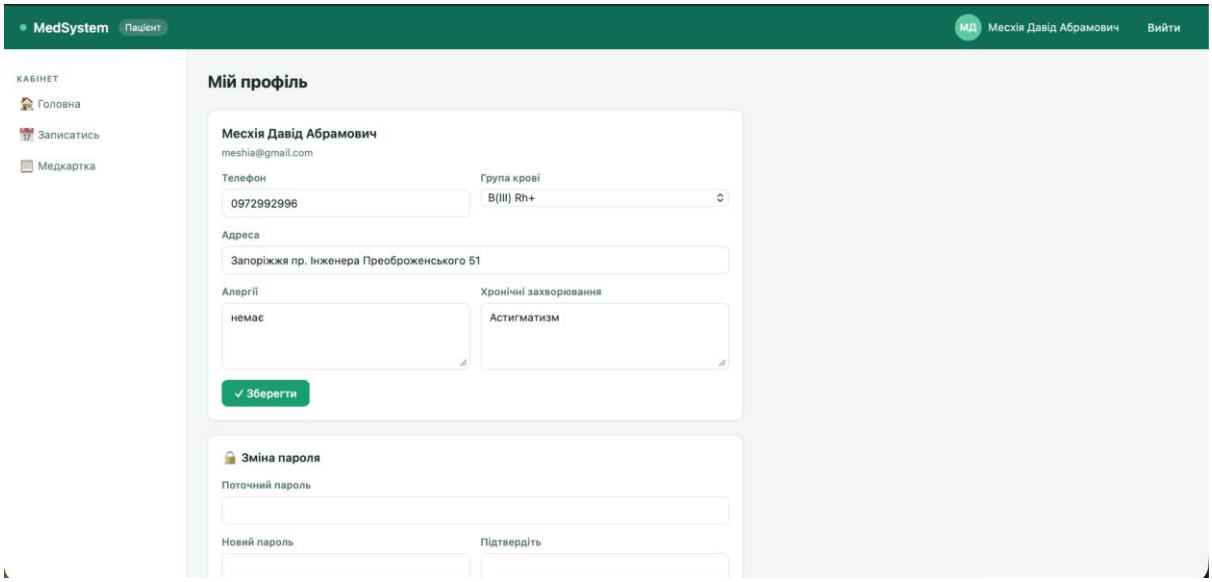


Рисунок 3.16 – Профіль пацієнта

Тут пацієнт сам може зазначити або відредагувати: номер телефону, групу крові, адресу проживання, алергії, хронічні захворювання.

Ця інформація зберігається в базі даних і буде доступна для перегляду лікарям.

Нижче (див. рис. 3.17) бачимо вікно для зміни паролю. Необхідно ввести старий пароль, а потім двічі новий і натиснути на кнопку ‘Змінити пароль’.

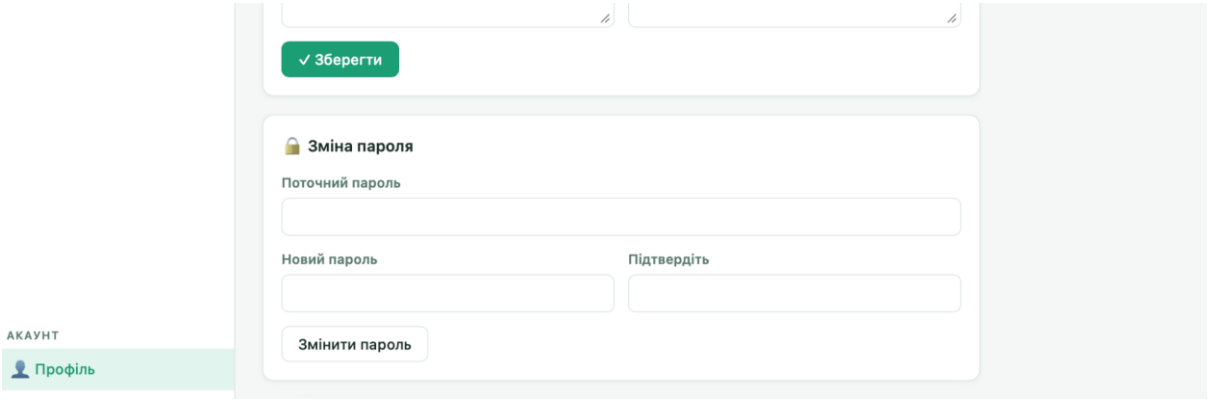


Рисунок 3.17 – Змінити пароль

3.3.3 Лікар

Лікар не може зареєструватись самостійно оскільки це працівник закритої системи клініки. Лікаря реєструє системний адміністратор і після чого необхідно змінити пароль через свій кабінеті.

Після вдалого входу лікар потрапляє до особистого кабінету (див. рис. 3.18).

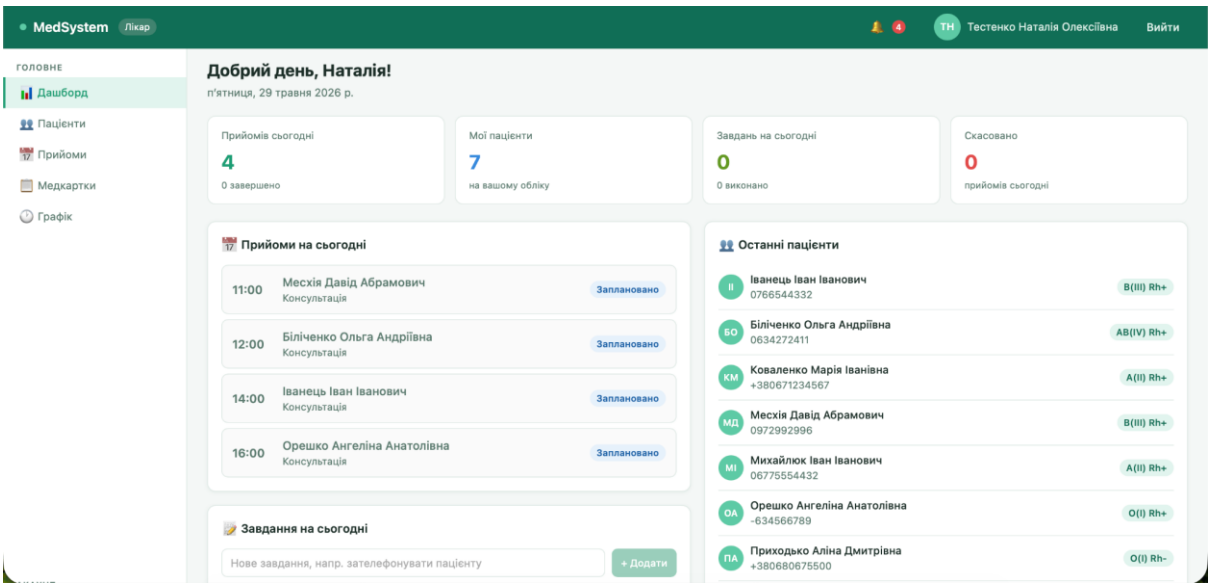


Рисунок 3.18 – Кабінет лікаря

Головна сторінка складається з наступних елементів:

- показники лікаря – верхній набір актуальної інформації про роботу.
- лікаря (інформація постійно оновлюється в реальному часі).
- прийом на сьогодні – список пацієнтів записаних на сьогодні (відображається: пацієнт, час, мета візиту, статус запису).
- завдання на сьогодні – список завдань, що лікар може сам собі ставити та відмічати, які завдання завершено (див. рис. 3.19).
- останні пацієнти – список пацієнтів, які записувались коли небудь на прийом.

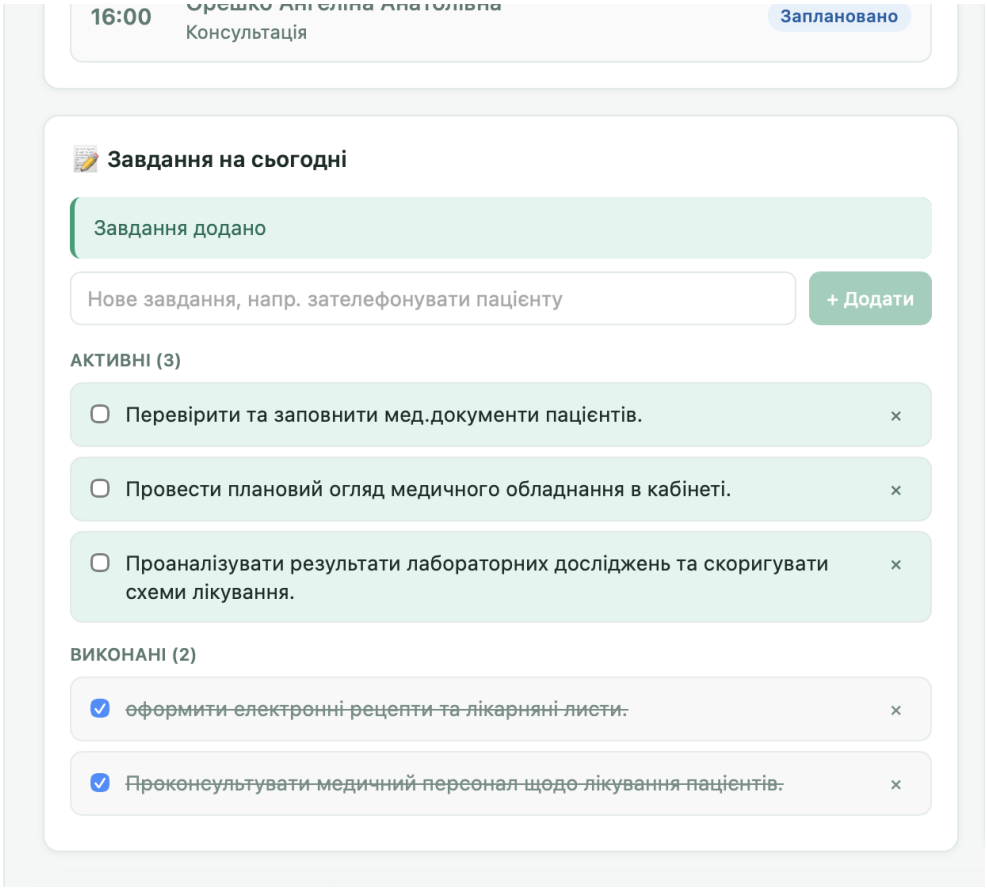


Рисунок 3.19 –Завдання на сьогодні

Тут у лікаря є можливість прописати особисті завдання – викреслювати, видаляти та редагувати.

У правому верхньому куті видно чий аккаунт зараз на екрані та кнопка ‘Вийти’, яка дозволяє покинути аккаунт.

Поруч кнопка ‘Дзвіночок’ – це панель повідомлень, де лікарю приходить повідомлення про якісь новини, наприклад: запис або скасування запису (див. рис. 3.20).

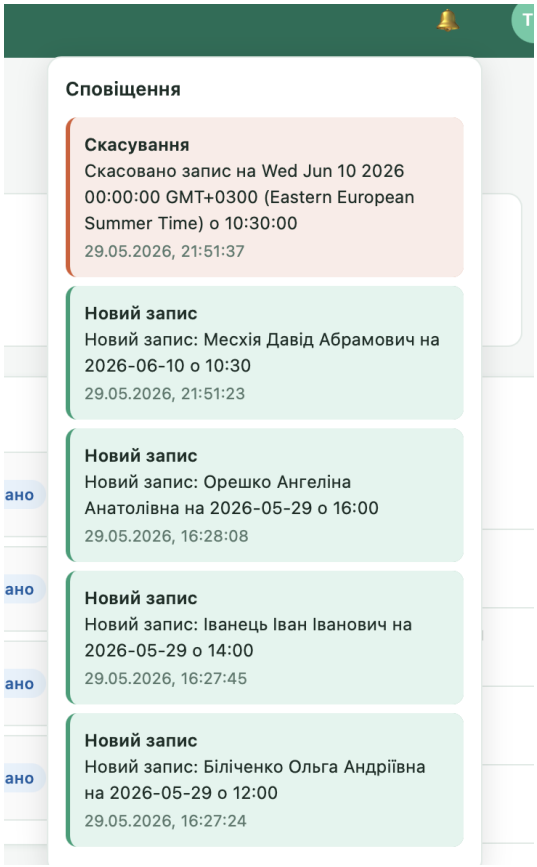


Рисунок 3.20 – Панель повідомлень

Також інші вкладки на лівій панелі:

1) дашборд – головна інформаційна сторінка (див. рис. 3.18).
2) пацієнти – список пацієнтів (див. рис. 3.21).
3) прийом – календар записів на цей тиждень (див. рис. 3.22).
4) медкарти – меню, де у лікаря є можливість знайти електронну медичну карту пацієнта (див.рис.3.29).
5) графік роботи – лікар вказує робочі години на тиждень (див.рис.3.33).
6) профіль – вкладка редагування профілю (див.рис.3.38).

Розглянемо детальніше кожен з пунктів.

Розділ ‘Пацієнти’ – це вкладка із списком пацієнтів (див. рис. 3.21).

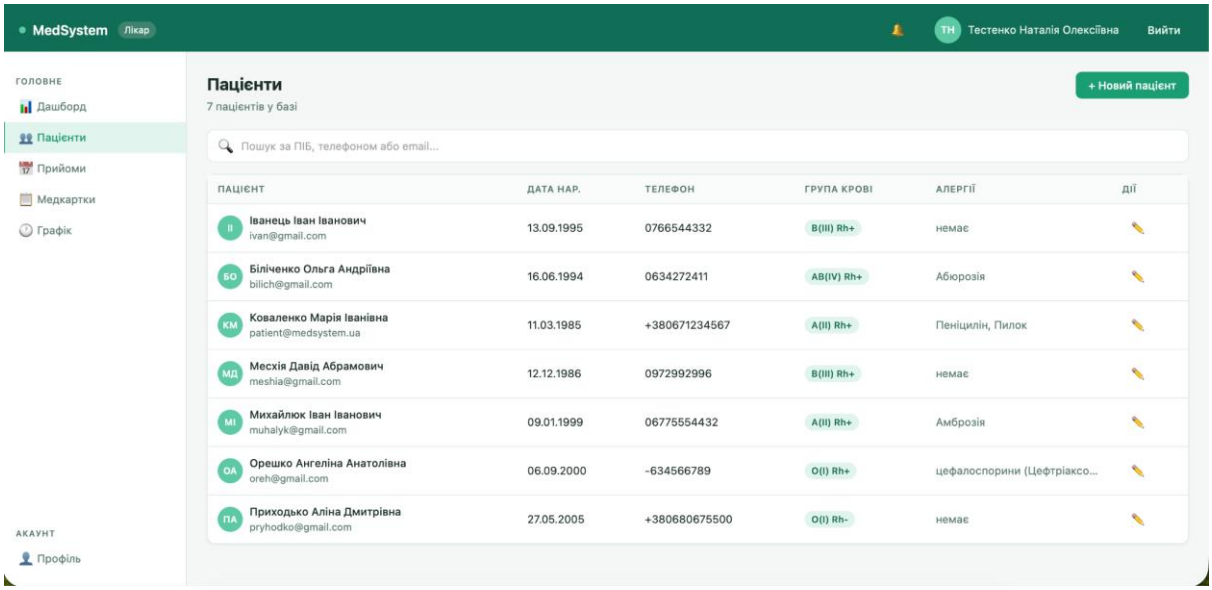


Рисунок 3.21 – Пацієнти

Тут у лікаря є можливість переглянути всіх пацієнтів, які є в базі даних. Тут про них можна дізнатися: ПІБ, дата народження, контактний номер телефону, група крові, наявні алергії.

Також є можливість редагування зареєстрованих пацієнтів (див. рис. 3.22), якщо пацієнт забув щось змінити, наприклад, адресу, номер телефона, або не знав про алергії та хронічні захворювання.

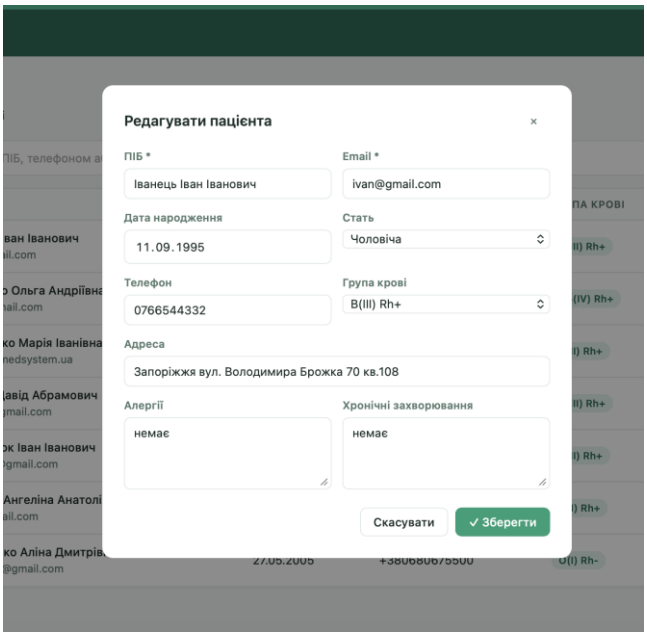


Рисунок 3.22 – Редагування пацієнта

Для зручного пошуку пацієнтів, згори, є пошукова строка (див. рис. 3.23).

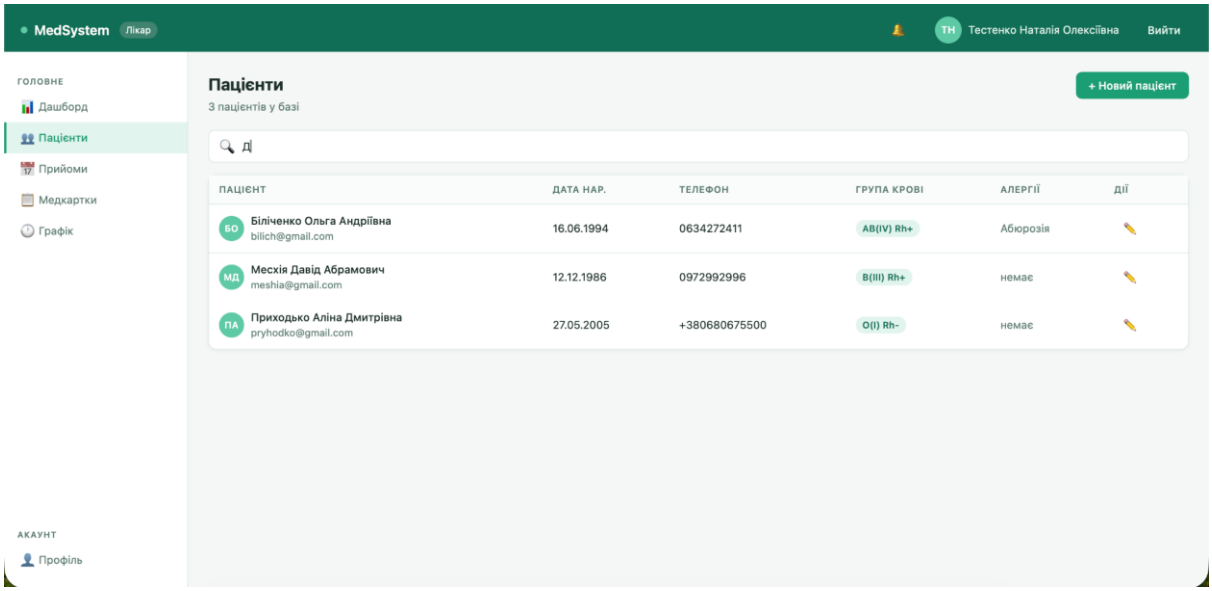


Рисунок 3.23 – Пошук

В верхньому правому куті знаходиться кнопка ‘+Новий пацієнт’ (див.рис.3.24). За допомогою цієї кнопки лікар має можливість самостійно створити аккаунт для пацієнта. Дана функція додана на випадок, якщо до лікаря прийшов пацієнт без запису або людина з якихось причин не може самотужки створити акаунт (наприклад людина похилого віку).

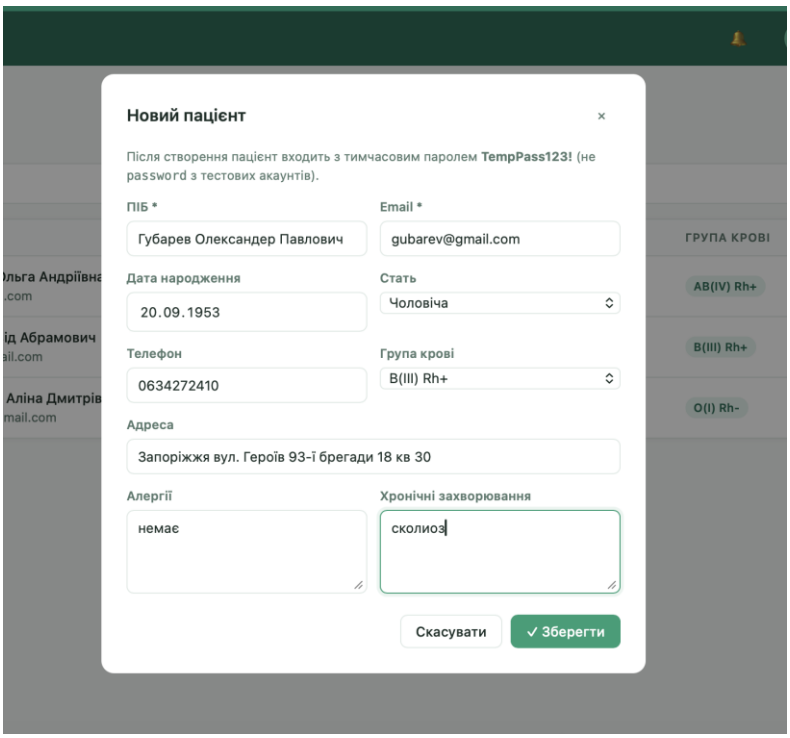


Рисунок 3.24 – Створення нового пацієнта

Тут лікарю потрібно заповнити всі персональні дані нового пацієнта такі як : ПІБ, електронна пошта, дата народження, стать, контактний телефон, група крові, домашня адреса, наявність алергії, хронічні захворювання.

Після зберігання випливає інформаційне вікно (див. рис. 3.25). Тут можемо побачити тимчасовий пароль та постійний логін який ми можемо або просто повідомити пацієнту або скопіювати за допомогою кнопки ‘Скопіювати’ та надіслати пацієнтові. Після чого можемо закрити вікно кнопкою ‘Закрити’.

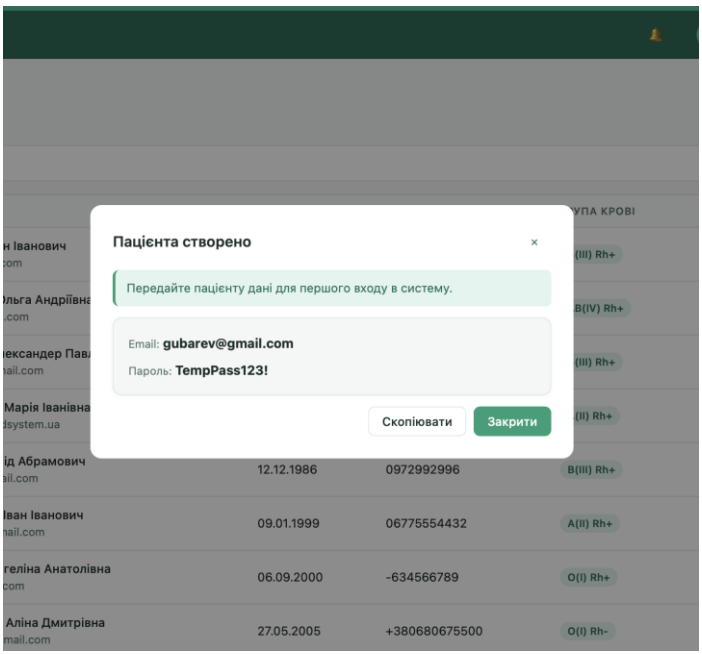


Рисунок 3.25 – Пацієнта створено

Наступним розглянемо вкладку ‘Прийом’ (див. рис. 3.26) – це календар на тиждень, де лікар може переглянути свої записи на будь-який день.

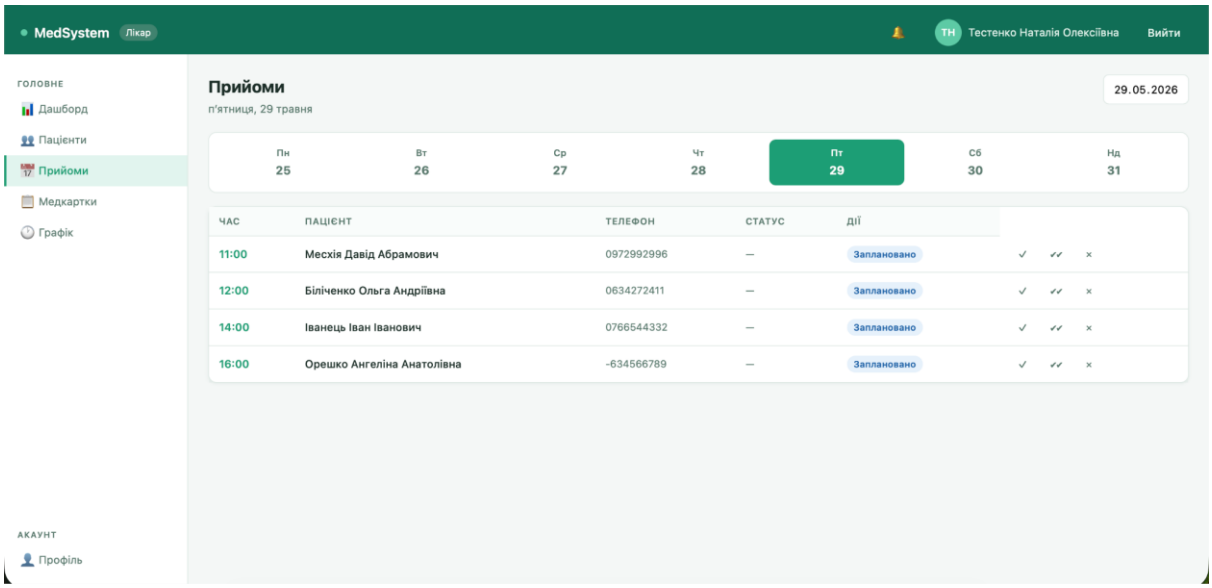


Рисунок 3.26 – Прийоми

Лікар бачить наступну інформацію: дата прийому, час прийому, ПІБ пацієнта, контактный телефон, статус прийому, дії, кнопки: ‘Підтвердити запис’, ‘Завершити прийом’, ‘Скасувати’ (див.рис.3.28).

В верхньому правому куті бачимо календар (див. рис. 3.27), де можна обрати інший тиждень. Тижні оновлюються автоматично, але за потреби можна переглянути інші дати.

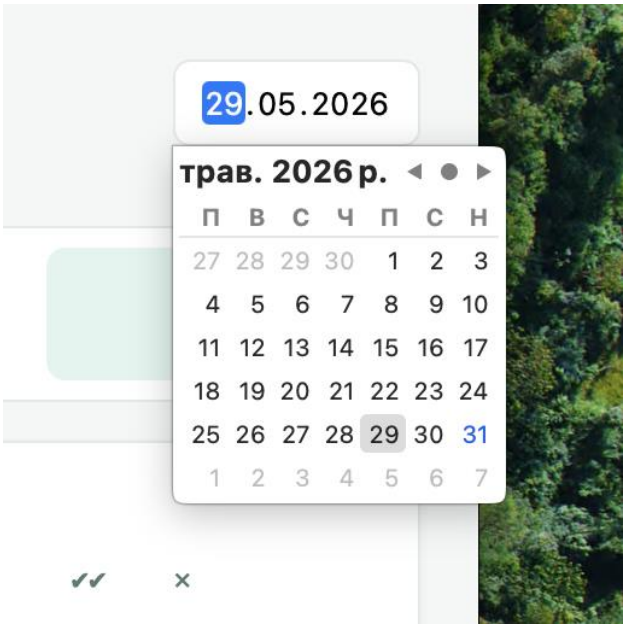


Рисунок 3.27 – Календар

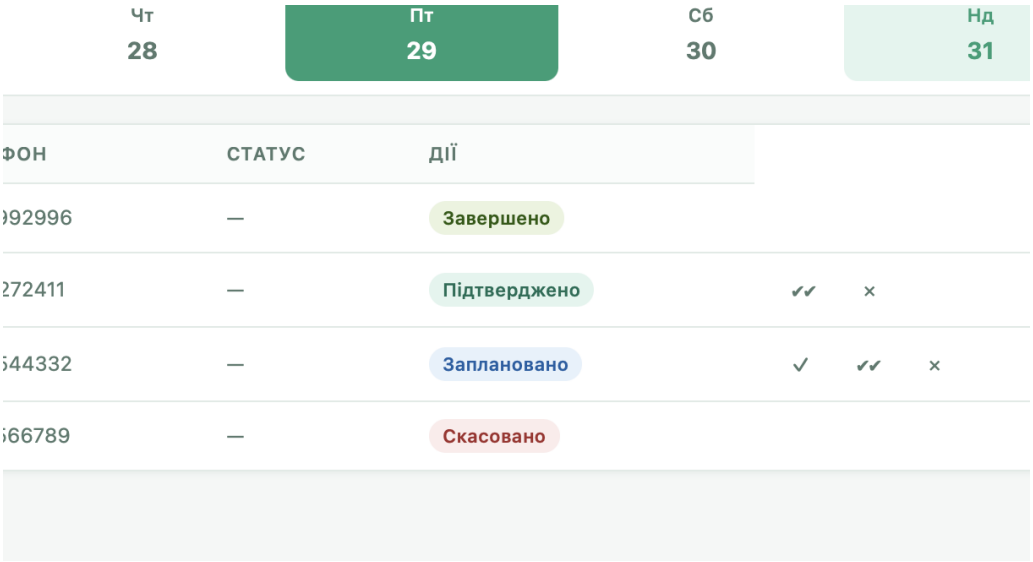


Рисунок 3.28 – Кнопки підтвердження та скачування

- одна галочка – підтвердити запис, натискають коли пацієнт явився на прийом. Після натискання программа пише ‘Підтверджено’ та залишаються на панелі дві кнопки: завершити та скасувати.
- дві галочки – завершити, натискають коли прийом завершено. Программа пише ‘Завершено’ та решта кнопок зникають.
- хрестик – скасувати запис, натискають, якщо запис скасовано з певних причин. Программа пише ‘Скасовано’ та решта кнопок зникає. При натисканні цієї кнопки программа видає вікно підтвердження (див. рис. 3.14).

Наступним пунктом розглянемо розділ ‘Медкарти’ (див. рис. 3.29).

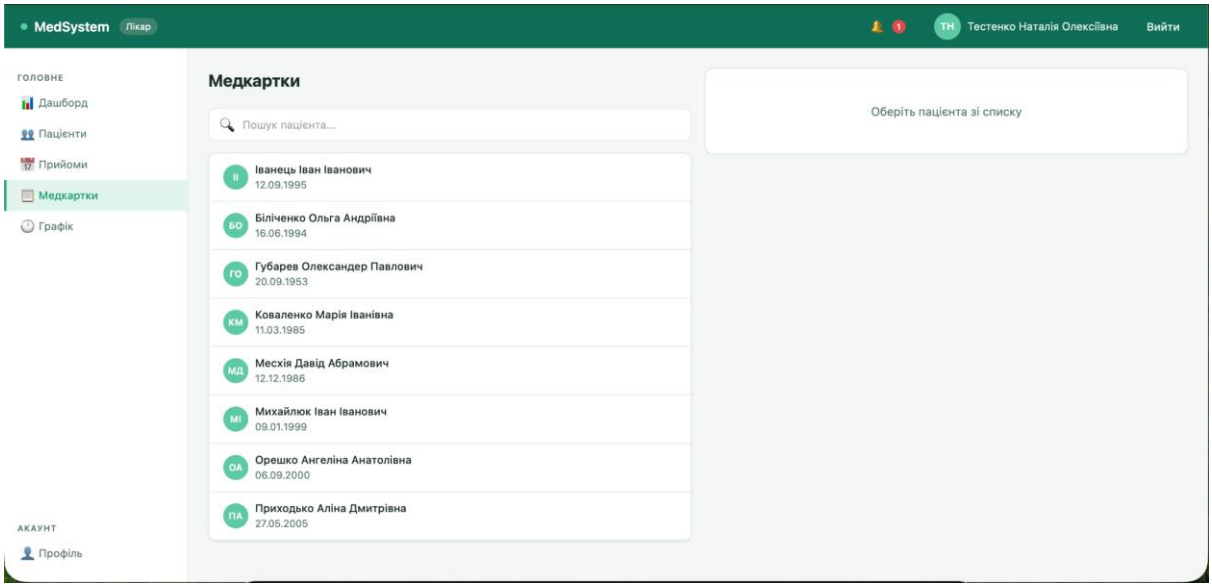


Рисунок 3.29 – Медкарти

В цьому розділі зібрані всі мед карти пацієнтів, що коли небудь були на прийомі у цього лікаря. Сторінка має два основних стовпці: в лівому – список пацієнтів, в правому – історія хвороб обраного пацієнта (див. рис. 3.30). Також згори можна побачити пошукову строку, яка працює так само як було описано раніше.

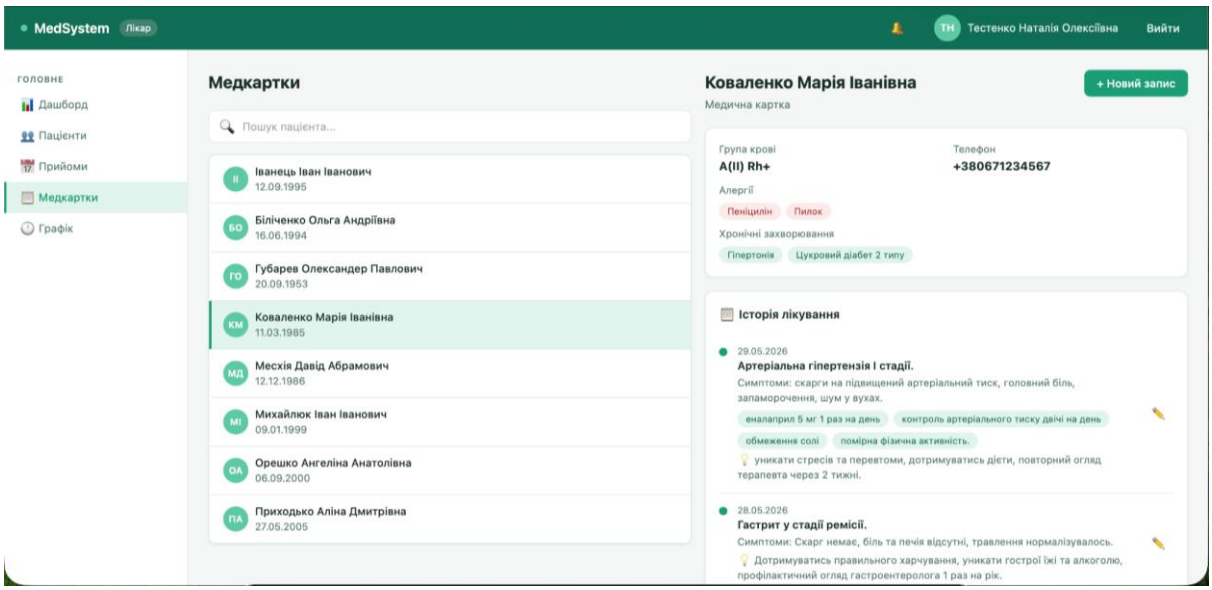


Рисунок 3.30 – Медична картка

Тут можна побачити всю історію відвідувань пацієнта у списку ‘Історія лікування’, де знаходиться вся необхідна інформація про його скарги та лікування.

В правому верхньому куті бачимо кнопку ‘+ Новий запис’ (див.рис.3.31), вона дозволяє створити новий запис в історії хвороб, що є основним робочим полем лікаря. Реалізацію цього компонента інтерфейсу лікаря наведено у додатку А, рис. А.9.

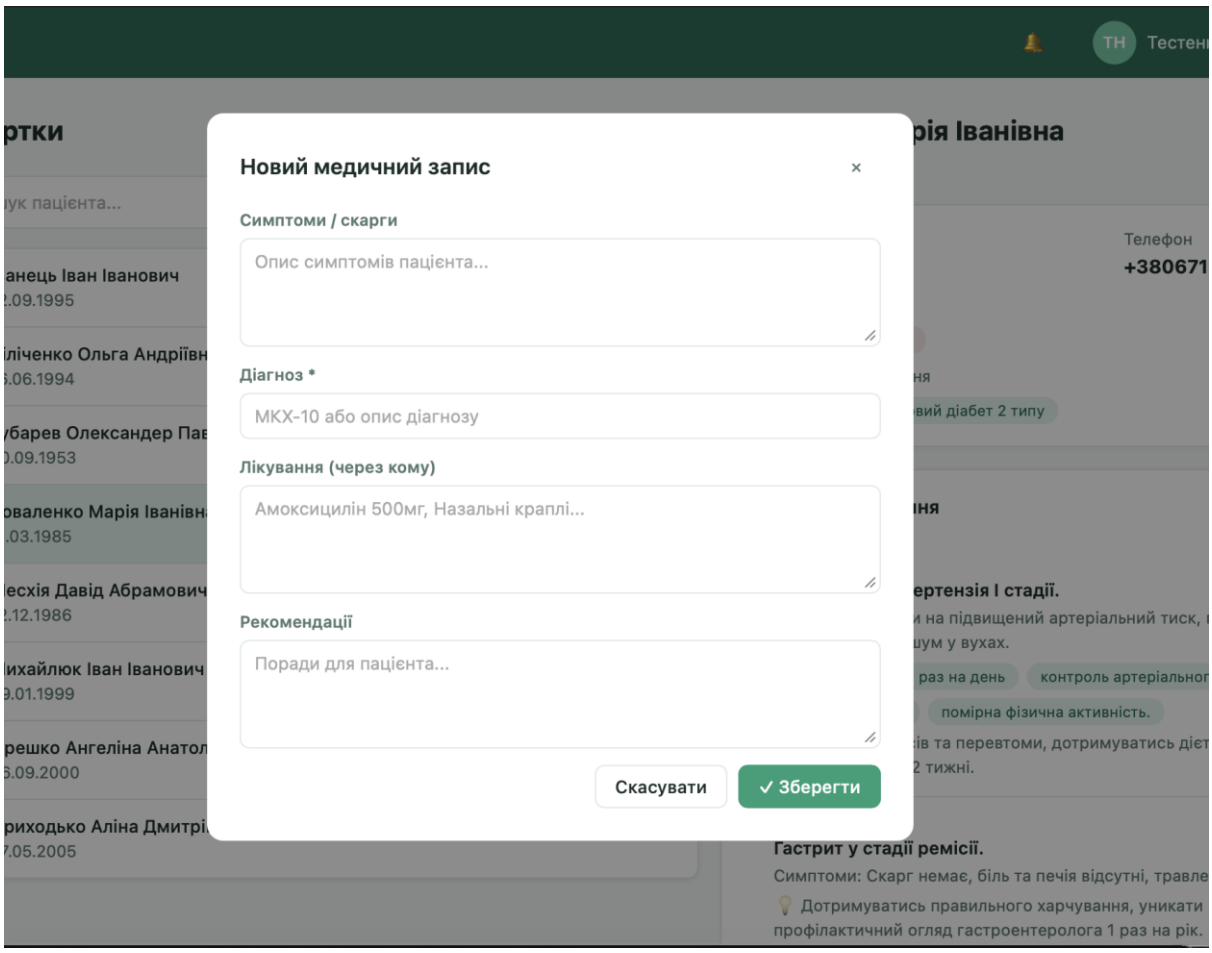


Рисунок 3.31 – Новий медичний запис

На цій формі маємо наступні текстові поля:

- симптоми / скарги – в це поле лікар заповнює всю інформацію про стан пацієнта при першому візиті. Це поле є обов’язковим для заповнення!
- діагноз – поле куди лікар вписує поставлений діагноз після проведення обстеження.
- лікування – поле в яке заповнюється рецепт лікування.
- рекомендації – поле куди лікар виписує рекомендації для покращення стану здоров’я пацієнта.

Знизу є дві кнопки: ‘Скасувати’ та ‘Зберегти’, яка зберігає запис на сервер.

Пацієнт може переглянути ці записи в будь який момент зі свого аккаунта після зберігання інформації лікарем.

Вже створені записи є можливість редагувати за допомогою кнопки ‘Олівець’ (див. рис. 3.32). При натисканні висвічується вже заповнена форма, яку можна відредагувати та зберегти заново, інформація оновлюється, як на сервері, так і у пацієнта.

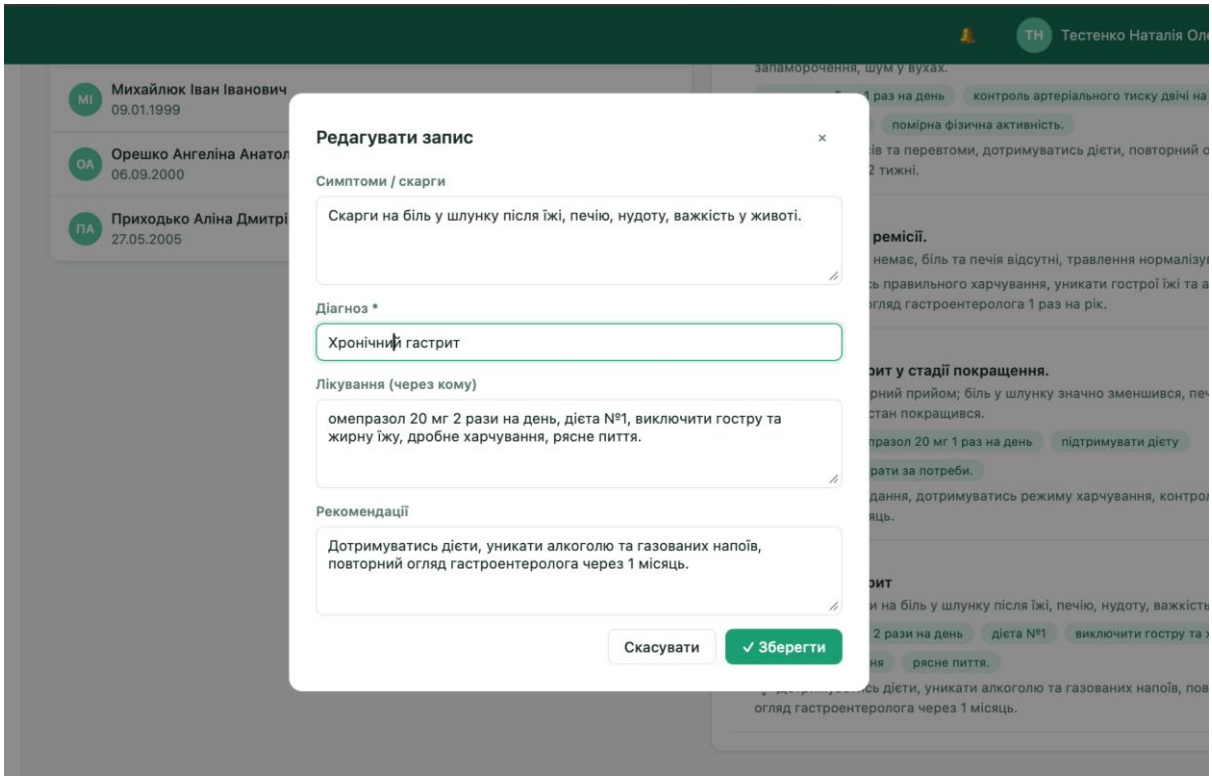


Рисунок 3.32 – Редагувати запис

Наступна вкладка – це графік роботи (див. рис. 3.33).

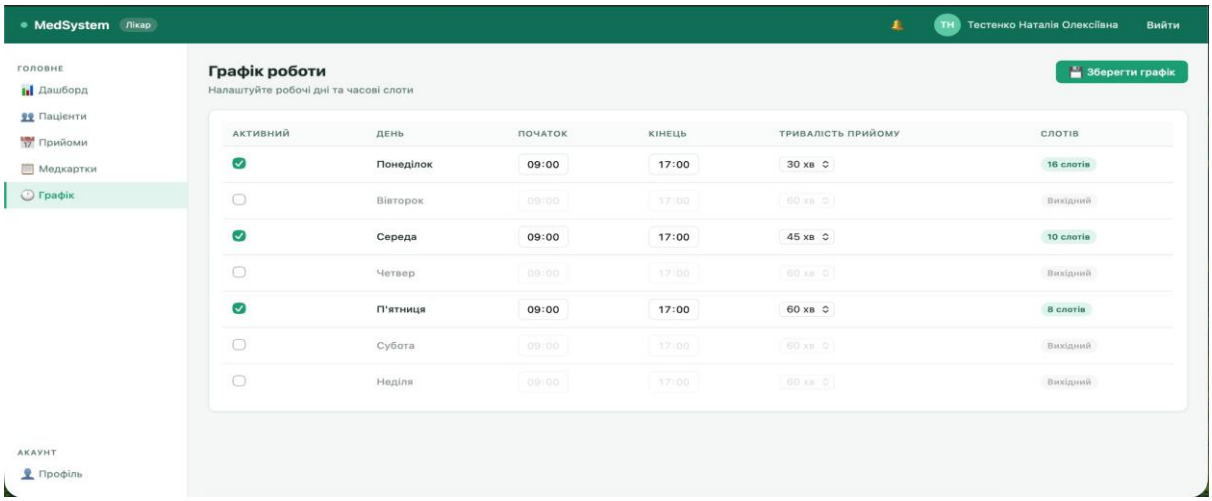


Рисунок 3.33 – Графік роботи

Це вкладка (див. рис. 3.34) на якій лікар налаштовує: робочі дні, робочі години та орієнтовний час прийому (в програмі іменуємий як ‘Слот’). Також відображаються дні тижня та календарна дата. За допомогою чекбоксів лікар вмикає або вимикає робочий день, потім налаштовує години прийому. Наступним налаштовує скільки часу триває один прийом. Останній стовпець показує кількість слотів (прийомів) в робочій день. Код управління розкладом та генерації відповідних часових слотів міститься у додатку А, рис. А.10.

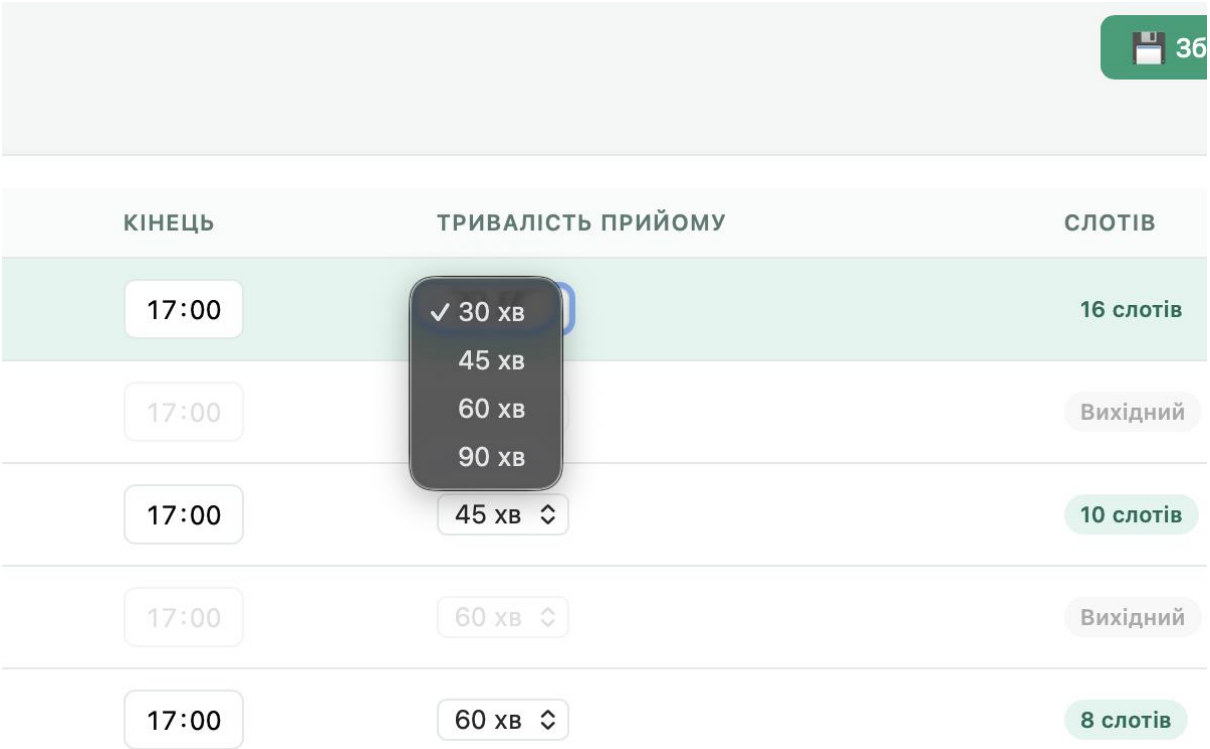


Рисунок 3.34 – Налаштування тривалості прийому

Для збереження змін необхідно натиснути на кнопку ‘Зберегти графік’, що знаходиться в правому верхньому куті вікна. Пісня натискання кнопка стає індикатором, що показує статус зберігання інформації. Спочатку статус ‘Збереження…’ (див. рис. 3.35), після успішного завершення операції виводиться статус ‘Збережено’ (див. рис. 3.36). Після очікування декількох секунд індикатор знову стає кнопкою ‘Зберегти графік’ (див. рис. 3.37).

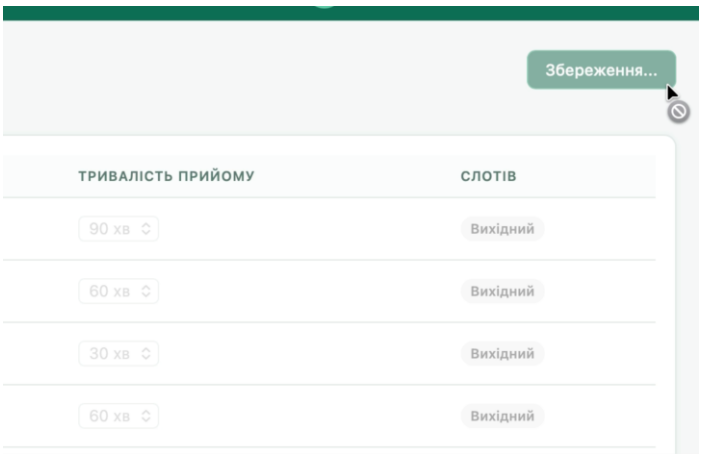


Рисунок 3.35 – Збереження

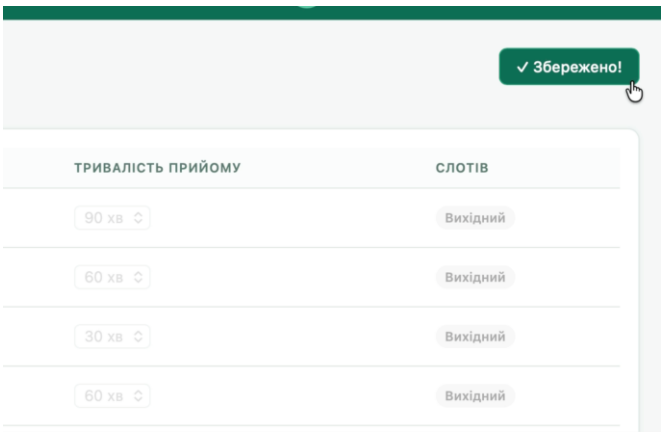


Рисунок 3.36 – Збережено

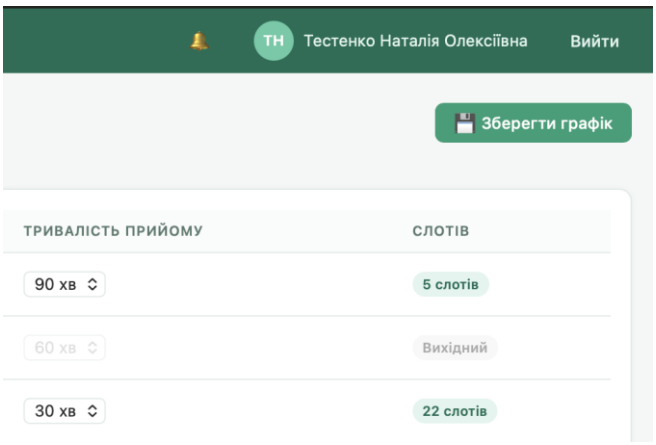


Рисунок 3.37 – Зберегти графік

Сформовані слоти у пацієнта відображатимуться, як на (див. рис. 3.11).

Наступний пункт – це вікно ‘Профіль’, де можна відредагувати дані про лікаря (див. рис. 3.38).

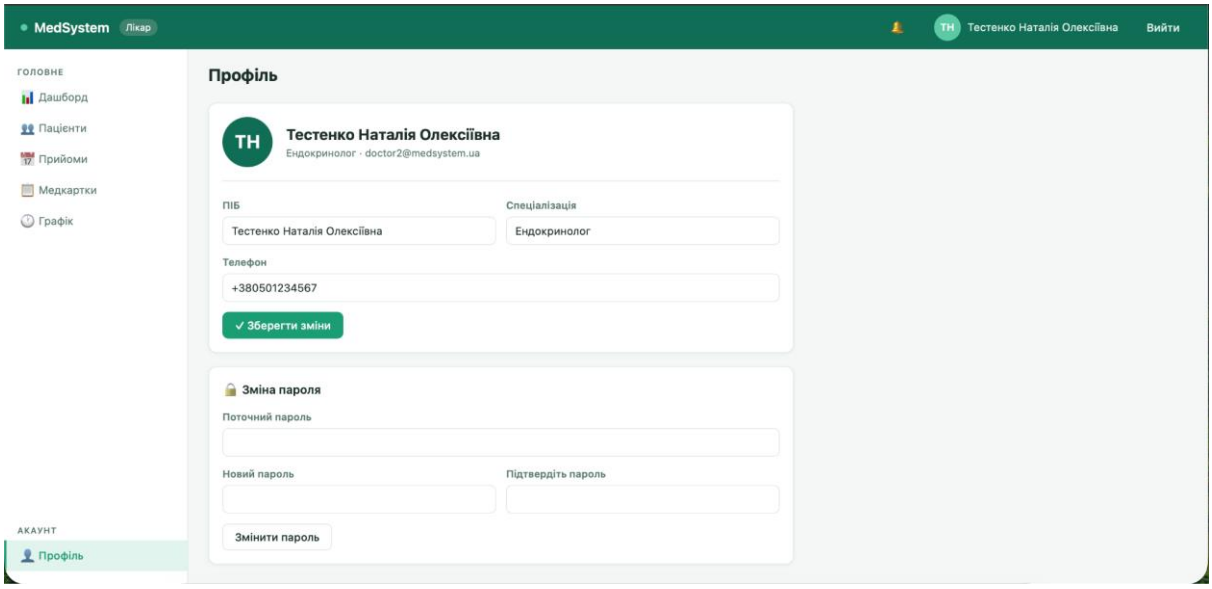


Рисунок 3.38 – Профіль

Вікно містить два поля. Перше поле – це персональні дані, такі як: ПІБ, контактний номер телефону лікаря, спеціальність.

В кінці поля є кнопка ‘Зберегти’, яка зберігає оновлену інформацію на сервер. Після успішної операції згори висвічується відповідна індикація ‘Профіль оновлено’ (див. рис. 3.39).

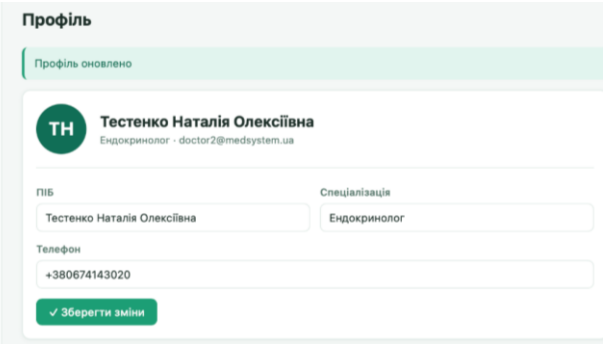


Рисунок 3.39 – Профіль оновлено

Наступне поле дозволяє змінити пароль аккаунта (див. рис. 3.40).

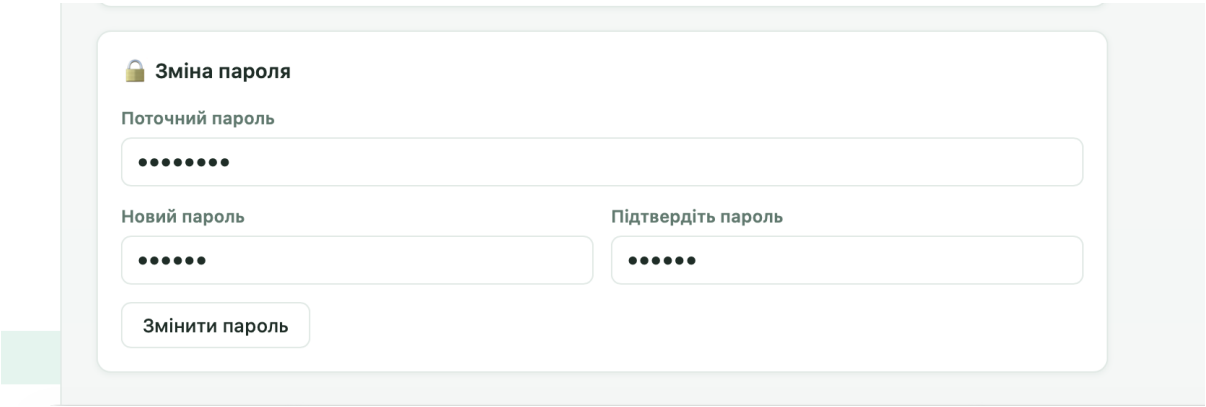


Рисунок 3.40 – Змінити пароль

Для зміни паролю необхідно ввести старий пароль та два рази новий, після чого натиснути кнопку ‘Змінити пароль’ та у підтвердження успішності операції засвітиться відповідна індикація. Індикація може бути як підтвердженням (див. рис. 3.41) так і помилкою (див. рис. 3.42) .

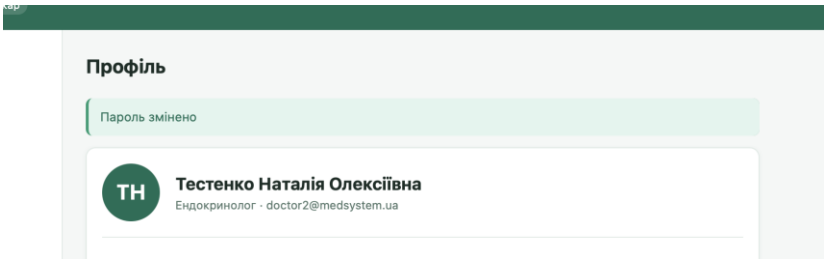


Рисунок 3.41 – Підтвердження

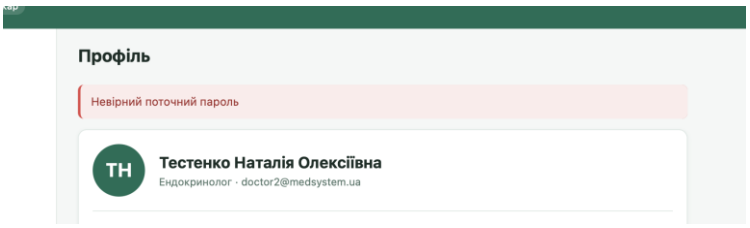


Рисунок 3.42 – Помилка

Розроблений інтерфейс забезпечує комфортну взаємодію пацієнтів і лікарів із системою завдяки логічній структурі сторінок, зрозумілій навігації та зручному розташуванню елементів керування. Реалізовані функції авторизації, керування профілем, запису на прийом, ведення електронних медичних карток і

налаштування графіка роботи забезпечують ефективне виконання основних бізнес-процесів медичного закладу.

3.4 Фрагменти програмної реалізації системи

Для реалізації функціональних можливостей медичної інформаційної системи було використано сучасні технології веброзробки. Клієнтська частина застосунку реалізована за допомогою бібліотеки React, що забезпечує створення динамічного користувацького інтерфейсу та швидку взаємодію з користувачем. Взаємодія між клієнтською та серверною частинами здійснюється через REST API шляхом обміну даними у форматі JSON.

Для кожного інтерфейсного елемента або сторінки наведено відповідний код програмної реалізації, що демонструє принцип роботи окремих компонентів системи.

Розглянуті фрагменти охоплюють модулі авторизації та реєстрації користувачів, особисті кабінети пацієнта та лікаря, механізми запису на прийом, управління електронними медичними картками, формування графіка роботи лікарів, а також налаштування профілів користувачів. Наведені приклади підтверджують працездатність розробленої системи та демонструють практичну реалізацію функціональних вимог, поставлених у рамках дипломного проєкту.

3.5 Висновки до третього розділу

У третьому розділі розроблено алгоритмічне та програмне забезпечення системи «MedSystem», зокрема модулі автентифікації, контролю доступу та фіксації змін. На базі СКБД PostgreSQL створено структуру бази даних із каскадними правилами та обмеженнями цілісності для захисту інформації.

Також реалізовано вебінтерфейс для лікарів і пацієнтів, який автоматизує запис на прийом, ведення медкарт і перегляд розкладу. Створені рішення гарантують надійне збереження даних та повний аудит подій відповідно до сучасних медичних стандартів.

ВИСНОВКИ

У результаті виконання дипломного проєкту було розроблено високонадійну базу даних для медичної інформаційної системи з підтримкою історії змін та створено веборієнтовану інформаційну систему «MedSystem», призначену для автоматизації взаємодії між лікарями та пацієнтами.

Під час виконання роботи було проведено аналіз предметної області, визначено функціональні та нефункціональні вимоги до системи, спроєктовано структуру бази даних, розроблено архітектуру програмного забезпечення та реалізовано механізми забезпечення безпеки і контролю доступу до медичної інформації.

Розроблене програмне забезпечення реалізовано за клієнт-серверною архітектурою із використанням сучасних вебтехнологій React, Node.js, Express.js та PostgreSQL. Система підтримує рольовий доступ для пацієнтів і лікарів, забезпечує автентифікацію користувачів за допомогою JWT-токенів, захищене зберігання паролів із використанням алгоритму bcrypt та контроль доступу до медичних даних на рівні бізнес-логіки застосунку.

Функціональні можливості системи включають реєстрацію та авторизацію користувачів, управління профілями пацієнтів і лікарів, ведення електронних медичних карток, створення та редагування медичних записів, планування прийомів, автоматичне формування часових слотів для запису пацієнтів, налаштування графіка роботи лікарів, систему повідомлень та ведення персональних завдань лікаря.

Особливістю розробленої системи є реалізація журналу аудиту змін, який забезпечує автоматичне збереження історії виконаних операцій над критично важливими даними. Це підвищує надійність зберігання інформації, спрощує контроль змін та сприяє дотриманню вимог щодо безпеки медичних даних.

Отримані результати підтверджують досягнення поставленої мети дипломного проєкту. Розроблена система є працездатною, масштабованою та

може використовуватися як основа для подальшого впровадження і розвитку медичних інформаційних систем у закладах охорони здоров’я.

ПЕРЕЛІК ПОСИЛАНЬ

1. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Pearson Education, 2004. 1024 p.

2. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2017. 1272 p.

3. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2019. 1376 p.

4. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation and Management. 6th ed. London : Pearson Education, 2015. 1440 p.

5. PostgreSQL Global Development Group. PostgreSQL Documentation. URL : https://www.postgresql.org/docs/ (дата звернення : 01.06.2026).

6. React Documentation. URL : https://react.dev/ (дата звернення : 01.06.2026).

7. Node.js Documentation. URL : https://nodejs.org/docs/latest/api/ (дата звернення : 01.06.2026).

8. Express.js Documentation. URL : https://expressjs.com/ (дата звернення : 01.06.2026).

9. JSON Web Tokens Introduction. URL : https://jwt.io/introduction (дата звернення : 01.06.2026).

10. Mozilla Developer Network. HTTP Overview. URL : https://developer.mozilla.org/ (дата звернення : 01.06.2026).

11. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

12. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2019. 880 p.

13. ISO/IEC 25010:2011. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE).

ДОДАТОК А

Фрагменти програмної реалізації системи

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Таблиця користувачів (загальна авторизаційна сутність)
CREATE TABLE IF NOT EXISTS users (
  id         UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  full_name  VARCHAR(255) NOT NULL,
  email      VARCHAR(255) UNIQUE NOT NULL,
  password   VARCHAR(255) NOT NULL,          -- bcrypt hash
  role       VARCHAR(20) NOT NULL CHECK (role IN ('doctor', 'patient')),
  created_at TIMESTAMP DEFAULT NOW()
);

-- Таблиця розширеного профілю пацієнтів
CREATE TABLE IF NOT EXISTS patients (
  id               UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id          UUID REFERENCES users(id) ON DELETE CASCADE,
  birth_date        DATE,
  gender           VARCHAR(10) CHECK (gender IN ('male', 'female')),
  phone            VARCHAR(30),
  address          TEXT,
  blood_group       VARCHAR(40),
  allergies        TEXT,
  chronic_diseases   TEXT,
  created_at        TIMESTAMP DEFAULT NOW()
);

-- Таблиця розширеного профілю лікарів
CREATE TABLE IF NOT EXISTS doctors (
  id            UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id       UUID REFERENCES users(id) ON DELETE CASCADE,
  specialization VARCHAR(255),
  phone         VARCHAR(30),
  dashboard_notes TEXT,
  dashboard_notes_updated_at TIMESTAMP,
  created_at    TIMESTAMP DEFAULT NOW()
);
```

```
-- Таблиця зв'язку між лікарями та пацієнтами (для обмеження прав доступу)
CREATE TABLE IF NOT EXISTS doctor_patients (
 doctor_id  UUID NOT NULL REFERENCES doctors(id) ON DELETE CASCADE,
 patient_id  UUID  NOT  NULL  REFERENCES  patients(id)  ON  DELETE
CASCADE,
 created_at TIMESTAMP DEFAULT NOW(),
 PRIMARY KEY (doctor_id, patient_id)
);

-- Графік роботи лікаря для розрахунку тайм-слотів
CREATE TABLE IF NOT EXISTS doctor_schedules (
 id                UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
 doctor_id         UUID REFERENCES doctors(id) ON DELETE CASCADE,
 work_day                    VARCHAR(10)  NOT  NULL  CHECK  (work_day  IN
('monday','tuesday','wednesday','thursday','friday','saturday','sunday')),
 start_time        TIME NOT NULL,
 end_time          TIME NOT NULL,
 appointment_duration INTEGER NOT NULL DEFAULT 60,  -- тривалість у
хвилинах
 is_active         BOOLEAN DEFAULT TRUE,
 UNIQUE(doctor_id, work_day)
);

-- Таблиця записів на прийом
CREATE TABLE IF NOT EXISTS appointments (
 id             UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
 patient_id     UUID REFERENCES patients(id) ON DELETE CASCADE,
 doctor_id      UUID REFERENCES doctors(id) ON DELETE CASCADE,
 appointment_date DATE NOT NULL,
 appointment_time TIME NOT NULL,
 status         VARCHAR(20) NOT NULL DEFAULT 'scheduled'
               CHECK (status IN ('scheduled','confirmed','completed','cancelled')),
 notes          TEXT,
 created_at     TIMESTAMP DEFAULT NOW(),
 UNIQUE(doctor_id, appointment_date, appointment_time)
);

-- Електронні медичні картки (історія лікування)
CREATE TABLE IF NOT EXISTS medical_records (
 id             UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
 patient_id     UUID REFERENCES patients(id) ON DELETE CASCADE,
 doctor_id      UUID REFERENCES doctors(id) ON DELETE CASCADE,
 appointment_id  UUID REFERENCES appointments(id) ON DELETE SET NULL,
 symptoms       TEXT,
```

```
 diagnosis       VARCHAR(500),
 treatment       TEXT,
 recommendations TEXT,
 created_at      TIMESTAMP DEFAULT NOW()
);

-- Журнал аудиту (високонадійне збереження історії змін медичних даних)
CREATE TABLE IF NOT EXISTS audit_log (
 id          UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
 user_id     UUID REFERENCES users(id) ON DELETE SET NULL,
 action      VARCHAR(100),  -- 'CREATE', 'UPDATE', 'DELETE'
 table_name  VARCHAR(100),
 record_id   UUID,
 details     JSONB,         -- збереження зліпку даних у форматі JSON
 created_at  TIMESTAMP DEFAULT NOW()
);
```

Рисунок А.1 – Скрипт розгортання структури бази даних медичної інформаційної системи та логування змін (schema.sql)

```
const { Pool } = require('pg');

// Створення пулу з'єднань для оптимізації навантаження на БД та стабільності підключень
const pool = new Pool({
 host:     process.env.DB_HOST     || 'localhost',
 port:     process.env.DB_PORT     || 5432,
 database: process.env.DB_NAME     || 'medsystem',
 user:     process.env.DB_USER     || 'postgres',
 password: process.env.DB_PASSWORD || 'postgres',
});

// Логування подій життєвого циклу підключень
pool.on('connect', () => console.log('✅ PostgreSQL підключено успішно'));
pool.on('error',   (err) => console.error('❌ Критична помилка PostgreSQL:', err));

module.exports = pool;
```

Рисунок А.2 – Конфігурація пулу з’єднань з СКБД для забезпечення відмовостійкості

```
const express = require('express');
const router = express.Router();
const { getByPatient, getMyRecords, create, update } =
require('../controllers/medicalRecordController');
const { authenticate, requireRole } = require('../middleware/auth');

// Визначення прав доступу та маршрутизація операцій над медичними записами
router.get('/mine',              authenticate, requireRole('patient'), getMyRecords);
router.get('/patient/:patientId', authenticate, requireRole('doctor'),  getByPatient);
router.post('/',                 authenticate, requireRole('doctor'),  create);
router.put('/:id',               authenticate, requireRole('doctor'),  update);

module.exports = router;
```

Рисунок А.3 – Опис кінцевих точок API серверної частини для електронних медичних карток

```
const pool = require('../db/pool');
const { assertDoctorPatient } = require('../utils/doctorPatient');

// Отримання медичних записів конкретного пацієнта (для лікаря)
const getByPatient = async (req, res) => {
  const doctorId = req.user.profileId;
  try {
    const hasAccess = await assertDoctorPatient(pool, doctorId, req.params.patientId);
    if (!hasAccess) {
      return res.status(404).json({ error: 'Пацієнта не знайдено або доступ обмежено'
});
    }
    const result = await pool.query(
      `SELECT mr.*, u.full_name AS doctor_name, d.specialization
       FROM medical_records mr
       JOIN doctors d ON d.id = mr.doctor_id
       JOIN users u   ON u.id = d.user_id
       WHERE mr.patient_id = $1
       ORDER BY mr.created_at DESC`,
      [req.params.patientId]
    );
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: 'Помилка отримання медичних записів' });
```

```
 }
};

// Створення медичного запису з використанням ізольованої транзакції ACID
const create = async (req, res) => {
 const { patient_id, appointment_id, symptoms, diagnosis, treatment, recommendations } = req.body;

 if (!patient_id || !diagnosis) {
  return res.status(400).json({ error: 'Пацієнт та діагноз обов\'язкові для заповнення' });
 }

 const client = await pool.connect();
 try {
  await client.query('BEGIN'); -- Початок транзакції

  const hasAccess = await assertDoctorPatient(client, req.user.profileId, patient_id);
  if (!hasAccess) {
   await client.query('ROLLBACK');
   return res.status(403).json({ error: 'Немає санкціонованого доступу до медичної картки' });
  }

  const result = await client.query(
   `INSERT INTO medical_records (patient_id, doctor_id, appointment_id, symptoms, diagnosis, treatment, recommendations)
    VALUES ($1,$2,$3,$4,$5,$6,$7) RETURNING *`,
   [patient_id, req.user.profileId, appointment_id || null, symptoms, diagnosis, treatment, recommendations]
  );

  if (appointment_id) {
   await client.query("UPDATE appointments SET status='completed' WHERE id=$1", [appointment_id]);
  }

  -- Запис операції створення у незмінний журнал аудиту (аудит-лог)
  await client.query(
   `INSERT INTO audit_log (user_id, action, table_name, record_id, details)
    VALUES ($1,'CREATE','medical_records',$2,$3)`,
   [req.user.id, result.rows[0].id, JSON.stringify({ patient_id, diagnosis })]
  );
```

```
    await client.query('COMMIT'); -- Успішне завершення транзакції
    res.status(201).json(result.rows[0]);
  } catch (err) {
    await client.query('ROLLBACK'); -- Відкат змін у разі критичного збою
    console.error(err);
    res.status(500).json({ error: 'Помилка створення медичного запису' });
  } finally {
    client.release(); -- Звільнення клієнта назад до пулу
  }
};

// Оновлення медичного запису пацієнта із фіксацією попереднього стану
const update = async (req, res) => {
  const { symptoms, diagnosis, treatment, recommendations } = req.body;
  try {
    const result = await pool.query(
      `UPDATE medical_records
       SET symptoms=$1, diagnosis=$2, treatment=$3, recommendations=$4
       WHERE id=$5 AND doctor_id=$6 RETURNING *`,
     [symptoms, diagnosis, treatment, recommendations, req.params.id,
req.user.profileId]
    );
    if (!result.rows.length) return res.status(404).json({ error: 'Запис не знайдено або
доступ обмежено' });

    -- Логування оновлення медичних даних із записом модифікованих полів
    await pool.query(
      `INSERT INTO audit_log (user_id, action, table_name, record_id, details)
       VALUES ($1,'UPDATE','medical_records',$2,$3)`,
      [req.user.id, req.params.id, JSON.stringify(req.body)]
    );

    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка оновлення запису' });
  }
};

module.exports = { getByPatient, create, update };
```

Рисунок А.4 – Контролер обробки медичних карток із механізмом транзакцій та аудиту змін

```
// Перевірка наявності зв'язку між лікарем та пацієнтом для захисту персональних даних
const assertDoctorPatient = async (pool, doctorId, patientId) => {
  const result = await pool.query(
    `SELECT 1 FROM doctor_patients WHERE doctor_id = $1 AND patient_id = $2`,
    [doctorId, patientId]
  );
  return result.rows.length > 0;
};

module.exports = { assertDoctorPatient };
```

Рисунок А.5 – Модуль верифікації прав доступу між лікарем та пацієнтом

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

export default function RegisterPage() {
  const { register } = useAuth();
  const navigate = useNavigate();
  const [form, setForm]       = useState({ full_name:'', email:'', password:'', phone:'', birth_date:'', gender:'female' });
  const [error, setError]   = useState('');
  const [loading, setLoading] = useState(false);

  const set = (k) => (e) => setForm(f => ({ ...f, [k]: e.target.value }));

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(''); setLoading(true);
    try {
      await register(form);
      navigate('/patient/dashboard');
    } catch (err) {
      setError(err.response?.data?.error || 'Помилка реєстрації');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="login-page">
```

```
    <div className="login-card" style={{ width: 420 }}>
      <div className="login-logo">
        <div className="login-logo-icon">✚</div>
        <span style={{ fontSize: 20, fontWeight: 700 }}>MedSystem</span>
      </div>
      <h2 style={{ fontSize: 18, fontWeight: 700, marginBottom: 4 }}>Реєстрація пацієнта</h2>
      <p className="text-muted" style={{ marginBottom: 18 }}>Заповніть дані для створення акаунту</p>

      {error && <div className="alert-error">{error}</div>}

      <form onSubmit={handleSubmit}>
        <div className="form-row">
          <div className="form-group">
            <label>ПІБ *</label>
            <input value={form.full_name} onChange={set('full_name')} placeholder="Прізвище Ім'я По батькові" required />
          </div>
          <div className="form-group">
            <label>Дата народження</label>
            <input type="date" value={form.birth_date} onChange={set('birth_date')} />
          </div>
        </div>
        <div className="form-row">
          <div className="form-group">
            <label>Email *</label>
            <input type="email" value={form.email} onChange={set('email')} placeholder="email@example.ua" required />
          </div>
          <div className="form-group">
            <label>Телефон</label>
            <input value={form.phone} onChange={set('phone')} placeholder="+380 XX XXX XX XX" />
          </div>
        </div>
        <div className="form-row">
          <div className="form-group">
            <label>Пароль *</label>
            <input type="password" value={form.password} onChange={set('password')} placeholder="мін. 6 символів" required minLength={6} />
```

```
          </div>
          <div className="form-group">
            <label>Стать</label>
            <select value={form.gender} onChange={set('gender')}>
              <option value="female">Жіноча</option>
              <option value="male">Чоловіча</option>
            </select>
          </div>
        </div>
        <button className="btn btn-primary" style={{ width: '100%', justifyContent:
'center' }} disabled={loading}>
          {loading ? 'Реєстрація...' : 'Зареєструватись'}
        </button>
      </form>
      <div style={{ textAlign: 'center', marginTop: 14 }}>
        <Link to="/login" style={{ color: 'var(--teal)', fontSize: 13, textDecoration:
'none' }}>
          Вже є акаунт? Увійти
        </Link>
      </div>
    </div>
  </div>
 );
}
```

Рисунок А.6 – Форма реєстрації нових користувачів-пацієнтів системи

```
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import { useAuth } from '../../context/AuthContext';
import api from '../../utils/api';

const STATUS_UA = { scheduled:'Заплановано', confirmed:'Підтверджено',
completed:'Завершено', cancelled:'Скасовано' };
const STATUS_CLS= { scheduled:'badge-blue', confirmed:'badge-teal',
completed:'badge-green', cancelled:'badge-red' };

export default function PatientDashboard() {
 const { user } = useAuth();
 const [appointments, setAppointments] = useState([]);
 const [records,     setRecords]     = useState([]);

 useEffect(() => {
```

```
  api.get('/appointments/mine').then(r=>setAppointments(r.data)).catch(()=>{});
  api.get('/medical-records/mine').then(r=>setRecords(r.data)).catch(()=>{});
 }, []);

 const upcoming = appointments.filter(a=>a.status!=='cancelled'&&a.status!=='completed'&&new Date(a.appointment_date)>=new Date());
 const lastRecord = records[0];

 const cancel = async (id) => {
  if (!window.confirm('Скасувати запис?')) return;
  await api.patch(`/appointments/${id}`, { status:'cancelled' });
  setAppointments(as => as.map(a => a.id===id ? {...a,status:'cancelled'} : a));
 };

 return (
  <div>
   <div className="page-header">
    <div>
     <div className="page-title">Добрий день, {user?.full_name?.split(' ')[1] || '}!</div>
     <div className="page-sub">{new Date().toLocaleDateString('uk-UA',{weekday:'long',day:'numeric',month:'long'})}</div>
    </div>
    <Link to="/patient/book"><button className="btn btn-primary">+ Записатись до лікаря</button></Link>
   </div>

   <div className="grid2">
    <div>
     <div className="card mb16">
      <div className="card-title">📅 Майбутні прийоми</div>
      {upcoming.length === 0
       ? <div className="text-muted">Немає запланованих прийомів.</div>
       : <div className="slots-list">
          {upcoming.map(a=>(
           <div key={a.id} className="slot busy">
            <span className="slot-time">{new Date(a.appointment_date).toLocaleDateString('uk-UA',{day:'2-digit',month:'2-digit'})}</span>
            <div style={{flex:1}}>
             <div style={{fontWeight:500}}>{a.doctor_name}</div>
             <div className="text-muted">{a.specialization} · {a.appointment_time?.slice(0,5)}</div>
```

```
              </div>
              <span                                                    className={`badge
${STATUS_CLS[a.status]}`}>{STATUS_UA[a.status]}</span>
              {a.status==='scheduled'&&<button  className="btn  btn-danger  btn-
sm" onClick={()=>cancel(a.id)}>Скасувати</button>}
            </div>
          ))}
        </div>
      }
    </div>

    {lastRecord && (
     <div className="card">
      <div className="card-title">📋 Останній прийом</div>
      <div
style={{fontWeight:500,marginBottom:4}}>{lastRecord.diagnosis}</div>
       <div                  className="text-muted                  mb12">{new
Date(lastRecord.created_at).toLocaleDateString('uk-UA')}                    ·
{lastRecord.doctor_name}</div>
       {lastRecord.recommendations  &&  <div  style={{background:'var(--teal-
light)',borderRadius:8,padding:'10px  12px',fontSize:13,color:'var(--teal-dark)'}}>💡
{lastRecord.recommendations}</div>}
     </div>
    )}
   </div>

   <div className="card">
    <div className="card-title">📜 Медична активність</div>
    {records.length === 0
     ? <p className="text-muted">Немає медичних записів</p>
     : <div className="timeline">
        {records.slice(0,5).map(r=>(
         <div key={r.id} className="tl-item">
          <div className="tl-dot"></div>
          <div>
           <div                                          className="tl-date">{new
Date(r.created_at).toLocaleDateString('uk-UA')}</div>
           <div className="tl-title">{r.diagnosis}</div>
           <div className="tl-desc">{r.doctor_name}</div>
          </div>
         </div>
        ))}
       </div>
    }
```

```
      </div>
    </div>
  </div>
 );
}
```

Рисунок А.7 – Інтерфейс особистого кабінету пацієнта

```
import React, { useEffect, useState } from 'react';
import api from '../../utils/api';

export default function PatientBooking() {
 const [doctors,   setDoctors]   = useState([]);
 const [selected,  setSelected]  = useState(null);
 const [date,      setDate]      = useState('');
 const [slots,     setSlots]     = useState([]);
 const [loading,   setLoading]   = useState(false);
 const [booked,    setBooked]    = useState(null);
 const [error,     setError]     = useState('');

 useEffect(() => { api.get('/doctors').then(r=>setDoctors(r.data)).catch(()=>{}); },
[]);

 useEffect(() => {
  if (!selected || !date) { setSlots([]); return; }
  setLoading(true);
  api.get(`/appointments/slots?doctor_id=${selected.id}&date=${date}`)
   .then(r=>setSlots(r.data.slots||[]))
   .catch(()=>setSlots([]))
   .finally(()=>setLoading(false));
 }, [selected, date]);

 const book = async (time) => {
  setError('');
  try {
   await api.post('/appointments', { doctor_id:selected.id, appointment_date:date,
appointment_time:time });
   setBooked({ doctor:selected.full_name, date, time });
   setSlots(s=>s.map(sl=>sl.time===time?{...sl,available:false}:sl));
  } catch (err) { setError(err.response?.data?.error||'Помилка запису'); }
 };

 if (booked) return (
```

```
  <div style={{textAlign:'center', padding:40}}>
    <div style={{fontSize:48,marginBottom:16}}>✅</div>
    <div style={{fontSize:20,fontWeight:700,marginBottom:8}}>Запис підтверджено!</div>
    <div className="text-muted">
      <strong>{booked.doctor}</strong><br/>
      {new Date(booked.date).toLocaleDateString('uk-UA',{weekday:'long',day:'numeric',month:'long'})} о {booked.time}
    </div>
    <button className="btn btn-primary" style={{marginTop:20}} onClick={()=>{setBooked(null);setDate('');setSlots([]);}}>
      Записатись ще раз
    </button>
  </div>
 );

 return (
  <div>
    <div className="page-header"><div className="page-title">Запис до лікаря</div></div>
    {error && <div className="alert-error">{error}</div>}

    <div className="card mb16">
      <div className="card-title">1 Оберіть лікаря</div>
      <div className="grid2">
        {doctors.map(d=>(
          <div key={d.id} onClick={()=>{setSelected(d);setDate('');setSlots([]);}}
            style={{ padding:'12px 14px', borderRadius:9, border:`2px solid ${selected?.id===d.id?'var(--teal)':'var(--border)'}`,
              background: selected?.id===d.id?'var(--teal-light)':'#fff', cursor:'pointer' }}>
            <div className="flex-center">
              <div>
                <div style={{fontWeight:600,fontSize:13}}>{d.full_name}</div>
                <div className="text-muted">{d.specialization}</div>
              </div>
            </div>
          </div>
        ))}
      </div>
    </div>

    {selected && (
```

```
      <div className="card mb16">
        <div className="card-title">2 Оберіть дату прийому</div>
        <input type="date" value={date} min={new Date().toISOString().split('T')[0]}
onChange={e=>setDate(e.target.value)} />
      </div>
    )}

    {date && (
      <div className="card">
        <div className="card-title">3 Оберіть доступний час</div>
        {loading ? <div className="spinner"></div> : slots.length === 0 ? (
          <p className="text-muted">Немає доступних слотів на цей день</p>
        ) : (
          <div className="slots-list">
            {slots.map(sl=>(
              <div key={sl.time} className={`slot ${sl.available?'free':'busy'}`}>
                <span className="slot-time">{sl.time}</span>
                {sl.available ? (
                  <button              className="btn              btn-primary              btn-sm"
onClick={()=>book(sl.time)}>Записатись</button>
                ) : (
                  <span className="badge badge-gray">Зайнято</span>
                )}
              </div>
            ))}
          </div>
        )}
      </div>
    )}
  </div>
 );
}
```

Рисунок А.8 – Модуль покрокового інтерактивного запису пацієнта на прийом

```
import React, { useEffect, useState } from 'react';
import api from '../../utils/api';

const EMPTY_RECORD = { symptoms:'', diagnosis:'', treatment:'',
recommendations:'', appointment_id:'' };

export default function DoctorMedRecords() {
```

```
 const [patients,  setPatients]  = useState([]);
 const [selected,  setSelected]  = useState(null);
 const [records,   setRecords]   = useState([]);
 const [modal,     setModal]     = useState(false);
 const [form,      setForm]      = useState(EMPTY_RECORD);
 const [editing,   setEditing]   = useState(null);
 const [saving,    setSaving]    = useState(false);
 const [search,    setSearch]    = useState('');

 useEffect(() => {
   api.get('/patients').then(r => setPatients(r.data)).catch(()=>{});
 }, []);

 const selectPatient = (p) => {
   setSelected(p);
   api.get(`/medical-records/patient/${p.id}`).then(r                    =>
setRecords(r.data)).catch(()=>{});
 };

 const save = async () => {
   setSaving(true);
   try {
     if (editing) {
       const res = await api.put(`/medical-records/${editing}`, form);
       setRecords(rs => rs.map(r => r.id===editing ? res.data : r));
     } else {
       const res = await api.post('/medical-records', { ...form, patient_id: selected.id });
       setRecords(rs => [res.data, ...rs]);
     }
     setModal(false);
   } catch (err) { alert('Помилка збереження медичного запису'); }
   finally { setSaving(false); }
 };

 const              filtered          =          patients.filter(p          =>
p.full_name.toLowerCase().includes(search.toLowerCase()));

 return (
   <div className="grid2" style={{ alignItems:'start' }}>
     <div>
       <div              className="page-header"><div              className="page-
title">Медкартки</div></div>
       <div className="search-bar">
         <input          placeholder="Пошук          пацієнта..."          value={search}
```

```
onChange={e=>setSearch(e.target.value)} />
      </div>
      <div className="card">
        {filtered.map(p => (
          <div key={p.id} onClick={() => selectPatient(p)} className="patient-item">
            {p.full_name}
          </div>
        ))}
      </div>
    </div>

    <div>
      {!selected ? (
        <div className="card">Оберіть пацієнта зі списку</div>
      ) : (
        <>
          <div className="page-header">
            <div className="page-title">{selected.full_name}</div>
            <button  className="btn  btn-primary"  onClick={()  =>  {
setForm({...EMPTY_RECORD,  patient_id:  selected.id});  setEditing(null);
setModal(true); }}>+ Новий запис</button>
          </div>
          <div className="card">
            <div className="timeline">
              {records.map(r => (
                <div key={r.id} className="tl-item">
                  <div                                          className="tl-date">{new
Date(r.created_at).toLocaleDateString('uk-UA')}</div>
                  <div className="tl-title">Діагноз: {r.diagnosis}</div>
                  <div className="tl-desc">Симптоми: {r.symptoms}</div>
                  <div className="tl-desc">Рекомендації: {r.recommendations}</div>
                </div>
              ))}
            </div>
          </div>
        </>
      )}
    </div>
  </div>
 );
}
```

Рисунок А.9 – Компонент інтерфейсу лікаря для ведення електронних медичних карток

```
import React, { useEffect, useState } from 'react';
import api from '../../utils/api';

const DAYS = [
  { key:'monday',    ua:'Понеділок' }, { key:'tuesday',   ua:'Вівторок'  },
  { key:'wednesday', ua:'Середа'    }, { key:'thursday',  ua:'Четвер'    },
  { key:'friday',    ua:"П'ятниця"  }, { key:'saturday',  ua:'Субота'    },
  { key:'sunday',    ua:'Неділя'    },
];

export default function DoctorSchedule() {
  const [schedule, setSchedule] = useState(
    DAYS.map(d => ({ work_day:d.key, start_time:'09:00', end_time:'17:00',
appointment_duration:60, is_active:false }))
  );
  const [saving, setSaving] = useState(false);

  useEffect(() => {
    api.get('/schedules/me').then(r => {
      if (r.data.length > 0) {
        setSchedule(DAYS.map(d => {
          const found = r.data.find(s => s.work_day === d.key);
          return found
            ? { ...found, start_time: found.start_time.slice(0,5), end_time:
found.end_time.slice(0,5) }
            : { work_day:d.key, start_time:'09:00', end_time:'17:00',
appointment_duration:60, is_active:false };
        }));
      }
    }).catch(()=>{});
  }, []);

  const save = async () => {
    setSaving(true);
    try {
      await api.put('/schedules/me', schedule.filter(s=>s.is_active));
    } catch (err) { alert('Помилка збереження графіка'); }
    finally { setSaving(false); }
  };

  return (
    <div>
      <div className="page-header">
```

```
      <div className="page-title">Графік роботи лікаря</div>
      <button className="btn btn-primary" onClick={save}
disabled={saving}>Зберегти графік</button>
    </div>
    <div className="card">
      <table>
        <thead>

<tr><th>Активний</th><th>День</th><th>Початок</th><th>Кінець</th><th>Тр
ивалість</th></tr>
        </thead>
        <tbody>
          {schedule.map((row, idx) => (
            <tr key={row.work_day}>
              <td>
                <input type="checkbox" checked={row.is_active} onChange={e =>
setSchedule(s => s.map((r, i) => i===idx ? { ...r, is_active:e.target.checked } : r))} />
              </td>
              <td>{DAYS[idx].ua}</td>
              <td><input type="time" value={row.start_time} onChange={e =>
setSchedule(s => s.map((r, i) => i===idx ? { ...r, start_time:e.target.value } : r))}
disabled={!row.is_active} /></td>
              <td><input type="time" value={row.end_time} onChange={e =>
setSchedule(s => s.map((r, i) => i===idx ? { ...r, end_time:e.target.value } : r))}
disabled={!row.is_active} /></td>
              <td>{row.appointment_duration} хв</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  </div>
 );
}
```

Рисунок А.10 – Управління розкладом прийомів та генерація часових слотів лікаря