ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ВЕБ-СИСТЕМА ДЛЯ ОБЛІКУ ПАЦІЄНТІВ МЕДИЧНОГО ЦЕНТРУ З ВИКОРИСТАННЯМ HTML 5, CSS 3, JAVASCRIPT ТА MICROSOFT SQL SERVER"»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Андрій ЛАЗНЕВИЙ
(підпис)

Виконав: Здобувач вищої освіти групи ТЦР-43

Андрій ЛАЗНЕВИЙ

Керівник: Андрій АРОНОВ
кандидат технічних наук

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лазневому Андрію Валерійовичу

1. Тема кваліфікаційної роботи: «Веб-система для обліку пацієнтів медичного центру з використанням HTML 5, CSS 3, JAVASCRIPT ТА MICROSOFT SQL SERVER»
керівник кваліфікаційної роботи _____,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року №51.
2. Строк подання кваліфікаційної роботи «18» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: інформаційні ресурси та програмні компоненти веб-системи обліку пацієнтів медичного центру; нормативні та методичні матеріали щодо використання сучасних веб-технологій, методів обробки та захисту персональних даних пацієнтів.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Аналіз предметної області та постановка задачі створення веб-системи для обліку пацієнтів медичного центру.
    2. Огляд існуючих інформаційних систем та технологій збору, збереження і обробки медичних даних, аналіз можливостей сучасних веб-фреймворків і бібліотек для реалізації систем управління пацієнтськими даними.
    3. Розробка, програмна реалізація, тестування та оцінювання ефективності роботи веб-системи обліку пацієнтів
5. Перелік графічного матеріалу: *презентація*

1. Аналіз технологій розробки.
2. Функціональні можливості системи.
3. Структура бази даних медичного центру.
4. Діаграма послідовності авторизації користувача.
5. Rest api та рольова модель доступу (rbac).
6. Апробація результатів дослідження.

6. Дата видачі завдання«20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та огляд сучасних веб-технологій і платформ для розробки систем обліку пацієнтів медичного центру	20.02.2026-23.02.2026	Виконано
2	Вивчення технічної документації веб-технологій	26.02.2026-3.03.2026	Виконано
3	Розробка архітектури програмної системи: модулі введення даних пацієнтів	10.03.2026-17.03.2026	Виконано
4	Програмна реалізація веб-системи обліку пацієнтів із використанням сучасних веб-технологій	20.03.2026-27.03.2026	Виконано
5	Реалізація та налаштування механізмів обробки даних, тестування системи на різних сценаріях використання	1.04.2026-13.04.2026	Виконано
6	Тестування застосунку	17.04.2026-26.04.2026	Виконано
7	Оформлення роботи: вступ, висновки, реферат	1.05.2026-3.05.2026	Виконано
8	Розробка демонстраційних матеріалів	5.02.2026-8.02.2026	Виконано
9	Попередній захист роботи	11.05.2026-22.05.2026	Виконано

Здобувач вищої освіти _____ Андрій ЛАЗНЕВИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 60 стор., 31 рис., 12 табл., 33 джерела.

Об'єкт дослідження – процеси обліку пацієнтів у медичних інформаційних системах та організація електронного документообігу в медичних центрах.

Предмет дослідження – методи та засоби розробки веб-систем для управління даними пацієнтів із використанням сучасних технологій веб-розробки та реляційних баз даних.

Мета роботи – автоматизація процесу веб-системи для обліку пацієнтів медичного центру з використанням HTML5, CSS3, JavaScript та Microsoft SQL Server, що забезпечує ефективне зберігання, обробку та доступ до медичних даних, а також реалізацію функцій запису на прийом і ведення медичної історії.

Короткий зміст роботи: в роботі розглянуто особливості організації медичних інформаційних систем та проаналізовано сучасні веб-технології для створення систем обліку пацієнтів. Проведено дослідження можливостей HTML5, CSS3, JavaScript і Microsoft SQL Server для реалізації веб-застосунку, визначено функціональні та нефункціональні вимоги до системи, а також спроєктовано структуру бази даних і клієнт-серверну архітектуру.

У результаті роботи розроблено веб-систему для обліку пацієнтів медичного центру, яка забезпечує реєстрацію пацієнтів, запис на прийом, зберігання медичних даних та взаємодію користувачів із системою через веб-інтерфейс. Реалізована система дозволяє підвищити ефективність роботи медичного персоналу, оптимізувати процес обробки інформації та забезпечити зручний доступ до даних.

КЛЮЧОВІ СЛОВА: ВЕБ-СИСТЕМА, ОБЛІК ПАЦІЄНТІВ, HTML5, CSS3, JAVASCRIPT, NODE.JS, MS SQL SERVER, JWT, CRUD, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах розвитку цифрових технологій та інформатизації сфери охорони здоров'я особливої актуальності набуває впровадження ефективних інформаційних систем для автоматизації процесів обліку та обробки даних пацієнтів. Традиційні підходи до ведення медичної документації, що базуються на паперових носіях або застарілих програмних рішеннях, є малоефективними, потребують значних часових витрат та підвищують ризик втрати або помилок у даних.

Застосування сучасних веб-технологій, таких як HTML5, CSS3 та JavaScript, у поєднанні з потужними системами управління базами даних, зокрема Microsoft SQL Server, дозволяє створювати гнучкі, масштабовані та доступні з будь-якого пристрою інформаційні системи. Такі системи забезпечують централізоване зберігання даних, швидкий доступ до інформації, автоматизацію запису пацієнтів, ведення історії хвороб та оптимізацію роботи медичного персоналу.

Крім того, актуальність дослідження обумовлена необхідністю підвищення якості медичного обслуговування, зменшення навантаження на персонал та забезпечення надійного зберігання персональних даних пацієнтів із дотриманням сучасних вимог інформаційної безпеки. У зв'язку з цим розробка веб-системи для обліку пацієнтів медичного центру є своєчасною та практично значущою задачею.

Мета роботи – автоматизація процесу веб-системи для обліку пацієнтів медичного центру з використанням HTML5, CSS3, JavaScript та Microsoft SQL Server, що забезпечує ефективне зберігання, обробку та доступ до медичних даних, а також реалізацію функцій запису на прийом і ведення медичної історії.

Об'єкт дослідження – процеси обліку пацієнтів у медичних інформаційних системах та організація електронного документообігу в медичних центрах.

Предмет дослідження – методи та засоби розробки веб-систем для управління даними пацієнтів із використанням сучасних технологій веб-розробки та реляційних баз даних.

Наукові завдання:

1. Провести аналіз предметної області обліку пацієнтів у медичних центрах та визначити основні проблеми існуючих підходів до ведення медичної інформації.

2. Дослідити сучасні веб-технології (HTML5, CSS3, JavaScript) та можливості їх застосування для створення інформаційних систем медичного призначення.

3. Проаналізувати можливості системи управління базами даних Microsoft SQL Server для організації зберігання та обробки медичних даних.

4. Розробити структуру та архітектуру веб-системи обліку пацієнтів медичного центру.

5. Спроєктувати логічну та фізичну модель бази даних для зберігання інформації про пацієнтів, записи на прийом та медичні послуги.

6. Реалізувати програмну частину веб-системи з використанням сучасних веб-технологій та забезпечити взаємодію з базою даних.

7. Забезпечити реалізацію основних функціональних можливостей системи (реєстрація пацієнтів, запис на прийом, пошук та перегляд інформації).

8. Провести тестування розробленої системи та оцінити ефективність її роботи.

Методи дослідження:

1. Аналіз сучасних підходів до побудови медичних інформаційних систем.

2. Проєктування клієнт-серверної архітектури веб-застосунку.

3. Розробка серверної частини з використанням Node.js та Express.

4. Проєктування та реалізація бази даних у Microsoft SQL Server.

5. Реалізація механізмів автентифікації та авторизації користувачів (JWT, RBAC).

6. Розробка клієнтської частини з використанням HTML5, CSS3 та JavaScript.

Використання сучасних веб-технологій, таких як HTML5, CSS3 та JavaScript, а також системи управління базами даних Microsoft SQL Server дозволяє

реалізувати ефективні механізми збору, обробки та відображення інформації про пацієнтів медичного центру. Для забезпечення зручної взаємодії користувача із системою застосовано веб-інтерфейс, створений із використанням сучасних бібліотек і технологій, що забезпечує наочне представлення даних та можливість ефективного керування процесами обліку пацієнтів.

Реалізована веб-система обліку пацієнтів дозволяє здійснювати введення, зберігання та обробку інформації у режимі реального часу, забезпечує швидкий доступ до даних про пацієнтів, запис на прийом, історію звернень та перелік наданих медичних послуг. Це сприяє підвищенню ефективності роботи медичного персоналу, оптимізації внутрішніх процесів медичного центру та покращенню якості обслуговування пацієнтів. Система також передбачає можливість адаптації до різних умов використання та подальшого розширення її функціональних можливостей.

Результати роботи можуть бути використані у навчальному процесі для демонстрації принципів розробки веб-застосунків та інформаційних систем, у дослідницьких проєктах, пов’язаних з обробкою та захистом медичних даних, а також у практичній діяльності медичних установ для автоматизації процесів обліку пацієнтів. Розроблена веб-система демонструє ефективність застосування сучасних веб-технологій як інструменту для створення надійних, зручних і масштабованих інформаційних сервісів у сфері охорони здоров’я.

Апробація. Лазневий Андрій Валерійович «НФОРМАЦІЙНА СИСТЕМА ОБЛІКУ ПАЦІЄНТІВ МЕДИЧНОГО ЦЕНТРУ» // VII Всеукраїнську науково-технічну конференцію «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. – К.: ДУІКТ, 2026. (подано до друку)

Лазневий Андрій Валерійович «ІНФОРМАЦІЙНА СИСТЕМА ОБЛІКУ ПАЦІЄНТІВ МЕДИЧНОГО ЦЕНТРУ» // Всеукраїнську науково-технічну конференцію «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026. (подано до друку)

1 ТЕОРЕТИЧНІ ОСНОВИ ВЕБ-СИСТЕМ ОБЛІКУ ПАЦІЄНТІВ

1.1 Особливості організації медичних інформаційних систем

Медичний центр це складна організаційна структура, діяльність якої спрямована на надання діагностичних, лікувальних та профілактичних послуг населенню. У сучасних умовах такі установи функціонують у середовищі постійного зростання обсягів інформації, що пов’язано з веденням медичних карток пацієнтів, історій хвороб, результатів обстежень, призначень лікарів та інших даних [1].

Важливість медичного центру полягає не лише у наданні медичних послуг, але й у забезпеченні безперервності лікувального процесу, своєчасного доступу до інформації про пацієнтів та підвищенні рівня організації роботи медичного персоналу. В умовах цифровізації суспільства медичні заклади все частіше переходять до використання інформаційних систем, які дозволяють автоматизувати процеси обліку, зменшити кількість помилок та підвищити оперативність обробки даних.

Медичні інформаційні системи (МІС) являють собою програмно-технічні комплекси, призначені для збору, зберігання, обробки та аналізу медичних даних. Вони забезпечують централізований доступ до інформації про пацієнтів, дозволяють вести електронні медичні картки, формувати звітність, а також автоматизувати процес запису на прийом та взаємодію між лікарями і пацієнтами. Використання таких систем значно підвищує ефективність роботи медичного центру та сприяє покращенню якості медичних послуг.

Основних функції, які забезпечують ефективну організацію роботи медичного центру: облік пацієнтів, ведення історії хвороби та запис на прийом (рис. 1.1).

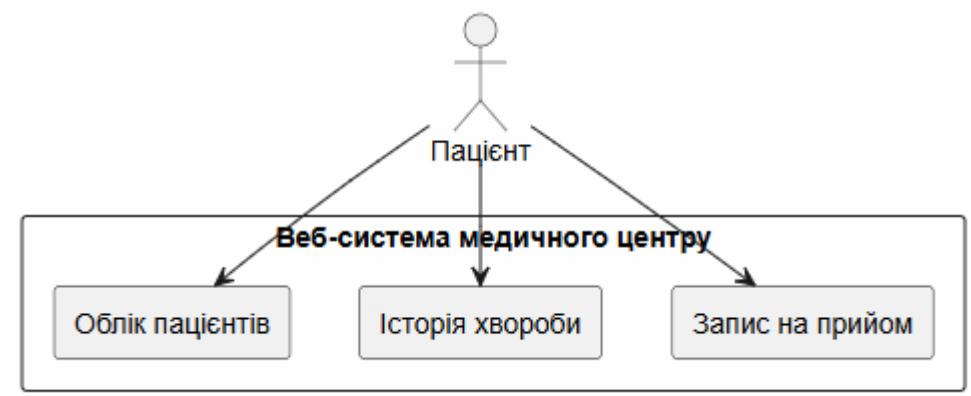

Рис. 1.1 Основні функції веб-системи обліку пацієнтів медичного центру

Функція обліку пацієнтів передбачає зберігання та управління персональними даними пацієнтів, включаючи їхні анкетні відомості, контактну інформацію та ідентифікаційні дані. Це дозволяє швидко знаходити необхідну інформацію та забезпечує зручність роботи медичного персоналу.

Ведення історії хвороби є важливою складовою медичної інформаційної системи, оскільки дозволяє накопичувати та зберігати інформацію про стан здоров’я пацієнта, результати обстежень, призначення лікарів та динаміку лікування. Це забезпечує безперервність медичного обслуговування та підвищує якість прийняття клінічних рішень [2].

Функція запису на прийом забезпечує можливість планування візитів пацієнтів до лікарів, формування розкладу роботи медичного персоналу та оптимізацію використання часу. Завдяки автоматизації цього процесу зменшується кількість черг, підвищується зручність для пацієнтів та ефективність роботи медичного центру.

Для наочного відображення взаємодії основних функцій медичної інформаційної системи доцільно представити їх у вигляді структурної схеми (Рис. 1.1).

У багатьох медичних установах досі існує низка проблем, пов’язаних з організацією обліку пацієнтів та веденням медичної документації. Основними з них є використання паперових носіїв, дублювання даних та наявність помилок у процесі обробки інформації.

Паперовий облік є одним із найбільш застарілих підходів до зберігання інформації, який потребує значних витрат часу на пошук та обробку даних. Крім того, паперові документи можуть бути втрачені, пошкоджені або недоступні у потрібний момент, що негативно впливає на якість медичного обслуговування.

Дублювання даних – це також проблема, яка виникає через відсутність єдиної централізованої системи. Інформація про одного і того ж пацієнта може зберігатися у різних документах або підрозділах, що ускладнює її актуалізацію та призводить до неузгодженості даних [3].

Ручне введення інформації підвищує ймовірність неточностей, пропусків або неправильного трактування даних. Це може негативно впливати на прийняття медичних рішень та створювати ризики для здоров’я пацієнтів.

Для кращого розуміння основних проблем традиційного підходу до обліку пацієнтів доцільно представити їх у вигляді структурної схеми (рис. 1.2).

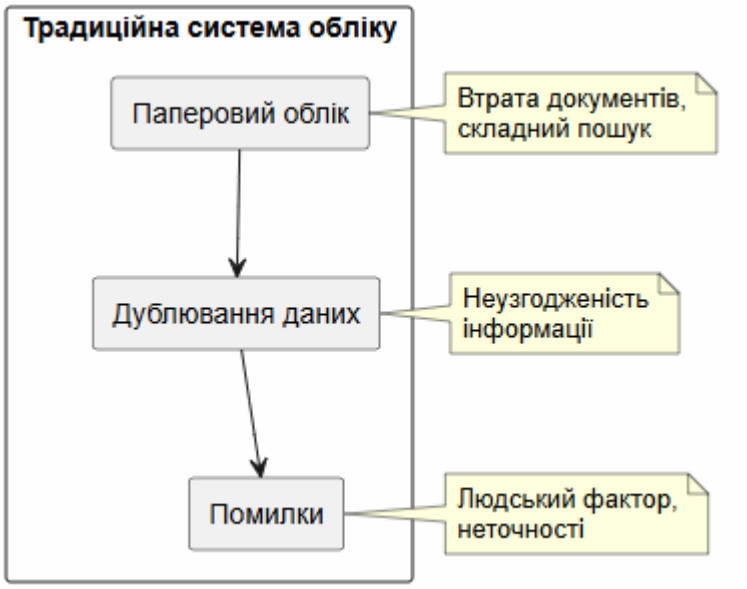

Рис. 1.2 Основні проблеми традиційного обліку пацієнтів

З урахуванням виявлених проблем традиційних підходів до обліку пацієнтів виникає необхідність формування чітких вимог до сучасних медичних інформаційних систем. До ключових вимог належать конфіденційність, надійність та доступність.

Конфіденційність – медичні дані належать до категорії персональної

інформації та підлягають захисту. Система повинна забезпечувати контроль доступу до даних, використання механізмів автентифікації та авторизації, а також захист інформації від несанкціонованого доступу. Це особливо важливо у контексті дотримання сучасних стандартів інформаційної безпеки.

Надійність системи передбачає її стабільну роботу без збоїв та втрати даних., для цього необхідно реалізувати механізми резервного копіювання, відновлення інформації, а також забезпечити цілісність даних під час їх обробки та зберігання. Надійність безпосередньо впливає на безперервність роботи медичного центру [4].

Доступність означає можливість отримання необхідної інформації у будь-який момент часу відповідними користувачами системи. Веб-система повинна забезпечувати швидкий доступ до даних, підтримку одночасної роботи декількох користувачів та адаптивність до різних пристроїв. Це дозволяє медичному персоналу оперативно отримувати інформацію та приймати обґрунтовані рішення.

Для узагальнення основних вимог до веб-системи обліку пацієнтів доцільно представити їх у вигляді структурної схеми (рис. 1.3).

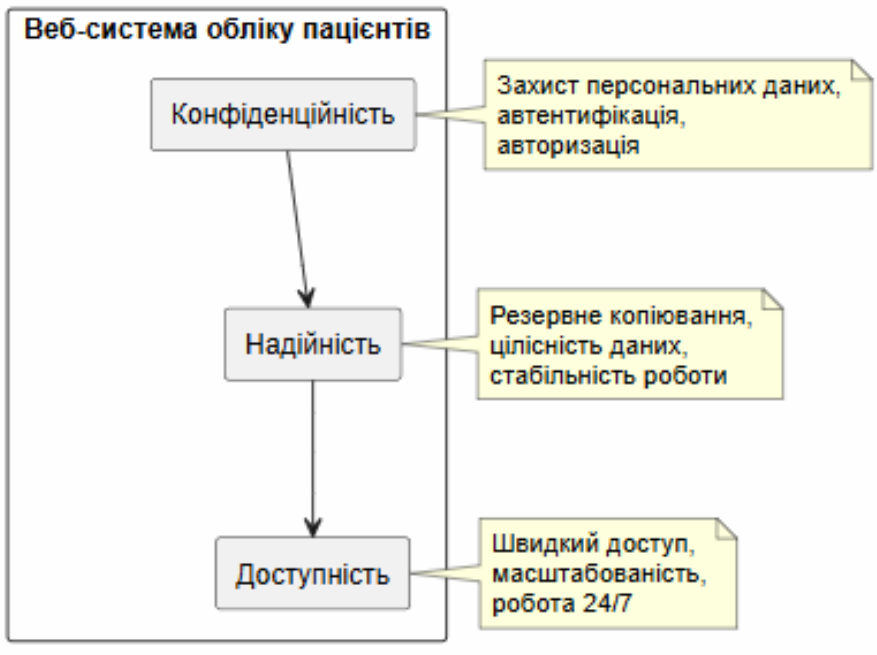

Рис. 1.3 Основні вимоги до веб-системи обліку пацієнтів

У центрі такої системи функціонування медичної інформаційної системи обліку пацієнтів знаходиться веб-застосунок, який забезпечує обробку запитів

користувачів, взаємодію з базою даних та відображення інформації.

Основними учасниками системи є пацієнт і лікар, які взаємодіють із веб-системою через інтерфейс користувача. Пацієнт може вводити свої дані, записуватися на прийом та переглядати інформацію, тоді як лікар має доступ до медичних записів, історії хвороби та результатів обстежень. Уся інформація обробляється системою та зберігається у централізованій базі даних, що забезпечує її цілісність і доступність. Загальна структура взаємодії компонентів веб-системи обліку пацієнтів представлена на рис. 1.4.

Рис. 1.4 Схема медичної інформаційної системи обліку пацієнтів

Впровадження веб-систем обліку пацієнтів є необхідним кроком у напрямі цифрової трансформації медичної галузі, що сприяє підвищенню якості обслуговування та оптимізації внутрішніх процесів медичних установ.

1.2 Аналіз сучасних веб-технологій (HTML5, CSS3, JavaScript)

Веб-технології відіграють ключову роль у розробці інформаційних систем, зокрема веб-систем обліку пацієнтів медичного центру. Використання таких технологій дозволяє створювати зручні, функціональні та адаптивні інтерфейси, які

забезпечують ефективну взаємодію користувачів із системою. Основу клієнтської частини веб-застосунку становлять HTML5, CSS3 та JavaScript, кожна з яких виконує важливу функцію у процесі розробки [5].

Мова розмітки HTML5 використовується для створення структури веб-сторінки (рис. 1.5). Вона дозволяє визначити логічну організацію контенту, включаючи форми введення даних, таблиці, кнопки та інші елементи інтерфейсу. У контексті веб-системи обліку пацієнтів HTML5 забезпечує формування сторінок для реєстрації пацієнтів, перегляду інформації, запису на прийом та інших функціональних можливостей системи. Використання сучасних семантичних тегів підвищує зручність розробки та покращує доступність веб-застосунку.

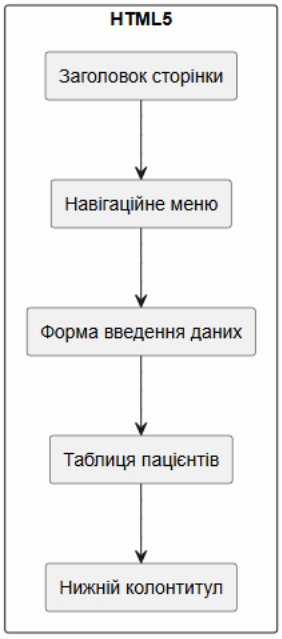

Рис. 1.5 HTML5 як основа структури веб-сторінки

CSS3 відповідає за візуальне оформлення та дизайн веб-інтерфейсу. Завдяки цій технології забезпечується стилізація елементів сторінки, включаючи кольори, шрифти, відступи та розташування компонентів [6]. Особливе значення має можливість створення адаптивного дизайну, що дозволяє коректно відображати веб-застосунок на різних пристроях, таких як комп’ютери, планшети та смартфони. У медичних інформаційних системах це є важливим фактором, оскільки доступ до даних може здійснюватися з різних платформ (рис. 1.6).

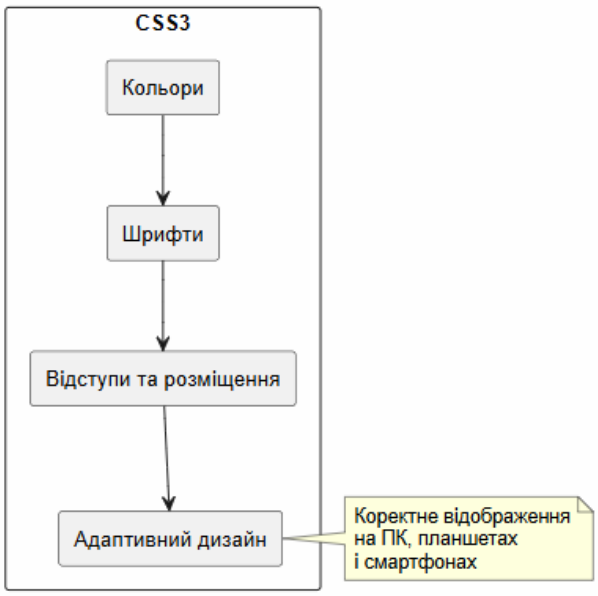

Рис. 1.6 CSS3 як засіб оформлення та адаптації інтерфейсу

JavaScript є мовою програмування, яка забезпечує реалізацію логіки роботи веб-застосунку та взаємодію з користувачем (рис. 1.7). За допомогою JavaScript здійснюється обробка подій, перевірка введених даних, динамічне оновлення сторінок без їх перезавантаження, а також взаємодія з серверною частиною через API. У веб-системі обліку пацієнтів ця технологія використовується для реалізації функцій додавання, редагування та пошуку інформації, а також для забезпечення зручності та швидкості роботи системи.

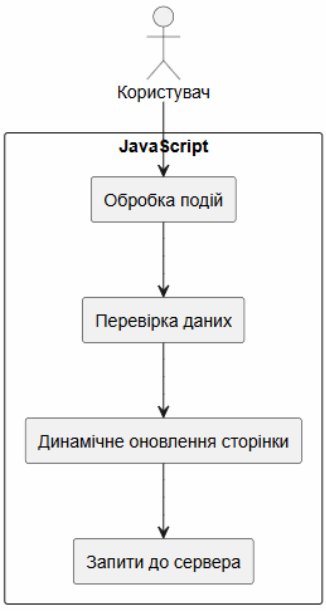

Рис. 1.7 JavaScript як засіб реалізації логіки та взаємодії

Кожна з технологій – HTML5, CSS3 та JavaScript – виконує окрему функцію у процесі розробки веб-застосунків, проте лише їх комплексне використання забезпечує створення повноцінної, зручної та ефективної системи [7].

З метою визначення ролі кожної технології у структурі веб-системи обліку пацієнтів, а також їх основних можливостей і переваг, наведено порівняльну характеристику у вигляді таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика веб-технологій HTML5, CSS3 та JavaScript

Технологія	Призначення	Основні можливості	Переваги	Роль у системі
HTML5	Структура веб-сторінки	Формування елементів інтерфейсу (форми, таблиці, кнопки), семантична розмітка	Простота використання, підтримка сучасних стандартів, сумісність з браузерами	Створення структури сторінок системи (реєстрація, запис на прийом, перегляд даних)
CSS3	Оформлення та дизайн	Стилізація елементів, адаптивний дизайн, анімації, розташування блоків	Покращення зовнішнього вигляду, адаптивність під різні пристрої, гнучкість	Забезпечення зручного та привабливого інтерфейсу користувача
JavaScript	Логіка та взаємодія	Обробка подій, валідація форм, робота з API, динамічне оновлення сторінки	Інтерактивність, швидкість роботи, можливість створення SPA	Реалізація функціональності системи та взаємодії з сервером

Фронтенд являє собою клієнтську частину веб-застосунку, що виконується у браузері користувача та забезпечує відображення інтерфейсу, обробку дій користувача та передачу даних на сервер. Саме фронтенд формує користувацький досвід, забезпечуючи зручність, інтуїтивність та ефективність роботи з системою.

У веб-системі обліку пацієнтів фронтенд реалізується за допомогою HTML5, CSS3 та JavaScript і включає форми введення даних, сторінки перегляду інформації, а також елементи керування записами пацієнтів [8].

Веб-застосунок функціонує на стороні сервера, а користувач взаємодіє з ним через браузер, що значно спрощує процес впровадження та обслуговування системи. Оновлення та модифікації виконуються централізовано, без потреби втручання з боку користувачів.

Веб-застосунки характеризуються високим рівнем масштабованості. Це означає, що система може бути легко розширена для обслуговування більшої кількості користувачів або додавання нових функціональних можливостей без суттєвих змін у її архітектурі. Така властивість є особливо важливою для медичних центрів, які можуть розширювати свою діяльність або збільшувати обсяг оброблюваних даних.

1.3 Реляційні бази даних у медичних системах (MS SQL Server)

Реляційна база даних являє собою структуровану систему зберігання інформації, у якій дані організовані у вигляді таблиць, що складаються з рядків і стовпців. Кожна таблиця відповідає певній сутності, наприклад пацієнтам, лікарям або записам на прийом, а зв’язки між таблицями встановлюються за допомогою ключів. Такий підхід дозволяє забезпечити цілісність даних, уникнути дублювання інформації та ефективно виконувати запити до бази даних.

Нереляційні бази даних, або NoSQL-системи, відрізняються більш гнучкою структурою зберігання даних. У таких системах інформація може зберігатися у вигляді документів, ключ-значення, графів або колонкових структур [9]. Це дозволяє працювати з неструктурованими або напівструктурованими даними та забезпечує високу масштабованість і продуктивність при обробці великих обсягів інформації.

Вибір між реляційними та нереляційними базами даних залежить від специфіки задачі. У медичних інформаційних системах, де важлива точність,

структурованість та цілісність даних, найчастіше використовуються саме реляційні бази даних. Вони забезпечують чітку організацію інформації про пацієнтів, медичні записи та інші сутності, що є критично важливим для надійної роботи системи.

Для узагальнення відмінностей між реляційними та нереляційними базами даних доцільно представити їх порівняльну характеристику (табл. 1.2).

Таблиця 1.2

Порівняльна характеристика реляційних та нереляційних баз даних

Критерій	Реляційні бази даних (SQL)	Нереляційні бази даних (NoSQL)
Структура даних	Таблична (рядки і стовпці)	Гнучка (документи, ключ-значення, графи)
Схема даних	Фіксована, заздалегідь визначена	Динамічна, може змінюватися
Зв’язки між даними	Чітко визначені (через ключі)	Часто відсутні або неявні
Цілісність даних	Висока (ACID-властивості)	Обмежена (часто BASE)
Масштабованість	Вертикальна	Горизонтальна
Продуктивність	Висока для складних запитів	Висока для великих обсягів даних
Використання	Банківські системи, медицина, облік	Big Data, соціальні мережі
Приклади	MS SQL Server, MySQL, PostgreSQL	MongoDB, Cassandra, Redis

Найдоцільніше рішення для реалізації таких завдань є Microsoft SQL Server, який належить до реляційних систем управління базами даних і широко застосовується у корпоративних інформаційних системах.

Перевагою MS SQL Server є високий рівень надійності, що особливо важливо для медичних інформаційних систем. У процесі роботи медичного центру дані про пацієнтів, записи на прийом, медичні послуги та персонал повинні зберігатися без втрат і спотворень. Система повинна гарантувати стабільну роботу навіть при значній кількості запитів, а також забезпечувати можливість резервного копіювання та відновлення інформації [10]. Використання MS SQL Server дозволяє

підтримувати цілісність даних, мінімізувати ризик їх втрати та забезпечити безперервність функціонування веб-системи.

У медичній сфері дані пацієнтів належать до конфіденційної інформації, тому доступ до них повинен бути обмеженим і контрольованим. MS SQL Server підтримує механізми автентифікації користувачів, розмежування прав доступу, керування ролями та журналювання дій. Це дозволяє забезпечити захист персональних даних від несанкціонованого перегляду, зміни або видалення. Для веб-системи обліку пацієнтів такі можливості є критично важливими, оскільки помилки або витік інформації можуть мати серйозні наслідки як для медичного закладу, так і для самих пацієнтів.

Система здатна ефективно працювати з великими обсягами структурованих даних, швидко виконувати SQL-запити, сортування, фільтрацію та пошук інформації. Для медичного центру це означає можливість оперативно отримувати відомості про пацієнта, знаходити записи на прийом, аналізувати завантаження лікарів та формувати необхідну звітність. Висока продуктивність бази даних безпосередньо впливає на швидкість роботи всієї веб-системи та зручність її використання персоналом.

У розроблюваній системі в базі даних доцільно зберігати кілька основних категорій інформації. Насамперед це дані про пацієнтів, до яких належать прізвище, ім’я, по батькові, дата народження, контактний номер телефону, адреса електронної пошти, адреса проживання та інші персональні відомості. Також важливими є дані про записи на прийом, які відображають дату і час візиту, лікаря, до якого записаний пацієнт, обрану послугу, а також статус запису. Окремо повинна зберігатися інформація про лікарів, зокрема їхні персональні дані, спеціалізація, контактна інформація, графік роботи та інші відомості, необхідні для організації прийому [11].

Для організації такої бази даних використовується ER-логіка (Entity-Relationship), яка передбачає подання системи у вигляді сутностей, їх атрибутів і зв’язків між ними. У даному випадку основними сутностями є «Пацієнт», «Лікар», «Запис на прийом» та «Послуга». Кожна з цих сутностей представлена окремою

таблицею в базі даних. Сутність «Пацієнт» пов’язана із сутністю «Запис на прийом», оскільки один пацієнт може мати декілька записів. Аналогічно сутність «Лікар» також пов’язана із записами, оскільки один лікар може проводити багато прийомів. Сутність «Послуга» пов’язується із записами для визначення того, яку саме медичну послугу отримує пацієнт під час візиту.

У реляційній моделі важливу роль відіграють ключі, які забезпечують ідентифікацію записів та підтримку зв’язків між таблицями. Первинний ключ (PK, Primary Key) використовується для унікальної ідентифікації кожного запису в таблиці. Наприклад, у таблиці пацієнтів це може бути PatientID, у таблиці лікарів – DoctorID, у таблиці послуг – ServiceID, а у таблиці записів – AppointmentID. Зовнішній ключ (FK, Foreign Key) використовується для встановлення зв’язків між таблицями [12]. Наприклад, у таблиці записів на прийом можуть міститися поля PatientID, DoctorID та ServiceID, які є зовнішніми ключами та посилаються відповідно на первинні ключі таблиць пацієнтів, лікарів і послуг.

Підхід дозволяє уникнути дублювання інформації, забезпечити цілісність даних і спростити процес виконання запитів.

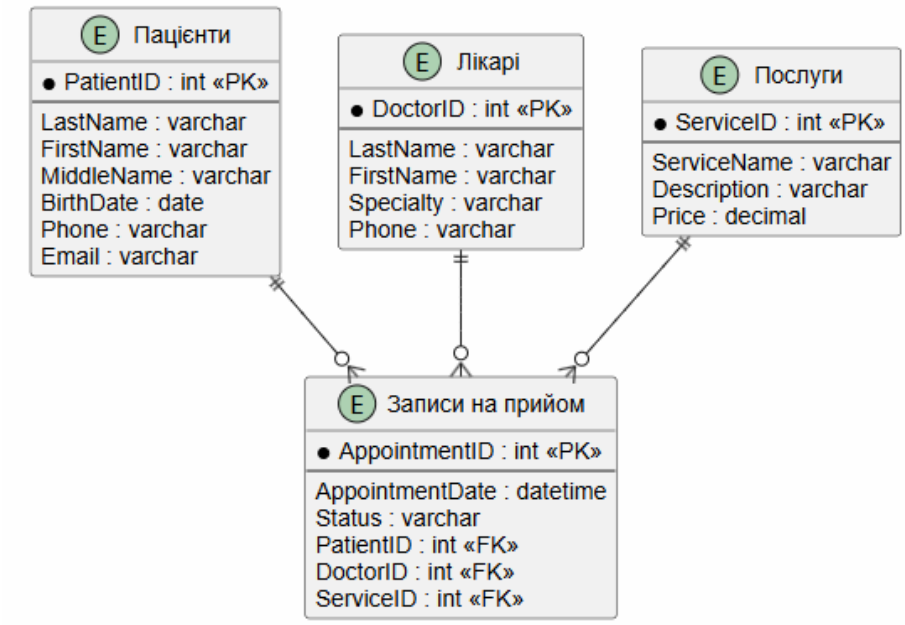

Рис. 1.8 ER-діаграма бази даних веб-системи

Наприклад, для отримання повної інформації про конкретний запис на прийом система може об’єднати дані одразу з кількох таблиць: відомості про

пацієнта, лікаря, послугу та параметри самого запису. Саме завдяки використанню реляційної моделі та правильно спроєктованих зв’язків база даних стає логічно впорядкованою, зручною в супроводі та придатною до подальшого розширення.

Для узагальнення структури бази даних веб-системи обліку пацієнтів та визначення ролі основних сутностей доцільно представити їх у вигляді таблиці. У ній наведено ключові об’єкти системи, їх призначення та основні атрибути, що використовуються для організації зберігання інформації (таблиця 1.3).

Таблиця 1.3

Основні сутності бази даних та їх призначення

Сутність	Призначення	Основні атрибути
Пацієнти	Зберігання персональних даних пацієнтів	PatientID, ПІБ, дата народження, телефон
Лікарі	Зберігання інформації про медичний персонал	DoctorID, ПІБ, спеціалізація
Послуги	Зберігання переліку медичних послуг	ServiceID, назва, опис, вартість
Записи на прийом	Фіксація візитів пацієнтів	AppointmentID, дата, статус, зовнішні ключі

Використання Microsoft SQL Server у веб-системі обліку пацієнтів є обґрунтованим рішенням, оскільки ця система управління базами даних забезпечує високий рівень надійності, безпеки та продуктивності. Реляційна модель даних із чітким визначенням сутностей, первинних і зовнішніх ключів дозволяє створити цілісну структуру зберігання інформації, необхідну для ефективної роботи медичного центру.

Висновок до розділу 1

Розглянуто теоретичні основи побудови веб-систем обліку пацієнтів медичного центру. Проведено аналіз особливостей організації медичних інформаційних систем, визначено їх основні функції, зокрема облік пацієнтів,

ведення історії хвороби та організацію запису на прийом. Також було виявлено основні проблеми традиційних підходів до ведення медичної документації, серед яких паперовий облік, дублювання даних та наявність помилок, що обґрунтовує необхідність впровадження сучасних інформаційних рішень.

Проаналізовано сучасні веб-технології, такі як HTML5, CSS3 та JavaScript, які є основою для розробки клієнтської частини веб-застосунків. Визначено їх роль у формуванні структури, дизайну та логіки роботи системи, а також обґрунтовано доцільність їх використання для створення зручного та функціонального інтерфейсу користувача. Окрему увагу приділено перевагам веб-застосунків, серед яких доступність, відсутність необхідності встановлення та масштабованість.

Розглянуто особливості використання реляційних баз даних у медичних інформаційних системах, зокрема можливості Microsoft SQL Server як надійної, безпечної та продуктивної системи управління даними. Описано структуру зберігання інформації, основні сутності бази даних та їх взаємозв’язки, а також принципи використання первинних і зовнішніх ключів для забезпечення цілісності даних.

2 ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ ОБЛІКУ ПАЦІЄНТІВ

2.1 Формування функціональних та нефункціональних вимог

На етапі проєктування веб-системи обліку пацієнтів медичного центру важливим завданням є визначення функціональних вимог, які описують основні можливості системи та її поведінку під час взаємодії з користувачем. Саме функціональні вимоги визначають, які дії повинна виконувати система для забезпечення ефективної роботи медичного персоналу та обслуговування пацієнтів [13].

До функціональних вимог розроблюваної системи належить можливість реєстрації пацієнтів. Система повинна забезпечувати введення, збереження та редагування персональних даних, що дозволяє формувати єдину базу пацієнтів медичного центру. Це спрощує доступ до інформації та підвищує ефективність роботи з даними.

Важливою функцією є запис пацієнтів на прийом до лікаря. Система повинна дозволяти обирати лікаря, дату та час візиту, а також забезпечувати збереження відповідної інформації у базі даних. Це дозволяє оптимізувати розклад роботи лікарів та зменшити навантаження на реєстратуру.

Також система повинна забезпечувати можливість перегляду інформації про пацієнтів та їхні записи. Користувачі повинні мати доступ до історії звернень, даних про лікарів і надані послуги, що сприяє більш ефективному прийняттю рішень.

Функція пошуку дозволяє швидко знаходити необхідну інформацію за заданими параметрами, такими як прізвище пацієнта, дата запису або лікар. Це значно скорочує час обробки інформації та підвищує зручність роботи із системою.

Система повинна підтримувати редагування даних, що дозволяє вносити зміни до інформації про пацієнтів, записи на прийом та інші сутності. Це забезпечує актуальність і достовірність даних у системі.

Для узагальнення функціональних вимог до веб-системи обліку пацієнтів доцільно представити їх у вигляді таблиці (таблиця 2.1).

Таблиця 2.1

Функціональні вимоги до веб-системи обліку пацієнтів

№	Функція	Опис
1	Реєстрація пацієнта	Введення, збереження та редагування персональних даних пацієнта
2	Запис на прийом	Створення запису до лікаря з вибором дати, часу та послуги
3	Перегляд даних	Отримання інформації про пацієнтів, записи та історію звернень
4	Пошук	Пошук пацієнтів або записів за заданими параметрами
5	Редагування	Оновлення та зміна існуючих даних у системі

Крім функціональних вимог, важливе значення мають нефункціональні вимоги, які визначають якісні характеристики системи та впливають на ефективність її використання. Нефункціональні вимоги описують умови, за яких система повинна працювати, а також рівень її продуктивності, безпеки та зручності для користувачів.

Веб-застосунок повинен забезпечувати швидке виконання запитів, оперативне відображення інформації та мінімальний час відгуку при взаємодії користувача із системою. Це особливо важливо для медичних установ, де швидкий доступ до даних може впливати на прийняття рішень.

Безпека повинна забезпечувати захист інформації від несанкціонованого доступу, включати механізми автентифікації та авторизації користувачів, а також гарантувати цілісність і конфіденційність даних.

Веб-система повинна підтримувати збільшення кількості користувачів, обсягу даних та функціональних можливостей, що є важливим для розвитку медичного центру.

Зручність використання (юзабіліті) визначає рівень комфорту взаємодії користувача із системою. Інтерфейс повинен бути інтуїтивно зрозумілим, простим

у використанні та не потребувати тривалого навчання. Це дозволяє медичному персоналу швидко адаптуватися до роботи із системою.

Для узагальнення нефункціональних вимог до веб-системи обліку пацієнтів їх представлено у вигляді таблиці (таблиця 2.2).

Таблиця 2.2

Нефункціональні вимоги до веб-системи обліку пацієнтів

№	Вимога	Опис
1	Швидкість	Забезпечення швидкого відгуку системи та обробки запитів
2	Безпека	Захист персональних даних, контроль доступу, автентифікація
3	Масштабованість	Можливість розширення системи при зростанні навантаження
4	Зручність	Інтуїтивний інтерфейс та простота використання

На рис. 2.1 представлено use case діаграму веб-системи обліку пацієнтів медичного центру. Вона відображає основних користувачів системи та перелік функцій, з якими вони взаємодіють у процесі роботи.

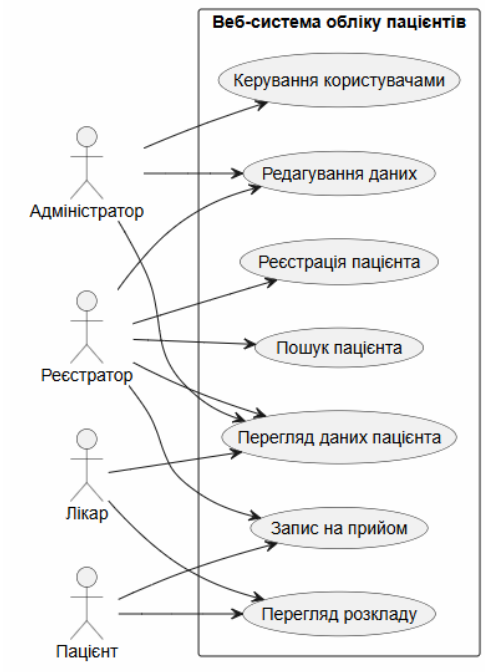

Рис. 2.1 Use Case діаграма веб-системи обліку пацієнтів медичного центру

Зокрема, реєстратор виконує операції реєстрації пацієнтів, запису на прийом, пошуку, перегляду та редагування даних. Лікар має доступ до перегляду даних пацієнтів і розкладу прийому, пацієнт може записуватися на прийом і переглядати доступний розклад, а адміністратор відповідає за керування користувачами та контроль даних у системі.

2.2 . Розробка архітектури клієнт-серверної системи

Клієнт-серверна система медичного центру реалізована за трирівневою архітектурою: рівень представлення (Frontend), рівень бізнес-логіки (Backend API) та рівень даних (SQL Server). Такий підхід забезпечує розділення відповідальності між компонентами, спрощує супровід системи та дозволяє масштабувати окремі частини незалежно.

На рівні представлення використано HTML/CSS/JavaScript, де клієнт формує HTTP-запити до REST API. На серверному рівні застосовано Node.js + Express, який виконує автентифікацію, авторизацію, обробку бізнес-правил і доступ до БД. Рівень даних реалізовано на SQL Server (Azure SQL Edge у Docker-середовищі), де зберігаються користувачі, пацієнти, лікарі, записи на прийом та медичні записи.

Мережеву взаємодію компонентів організовано через Docker Compose: вебклієнт доступний на порту 8888, API - на 3000, СУБД - на 1433. Це підтверджується конфігурацією розгортання у файлі docker-compose.yml. Серверна частина централізовано ініціалізує маршрути API (/api/auth, /api/patients, /api/doctors, /api/appointments, /api/medical-records) та підключення до БД, що реалізовано у server.js. На клієнті є окремий модуль доступу до API з базовим URL, обробкою fetch-запитів і автоматичним додаванням JWT-токена в заголовок Authorization, що реалізовано у api.js.

Безпека доступу до ресурсів побудована на JWT-механізмі та рольовій моделі (admin, doctor, patient). Перевірка токена виконується в middleware auth.js, а перевірка ролей - у roles.js. Це дозволяє контролювати доступ до критичних операцій, наприклад створення/видалення пацієнтів лише для ролі адміністратора

[16].

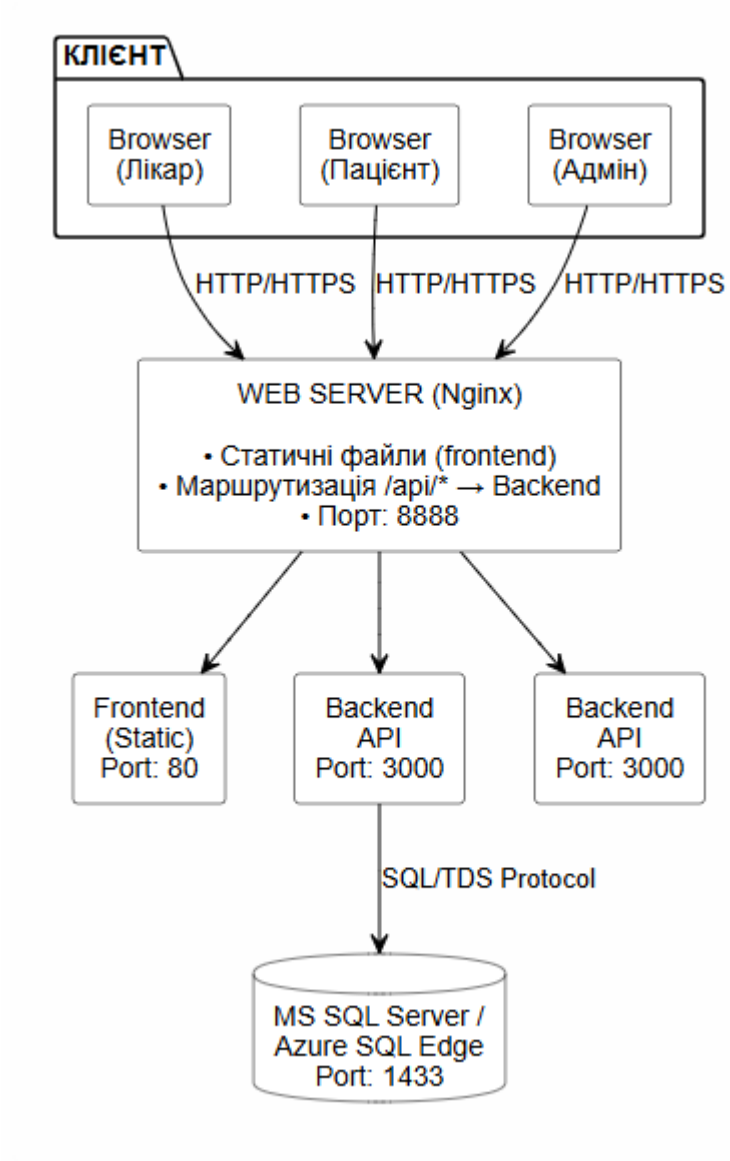

Рис. 2.1 Архітектура системи

На рисунку 2.2 представлено конфігурацію розгортання компонентів системи у середовищі Docker Compose.

```yaml
version: '3.8'

services:
  # Azure SQL Edge (підтримує ARM/Apple Silicon)
  sqlserver:
    image: mcr.microsoft.com/azure-sql-edge:latest
    container_name: medical-center-db
    environment:
      ACCEPT_EULA: Y
      MSSQL_SA_PASSWORD: "MedCenter2024!"
      MSSQL_PID: Developer
      MSSQL_USER: sa
    ports:
      - "1433:1433"
    volumes:
      - sqlserver_data:/var/opt/mssql
    networks:
      - medical-network
    restart: unless-stopped

  # Backend API
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: medical-center-api
    environment:
      DB_SERVER: sqlserver
      DB_NAME: MedicalCenter
      DB_USER: sa
      DB_PASSWORD: "MedCenter2024!"
      JWT_SECRET: your-secret-key-change-in-production
      PORT: 3000
      NODE_ENV: production
    ports:
      - "3000:3000"
    depends_on:
      - sqlserver
    networks:
      - medical-network
    restart: unless-stopped
    volumes:
      - ./backend:/app
      - /app/node_modules

  # Frontend
```

Рис. 2.2 Конфігурація контейнерів клієнт-серверної системи

На рисунку 2.2 показано, що кожен компонент розгортається як окремий сервіс із власними параметрами середовища та мережевими портами, що забезпечує модульність і керованість інфраструктури [16, 17].

На рисунку 2.3 наведено фрагмент коду ініціалізації серверного застосунку

та підключення маршрутів API.

h Medical-center — server.js — Medical-center

backend > JS server.js > ...

```
1   const express = require('express');
2   const cors = require('cors');
3   require('dotenv').config();
4
5   const { connectDB } = require('./config/database');
6   const authRoutes = require('./routes/auth');
7   const patientRoutes = require('./routes/patients');
8   const doctorRoutes = require('./routes/doctors');
9   const appointmentRoutes = require('./routes/appointments');
10  const medicalRecordRoutes = require('./routes/medical-records');
11  const errorHandler = require('./middleware/errorHandler');
12
13  const app = express();
14  const PORT = process.env.PORT || 3000;
15
16  app.use(cors());
17  app.use(express.json());
18  app.use(express.urlencoded({ extended: true }));
19
20  // Routes
21  app.use('/api/auth', authRoutes);
22  app.use('/api/patients', patientRoutes);
23  app.use('/api/doctors', doctorRoutes);
24  app.use('/api/appointments', appointmentRoutes);
25  app.use('/api/medical-records', medicalRecordRoutes);
26
27  // Error handling
28  app.use(errorHandler);
29
30  // 404 handler
31  app.use((req, res) => {
32    res.status(404).json({ message: 'Маршрут не знайдено' });
33  });
34
35  // Connect to database and start server
36  async function startServer() {
37    try {
38      await connectDB();
39      app.listen(PORT, () => {
40        console.log(`Server is running on port ${PORT}`);
41      });
42    } catch (error) {
43      console.error('Failed to start server:', error);
44      process.exit(1);
45    }
46  }
47
48  startServer();
49
```

Рис. 2.3 Ініціалізація REST API на сервері

На рисунку 2.3 показано централізовану точку конфігурації API, що спрощує подальше розширення системи новими модулями [18].

На рисунку 2.4 представлено реалізацію універсального клієнтського запиту до серверного API.

```
// API Configuration
const API_BASE_URL = 'http://localhost:3000/api';

// HTTP Request Helper
async function apiRequest(endpoint, options = {}) {
  const url = `${API_BASE_URL}${endpoint}`;

  const config = {
    headers: {
      'Content-Type': 'application/json',
      ...options.headers
    },
    ...options
  };

  // Add auth token if available
  const token = localStorage.getItem('token');
  if (token) {
    config.headers['Authorization'] = `Bearer ${token}`;
  }

  if (config.body && typeof config.body === 'object') {
    config.body = JSON.stringify(config.body);
  }

  try {
    const response = await fetch(url, config);
    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.message || 'Помилка запиту');
    }

    return data;
  } catch (error) {
    console.error('API Error:', error);
    throw error;
  }
}

// API Methods
const api = {
  // Auth
  login: (email, password) => apiRequest('/auth/login', {
    method: 'POST',
    body: { email, password }
  }),

  register: (userData) => apiRequest('/auth/register', {
```

Рис. 2.4 Клієнтський модуль взаємодії з API

На рисунку 2.4 показано, що клієнт використовує єдину функцію запитів, яка уніфікує обробку HTTP-викликів та механізм авторизації.

На рисунку 2.5 представлено компонентну UML-діаграму клієнт-серверної архітектури системи.

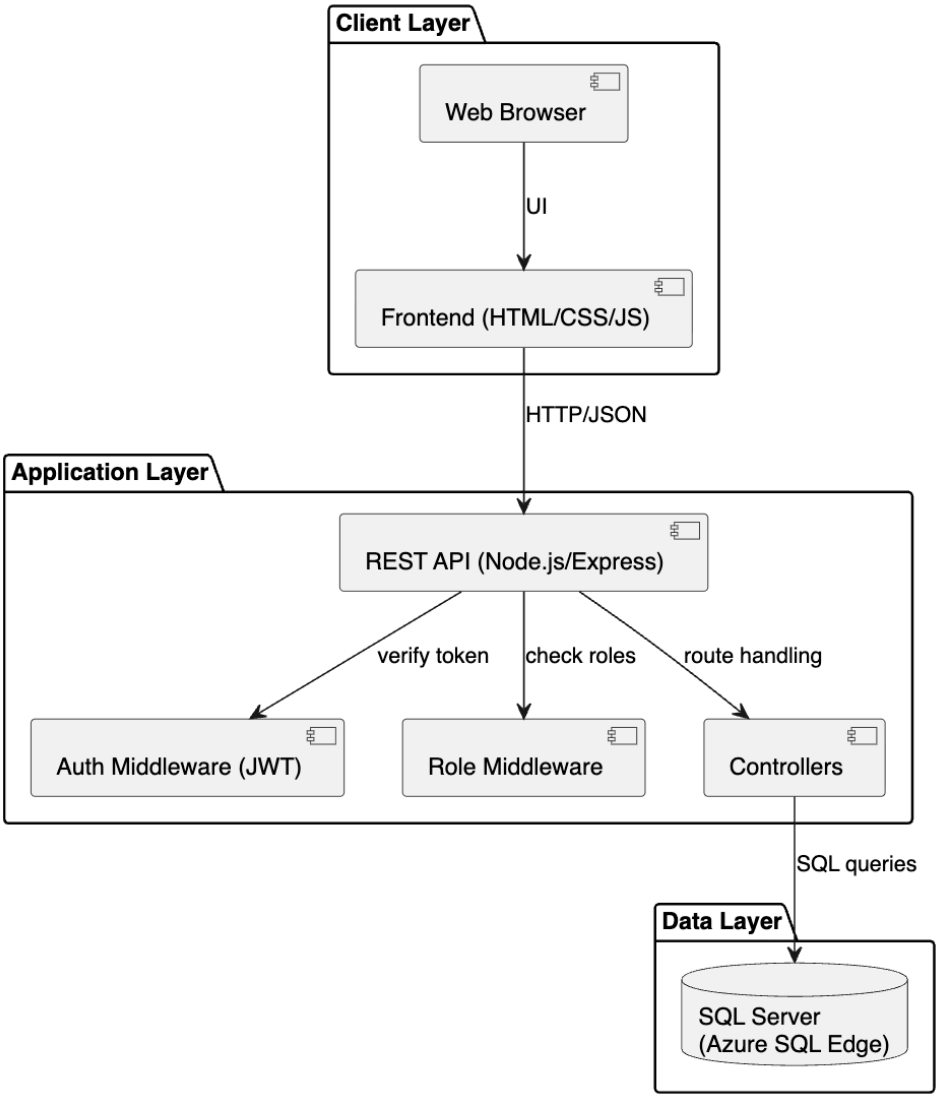

Рис. 2.5 Компонентна діаграма архітектури

На рисунку 2.5 показано поділ системи на три логічні рівні та основні напрями взаємодії між ними, що відповідає класичній моделі client-server і забезпечує структурованість програмного рішення.

На рисунку 2.6 наведено UML-діаграму послідовності процесу автентифікації користувача [20].

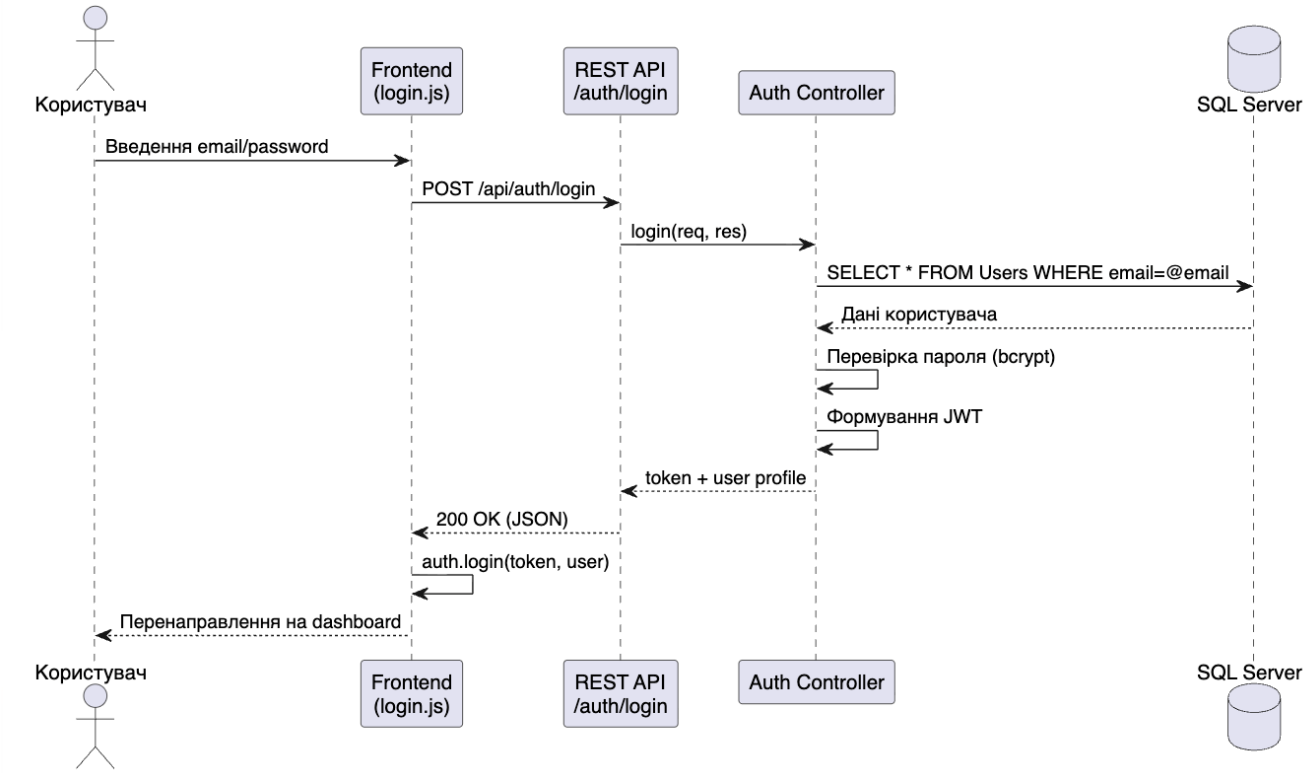

Рис. 2.6 Діаграма послідовності авторизації

На рисунку 2.6 представлено поетапний сценарій входу: від надсилання облікових даних до отримання JWT-токена та переходу користувача до відповідної панелі керування.

2.3 Проєктування структури бази даних та моделей даних

Проєктування бази даних виконано за реляційною моделлю з виділенням ключових сутностей предметної області: користувачі, пацієнти, лікарі, записи на прийом та медичні записи. Фізична схема реалізована в SQL Server у скрипті schema.sql.

Базова логіка моделі полягає в тому, що таблиця Users є центральною для автентифікації та ролей доступу, а таблиці Patients і Doctors розширюють профіль користувача за принципом зв’язку 1:1 через user_id. Операційні дані прийомів зберігаються в Appointments, а результати обстежень і лікування – в MedicalRecords, що забезпечує історичність медичної інформації [21, 22].

Для забезпечення цілісності даних використано:

– первинні ключі (IDENTITY, PRIMARY KEY);

– зовнішні ключі (FOREIGN KEY) між усіма залежними таблицями;

– обмеження доменів (наприклад CHECK для ролей та статусів);

– унікальні обмеження (UNIQUE) для email та medical_card_number;

– індекси на зовнішніх ключах для пришвидшення запитів.

На рисунку 2.7 представлено ER-діаграму спроєктованої бази даних з основними сутностями та зв’язками між ними

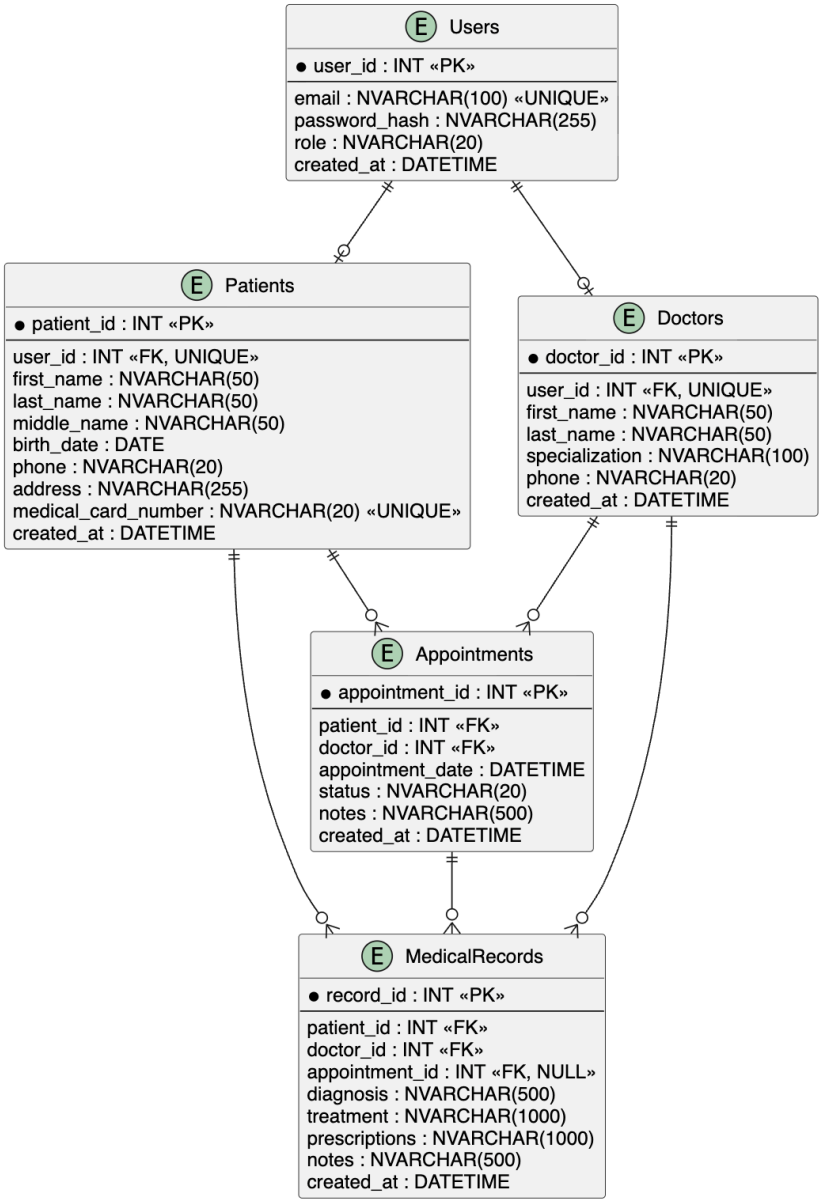

Рис. 2.7 ER-діаграма структури бази даних медичного центру

На рисунку 2.7 показано, що модель даних підтримує рольову структуру користувачів і повний цикл медичного обслуговування: від запису на прийом до збереження результатів лікування [22].

На рисунку 2.8 представлено логічні правила зв’язків і кардинальності, що використані під час проєктування БД

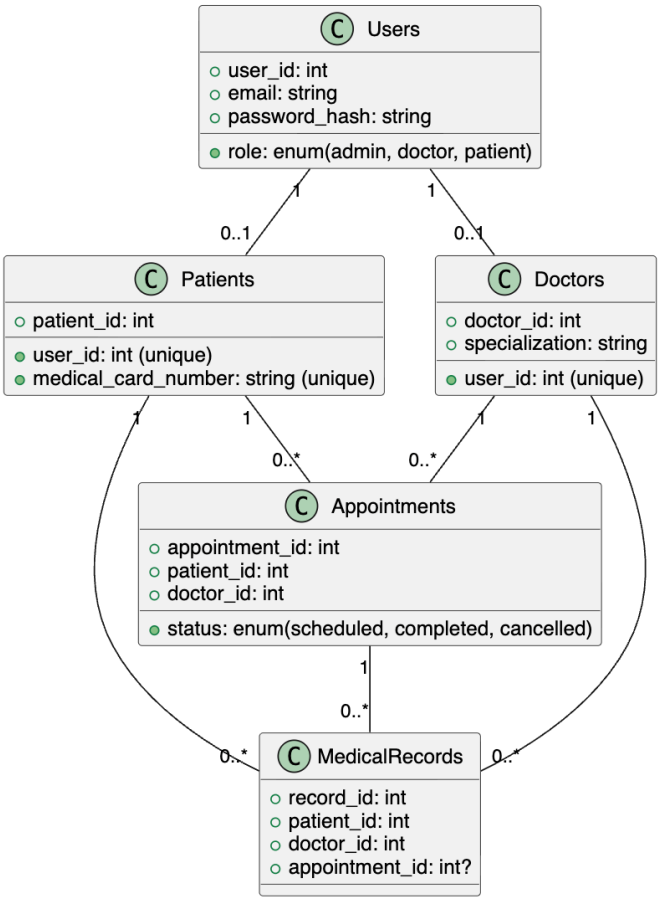

Рис. 2.8 Логічна модель даних та правила цілісності

На рисунку 2.8 представлено, що обмеження UNIQUE, FOREIGN KEY і перелічувані статуси/ролі забезпечують коректність та узгодженість даних на рівні СУБД.

Висновок до розділу 2

У другому розділі було послідовно виконано проєктування та практичну реалізацію архітектури клієнт-серверної інформаційної системи медичного центру.

Обрана трирівнева структура «frontend-backend-database» забезпечила логічне розділення відповідальності між інтерфейсом користувача, серверною бізнес-логікою та зберіганням даних. Реалізація через REST API, контейнеризація сервісів і рольова модель доступу створили технічну основу для стабільної та безпечної роботи системи.

У межах розділу також було обґрунтовано й побудовано структуру бази даних відповідно до предметної області. Сформовано ключові сутності (Users, Patients, Doctors, Appointments, MedicalRecords), встановлено зв’язки між ними та визначено обмеження цілісності, що гарантують коректність і узгодженість даних. Додатково передбачено механізми оптимізації запитів за рахунок індексації, що є важливим для продуктивності системи під час масштабування обсягу інформації.

Отже, у розділі 2 закладено повноцінний проєктний фундамент програмного продукту: від архітектурних рішень до фізичної моделі даних і засобів її розгортання. Отримані результати підтверджують, що реалізована структура є придатною для подальшого розвитку функціоналу, тестування та впровадження в умовах реального медичного закладу. Це забезпечує логічний перехід до наступних розділів, де розглядаються питання практичної реалізації, перевірки та оцінювання ефективності створеної системи.

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ

3.1 Реалізація серверної частини та API системи

Серверну частину системи реалізовано на платформі Node.js із використанням фреймворку Express. Центральна точка ініціалізації застосунку знаходиться у файлі server.js, де налаштовано middleware обробки JSON-запитів, CORS, підключення маршрутів API та глобальну обробку помилок. Після запуску сервер ініціює підключення до бази даних через модуль database.js, що забезпечує готовність шару даних до обслуговування HTTP-запитів.

Архітектура API побудована за модульним принципом: окремі маршрути винесено у файли auth.js, patients.js, doctors.js, appointments.js, medical-records.js. Така структура спрощує супровід коду, розширення функціоналу та тестування окремих бізнес-модулів. Логіка автентифікації та входу користувачів реалізована в контролері authController.js, де виконується перевірка облікових даних, хешування паролів і генерація JWT-токена.

Безпека доступу до ресурсів реалізована у два етапи: перевірка валідності токена в middleware auth.js та перевірка ролі користувача в roles.js. Це дозволяє контролювати доступ до критичних API-операцій, наприклад додавання чи видалення пацієнтів лише для адміністратора. У підсумку серверна частина забезпечує стандартизований REST-інтерфейс, захищений доступ до даних і надійний фундамент для подальшого масштабування системи.

На рисунку 3.1 представлено структуру взаємодії основних модулів серверної частини системи [24, 25].

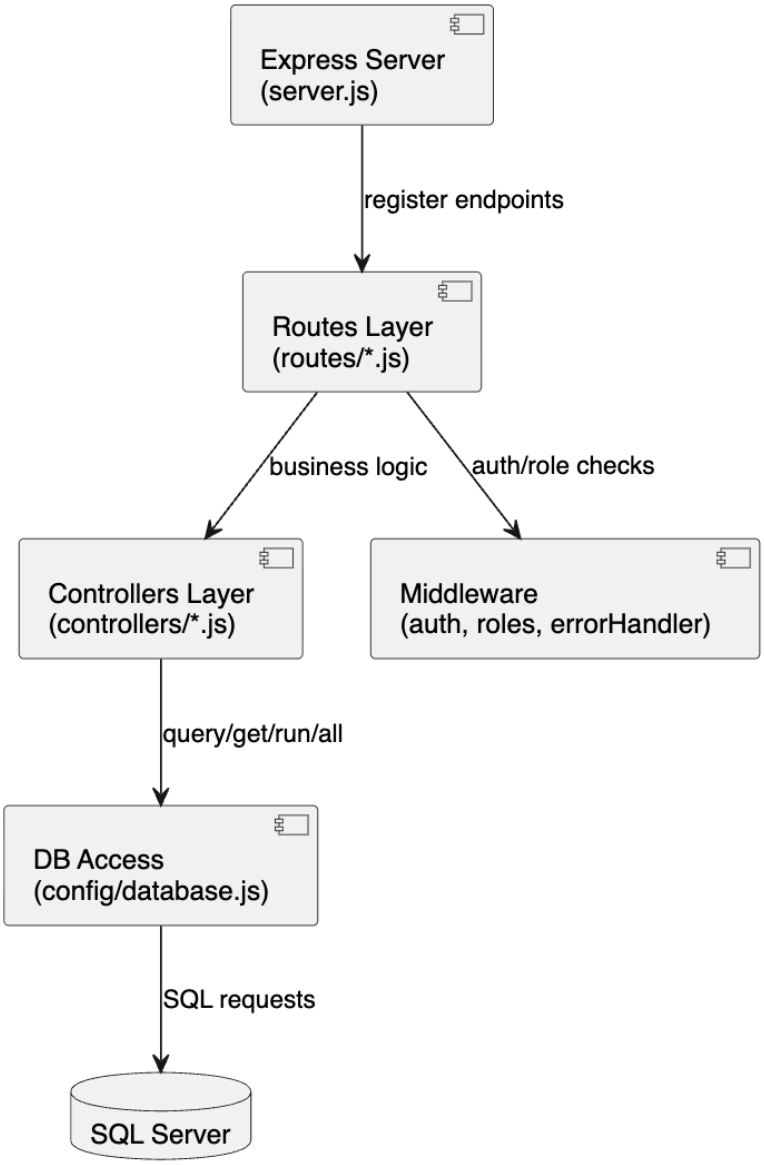

Рис. 3.1 Компонентна схема серверної частини та API

На рисунку 3.1 показано модульну організацію серверного коду, у якій маршрути делегують виконання контролерам, а доступ до даних ізольовано в окремому шарі.

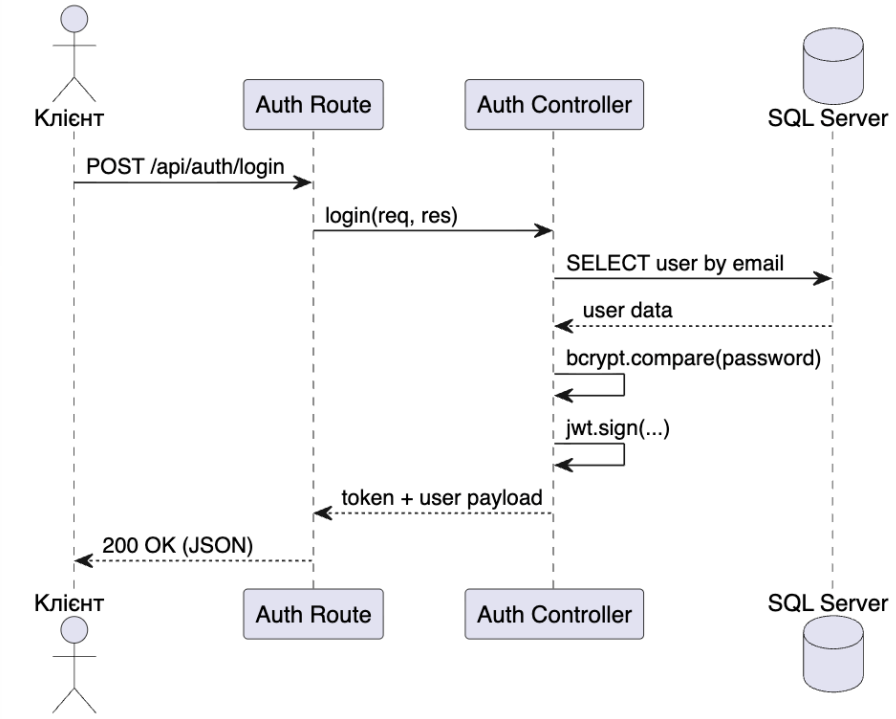

Рис. 3.2 Послідовність виконання запиту POST /api/auth/login

На рисунку 3.2 представлено повний цикл обробки запиту входу: від отримання облікових даних до формування JWT-токена та повернення авторизованої відповіді клієнту [26].

Таблиця 3.1

Технологічний стек серверної частини та API системи

Компонент	Технологія	Версія	Призначення
Runtime	Node.js	20.x	Серверне середовище
Framework	Express.js	4.18.2	Веб-фреймворк
Database	MS SQL Server / Azure SQL Edge	2022	Реляційна БД
ORM/Driver	mssql (Tedious)	10.0.4	Драйвер для SQL
Auth	JWT (jsonwebtoken)	9.0.2	Аутентифікація
Password Hashing	bcryptjs	2.4.3	Хешування паролів
Config	dotenv	16.3.1	Змінні середовища
CORS	cors	2.8.5	Крос-доменнізапити

```
backend/
├── config/
│   └── database.js          # Конфігурація MS SQL
├── controllers/
│   ├── authController.js    # Аутентифікація
│   ├── patientController.js # Управління пацієнтами
│   ├── doctorController.js  # Управління лікарями
│   ├── appointmentController.js # Прийоми
│   └── medicalRecordController.js # Медичні записи
├── middleware/
│   ├── auth.js              # JWT перевірка
│   └── roles.js             # RBAC (Role-Based Access Control)
├── routes/
│   ├── auth.js              # /api/auth
│   ├── patients.js          # /api/patients
│   ├── doctors.js           # /api/doctors
│   ├── appointments.js      # /api/appointments
│   └── medical-records.js   # /api/medical-records
├── .env                     # Змінні середовища
├── server.js                # Точка входу
└── package.json
```

Рис. 3.3 Серверна структура проекту

API Endpoints

Аутентифікація

На таблиці 3.2 продемонстровано перелік методів і прав доступу для кінцевих точок автентифікації користувачів (/api/auth), зокрема операцій реєстрації та входу в систему [26].

Таблиця 3.2

Методи та права доступу для API автентифікації (/api/auth)

Метод	Endpoint	Опис	Доступ
POST	/api/auth/register	Реєстрація	Публічний
POST	/api/auth/login	Вхід	Публічний

На таблиці 3.3 розглянуто основні методи API для роботи з даними пацієнтів (/api/patients) та відповідні обмеження доступу за ролями користувачів.

Таблиця 3.3

Методи та права доступу для API керування пацієнтами (/api/patients)

Метод	Endpoint	Опис	Доступ
GET	/api/patients	Список	Admin, Doctor
GET	/api/patients/:id	Деталі	Admin, Doctor, Owner
POST	/api/patients	Створити	Admin
PUT	/api/patients/:id	Оновити	Admin
DELETE	/api/patients/:id	Видалити	Admin

На таблиці 3.4 показано склад методів API для модуля лікарів (/api/doctors) і розмежування прав доступу між адміністратором, лікарем та пацієнтом.

Таблиця 3.4

Методи та права доступу для API керування лікарями (/api/doctors)

Метод	Endpoint	Опис	Доступ
GET	/api/doctors	Список	Admin, Doctor, Patient
GET	/api/doctors/:id	Деталі	Admin, Doctor
GET	/api/doctors/by-user/:userId	За User ID	Doctor (owner)
POST	/api/doctors	Створити	Admin
PUT	/api/doctors/:id	Оновити	Admin, Owner

На таблиці 3.5 продемонстровано набір кінцевих точок для керування записами на прийом (/api/appointments), включаючи операції створення, оновлення та видалення записів.

Таблиця 3.5

Методи та права доступу для API записів на прийом (/api/appointments)

Метод	Endpoint	Опис	Доступ
GET	/api/appointments	Список	All roles
GET	/api/appointments/without-records/:patientId	Без записів	Doctor
POST	/api/appointments	Створити	Admin, Patient
PUT	/api/appointments/:id	Оновити	Admin, Doctor
DELETE	/api/appointments/:id	Видалити	Admin

3.2 Розробка бази даних та логіки обробки даних

У межах цього підрозділу реалізовано прикладний рівень роботи з даними: підключення до SQL Server, виконання параметризованих запитів та бізнес-логіку CRUD-операцій для пацієнтів, лікарів, записів на прийом і медичних карток. Базовий доступ до БД інкапсульовано у модулі database.js, де реалізовано функції query/get/run/all, повторні спроби з’єднання, ініціалізацію таблиць та стартові дані.

Логіка обробки даних винесена у контролери: patientController.js, doctorController.js, appointmentController.js, medicalRecordController.js. Для створення записів використовуються INSERT ... OUTPUT INSERTED, що дає змогу одразу отримати ідентифікатор нової сутності; для читання застосовано JOIN-запити, які повертають розширені дані (наприклад, ПІБ лікаря/пацієнта разом із записом на прийом) [27].

Окрему увагу приділено прикладним правилам обробки: генерація номера медичної картки, фільтрація записів за роллю користувача, пошук прийомів без медичного висновку (/appointments/without-records/:patientId), а також перевірка існування запису перед оновленням або видаленням. Це забезпечує цілісність і коректність обробки даних на рівні серверної логіки.

На рисунку 3.4 показано структуру шару доступу до даних і зв’язок контролерів із БД.

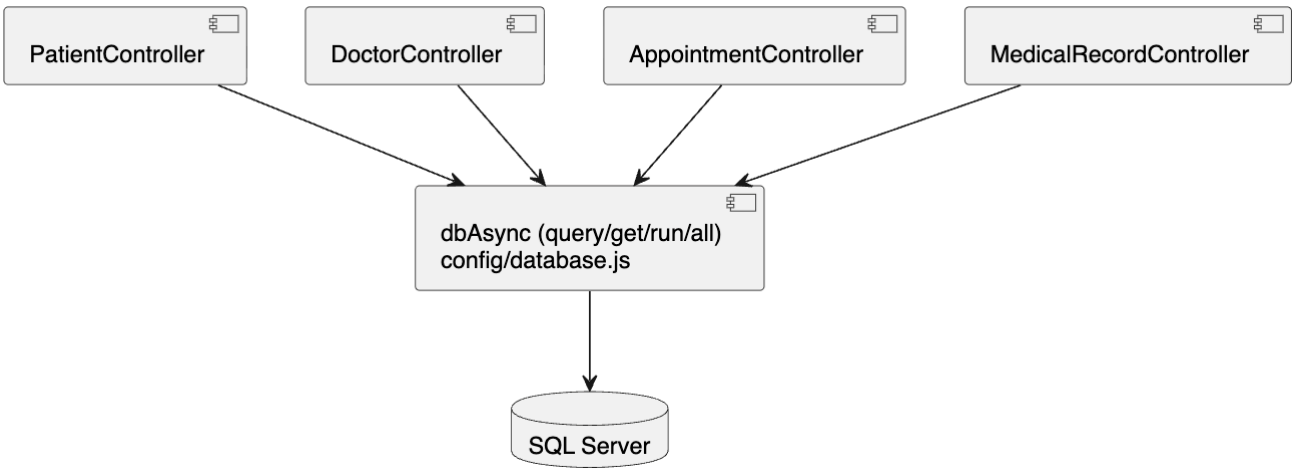

Рис. 3.4 Схема взаємодії контролерів із модулем доступу до даних

На рисунку 3.4 продемонстровано, що всі контролери використовують єдиний уніфікований модуль доступу до БД, що спрощує супровід і стандартизує обробку SQL-запитів [28].

На рисунку 3.5 представлено послідовність створення медичного запису з подальшим читанням створеного рядка.

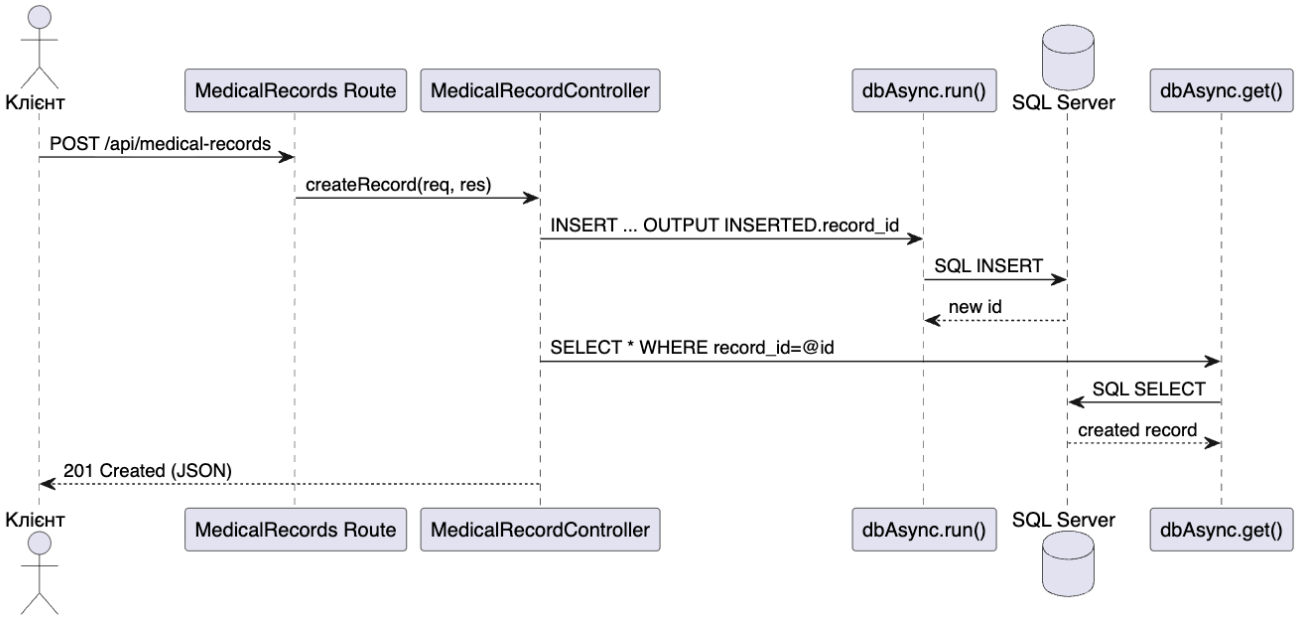

Рис. 3.5 Послідовність обробки запиту POST /api/medical-records

На рисунку 3.5 розглянуто типовий шаблон обробки даних у системі: запис у БД, отримання ідентифікатора та повернення клієнту повного об’єкта.

На рисунку 3.6 продемонстровано рольову фільтрацію даних для модуля

записів на прийом.

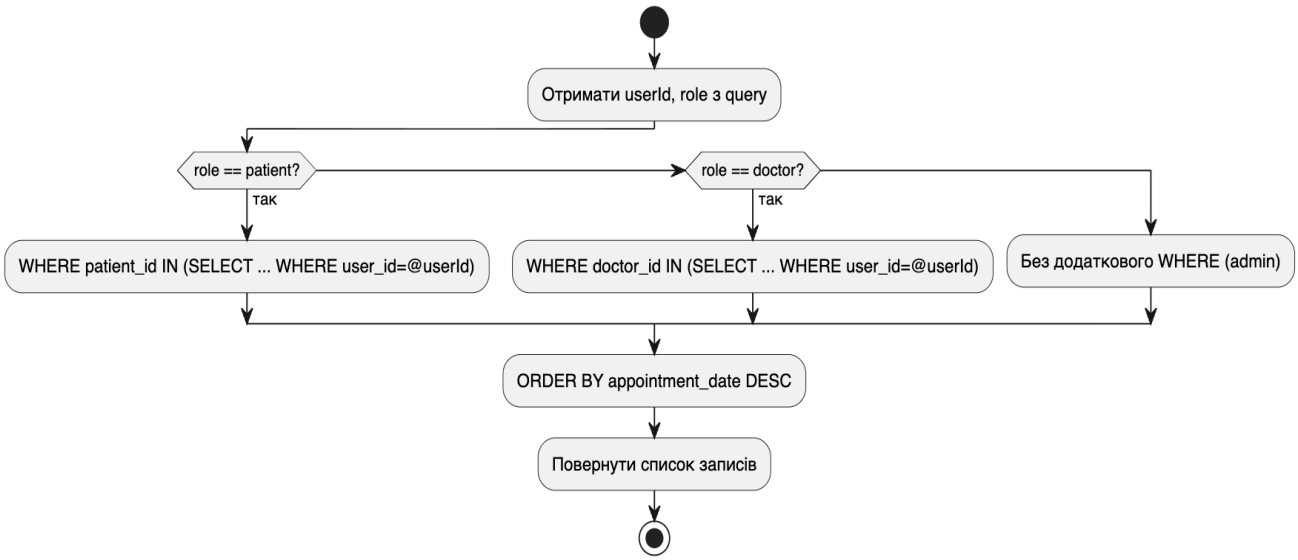

Рис. 3.6 Логіка фільтрації записів на прийом за ролями

На рисунку 3.6 показано, що серверна частина застосовує різні умови вибірки залежно від ролі, забезпечуючи контроль доступу до даних на рівні бізнес-логіки.

3.3 Розробка клієнтської частини та взаємодія з сервером

На рисунку 3.7 показано структуру frontend-частини проєкту, організовану за функціональними модулями, що забезпечує зручність розробки, підтримки та масштабування веб-застосунку. Основними складовими структури є каталоги стилів (css), клієнтської логіки (js), а також сторінки, розподілені відповідно до ролей користувачів у системі – адміністратора (pages/admin), лікаря (pages/doctor) та пацієнта (pages/patient). Каталог css містить файли стилізації інтерфейсу, які відповідають за візуальне оформлення сторінок, адаптивність дизайну та єдину стилістичну концепцію системи. У директорії js реалізовано клієнтську логіку застосунку, включаючи обробку подій, взаємодію з API, валідацію даних та динамічне оновлення елементів інтерфейсу.

```
frontend/
├── css/
│   ├── variables.css    # CSS змінні (кольори, розміри)
│   ├── reset.css        # CSS reset
│   ├── base.css         # Базові стилі
│   ├── components.css   # Компоненти (кнопки, форми, таблиці)
│   └── layout.css       # Макет (header, sidebar)
├── js/
│   ├── api.js           # API клієнт
│   ├── auth.js          # Управління сесією
│   └── utils.js         # Утиліти (форматування дат)
└── pages/
    ├── admin/           # Сторінки адміна
    │   ├── dashboard.html
    │   ├── patients.html
    │   ├── doctors.html
    │   └── medical-card.html
    ├── doctor/          # Сторінки лікаря
    │   ├── dashboard.html
    │   ├── patients.html
    │   ├── appointments.html
    │   ├── medical-card.html
    │   └── profile.html
    └── patient/         # Сторінки пацієнта
        └── medical-card.html
```

Рис. 3.7 Архітектура Frontend частини

CSS Архітектура (BEM-подібна)

JavaScript модулі

API Client (api.js)

```
/* Блок */
.btn { }
.btn-primary { }
.btn-secondary { }

/* Елемент */
.form-input { }
.form-label { }
.form-group { }

/* Модифікатор */
.card-header { }
.card-body { }
.card-footer { }
```

Auth модуль (auth.js)

```
const API_BASE_URL = '/api';
async function apiRequest(endpoint, options = {}) {
const token = localStorage.getItem('token');
const response = await fetch (`${API_BASE_URL}${endpoint}`, { headers: {
});
}
},
'Authorization': `Bearer ${token}`,
'Content-Type': 'application/json'
...options
return response.json();
```

Nginx Configuration

```
server {
listen 80;
root /usr/share/nginx/html;
# Статика
location / {
}
try_files $uri $uri/ /index.html;
# API proxy
location /api/ {
}
proxy_pass http://backend: 3000/api/; proxy_set_header Host $host;
}
```

Аутентифікація

JWT Tokens з терміном дії 24 години

Хешування паролів bcryptjs (10 salt rounds)

Захищені роути через middleware [29]

Авторизація (RBAC)

Таблиця 3.7

Рольова модель доступу користувачів у системі

Роль	Права
admin	Повний доступ
doctor	Читання пацієнтів, редагування своїх записів
patient	Читання тільки своєї медкарти

SQL Injection захист

1. Параметризовані запити (@param)
2. Input validation
3. Prepared statements

Потоки даних

Створення медичного запису

1. Лікар відкриває medical-card.html?patientId=1
2. Frontend завантажує дані пацієнта GET /api/patients/1
3. Frontend завантажує історію GET /api/medical-records/1
4. Лікар натискає "Новий запис"
5. Frontend завантажує доступні прийоми GET /api/appointments/without-records/1
6. Лікар заповнює форму (діагноз, лікування, призначення)
7. Frontend відправляє POST /api/medical-records
8. Backend перевіряє JWT, права доступу
9. Backend виконує SQL INSERT INTO MedicalRecords
10. Backend повертає створений запис
11. Frontend оновлює список записів

Аутентифікація

1. Користувач вводить email/password на login.html
2. Frontend відправляє POST /api/auth/login
3. Backend перевіряє хеш пароля (bcrypt.compare)
4. Backend генерує JWT токен
5. Backend повертає {token, user}

6. Frontend зберігає в localStorage
7. Frontend редиректить на dashboard відповідно до ролі [30]

На рисунку 3.6 представлено інтерфейс головної сторінки системи обліку пацієнтів. У верхній частині розміщено навігаційну панель із назвою застосунку, пунктами меню та кнопкою входу до системи. Центральний блок містить основний інформаційний меседж платформи та короткий опис її призначення для медичних закладів.

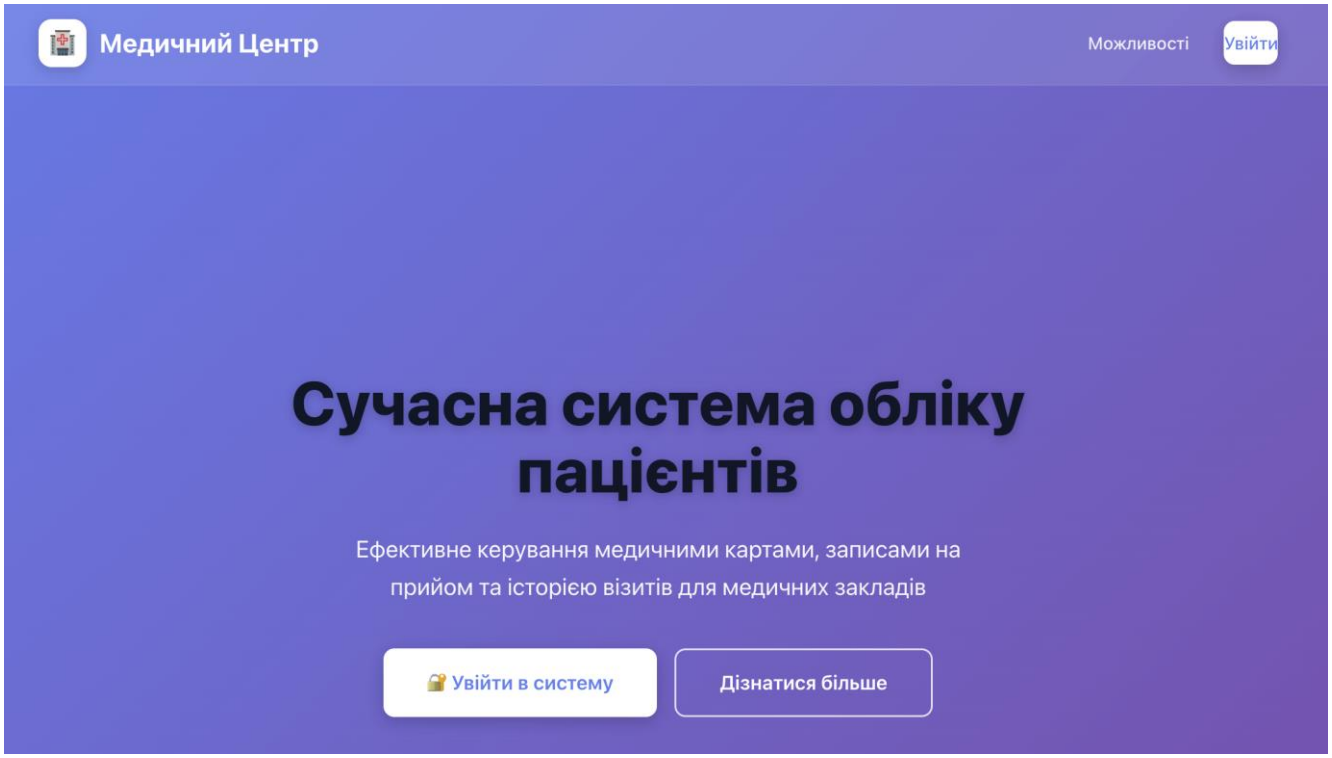

Рис. 3.8 Головна сторінка вебзастосунку медичного центру

У нижній частині головного екрану реалізовано кнопки цільових дій: перехід до авторизації та ознайомлення з додатковою інформацією. Така структура сторінки забезпечує зрозумілий сценарій першої взаємодії користувача із системою та швидкий доступ до ключових функцій.

На рисунку 3.9 показано сторінку входу до інформаційної системи медичного центру. Інтерфейс містить обов’язкові поля для введення електронної пошти та пароля, кнопку підтвердження входу, а також блок із тестовими обліковими записами для демонстраційного доступу.

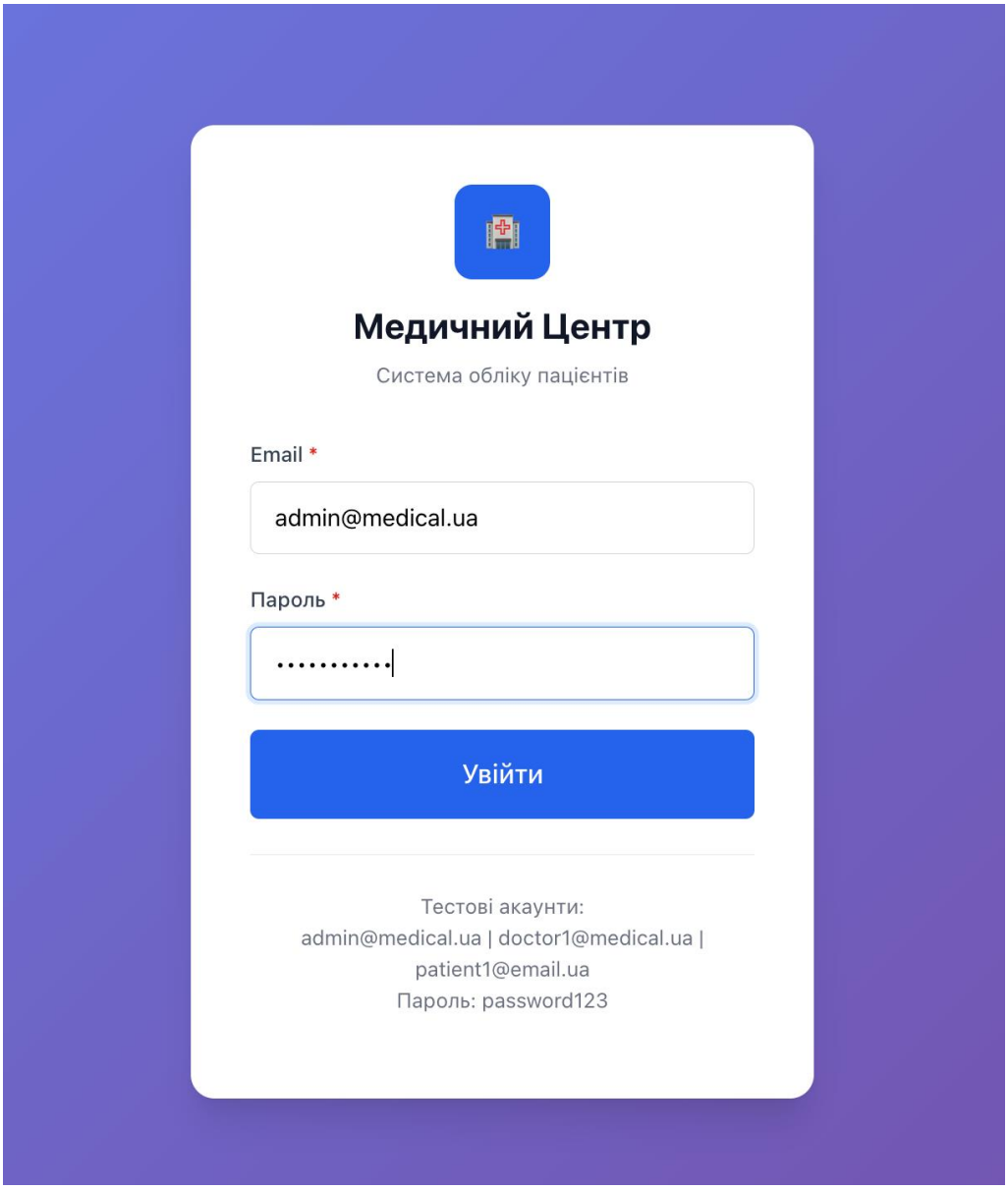

Рис. 3.9 Форма автентифікації користувача в системі

На рисунку 3.10 представлено головний екран адміністратора після успішної авторизації. Інтерфейс містить верхню навігацію за основними модулями системи (головна, пацієнти, лікарі, записи), блок профілю поточного користувача та кнопку завершення сесії.

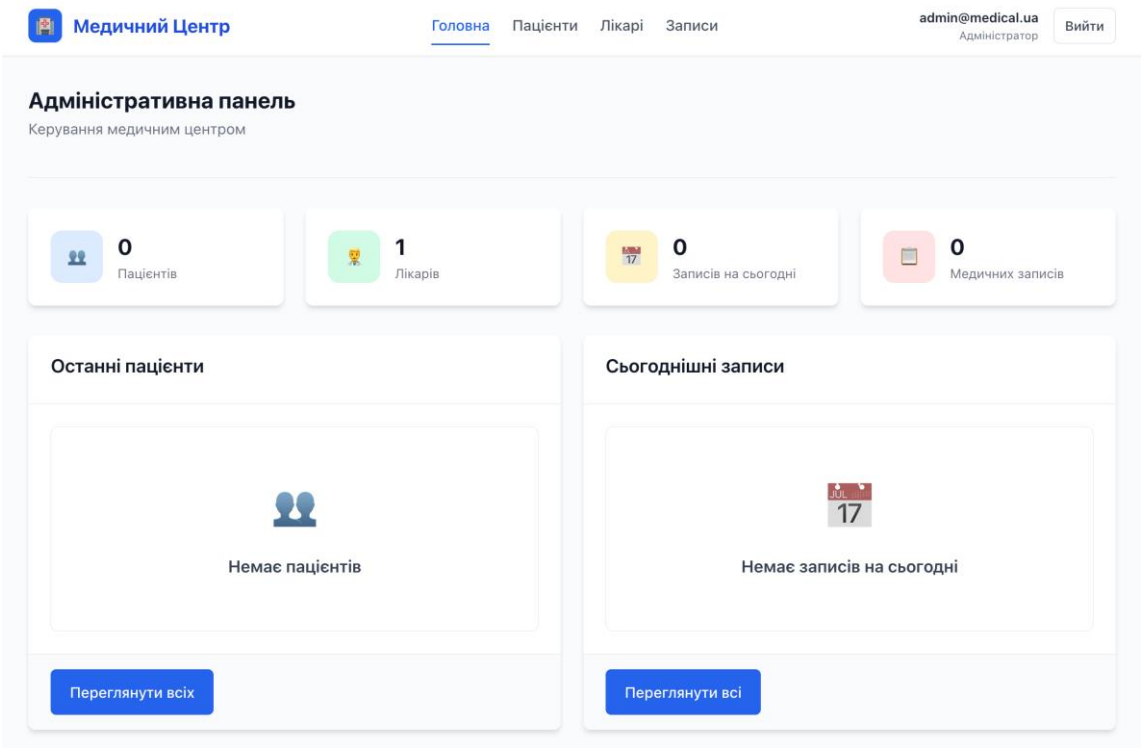

Рис. 3.10 Адміністративна панель керування медичним центром

Центральна частина панелі реалізує інформаційно-аналітичний підхід: відображаються ключові показники стану системи (кількість пацієнтів, лікарів, записів на поточну дату та медичних записів). Нижче розміщено оперативні віджети для перегляду останніх пацієнтів і сьогоднішніх записів, що забезпечує швидкий доступ до пріоритетних даних для адміністративного контролю.

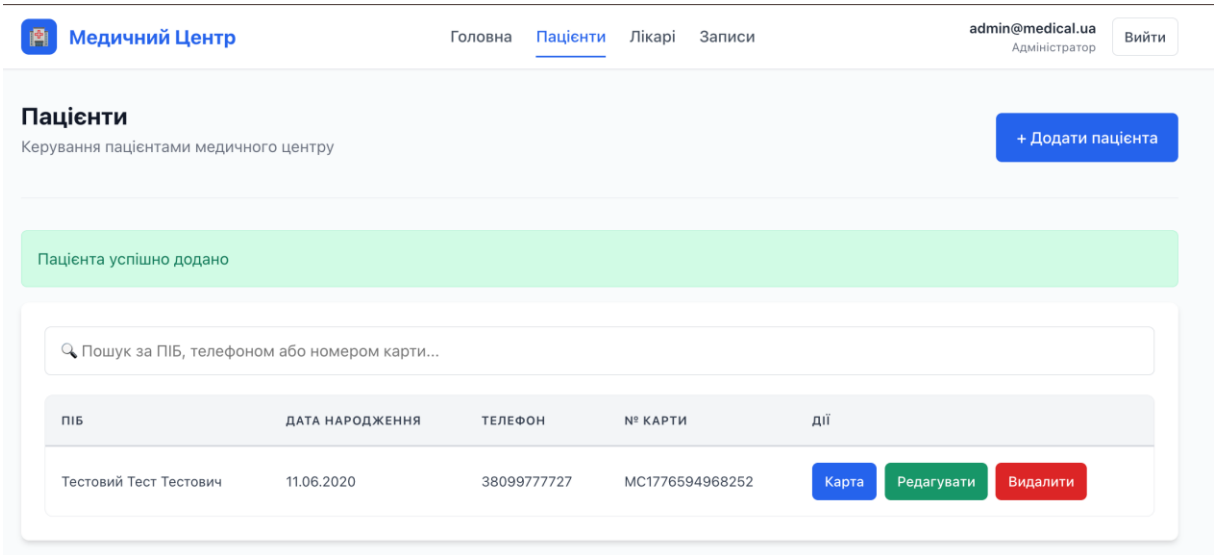

Рис. 3.11 Інтерфейс керування пацієнтами в адміністративному модулі

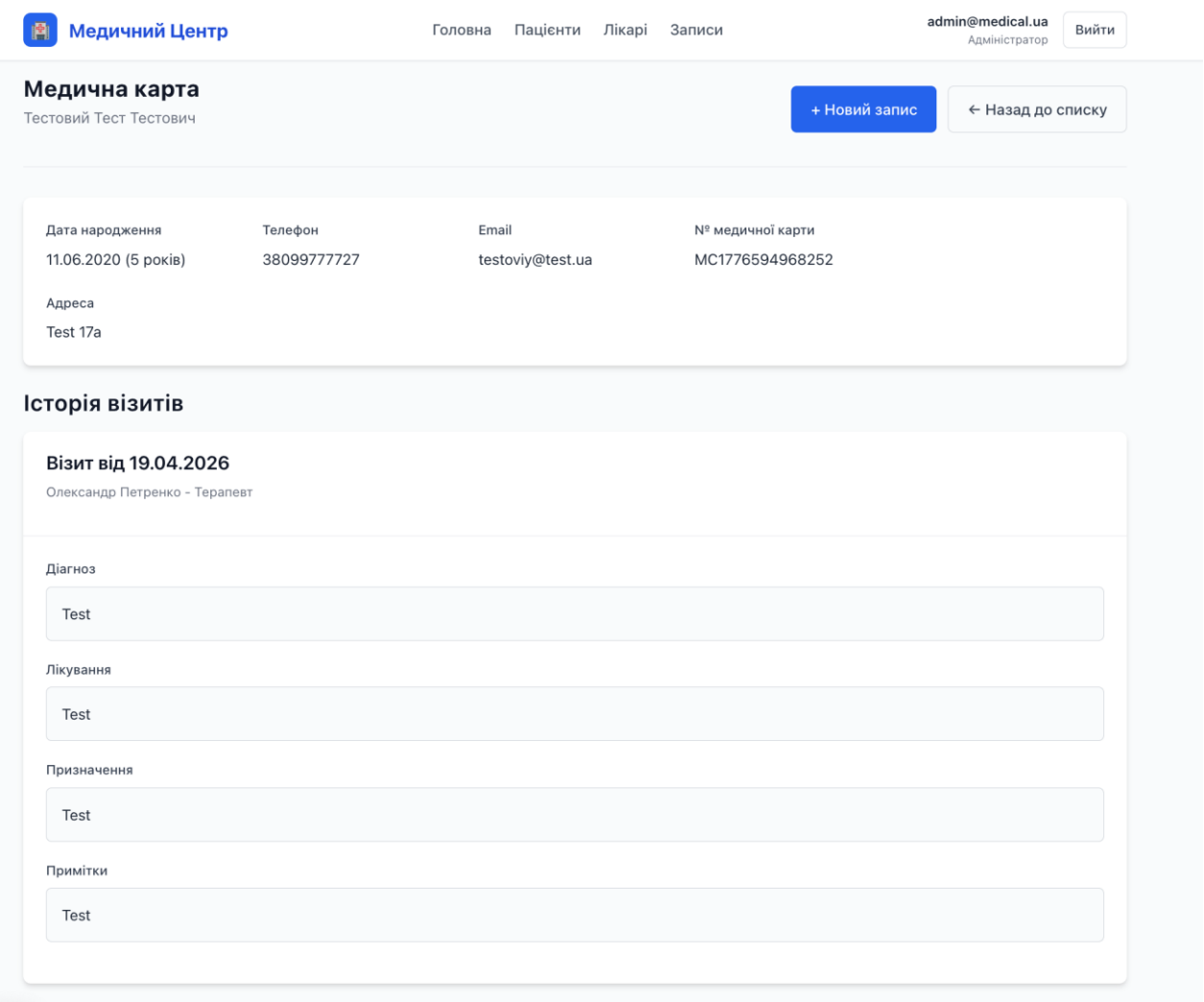

Рис. 3.12 Інтерфейс перегляду медичної карти пацієнта

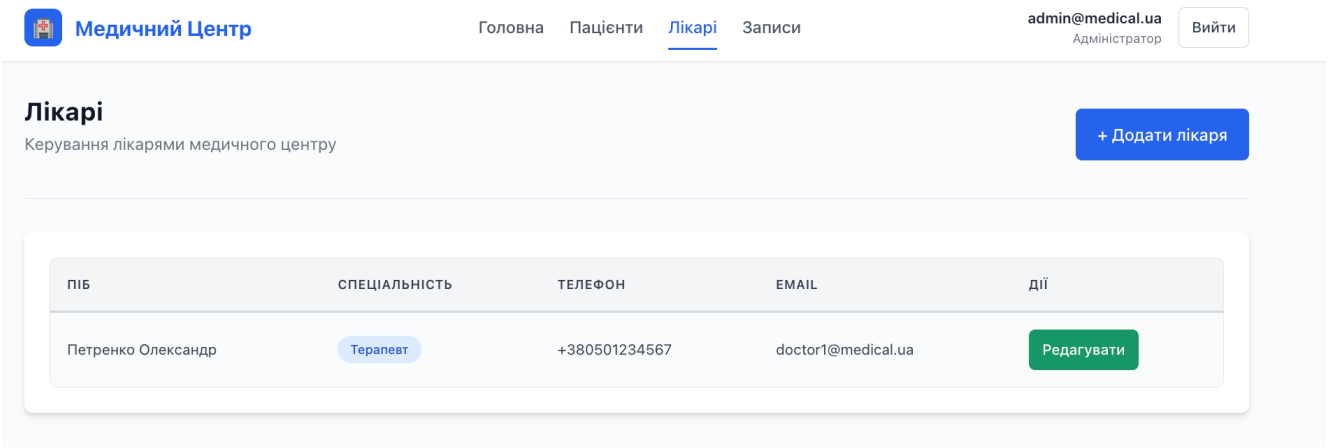

Рис. 3.13 Інтерфейс керування лікарями в адміністративному модулі

Представлена форма забезпечує централізоване адміністрування кадрових даних і підтримує актуальність інформації про медичний персонал. Використання окремого модуля для лікарів спрощує організацію записів на прийом та подальшу прив’язку медичних висновків до конкретного фахівця [30, 31].

Висновок до розділу 3

У третьому розділі було реалізовано прикладний рівень інформаційної системи медичного центру: серверну частину, API-взаємодію та механізми обробки даних. На базі Node.js і Express сформовано модульну структуру маршрутизації та контролерів, що забезпечує розділення відповідальності між автентифікацією, роботою з пацієнтами, лікарями, записами на прийом і медичними записами. Запроваджена рольова модель доступу (admin, doctor, patient) і JWT-автентифікація забезпечили контрольований та безпечний доступ до функцій системи.

Також у межах розділу опрацьовано логіку доступу до бази даних SQL Server через уніфікований шар запитів, що дозволило стандартизувати виконання операцій читання, створення, оновлення та видалення даних. Реалізовано прикладні сценарії обробки: зв’язування користувачів із профілями, ведення історії візитів, формування медичних записів, а також фільтрацію інформації відповідно до ролі користувача. Це забезпечило цілісність даних і коректність бізнес-процесів на рівні серверної логіки.

Окрему увагу приділено інтерфейсній частині, де реалізовано ключові екрани системи: головну сторінку, форму входу, адміністративну панель, сторінки керування пацієнтами та лікарями, а також перегляд медичної карти. Отриманий результат підтверджує, що система має завершену функціональну основу для практичного використання, а також є придатною до подальшого розширення, тестування та впровадження в реальних умовах медичного закладу.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розглянуто теоретичні основи створення веб-систем обліку пацієнтів, зокрема специфіку організації медичних інформаційних систем, вимоги до захисту персональних даних і особливості цифровізації медичних процесів. Проаналізовано сучасні веб-технології (HTML5, CSS3, JavaScript) та можливості їх застосування для побудови зручного інтерфейсу користувача. Окремо виокремлено роль реляційних баз даних у забезпеченні цілісності, структурованості та надійного зберігання медичної інформації на прикладі MS SQL Server.

У межах проєктної частини сформовано функціональні та нефункціональні вимоги до системи, що охоплюють облік пацієнтів, керування лікарями, записами на прийом і медичними картами. На цій основі розроблено архітектуру клієнт-серверної системи з розподілом на рівень представлення, серверну логіку та рівень даних. Також спроєктовано структуру бази даних і моделі даних, визначено ключові сутності, зв’язки між ними, обмеження цілісності та механізми оптимізації доступу до даних.

У практичній частині реалізовано серверну частину системи на Node.js/Express із REST API, автентифікацією на основі JWT та рольовою моделлю доступу (admin, doctor, patient). Розроблено логіку обробки даних для основних бізнес-процесів: створення, перегляду, оновлення та видалення пацієнтів, лікарів, записів і медичних записів. Продемонстровано роботу клієнтської частини та інтеграцію з сервером через API, а також реалізацію ключових інтерфейсів для різних ролей користувачів.

Розроблено працездатну веб-систему обліку пацієнтів, яка поєднує теоретично обґрунтовані підходи та практично реалізовані програмні рішення. Отримані результати мають прикладну цінність і можуть бути використані як основа для подальшого розвитку системи, зокрема розширення аналітики, впровадження журналювання дій користувачів, інтеграції з зовнішніми медичними сервісами та підвищення рівня інформаційної безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft. SQL Server Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/sql/ (дата звернення: 10.04.2026).
2. Mozilla. HTML5 Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 10.04.2026).
3. Mozilla. CSS3 Documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 10.04.2026).
4. Mozilla. JavaScript Guide [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 11.04.2026).
5. W3C. HTML5 Specification [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/html5/ (дата звернення: 11.04.2026).
6. W3C. CSS Specifications [Електронний ресурс]. – Режим доступу: https://www.w3.org/Style/CSS/ (дата звернення: 11.04.2026).
7. ECMAScript. JavaScript Language Specification [Електронний ресурс]. – Режим доступу: https://tc39.es/ecma262/ (дата звернення: 12.04.2026).
8. Elmasri R., Navathe S. Fundamentals of Database Systems. – 7th ed. – Pearson, 2016. – 1272 p.
9. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. – 7th ed. – McGraw-Hill, 2020. – 1376 p.
10. Date C. J. An Introduction to Database Systems. – 8th ed. – Pearson, 2004. – 1024 p.
11. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. – 6th ed. – Pearson, 2015. – 1440 p.
12. Coronel C., Morris S. Database Systems: Design, Implementation, and Management. – 13th ed. – Cengage Learning, 2018. – 704 p.
13. Pressman R. Software Engineering: A Practitioner’s Approach. – 8th ed. – McGraw-Hill, 2015. – 976 p.

14. Sommerville I. Software Engineering. – 10th ed. – Pearson, 2016. – 816 p.

15. OWASP Foundation. OWASP Top Ten Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/ (дата звернення: 12.04.2026).

16. ISO/IEC. ISO/IEC 27001 Information Security Management [Електронний ресурс]. – Режим доступу: https://www.iso.org/isoiec-27001-information-security.html (дата звернення: 13.04.2026).

17. HealthIT.gov. What is a Health IT System? [Електронний ресурс]. – Режим доступу: https://www.healthit.gov/topic/health-it-and-health-information-exchange-basics (дата звернення: 13.04.2026).

18. Oracle. Database Concepts [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/database/ (дата звернення: 13.04.2026).

19. Microsoft Learn. What's new in SQL Server 2022 [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2022?view=sql-server-ver17 (дата звернення: 19.04.2026).

20. Microsoft Learn. Primary and foreign key constraints (SQL Server) [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver17 (дата звернення: 19.04.2026).

21. Microsoft Learn. Data types (Transact-SQL) [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql (дата звернення: 19.04.2026).

22. Microsoft Learn. CREATE INDEX (Transact-SQL) [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-index-transact-sql?redirectedfrom=MSDN&view=sql-server-ver16 (дата звернення: 19.04.2026).

23. Microsoft Learn. SELECT (Transact-SQL) [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql?view=sql-server-ver17 (дата звернення: 19.04.2026).

24. Microsoft Learn. Azure SQL Edge documentation [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/en-us/azure/azure-sql-edge/ (дата звернення: 19.04.2026).

25. Docker Docs. Compose file reference [Електронний ресурс]. – 2025. – Режим доступу: https://docs.docker.com/reference/compose-file/ (дата звернення: 19.04.2026).

26. Docker Docs. Networking in Compose [Електронний ресурс]. – 2025. – Режим доступу: https://docs.docker.com/compose/how-tos/networking/ (дата звернення: 19.04.2026).

27. Docker Docs. Define and manage volumes in Docker Compose [Електронний ресурс]. – 2025. – Режим доступу: https://docs.docker.com/reference/compose-file/volumes/ (дата звернення: 19.04.2026).

28. Docker Docs. Docker Desktop [Електронний ресурс]. – 2025. – Режим доступу: https://docs.docker.com/desktop/ (дата звернення: 19.04.2026).

29. Express.js. Routing [Електронний ресурс]. – 2025. – Режим доступу: https://expressjs.com/en/guide/routing.html (дата звернення: 19.04.2026).

30. Express.js. Using middleware [Електронний ресурс]. – 2025. – Режим доступу: https://expressjs.com/en/guide/using-middleware.html (дата звернення: 19.04.2026).

31. Express.js.Security Best Practices for Express in Production [Електронний ресурс]. – 2025. – Режим доступу: https://expressjs.com/en/advanced/best-practice-security.html (дата звернення: 19.04.2026).

32. MDN Web Docs. Fetch API [Електронний ресурс]. – 2025. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 19.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

«ВЕБ-СИСТЕМА ДЛЯ ОБЛІКУ ПАЦІЄНТІВ МЕДИЧНОГО ЦЕНТРУ З ВИКОРИСТАННЯМ HTML 5, CSS 3, JAVASCRIPT ТА MICROSOFT SQL SERVER»

Виконав студент: групи ТЦР-43 Андрій ЛАЗНЕВИЙ

Керівник роботи: кандидат технічних наук Андрій АРОНОВ

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ 2

Мета роботи:

Розробка веб -системи для обліку пацієнтів медичного центру з використанням HTML5, CSS3, JavaScript та Microsoft SQL Server, що забезпечує ефективне зберігання, обробку та доступ до медичних даних, реалізацію функцій запису на прийом і ведення медичної історії.

Об'єкт дослідження:

Процеси обліку пацієнтів у медичних інформаційних системах та організація електронного документообігу в медичних центрах.

Предмет дослідження:

Методи та засоби розробки веб -систем для управління даними пацієнтів із використанням сучасних технологій веб -розробки та реляційних баз даних.

НАУКОВІ ЗАВДАННЯ ТА МЕТОДИ ДОСЛІДЖЕННЯ

Наукові завдання:

- Аналіз предметної області обліку пацієнтів і проблем існуючих підходів
- Дослідження веб-технологій (HTML5, CSS3, JavaScript) для медичних систем
- Аналіз можливостей MS SQL Server для зберігання медичних даних
- Розробка структури та трирівневої архітектури системи
- Проєктування логічної та фізичної моделі бази даних
- Реалізація REST API з JWT-автентифікацією та RBAC
- Розробка клієнтської частини та взаємодії з сервером
- Тестування системи та оцінка ефективності

Методи дослідження:

- Аналіз сучасних підходів до побудови медичних інформаційних систем
- Проєктування клієнт-серверної архітектури веб-застосунку
- Розробка серверної частини з Node.js та Express
- Проєктування та реалізація бази даних у MS SQL Server
- Реалізація механізмів автентифікації (JWT) та авторизації (RBAC)
- Розробка клієнтської частини з HTML5, CSS3 та JavaScript

АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

Таблиця 1 – Порівняльна характеристика веб-технологій HTML5, CSS3 та JavaScript

Технологія	Призначення	Роль у системі
HTML5	Структура веб-сторінки	Форми реєстрації та перегляду даних пацієнтів
CSS3	Оформлення та адаптивний дизайн	Зручний інтерфейс на комп'ютері та планшеті
JavaScript	Логіка та взаємодія з REST API	CRUD-операції, динамічне оновлення сторінок

Таблиця 2 – Порівняльна характеристика реляційних та нереляційних баз даних

Критерій	Реляційні (SQL)	Нереляційні (NoSQL)
Структура даних	Таблична (рядки і стовпці)	Гнучка (документи, ключ-значення)
Цілісність	Висока (ACID-властивості)	Обмежена (BASE)
Масштабованість	Вертикальна	Горизонтальна
Використання	Медицина, банківські системи, облік	Big Data, соціальні мережі
Приклади СУБД	MS SQL Server, MySQL, PostgreSQL	MongoDB, Cassandra, Redis

Обґрунтовано вибір MS SQL Server як надійної, безпечної та продуктивної СУБД для зберігання структурованих медичних даних.

ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ СИСТЕМИ 5

Рисунок 1 Use Case діаграма веб -системи обліку пацієнтів медичного центру

СТРУКТУРА БАЗИ ДАНИХ МЕДИЧНОГО ЦЕНТРУ 7

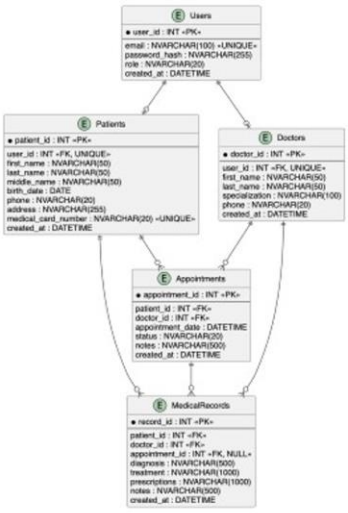

Рисунок 2 – ER-діаграма структури бази даних медичного центру

ДІАГРАМА ПОСЛІДОВНОСТІ АВТОРИЗАЦІЇ КОРИСТУВАЧА 8

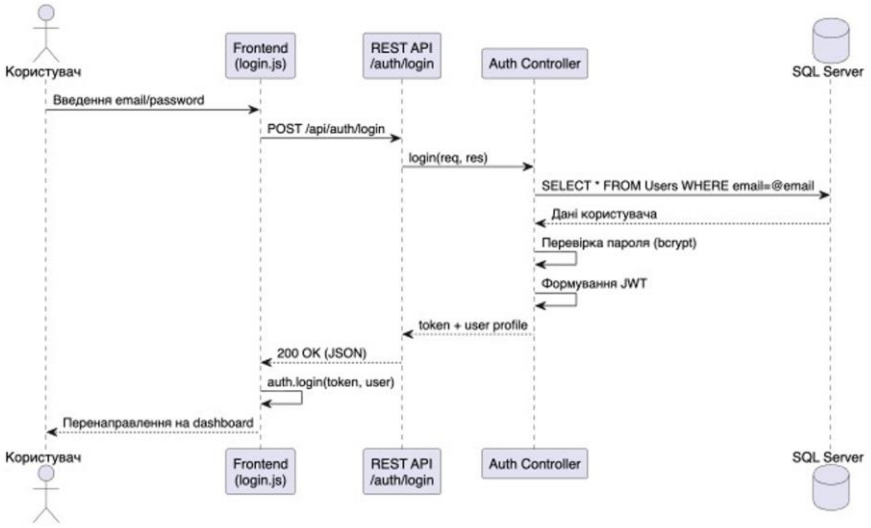

Рисунок 3 – Діаграма послідовності процесу авторизації користувача

ФУНКЦІОНАЛЬНІ ТА НЕФУНКЦІОНАЛЬНІ ВИМОГИ 9

Таблиця 3 – Функціональні вимоги до веб -системи обліку пацієнтів

№	Функція	Опис
1	Реєстрація пацієнта	Введення, збереження та редагування персональних даних пацієнта
2	Запис на прийом	Створення запису до лікаря з вибором дати, часу та послуги
3	Перегляд даних	Доступ до інформації про пацієнтів, записи та історію звернень
4	Пошук	Пошук пацієнтів або записів за ПІБ, датою, лікарем
5	Редагування	Оновлення та зміна існуючих даних у системі

Таблиця 4 – Нефункціональні вимоги до веб -системи обліку пацієнтів

№	Вимога	Опис
1	Швидкість	Швидкий відгук системи; час виконання запитів — до 2 с
2	Безпека	JWT-автентифікація, RBAC, параметризовані SQL -запити, bcrypt
3	Масштабованість	Контейнеризація (Docker), можливість горизонтального розширення
4	Зручність	Адаптивний інтерфейс, підтримка різних ролей (admin/doctor/patient)

REST API ТА РОЛЬОВА МОДЕЛЬ ДОСТУПУ (RBAC) 10

Таблиця 5 – Основні ендпоінти REST API системи

Метод	Ендпоінт	Опис	Роль доступу
POST	/api/auth/login, /register	Автентифікація / реєстрація	Публічний
GET / POST	/api/patients	Список / Створення пацієнтів	Admin, Doctor / Admin
PUT / DEL	/api/patients/:id	Оновлення / Видалення пацієнта	Admin
GET	/api/doctors	Список лікарів	All roles
GET / POST	/api/appointments	Записи на прийом	All / Admin, Patient
PUT	/api/appointments/:id	Оновлення запису	Admin, Doctor
POST	/api/medical -records	Додавання медичного запису	Doctor

Рольова модель доступу:

admin — Повний доступ: перегляд, створення, редагування та видалення всіх сутностей системи

doctor — Перегляд пацієнтів і своїх записів; додавання медичних висновків і діагнозів

patient — Читання власної медичної карти; запис на прийом; перегляд розкладу лікарів

ДЕМОНСТРАЦІЯ ІНТЕРФЕЙСУ СИСТЕМИ 11

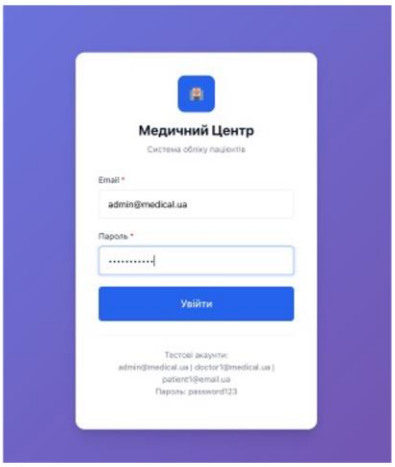

Рисунок 4 – Форма автентифікації користувача

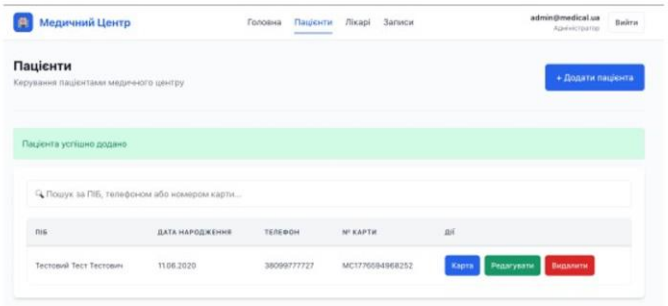

Рисунок 5 – Інтерфейс управління пацієнтами (адміністратор)

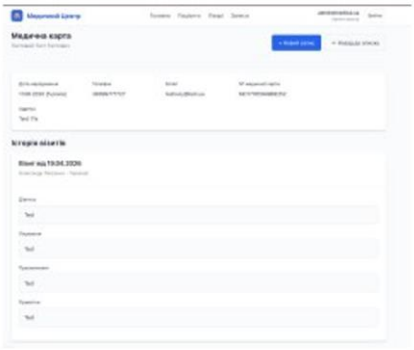

Рисунок 6 – Інтерфейс медичної карти пацієнта

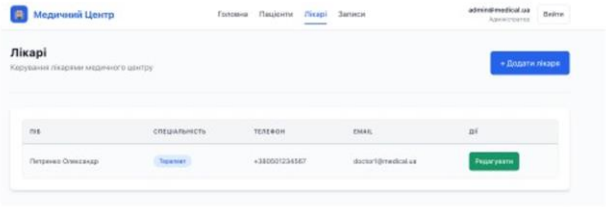

Рисунок 7 – Інтерфейс спец-карти лікаря

Апробація результатів дослідження

1. Лазневий А. В.. Аронов А. О. Інформаційна система обліку пацієртів медичного центру // Всеукраїнська наукова технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» Збірник тез. К.: ДУІКТ, 2026.
2. Лазневий А. В.. Аронов А. О. Інформаційна система обліку пацієртів медичного центру //Всеукраїнська наукова технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» Збірник тез. К.: ДУІКТ, 2026.

ВИСНОВКИ

1. Розглянуто теоретичні основи МІС та виявлено проблеми традиційного обліку пацієнтів, що обґрунтовує необхідність цифровізації

2. Проаналізовано веб-технології HTML5, CSS3, JavaScript та обґрунтовано вибір MS SQL Server як СУБД

3. Сформовано функціональні та нефункціональні вимоги; розроблено трирівневу клієнт-серверну архітектуру

4. Спроєктовано структуру БД (Users, Patients, Doctors, Appointments, MedicalRecords) із забезпеченням цілісності

5. Реалізовано REST API на Node.js/Express із JWT-автентифікацією та рольовою моделлю доступу (admin/doctor/patient)

6. Розроблено клієнтську частину системи: авторизація, адмін-панель, управління пацієнтами та лікарями, медична карта

7. Отримано працездатну веб-систему, придатну до подальшого розширення та впровадження в реальних умовах медичного закладу

ДОДАТОК Б. ЛІСТИНГИ

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Медичний Центр - Система обліку пацієнтів</title>
  <link rel="stylesheet" href="frontend/css/variables.css">
  <link rel="stylesheet" href="frontend/css/reset.css">
  <link rel="stylesheet" href="frontend/css/base.css">
  <link rel="stylesheet" href="frontend/css/components.css">
  <style>
    * {
      box-sizing: border-box;
    }

    .landing-page {
      min-height: 100vh;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      display: flex;
      flex-direction: column;
      font-family: var(--font-family, 'Segoe UI', system-ui, sans-serif);
    }

    .landing-header {
      background: rgba(255, 255, 255, 0.1);
      backdrop-filter: blur(10px);
      border-bottom: 1px solid rgba(255, 255, 255, 0.2);
      position: sticky;
      top: 0;
      z-index: 100;
    }

    .landing-header-content {
      max-width: 1200px;
      margin: 0 auto;
      padding: 1rem 2rem;
      display: flex;
      align-items: center;
      justify-content: space-between;
    }

    .landing-logo {
      display: flex;
      align-items: center;
      gap: 0.75rem;
      color: white;
      font-size: 1.5rem;
      font-weight: 700;
      text-decoration: none;
      transition: transform 0.3s ease;
```

```
}

.landing-logo:hover {
  transform: scale(1.02);
}

.landing-logo-icon {
  width: 44px;
  height: 44px;
  background: white;
  color: #667eea;
  border-radius: 12px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 1.5rem;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}

.landing-nav {
  display: flex;
  align-items: center;
  gap: 2rem;
}

.landing-nav a {
  color: rgba(255, 255, 255, 0.9);
  font-weight: 500;
  text-decoration: none;
  transition: all 0.3s ease;
  padding: 0.5rem 0;
  position: relative;
}

.landing-nav a:hover {
  color: white;
}

.landing-nav a::after {
  content: '';
  position: absolute;
  bottom: 0;
  left: 0;
  width: 0;
  height: 2px;
  background: white;
  transition: width 0.3s ease;
}

.landing-nav a:hover::after {
  width: 100%;
}
```

```
.landing-btn {
  padding: 0.75rem 1.5rem;
  border-radius: 10px;
  font-weight: 600;
  transition: all 0.3s ease;
  text-decoration: none;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  gap: 0.5rem;
}

.landing-btn-primary {
  background: white;
  color: #667eea !important;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}

.landing-btn-primary:hover {
  background: rgba(255, 255, 255, 0.95);
  color: #5a67d8 !important;
  transform: translateY(-2px);
  box-shadow: 0 6px 20px rgba(0, 0, 0, 0.25);
}

.landing-hero {
  flex: 1;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 4rem 2rem;
  position: relative;
  overflow: hidden;
}

.landing-hero::before {
  content: '';
  position: absolute;
  top: -50%;
  right: -20%;
  width: 600px;
  height: 600px;
  background: rgba(255, 255, 255, 0.05);
  border-radius: 50%;
  animation: float 20s infinite ease-in-out;
}

.landing-hero::after {
  content: '';
  position: absolute;
  bottom: -30%;
  left: -10%;
  width: 400px;
```

```
  height: 400px;
  background: rgba(255, 255, 255, 0.05);
  border-radius: 50%;
  animation: float 15s infinite ease-in-out reverse;
}

@keyframes float {
  0%, 100% { transform: translate(0, 0) scale(1); }
  50% { transform: translate(30px, -30px) scale(1.05); }
}

.landing-hero-content {
  max-width: 800px;
  text-align: center;
  color: white;
  position: relative;
  z-index: 1;
}

.landing-hero h1 {
  font-size: 3.5rem;
  font-weight: 800;
  margin-bottom: 1.5rem;
  line-height: 1.1;
  text-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
}

.landing-hero p {
  font-size: 1.35rem;
  opacity: 0.95;
  margin-bottom: 2.5rem;
  line-height: 1.6;
  max-width: 600px;
  margin-left: auto;
  margin-right: auto;
}

.landing-hero-buttons {
  display: flex;
  gap: 1rem;
  justify-content: center;
  flex-wrap: wrap;
}

.landing-btn-large {
  padding: 1rem 2.5rem;
  font-size: 1.125rem;
}

.landing-btn-outline {
  background: transparent;
  border: 2px solid rgba(255, 255, 255, 0.8);
  color: white;
```

```
}

.landing-btn-outline:hover {
  background: white;
  color: #667eea;
  transform: translateY(-2px);
}

.landing-features {
  background: linear-gradient(180deg, #f8fafc 0%, #ffffff 100%);
  padding: 5rem 2rem;
}

.landing-features-header {
  text-align: center;
  max-width: 600px;
  margin: 0 auto 3rem;
}

.landing-features-header h2 {
  font-size: 2.25rem;
  font-weight: 700;
  color: #1f2937;
  margin-bottom: 1rem;
}

.landing-features-header p {
  color: #6b7280;
  font-size: 1.125rem;
}

.landing-features-grid {
  max-width: 1200px;
  margin: 0 auto;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
  gap: 2rem;
}

.landing-feature {
  text-align: center;
  padding: 2.5rem 2rem;
  background: white;
  border-radius: 16px;
  box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.05), 0 2px 4px -1px rgba(0, 0, 0, 0.03);
  transition: all 0.3s ease;
  border: 1px solid #f3f4f6;
}

.landing-feature:hover {
  transform: translateY(-5px);
  box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04);
}
```

```
.landing-feature-icon {
  width: 72px;
  height: 72px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  border-radius: 20px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 2rem;
  margin: 0 auto 1.5rem;
  box-shadow: 0 8px 16px rgba(102, 126, 234, 0.3);
}

.landing-feature h3 {
  font-size: 1.25rem;
  font-weight: 700;
  margin-bottom: 0.75rem;
  color: #1f2937;
}

.landing-feature p {
  color: #6b7280;
  line-height: 1.6;
  font-size: 0.95rem;
}

.landing-stats {
  background: white;
  padding: 4rem 2rem;
  border-top: 1px solid #e5e7eb;
}

.landing-stats-grid {
  max-width: 1000px;
  margin: 0 auto;
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 3rem;
  text-align: center;
}

.landing-stat h4 {
  font-size: 2.5rem;
  font-weight: 800;
  color: #667eea;
  margin-bottom: 0.5rem;
}

.landing-stat p {
  color: #6b7280;
  font-weight: 500;
```

```
    }

    .landing-footer {
      background: #1f2937;
      color: #9ca3af;
      text-align: center;
      padding: 2.5rem 2rem;
    }

    .landing-footer p {
      margin: 0;
    }

    /* Responsive */
    @media (max-width: 768px) {
      .landing-hero h1 {
        font-size: 2.25rem;
      }

      .landing-hero p {
        font-size: 1.1rem;
      }

      .landing-header-content {
        padding: 1rem;
      }

      .landing-logo {
        font-size: 1.25rem;
      }

      .landing-logo-icon {
        width: 38px;
        height: 38px;
        font-size: 1.25rem;
      }

      .landing-nav {
        gap: 1rem;
      }

      .landing-features-grid {
        grid-template-columns: 1fr;
      }
    }
  </style>
</head>
<body>
  <div class="landing-page">
    <header class="landing-header">
      <div class="landing-header-content">
        <a href="/" class="landing-logo">
```

```
      <span class="landing-logo-icon">🏥</span>
      <span>Медичний Центр</span>
    </a>

    <nav class="landing-nav">
      <a href="#features">Можливості</a>
      <a href="frontend/pages/login.html" class="landing-btn landing-btn-primary">Увійти</a>
    </nav>
  </div>
 </header>

 <section class="landing-hero">
  <div class="landing-hero-content">
    <h1>Сучасна система обліку пацієнтів</h1>
    <p>Ефективне керування медичними картами, записами на прийом та історією візитів для медичних закладів</p>

    <div class="landing-hero-buttons">
      <a href="frontend/pages/login.html" class="landing-btn landing-btn-primary landing-btn-large">
        🔐 Увійти в систему
      </a>
      <a href="#features" class="landing-btn landing-btn-outline landing-btn-large">
        Дізнатися більше
      </a>
    </div>
  </div>
 </section>
</div>

<section id="features" class="landing-features">
 <div class="landing-features-header">
 <h2>Можливості системи</h2>
  <p>Все необхідне для ефективної роботи медичного закладу</p>
 </div>

 <div class="landing-features-grid">
  <div class="landing-feature">
    <div class="landing-feature-icon">👥</div>
    <h3>Реєстрація пацієнтів</h3>
    <p>Зручне додавання та керування особистими даними пацієнтів з унікальними номерами медичних карт</p>
  </div>

  <div class="landing-feature">
    <div class="landing-feature-icon">📅</div>
    <h3>Запис на прийом</h3>
    <p>Онлайн-запис до лікарів різних спеціальностей з переглядом розкладу</p>
  </div>

  <div class="landing-feature">
    <div class="landing-feature-icon">📋</div>
    <h3>Медичні карти</h3>
    <p>Ведення повної історії візитів, діагнозів, призначень та лікування</p>
  </div>
```

```
      <div class="landing-feature">
        <div class="landing-feature-icon">🔐</div>
        <h3>Ролі та доступ</h3>
        <p>Розділення прав доступу для адміністраторів, лікарів та пацієнтів</p>
      </div>
    </div>
  </section>

  <footer class="landing-footer">
    <p>© 2024 Медичний Центр. Система обліку пацієнтів.</p>
  </footer>

  <script src="frontend/js/auth.js"></script>
  <script>
    // Redirect if already logged in
    if (auth.isLoggedIn()) {
      auth.redirectToDashboard();
    }
  </script>
</body>
</html>
```