ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему *Розробка веб-системи прийому пацієнтів медичних установ*

Виконав: здобувач *3* року навчання
групи *ЗПРс*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Подобін А. Л.
(підпис, прізвище та ініціали)

Керівник *Величко Ю.І.*
(підпис, прізвище та ініціали)

Рецензент *к.т.н., доцент Корніловська Н.В*
(прізвище та ініціали)

Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*
Кафедра, циклова комісія *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *бакалавр*
Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)
Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри *програмних засобів і технологій*
_____ О.Є. Огнєва
«___» _____ 2026 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Подобін Артем Леонідович
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розробка веб-системи прийому пацієнтів медичних установ*

керівник проєкту (роботи) к.т.н. доцент Величко Ю.І.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «_16_» ___січня_ 2026 року №_83-с_

2. Строк подання здобувачем проєкту(роботи) 10.06.2026

3. Вихідні дані до проєкту (роботи) літературні та періодичні джерела, матеріали переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1.Дослідження предметної області веб-сайтів медичних установ та обґрунтування технологічного стеку
2. Проектування архітектури клієнт-серверного застосунку бази даних та програмного інтерфейсу API
3. Програмна реалізація відео-консультації за допомогою Firebase
4. Тестування та впровадження продукту.

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

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання 16.01.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	10.02.2026	Виконано
2	Підбір літератури	11.02.2026-21.02.2026	Виконано
3	Аналіз предметної області	22.02.2026-20.03.2026	Виконано
4	Розробка та обґрунтування завдання	21.03.2026-15.03.2026	Виконано
5	Розробка концептуальної моделі	16.03.2026-25.03.2026	Виконано
6	Моделювання та проектування системи	26.03.2026-07.04.2026	Виконано
7	Моделювання та проектування бази даних	08.04.2026-15.04.2026	Виконано
8	Розробка інтерфейсу сайту	16.04.2026-22.04.2026	Виконано
9	Тестування сайту	23.04.2026-27.04.2026	Виконано
10	Оформлення пояснювальної записки	28.04.2026-29.05.2026	Виконано
11	Захист кваліфікаційної роботи		Виконано

Здобувач _____ Подобін А.Л.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Величко Ю. І.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка містить: 74 сторінки, 34 рисунків, 1 таблиці, 30 джерел, 2 додатки.

Об’єкт дослідження - процес організації прийому пацієнтів, ведення електронної медичної документації та надання дистанційних медичних консультацій у приватних медичних установах

Предмет дослідження – методи, технології та програмні засоби розробки веб-системи для авторизації діяльності медичних закладів і взаємодії між пацієнтами та медичним персоналом.

Мета роботи - проектування та програмна реалізація веб-додатку прийому пацієнтів медичних установ, яка забезпечує реєстрацію користувача, введення електронних медичних карток, управління записами на прийом, перегляд результатів лабораторних досліджень та проведення онлайн-консультації.

У кваліфікаційній роботі проведено аналіз сучасних інформаційних систем у сфері охорони здоров’я та досліджено особливості автоматизації процесів обслуговування пацієнтів у приватних медичних закладах. На основі проведеноо аналізу обґрунтовано вибір технологічного стеку, до складу якого увійшли фреймворк Laravel для реалізації серверної частини, бібліотека React для клієнтського інтерфейсу та система керування базами даних MySQL для зберігання інформації.

Окрему увага приділено реалізації модуля авторизації та онлайн-консультацій між пацієнтом та сімейний лікарем. Використання сучасних веб-технологій забезпечує можливість проведення дистанційних консультацій у режимі реального часу, що підвищує доступність медичних послуг та покращує взаємодію між учасниками системи.

КЛЮЧОВІ СЛОВА: ВЕБ-СИСТЕМА, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА МЕДИЧНА КАРТКА, ОНЛАЙН-КОНСУЛЬТАЦІЇ, ЗАПИС НА ПРИЙМ, REACT, LARAVEL, MYSQL, WEBRTC, КЛІЄНТ-СЕРВЕРНА АХРІТЕКТУРА.

АНОТАЦІЯ

Кваліфікаційну роботу присвячено проектуванню та розробці веб-системи прийому пацієнтів медичних установ. Актуальність теми зумовлена необхідністю цифровізації процесів надання медичних послуг, автоматизації ведення медичної документації та забезпечення зручної взаємодії між пацієнтами й медичним персоналом. Метою проекту було створення сучасної інформаційної системи, яка надає можливість пацієнтам отримувати інформацію про медичний заклад, записуватися на прийом до лікарів, переглядати власну медичну картку та результати лабораторних досліджень, а також брати участь в онлайн-консультаціях.

Під час виконання роботи було проведено аналіз існуючих медичних інформаційних систем та сучасних підходів до побудови веб-додаток. Для реалізації програмного інтерфейсу обрано архітектурний стиль REST, який забезпечує зручну взаємодію між клієнтською та серверною частинами системи. Серверну частину розроблено за допомогою фреймворку Laravel мовою програмування PHP. Використання можливостей фреймворку дозволило реалізувати безпечну систему автентифікації користувачів, розмежування прав доступу та механізми валідації даних. Особливу увагу приділено проектуванню бази даних та організації зберігання медичної інформації. Основним сховищем даних виступає реляційна система керування базами даних MySQL. Структура бази даних забезпечує зберігання інформації про користувачів системи, лікарів, пацієнтів, медичні картки, результати лабораторних аналізів, записи на прийом та онлайн-консультації. Для роботи з базою даних використано ORM Eloquent, що дозволило спростити взаємодію між програмним кодом та реляційними таблицями. Клієнтську частину системи реалізовано за допомогою бібліотеки React у форматі Single Page Application. Такий підхід забезпечує швидку взаємодію користувача з веб-додатком без необхідності повного перезавантаження сторінок. Інтерфейс системи адаптований до різних категорій користувачів та включає особистий кабінет пацієнта, сторінки перегляду медичної інформації, модуль запису на

прийом та спеціалізовані панелі для медичного персоналу. Для лікарів реалізовано окрему медичну панель, яка надає можливість переглядати актуальні записи на прийом, працювати з електронними медичними картками пацієнтів, формувати висновки за результатами консультацій та зберігати медичні дані в централізованій системі. Такий підхід дозволяє зменшити обсяг паперової документації та підвищити ефективність роботи медичного персоналу. Важливою складовою розробленої системи є модуль відео-консультації між пацієнтом та сімейним лікарем. Реалізований функціонал забезпечує проведення дистанційних консультацій у режимі реального часу, що особливо актуально для пацієнтів, які не мають можливості особисто відвідати медичний заклад. Інтеграція засобів відеозв'язку сприяє підвищенню доступності медичних послуг та покращенню комунікації між учасниками лікувального процесу.

У результаті виконання роботи створено повнофункціональну веб-систему, яка автоматизує процеси запису на прийом, ведення електронної медичної документації та дистанційного консультування пацієнтів. Розроблений програмний продукт може бути використаний як основа для подальшого впровадження цифрових технологій у діяльність медичних установ та розширення функціональних можливостей електронних систем охорони здоров’я.

ABSTRACT

The qualification work is devoted to the design and development of a web system for receiving patients in medical institutions. The relevance of the topic is due to the need to digitize the processes of providing medical services, automate medical documentation, and ensure convenient interaction between patients and medical personnel. The goal of the project was to create a modern information system that allows patients to receive information about the medical institution, make appointments with doctors, view their own medical records and laboratory test results, and participate in online consultations.

During the work, an analysis of existing medical information systems and modern approaches to building web applications was conducted. The REST architectural style was chosen to implement the software interface, which provides convenient interaction between the client and server parts of the system. The server part was developed using the Laravel framework in the PHP programming language. Using the capabilities of the framework allowed the implementation of a secure user authentication system, delimitation of access rights, and data validation mechanisms. Particular attention was paid to the design of the database and the organization of storage of medical information. The main data repository is the MySQL relational database management system. The database structure provides storage of information about system users, doctors, patients, medical records, laboratory test results, appointment records and online consultations. The Eloquent ORM was used to work with the database, which simplified the interaction between the program code and relational tables. The client part of the system was implemented using the React library in the Single Page Application format. This approach ensures fast user interaction with the web application without the need to completely reload the pages. The system interface is adapted to different categories of users and includes a patient's personal account, medical information viewing pages, an appointment recording module and specialized panels for medical staff.

A separate medical panel has been implemented for doctors, which provides the ability to view current appointment records, work with patients' electronic medical records, form conclusions based on the results of consultations and store medical data in

a centralized system. This approach allows to reduce the volume of paper documentation and increase the efficiency of medical personnel.

An important component of the developed system is the video consultation module between the patient and the family doctor. The implemented functionality provides remote consultations in real time, which is especially important for patients who do not have the opportunity to personally visit a medical institution. The integration of video communication tools helps to increase the accessibility of medical services and improve communication between participants in the treatment process.

As a result of the work, a fully functional web system was created that automates the processes of making an appointment, maintaining electronic medical records and remote consultation of patients. The developed software product can be used as a basis for further implementation of digital technologies in the activities of medical institutions and expanding the functionality of electronic health systems.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах цифровізації сфери охорони здоров’я зростає потреба у впровадженні інформаційних систем, що автоматизують взаємодію між медичними установами, лікарями та пацієнтами. Традиційні методи запису на прийом, ведення медичної документації та обміну інформацією залишаються малоефективними, оскільки потребують значних часових витрат, створюють ризик помилок і не забезпечують оперативного доступу до даних.

Використання веб-технологій дозволяє централізувати медичну інформацію, спростити процес запису на прийом та підвищити ефективність роботи персоналу. Особливої актуальності набуває розвиток телемедицини та дистанційної взаємодії між лікарем і пацієнтом, що забезпечує доступ до медичних послуг незалежно від місця перебування користувача.

Об’єкт дослідження – процес взаємодії між пацієнтами, лікарями та працівниками медичних установ із використанням веб-технологій.

Предмет дослідження – методи та програмні засоби розробки веб-систем для автоматизації запису на прийом, ведення електронної медичної документації та організації дистанційних консультацій.

Метоа роботи - проєктування та розробка веб-системи прийому пацієнтів медичної установи, яка забезпечує онлайн-запис до лікаря, ведення електронних медичних карток, обробку медичних даних та проведення відео-консультацій.

Для досягнення поставленої мети необхідно:

- проаналізувати предметну область та існуючі рішення;
- визначити функціональні та нефункціональні вимоги до системи;
- спроєктувати архітектуру веб-застосунку;
- розробити серверну та клієнтську частини системи;
- реалізувати основні модулі: автентифікацію, запис на прийом, електронну медичну картку та онлайн-консультації;
- провести тестування основного функціоналу системи.

Практичним результатом роботи є реалізовані основні модулі веб-системи медичної установи, які забезпечують автоматизацію ключових процесів взаємодії між пацієнтами та медичним персоналом. Система дозволяє здійснювати онлайн-запис на прийом, керувати медичними даними пацієнтів, формувати електронні медичні записи та організовувати дистанційні консультації. Запропоновані рішення можуть бути використані як основа для впровадження у медичних закладах різного рівня та подальшого розвитку функціоналу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАЧІ

1.1. Аналіз предметної області

Сфера охорони здоров’я є невід’ємною частиною сучасного суспільства. Щодня медичні заклади обслуговують велику кількість пацієнтів, обробляють значні обсяги інформації та забезпечують взаємодію між медичним персоналом і відвідувачами. Через такі навантаження виникає потреба у впровадженні сучасних інформаційних технологій для оптимізації процесів та підвищення якості надання медичних послуг.

Одним із ключових процесів діяльності медичного закладу є організація прийому пацієнтів. Традиційні способи запису на прийом або звернення до лікаря здійснюються шляхом особистого відвідування медичного закладу або телефонного дзвінка. Такий підхід є поширеним і сьогодні, однак він має низку суттєвих недоліків. Працівники реєстратури витрачають значну кількість часу на обробку дзвінків, уточнення даних пацієнтів та внесення інформації до журналу обліку. У свою чергу, пацієнти часто стикаються з необхідністю очікування відповіді та відсутністю актуальної інформації про вільний час прийому. Цифровізація поступово охоплює медичну сферу. Розвиток веб-технологій, хмарних сервісів і мобільних пристроїв створив умови для переходу від паперової документації до електронних систем. Все більше медичних закладів впроваджують програмні рішення, які автоматизують адміністративні процеси та спрощують взаємодію між персоналом і пацієнтами.

Сучасні веб-системи прийому пацієнтів є не лише інструментом онлайн-запису. Залежно від функціональних можливостей вони можуть включати модулі керування електронними медичними картками, систему нагадувань про візити, відео-консультації, управління розкладом персоналу та формування звітності. Такі модулі створюють єдиний інформаційний простір, у межах якого всі учасники процесу можуть оперативно отримувати необхідну інформацію.

Потреба в таких рішеннях особливо гостро проявилася під час пандемії COVID-19. Медичним закладам доводилося мінімізувати скупчення людей та

організовувати дистанційну взаємодію з пацієнтами. Електронні системи запису та онлайн-консультацій у багатьох країнах стали важливим інструментом надання медичних послуг у цей період. Після завершення пандемії попит на цифрові медичні сервіси не зменшився, а навпаки продовжив зростати завдяки їх зручності та доступності.

Перевагою веб-систем є цілодобовий доступ до функціоналу незалежно від місця перебування користувача. Пацієнт може переглядати графік роботи лікаря, записуватися на прийом або скасовувати візит у будь-який зручний час. Для медичного персоналу це означає ефективніше використання робочого часу та спрощення планування навантаження.

Водночас із перевагами існує низка вимог, які необхідно враховувати під час розробки таких систем. Основною вимогою є захист персональних даних. Медична інформація пацієнтів належить до категорії конфіденційних даних, тому система повинна забезпечувати надійну автентифікацію користувачів, контроль доступу та захист інформації. Не менш важливими є стабільність роботи програмного забезпечення, масштабованість, швидкодія та зручність інтерфейсу. Процеси, що підлягають автоматизації в медичних установах, можна визначити таким чином:

- інформування пацієнтів про клініку, її відділення, лікарів та перелік послуг із зазначенням цін;
- реєстрація нових пацієнтів та їх авторизація в особистому електронному кабінеті;
- онлайн-запис на прийом до конкретного лікаря з вибором зручного часу;
- ведення амбулаторної картки пацієнта з можливістю доступу для відповідних лікарів;
- фіксація результатів кожного прийому лікарем із збереженням у базі даних;
- завантаження та перегляд результатів лабораторних аналізів;
- проведення онлайн-консультацій між сімейним лікарем та пацієнтом.

Таким чином, розробка комплексної інформаційної системи для медичної установи є обґрунтованою необхідністю, що відповідає сучасним технологічним стандартам і реальним потребам учасників медичного процесу. Система повинна забезпечити зручну та безпечну взаємодію між пацієнтами, лікарями та адміністраторами в межах єдиної інтегрованої платформи, охоплюючи повний цикл надання медичних послуг.

1.2. Огляд та аналіз існуючих рішень

Обґрунтування актуальності та визначення вимог до розробки системи здійснювалося на основі дослідження веб-сайтів кількох приватних клінік, що функціонують на території України. До аналізу були залучені такі заклади: «Medibor», «Добробут», «Oxford Medical», «Асклепій» та «Нікамед». Кожен із цих закладів відповідає вимогам законодавства України щодо надання медичних послуг у цифровому форматі. Усі вони мають структуровані інформаційні системи, які надають пацієнтам відомості про відділення, послуги та лікарів. Це дозволяє користувачам обрати потрібного спеціаліста відповідно до його опису та профілю діяльності.

Приватна клініка «Medibor»(рис. 1.1) має добре структуровану навігацію веб-сайту, містить опис послуг і їх вартість, а також перелік лікарів, які надають ці послуги. Окремо представлена інформація про філії клініки в різних районах міста, що забезпечує її доступність для пацієнтів. Веб-сайт виконаний у стриманому корпоративному стилі з чіткою ієрархічною структурою меню. Основна навігаційна панель містить такі розділи: послуги, лікарі, філії, ціни, новини та вакансії. Розділ послуг реалізовано у вигляді багаторівневого меню з поділом на категорії та детальним описом кожної послуги. Попри широкий функціонал, глибока вкладеність підрозділів може ускладнювати навігацію для користувачів, які вперше відвідують сайт. На сайті також передбачено особистий кабінет пацієнта, який реалізований через зовнішній сервіс. Однак перенаправлення користувача на сторонній ресурс може знижувати рівень довіри та порушувати цілісність інтерфейсу.

Запис на прийом реалізовано у форматі зворотного зв’язку. Пацієнт подає заявку, після чого з ним зв’язується оператор і пропонує доступні дату та час. Такий підхід є недостатньо зручним, оскільки при великій кількості звернень оператори можуть не встигати своєчасно обробляти всі заявки. Це призводить до затримок у підтвердженні запису, які можуть становити кілька хвилин або більше з моменту подачі заявки [5].

Після подання заявки пацієнт змушений очікувати зворотного зв’язку, що створює певні незручності, особливо у періоди високого навантаження. Медичний персонал, відповідальний за обробку записів, може не встигати своєчасно відповідати на всі звернення, що негативно впливає на користувацький досвід і може спонукати пацієнтів звертатися до альтернативних сервісів. Відсутність системи онлайн-запису з миттєвим підтвердженням є суттєвим недоліком з точки зору сучасних стандартів цифрового медичного обслуговування.

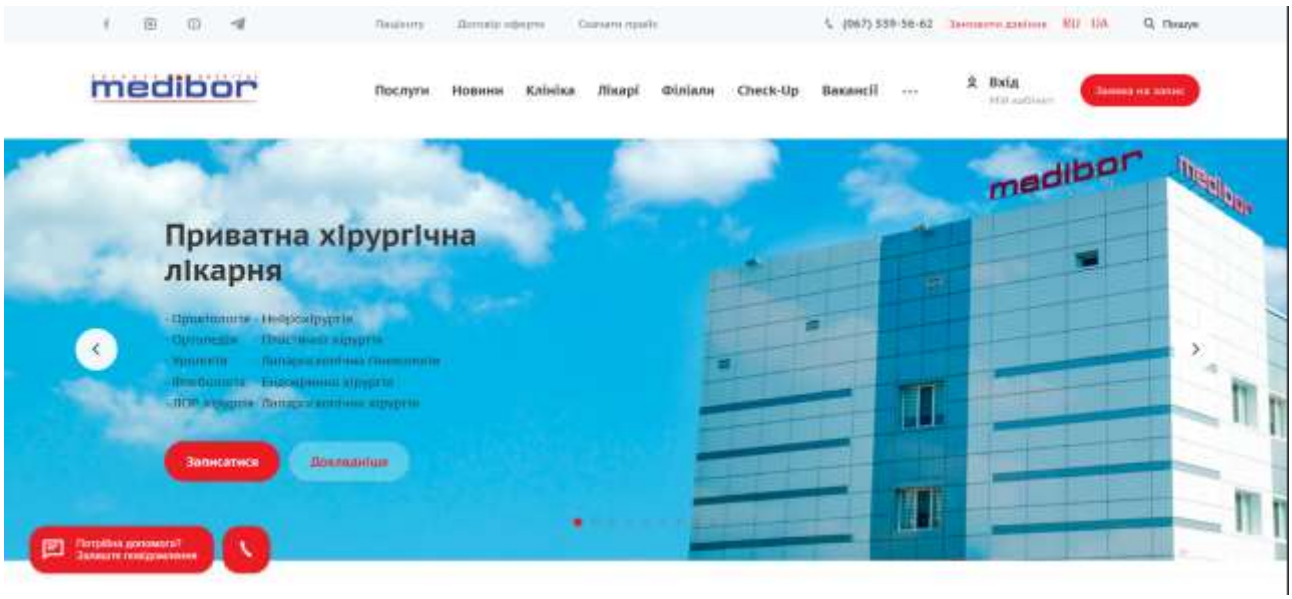


Рисунок 1.1. Веб-сайт приватної клініки Medibor

Медичний центр «Асклепій»(рис. 1.2) є багатопрофільним закладом, який функціонує на базі кількох філіалів у Житомирі та Бердичеві. Веб-сайт клініки вирізняється розгалуженою структурою напрямків, що охоплюють дорослу і дитячу поліклініки, інструментальну діагностику, відновлення та реабілітацію, а

також відділення оперативних втручання як для дорослих, так і для дітей. Навігація по напрямках реалізована через розгорнуте меню з іконками категорій, що спрощує орієнтацію для нових відвідувачів сайту. Перелік лікарів подано з фотографіями, рейтингом, кількістю відгуків та роками досвіду. Кожен лікар має індивідуальну сторінку з описом спеціалізації та кнопкою запису. Однак кнопка «Записатись» перенаправляє користувача на зовнішню систему e-zapis.com, що, як і у випадку з «Medibor», порушує єдність платформи та може знизити рівень довіри пацієнта. Перевагою «Асклепія» є наявність окремого розділу для замовлення аналізів з можливістю виклику медичної сестри на дім, що розширює доступність послуг. Також присутній розділ «Акції та новини», що свідчить про активну інформаційну роботу з пацієнтами. Особистий кабінет реалізований через сторонній ресурс patient-docs.com, що є аналогічним недоліком до вже розглянутих систем[3].

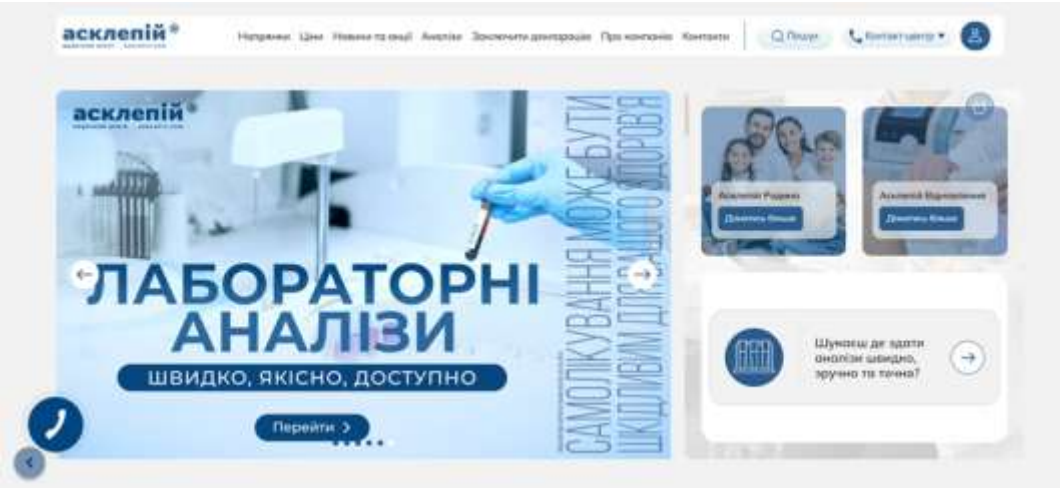


Рисунок 1.2. Веб-сайт приватної клініки Асклепія

Медичний заклад «Oxford Medical» також має добре структуровану навігацію веб-сайту та пропонує широкий спектр послуг у різних галузях медицини. Характерною особливістю закладу є поділ напрямків за віковими категоріями, що спрощує пошук необхідних спеціалістів для різних груп пацієнтів. Клініка позиціонує себе як мережевий медичний заклад із відпрацьованими стандартами якості, що позитивно впливає на структуру та зручність використання веб-ресурсу.

Проте, як і в більшості конкурентів, система запису на прийом не забезпечує миттєвого підтвердження. Після подання заявки пацієнт повинен очікувати на зворотний зв’язок від оператора. Крім того, відсутність інтегрованого особистого кабінету в межах власної платформи також є характерним недоліком цього сервісу [7].

Клініка «Добробут» є найбільш функціонально розвиненою серед досліджених медичних установ. Вона пропонує розширені можливості онлайн-запису, а також послугу виклику лікаря додому. Інформаційна система клініки містить детальний опис відділень і медичних послуг. Значна увага приділена представленню лікарів: для кожного спеціаліста наведено інформацію про спеціалізацію, досвід роботи та професійний профіль. Це дає пацієнтам можливість детально ознайомитися з лікарями та обрати найбільш відповідного фахівця [4].

Медичний центр «Нікамед» є найбільш вузькоспеціалізованою установою серед досліджених. Вебсайт клініки має просту структуру та охоплює шість основних напрямків: акушерство і гінекологію, ультразвукову діагностику, неврологію, ендокринологію, лабораторну діагностику та стоматологію. Сайт містить детальний прайс-лист із вартістю кожної процедури, що є безперечною перевагою з точки зору прозорості для пацієнтів.

Водночас «Нікамед» поступається конкурентам за рівнем цифровізації сервісів. Режим роботи обмежений трьома днями на тиждень — понеділком, середою та п’ятницею з 10:00 до 16:00, що суттєво знижує доступність медичних послуг. Запис на прийом здійснюється лише через телефонний дзвінок або форму зворотного зв’язку без можливості самостійного вибору дати та часу онлайн. Профілі лікарів представлені, однак містять мінімальний обсяг інформації. Особистий кабінет пацієнта на сайті відсутній, що робить «Нікамед» найменш цифровізованим закладом серед досліджуваних [6].

Проведений аналіз дозволяє визначити основні переваги сучасних медичних інформаційних систем:

- структурований перелік медичних послуг із поділом за напрямками та цільовою аудиторією;

- можливість завантаження прайс-листів у форматі PDF;
- наявність окремих розділів для різних філій;
- актуальна інформація про лікарів, їх спеціалізації та графік роботи;
- підтримка широкого спектра діагностичних процедур і медичних послуг;
- можливість онлайн-запису на прийом із вибором лікаря, дати та часу;
- широкий каталог профільних медичних центрів;
- зручний пошук лікарів за спеціалізацією.
- Водночас для досліджених систем характерні й певні недоліки:
- особистий кабінет пацієнта реалізований через сторонні сервіси, а не в межах основної платформи;
- у деяких медичних установах відсутня можливість безпосереднього онлайн-запису через сайт, натомість передбачено лише подання заявки;
- відсутня можливість проведення онлайн-консультацій із лікарем;
- окремі системи орієнтовані на великі мережі медичних закладів, що може ускладнювати їх використання для невеликих клінік.

Проведений аналіз аналогів дозволив визначити як сучасні тенденції розвитку медичних інформаційних систем, так і їхні основні недоліки. Отримані результати можуть бути використані під час проєктування власної системи для уникнення незручностей та підвищення ефективності роботи майбутнього програмного продукту. Навіть найбільш розвинена з розглянутих систем — «Добробут» — не забезпечує повного циклу взаємодії між пацієнтом і лікарем у межах єдиної інтегрованої платформи, яка б поєднувала доступ до електронної медичної картки, результатів аналізів та проведення онлайн-консультацій.

1.3. Обґрунтування вибору інструментів та технологій розробки

Вибір технологічного стеку один з найважливіших архітектурних рішень при розробці будь-якого веб-додатку. Правильно підібрані інструменти для розробки залежать не лише від зручності самої розробки, а й продуктивності готових рішень,

простота його супроводу і можливості масштабування у майбутньому. Вибір кожного інструмента та технологій буде обґрунтовано починаючи від архітектурного підходу, серверного фреймворку, клієнтської бібліотеки, реляційної бази даних та хмарних сервісів.

1.3.1. Вибір архітектурного підходу

На початку проектування веб-додатку було розглянуто дві основні архітектурні підходи: традиційні монолітний та сучасний підхід із розділенням на окремий клієнтський застосунок (SPA) і серверний програмний інтерфейс (REST API).

Реалізація монолітного підходу, зокрема, виконується за допомогою засобами шаблона Blade у Laravel, який передбачає формування HTML-сторінок безпосередньо на сервері. Такий варіант є простіший для невеликих проектів, але має свої суттєві обмеження: кожна дія користувача викликає повне перезавантаження сторінки, негативно впливаючи на швидкодію та комфорт користування системою. Підхід SPA у поєднані з REST API усуває подібні недоліки у системі, що робить її комфортною для використанні. Клієнтська частина завантажується один раз при завантажені веб-додатку і надалі динамічно оновлює лише потрібні компоненти інтерфейсу, обмінюючись із сервером структурованим JSON-відповідями. Воно забезпечує плавність роботи застосунку, що важливо для медичної системи адже лікар чи пацієнт не повинні відчувати затримок при переключення між розділами особистого кабінету, перегляду амбулаторної картки або замовлення даних[17; 18;].

Розділення на API і клієнтську частину дає змогу в майбутньому підключити мобільний застосунок до того ж API без жодних змін на сервері. Саме тому архітектура SPA та REST API була обрана як основна для розробки веб-додатку.

1.3.2. Вибір серверного серверної частини

Під час вибору технології для реалізації серверної частини у форматі REST API було розглянуто кілька популярних рішень: Laravel (PHP), Node.js з Express, Django (Python) та FastAPI (Python).

Laravel є сучасним PHP-фреймворком із чіткою архітектурою MVC. Його перевагами є потужна ORM Eloquent, розвинена система міграцій баз даних, зручні засоби маршрутизації запитів та широка екосистема готових пакетів. Розробка REST API у Laravel здійснюється за допомогою спеціалізованих інструментів, зокрема контролерів, ресурсів API (API Resources) та механізмів серіалізації даних. Додатковою перевагою є проста інтеграція з Firebase для перевірки токенів автентифікації [17].

Node.js у поєднанні з фреймворком Express забезпечує високу гнучкість під час розробки серверних застосунків. Водночас відсутність єдиного підходу до організації структури проєкту може ускладнювати підтримку коду та його подальший розвиток у командній роботі.

Django є потужним і безпечним фреймворком для розробки вебзастосунків на мові Python. Однак використання Python потребує окремого налаштування середовища виконання та менш природно інтегрується з поширеною PHP-орієнтованою інфраструктурою хостинг-провайдерів.

Під час аналізу також розглядався FastAPI, який вирізняється високою продуктивністю та зручними засобами створення REST API. Проте цей фреймворк менш поширений у корпоративних проєктах порівняно з Laravel і має меншу кількість готових рішень для інтеграції з використовуваними в роботі технологіями.

З урахуванням чіткої архітектури, великої кількості навчальних матеріалів і документації, зручної роботи з реляційними базами даних та простоти розробки REST API було обрано фреймворк Laravel. Додатковим аргументом на його користь стала можливість швидкої інтеграції з сервісом Firebase. Перевірка токенів автентифікації може бути реалізована за допомогою пакета kreait/laravel-firebase без необхідності створення складних власних механізмів. Це забезпечує природну

та ефективну взаємодію між обраними технологіями й спрощує реалізацію системи автентифікації. Детально наведено порівняння фреймворків у таблиці 1.1.

Таблиця 1.1. Порівняння серверних фреймворків

Критерії	Laravel	Node.js/Express	Django	FastApi
Мова	PHP	JavaScript	Python	Python
Будова ORM	Так(Eloquent)	Ні(потрібно окремо)	Так	Частково
Документація	Відмінна	Фрагментовано	Добра	Добра
Авторизація/JWT	Вбудована	Зовнішні пакети	Вбудована	Зовнішні пакети
Порядок у структурі	Чіткий MVC	Вільна структура	Чіткий MTV	Вільна структура
Розробка API	Відміна	Відміна	Добра	Відмінна

1.3.3. Вибір клієнтської бібліотеки React

Вибір клієнтської бібліотеки для розроби веб-додатку розглядалось серед трьох домінуючих рішень: React, Vue.js та Angular. Кожен має власну філософію та сильні сторони. Vue.js вирізняється простою та швидким стартом, однак його екосистема у контексті складної маршрутизації та управління глобальним станом, поступається React за зрілістю та кількістю готових рішень. Angular є повноцінним фреймворком із жорсткою структурою, вбудованим TypeScript та двостороннім зв’язуванням даних. Він добре підходить для великих корпоративних застосунків, але для середнього за масштабом медичного застосунку його вхідний поріг та надлишкова складність є невиправданим. React є бібліотекою для побудови інтерфейсів на основі компонентного підходу. Попри те що React лише UI-бібліотека, а не повноцінний фреймворк, його екосистема (React Router для маршрутизації, Redux або Context API для стану, бібліотеки компонентів) покриває всі потреби для розробки веб-додатку медичної системи. React є найпопулярнішим рішенням у промисловій розробці, має найбільше спільноти, документацій та

прикладів проектів. Компонентний підхід дозволяє легко виокремити повторювання елементів інтерфейсу: картку лікаря, запис на прийом, блоку аналізів та інше в окремих незалежних компонентах[10; 25].

1.3.4. Вибір реляційної бази даних MySQL

Зберігання даних системи є важливим рішенням в майбутньому проекті і правильно підібраний інструмент допоможе безпечно та зручно зберігати дані, такі як: інформація про пацієнтів та лікарів, про записи на прийоми, про медичні картки, аналізи та послуги. Для реалізації цих вимог розлягались реляційні бази даних(MySQL, PostgreSQL) та нереляційні(MongoDB).

MongoDB як документ орієнтована БД є зручною для збереження слабкого структурованих або частково змінюваних даних. Однак медична система має чіткі визначені та взаємопов’язану схему даних: пацієнт пов'язаний із записами на прийом, прийоми з лікарями та результатами. Такі зв’язки природньо виражаються реляційною моделлю з foreign key та JSON-запитами. Використання MongoDB в подібній структурі призвело до надлишкового дублювання даних та ускладнення запитів. PostgreSQL це альтернатива MySQL з широким набором типів даних та кращою підтримкою складних запитів. Утім, для масштабу в розробці майбутнього веб-додатку ці переваги не є критичними, а обраний фреймворк Laravel має найкращу набивну підтримку саме з MySQL через Eloquent ORM. MySQL широко розповсюджений рушій реляційної баз даних із відмінною інтеграцією з Laravel, простим адмініструванням та достатнього продуктивністю для медичного порталу з кількома сотнями або тисячами пацієнтів. Міграції в Laravel дозволяють версіонувати схему бази даних та легко відтворювати її на будь-якому середовищі. Тому MySQL обрано як основу базу даних системи[22; 23].

1.3.5. Вибір сервісу Firebase для автентифікація та відео-консультації

Firebase це хмарна платформа від Google, що пропонує широкий набір готових бекенд-сервісів. У майбутньому веб-додатку медичної системи Firebase буде використовуватись для двох окремих завдань: автентифікації користувачів в

систему та реалізація онлайн-консультації на базі WebRTC. При обрання підходу до автентифікації розглядались три варіанта: власна реалізація на основі Laravel Sanctum, використання Laravel Passort(OAuth 2.0) та Firebase Authentication. Перший варіант є простим і ефективним засобом для API-запитів, але вимагає самостійної реалізації відновлення паролю, верифікації електронної пошти, захисту від брутфорсу та інших механізмів безпеки. Laravel Passort реалізує повноцінний OAuth 2.0, але є надлишково складним для системи, де не передбачено сторонніх OAuth-клієнтів[13].

Firebase надає готову, перевірену та безпечну систему керування обліковими записами, яка бере на себе реєстрацію, вхід до облікового запису, верифікацію пошти, відновлення паролю та захист від атак. Використовувана клієнтська частина React взаємодіє напряму з Firebase Auth SDK, отримуючи id token користувача, який потім передається у заголовок кожного запиту до серверної частини Laravel API. Laravel в свою чергу верифікує цей токен за допомогою публічних ключів Google, не зберігаючи пароль та не управляючи сесіями самостійно. Така схема значно спрощує серверну частину та підвищує рівень безпеки системи. Онлайн-консультація між пацієнтом та лікарем є одним із важливим функціональним модулів системи. Розглядались декілька підходів для цієї реалізації модуля: використання готових сервісів(Agora, Twilio Video), Sokect.io з власним сигнальним сервером там WebRTC у поєднані з Firebase Realtime Database. Agora та Twilio мають зручне SDK для відеозв'язку, проте вони є платними сервісами з тарифікацією за хвилини або підключення, що у рамках роботи є дорогим рішенням. WebRTC це відкритий стандарт для передачі аудіо, відео та даних напряму між браузерами без проміжного сервера. Для встановлення WebRTC-зв’язку необхідний сигнальний механізм, який буде обмінювати мережевими описами між двома сторонами. Firebase Realtime Database є ідеальним рішенням у поставленому завданню: забезпечення миттєву синхронізацію даних між клієнтами в реальному часі. Один клієнт записує свій SDP-опис та ICE-кандидати у Firebase, інший одразу ж їх зчитує та відповідає своїми що встановлює

з’єднання. Переваги такого підходу дозволить після встановлення з’єднання медіапотік передавати напряму між браузерами без навантаження на сервер[14].

1.4. Постановка задачі проектування

Визначальним етапом в проектуванні будь-якої інформаційної системи є правильно постанова задач, яка дозволяє чітко визначити її мету, перелік суб’єктів взаємодії та функціональні можливості, які система повинна надавати кожному з них. Правильно сформульована задача є основою для подальшого проектування архітектури, БД та інтерфейсу користувача. Загальна мета в майбутньому веб-додатку MedPlus формується за допомогою визначення її суб’єктів, опису функціональних вимог до кожного актора у вигляді варіантів використання та у вигляді діаграми відповідно до нотації до UML.

Метою розробки є створення комплексної веб-системи MedPlus для автоматизації процесів прийому пацієнтів у приватній медичній клініці. Система має забезпечувати повний цикл взаємодії між пацієнтом та медичним закладом від отримання загальної інформації про клініку та запису на прийом та ведення електронної медичної картки, перегляду медичної картки пацієнтом та проведення онлайн-консультації. Система MedPlus є двокомпонентною: вона включає публічний інформаційний сайт клініки, доступний без реєстрації та захищений функціональний застосунок із розмежованим доступом для різних категорій користувачів. Веб-додаток має вирішити проблеми характерні для традиційного паперового обліку та телефонного запису:

- неможливість запису на прийом у позаробочий час клініки;
- відстутність у пацієнтів доступу до власних медичних записів в електронному вигляді;
- ручне ведення амбулаторних карток, що ускладнює пошук та аналіз інформації;
- відстутність зручного інструменту для дистанційних консультацій між пацієнтом та лікарем;

- відстутнітсь диференційного цифрового доступу до медичних записів для лікарів різних спеціалізацій;

1.5. Висновок до розділу 1

У першому розділі було проведено комплексний аналіз предметної області, досліджено існуючі рішення на ринку медичних інформаційних систем, обґрунтовано вибір технологічного стеку та сформульовано постановку задачі проектування веб-системи MedPlus.

В ході аналізу предметної області було встановлено, що приватні медичні заклади функціонують як складні бізнес-системи, в яких одночасно взаємодіють три ключові групи учасників: пацієнти, лікарі та адміністратори. Кожна з цих груп має власні потреби та вимоги до інформаційної системи. Було визначено перелік ключових процесів, що підлягають автоматизації реєстрації та авторизації пацієнтів, онлайн-запис на прийом, ведення електронної амбулаторної картки, зберігання та перегляд результатів аналізів, а також проведення дистанційних консультацій між сімейним лікарем і пацієнтом. Огляд та порівняльний аналіз п'яти діючих веб-систем приватних клінік «Medibor», «Асклепій», «Oxford Medical», «Добробут» та «Нікамед» дозволив виявити як переваги наявних рішень, так і системні прогалини, характерні для більшості з них. Серед спільних переваг досліджених систем варто виділити структурований каталог послуг, представлення профілів лікарів та наявність цінової інформації. Водночас практично в усіх закладах особистий кабінет пацієнта реалізований через сторонні сервіси, що порушує єдність інтерфейсу та знижує рівень довіри користувача. Більшість систем не забезпечують миттєвого підтвердження онлайн-запису, жодна з досліджених систем не пропонує інтегрованого доступу пацієнта до власної медичної картки та результатів аналізів у межах єдиної платформи, а можливість проведення онлайн-прийом у власному інтерфейсі відсутня в усіх аналогах без виключення. Ці прогалини стали основою для формулювання функціональних вимог до розроблюваної веб-додатку MedPlus.

Обґрунтування вибору технологічного стеку показало, що для реалізації поставлених завдань оптимальним є поєднання архітектурного підходу SPA з REST API, серверного фреймворку Laravel, клієнтської бібліотеки React, реляційної бази даних MySQL та хмарної платформи Firebase. Архітектура SPA забезпечує плавну та комфортну роботу користувача без перезавантаження сторінок, що є особливо важливим для медичної системи, де лікар або пацієнт повинен мати змогу швидко переходити між розділами кабінету без затримок. Laravel було обрано завдяки чіткій структурі MVC, потужній ORM Eloquent, зручній системі міграцій та природній інтеграції з Firebase через пакет kreait/laravel-firebase. React забезпечує компонентну архітектуру клієнтської частини, що спрощує повторне використання елементів інтерфейсу та полегшує подальший супровід коду. Таким чином, перший розділ формує повноцінне теоретичне та аналітичне підґрунтя для подальшого проектування і розробки веб-системи для приватної клініки MedPlus. Виявлені прогалини у наявних рішеннях підтверджують актуальність та доцільність розробки власної інтегрованої платформи, а обраний технологічний стек і сформульовані вимоги до акторів системи визначають чіткий вектор для архітектурних і проектних рішень наступних розділів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ ПРИЙОМУ МЕДИЧНОЇ УСТАНОВ

2.1. Загальна архітектура системи та схема взаємодії клієнт–сервер

Проєктування веб-системи потребує чіткого розуміння взаємодії між її основними компонентами. В основу системи прийому пацієнтів медичної установи покладено класичну трирівнева клієнт-серверну архітектуру з використанням REST API як механізму обміну даними між рівнями. Такий підхід є фактичним стандартом для сучасних медичних інформаційних систем, оскільки дозволяє незалежно розробляти, підтримувати та масштабувати клієнтську і серверну частини застосунку.

Архітектура системи поділяється на три логічні відокремлені рівні: рівень представлення (клієнтська частина на React), рівень бізнес-логіки (серверний API на базі Laravel) та рівень зберігання даних (база даних MySQL). Взаємодія між рівнями здійснюється виключно через HTTP-запити з передачею даних у форматі JSON. Це забезпечує передбачувану поведінку системи та спрощує її подальший супровід і розвиток.

Для реалізації автентифікації було обрано підхід, відмінний від класичної схеми, за якої облікові дані користувачів зберігаються у власній базі даних, а сервер самостійно керує сесіями та токенами доступу. Натомість використовується сервіс Firebase Authentication, який забезпечує безпечне керування процесами автентифікації та авторизації користувачів.

Таке рішення має чітке технічне обґрунтування. Зберігання паролів і керування сесіями належать до найбільш критичних з погляду безпеки компонентів будь-якої інформаційної системи. Передача цих функцій спеціалізованому сервісу з вбудованими механізмами захисту дозволяє суттєво знизити ризик витоку персональних даних та помилок під час реалізації власних механізмів безпеки [12; 18; 20].

Процес автентифікації в системі відбувається таким чином. Користувач вводить свої облікові дані у клієнтський застосунок, розроблений на React. За

допомогою Firebase SDK ці дані передаються безпосередньо до сервісу Firebase для перевірки. У разі успішної автентифікації користувач отримує підписаний JWT-токен, який автоматично додається до кожного наступного HTTP-запиту в заголовок «Authorization: Bearer».

Після отримання запиту серверна частина перевіряє дійсність токена через спеціалізований middleware, який виконує його валідацію за допомогою Firebase. Після успішної перевірки запит передається до наступного middleware, що визначає роль користувача та перевіряє його права доступу. Лише після проходження всіх етапів перевірки запит спрямовується до відповідного контролера для виконання необхідної бізнес-логіки.

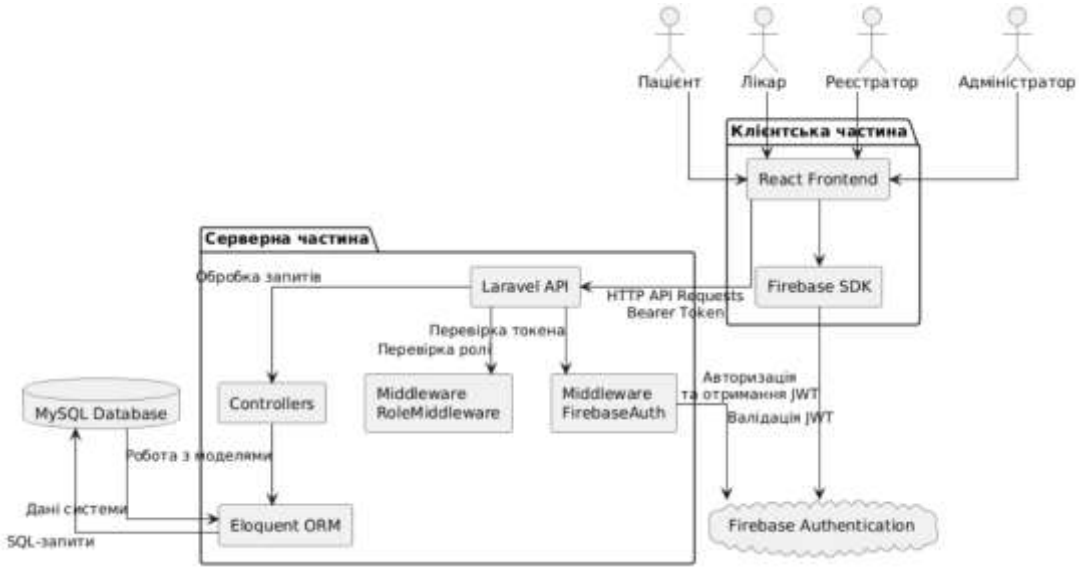


Рисунок 2.1. Компонентна діаграма архітектури системи

Рольова модель є одним із ключових архітектурних рішень системи. У ній передбачено чотири типи користувачів: пацієнт, лікар, працівник реєстратури та адміністратор. Для кожної ролі визначено окремий набір прав доступу до ресурсів REST API. Реалізація рольового контролю на рівні серверних middleware дозволяє централізовано керувати доступом до функціоналу системи без дублювання перевірок у кожному контролері.

Серверна частина побудована відповідно до архітектурного шаблону MVC, який органічно інтегрований у фреймворк Laravel. Контролери обробляють вхідні

HTTP-запити та формують відповіді клієнту. Моделі забезпечують взаємодію з базою даних за допомогою вбудованого інструмента Eloquent ORM, а система маршрутизації визначає, який контролер відповідає за обробку конкретного endpoint. Така організація програмного коду забезпечує чіткий розподіл відповідальності між компонентами системи та спрощує її подальшу підтримку. Завдяки цьому внесення змін до окремих модулів не впливає на роботу інших частин застосунку та знижує ризик виникнення помилок.

База даних MySQL використовується для зберігання основних сутностей системи, серед яких профілі пацієнтів і лікарів, записи на прийоми, електронні медичні картки, результати лабораторних аналізів та перелік медичних послуг. Взаємодія з базою даних здійснюється виключно через Eloquent ORM, що дозволяє працювати з даними на рівні об’єктів замість використання сирих SQL-запитів. Крім того, такий підхід підвищує безпеку застосунку та забезпечує захист від SQL-ін’єкцій [16].

2.2. Проектування структури реляційної бази даних

Наступним важливим етапом проєктування веб-додатка є розробка структури бази даних, оскільки від правильності її побудови залежить як коректність роботи бізнес-логіки, так і продуктивність системи загалом. Для зберігання даних було обрано реляційну систему керування базами даних MySQL. Спроєктована структура відповідає особливостям предметної області та охоплює всі ключові сутності системи:

- користувачів системи (пацієнтів, лікарів, працівників реєстратури та адміністраторів);
- записи пацієнтів на прийом до лікарів;
- електронні медичні картки пацієнтів;
- результати лабораторних досліджень;
- онлайн-консультації;
- інформацію про медичні послуги та графіки роботи лікарів

База даних складається з тринадцяти таблиць, кожна з яких відповідає окремій сутності або зв’язку предметної області(рис. 2.2). Центральною таблицею системи автентифікації є users, яка зберігає мінімально необхідні дані для ідентифікації користувача: унікальний ідентифікатор Firebase (firebase_uid) та роль (role). Розширена інформація про користувачів зберігається в окремих таблицях залежно від їхньої ролі.

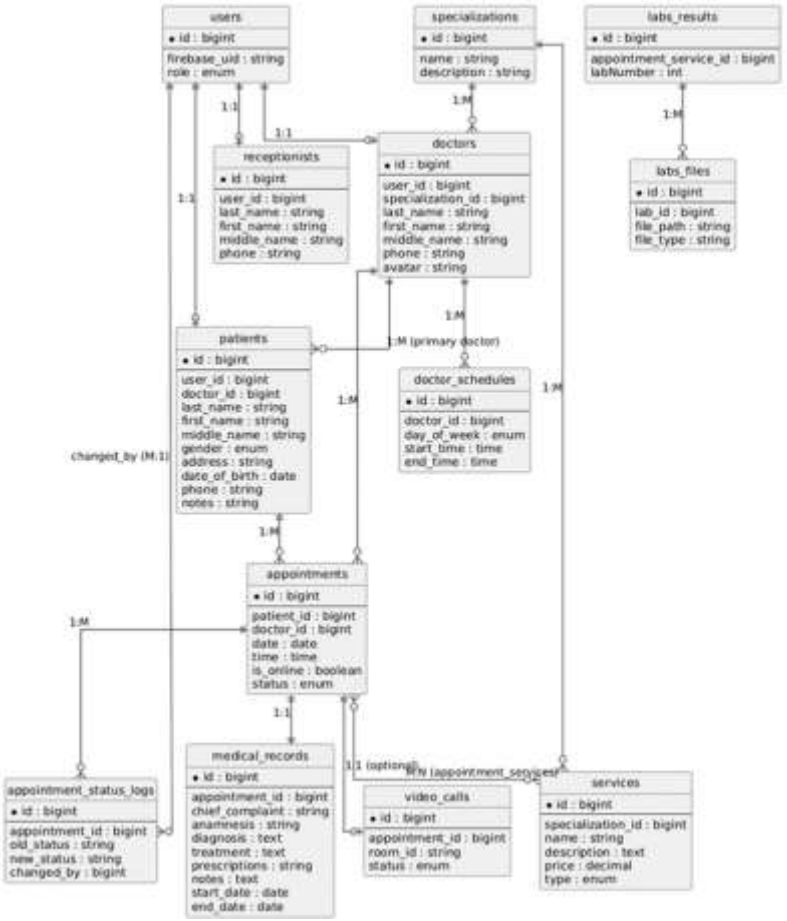


Рисунок 2.2. Схема бази даних веб-системи MedPlus

Таблиця doctors містить інформацію про лікарів: прізвище, ім’я та по батькові, номер телефону, посилання на аватар, а також зовнішні ключі на таблиці users (user_id) та specializations (specialization_id). Зв’язок із таблицею users є обов’язковим під час створення профілю лікаря адміністратором системи. Для зберігання даних працівників реєстратури використовується аналогічна таблиця receptionists.

Персональні дані пацієнтів зберігаються в таблиці patients. Її структура подібна до таблиці лікарів, однак додатково містить поля для зберігання статі, адреси проживання, дати народження та службових нотаток. Поле doctor_id визначає сімейного лікаря, закріпленого за пацієнтом, що надає лікарю доступ до його медичної документації.

Таблиці specializations та doctor_schedules виконують роль довідників. У таблиці specializations зберігається перелік медичних спеціальностей та їх опис. Це спрощує пошук лікарів за напрямком діяльності та формування переліку відповідних послуг. Таблиця doctor_schedules містить графіки роботи лікарів і використовується під час запису пацієнтів на прийом.

Однією з центральних таблиць системи є appointments, яка містить інформацію про записи пацієнтів на прийом до лікаря. У ній зберігаються дата та час прийому, ознака онлайн-консультації (is_online) і поточний статус запису (expected, completed, cancelled, no_show).

Для збереження історії змін статусів використовується таблиця appointment_status_logs. Вона містить попередній і новий статус запису, а також інформацію про лікаря, який виконав зміну. Між таблицями appointments та appointment_status_logs реалізовано зв’язок типу «один до багатьох».

Таблиця services містить перелік медичних послуг із детальним описом та вартістю. Кожна послуга пов’язана з певною медичною спеціалізацією. Для реалізації зв’язку «багато до багатьох» між прийомами та послугами використовується проміжна таблиця appointment_services. Крім зовнішніх ключів, вона зберігає вартість послуги на момент створення запису, що дозволяє зберігати історичну достовірність даних у разі зміни цін.

Зберігання електронної медичної картки пацієнта реалізовано в таблиці medical_records, яка містить такі поля:

- chief_complaint — скарги пацієнта;
- anamnesis — анамнез захворювання;
- initial_review — результати первинного огляду;
- diagnosis — встановлений діагноз;

- treatment — призначене лікування;
- prescriptions — рекомендації лікаря;
- notes — додаткові коментарі.

Між таблицями appointments та medical_records реалізовано зв’язок типу «один до одного».

Для зберігання результатів лабораторних досліджень використовуються таблиці lab_results та lab_files. Вони пов’язані з конкретною послугою в межах прийому через поле appointment_service_id. Таблиця lab_files містить шлях до файлу (file_path) та його тип (file_type), наприклад PNG, DOCX або PDF. Між таблицями реалізовано зв’язок типу «один до багатьох».

Останньою таблицею структури є video_consultations, яка використовується для зберігання інформації про відеоконсультації. Вона містить унікальний ідентифікатор відеокімнати та поточний стан з’єднання (waiting, active, ended) [22; 23].

Одним із критеріїв якості реляційної бази даних є відповідність нормальним формам. Нормалізація дозволяє зменшити надлишковість даних, уникнути аномалій оновлення та забезпечити цілісність інформації. Спроєктована база даних відповідає вимогам третьої нормальної форми (3НФ).

Перша нормальна форма (1НФ) вимагає, щоб усі атрибути містили лише атомарні значення, а кожен запис мав унікальний ідентифікатор. У всіх таблицях використовується первинний ключ id типу BIGINT з автоінкрементом. Кожне поле містить лише одне значення, а повторювані групи відсутні. Наприклад, ПІБ користувача зберігається в окремих полях last_name, first_name та middle_name. Отже, структура бази даних відповідає вимогам 1НФ.

Друга нормальна форма (2НФ) вимагає виконання умов 1НФ та відсутності часткових залежностей неключових атрибутів від первинного ключа. Оскільки в усіх таблицях використовується простий первинний ключ id, кожен неключовий атрибут повністю залежить від нього. Наприклад, у таблиці appointments поля date, time, status та is_online залежать виключно від ідентифікатора запису.

Третя нормальна форма (3НФ) передбачає відсутність транзитивних залежностей між неключовими атрибутами. У спроєктованій базі даних такі залежності усунуті шляхом виділення окремих сутностей у самостійні таблиці. Наприклад, інформація про спеціалізації лікарів не дублюється в таблиці doctors, а зберігається в окремій таблиці specializations, на яку посилається зовнішній ключ specialization_id. Аналогічно вартість послуг зберігається в таблиці services, а до таблиці appointment_services копіюється лише під час створення запису для фіксації історичного значення. Це підтверджує відповідність структури бази даних вимогам третьої нормальної форми [22].

2.3. Проектування поведінки системи

Після визначення структури бази даних та архітектури компонентів необхідно спроєктувати динамічну поведінку системи. Це дозволяє зрозуміти, як система реагує на дії користувачів і як виконуються ключові процеси всередині неї. Моделювання поведінки дає змогу виявити потенційні проблеми ще на етапі проєктування, чітко визначити відповідальність кожного компонента та зафіксувати очікувану логіку роботи системи у вигляді специфікації для подальшої реалізації.

Проєктування поведінки системи є особливо важливим для медичних інформаційних сервісів, оскільки значна частина функцій пов’язана з багатокроковою взаємодією між різними ролями користувачів, серверною логікою та базою даних. У таких системах навіть незначні помилки в послідовності виконання операцій можуть призводити до втрати даних, конфліктів під час запису на прийом або порушення цілісності медичної інформації. Саме тому використання UML-діаграм послідовностей дозволяє ще на етапі проєктування детально описати сценарії роботи системи та визначити механізми обміну даними між її компонентами. Моделювання поведінки системи спрощує подальшу реалізацію серверної та клієнтської частин, оскільки кожен етап взаємодії фіксується як окрема послідовність дій, що також полегшує тестування функціоналу [21; 24].

Автентифікація є першим і обов’язковим етапом будь-якої взаємодії з системою, оскільки в архітектурі відповідальність за перевірку облікових даних делегована зовнішньому сервісу Firebase. Процес входу включає взаємодію п’яти компонентів: користувача, клієнтського застосунку React, Firebase Authentication, серверного Laravel API та middleware для перевірки токенів. Діаграма послідовності для цього процесу наведено на рис. 2.3

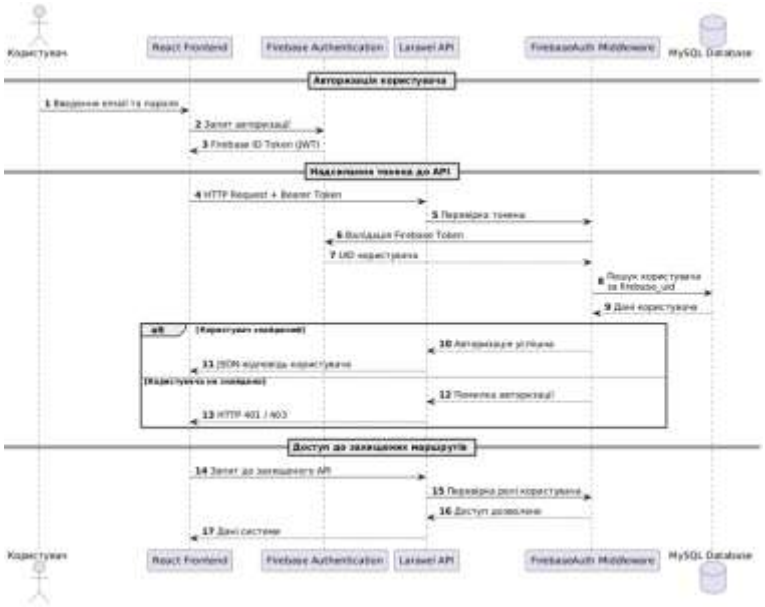


Рисунок 2.3. Діаграма послідовності авторизації

Користувач вводить свої дані у форму входу на клієнтській частині системи. У свою чергу, React-застосунок через Firebase SDK надсилає запит автентифікації безпосередньо до хмарного сервісу Firebase. У разі успішної перевірки сервіс повертає підписаний JWT-токен, який містить унікальний ідентифікатор користувача (uid) та термін дії токена.

Отриманий токен клієнт зберігає локально та додає до заголовка кожного наступного HTTP-запиту до Laravel API у форматі «Authorization: Bearer <token>». Серверна частина не довіряє токену без перевірки, тому спеціальний FirebaseAuth middleware перехоплює запит і надсилає токен до Firebase для валідації цифрового підпису. Після підтвердження автентичності токена сервіс повертає UID користувача.

Після отримання UID middleware виконує пошук у локальній базі даних MySQL у таблиці users за полем firebase_uid. Якщо запис знайдено, користувач вважається авторизованим і запит передається до відповідного контролера. У протилежному випадку сервер повертає HTTP-код помилки 401 або 403, що означає: обліковий запис існує у Firebase, але ще не зареєстрований у системі [13; 20; 21].

Перевірка ролі виконується після успішної ідентифікації користувача та підтвердження його UID у таблиці users. Роль зчитується з бази даних і перевіряється через RoleMiddleware перед передачею запиту до контролера. Це означає, що авторизований користувач не має доступу до ендпоінтів, які не відповідають його ролі. Наприклад, користувач із роллю «лікар» не може виконувати запити до API, призначених для пацієнтів.

Такий дворівневий підхід до захисту дозволяє спочатку перевіряти справжність токена, а потім права доступу до конкретних ресурсів. Це поєднує надійність зовнішнього хмарного сервісу з гнучкістю власної рольової моделі.

Наступним ключовим етапом після автентифікації користувача є запис на прийом до лікаря. Цей процес є одним із центральних у медичній інформаційній системі, оскільки забезпечує взаємодію між пацієнтом, лікарем і серверною частиною із використанням даних, що зберігаються в базі даних. Змодельована діаграма послідовності взаємодії між компонентами продемонстровано в додатку Б.

У процесі запису на прийом беруть участь чотири основні компоненти: пацієнт, клієнтський інтерфейс системи, серверна частина та база даних MySQL. Основною метою цього процесу є забезпечення коректного вибору спеціалізації, лікаря, медичних послуг та доступного часу прийому з урахуванням робочого графіка лікаря.

Процес починається з відкриття пацієнтом форми запису на прийом. Клієнтський застосунок надсилає HTTP-запит до серверного API для отримання списку доступних медичних спеціалізацій. Сервер, звертаючись до бази даних, повертає відповідні записи для відображення на клієнтській частині.

Після вибору спеціалізації виконується наступний GET-запит із параметром specialization_id для отримання списку лікарів, що працюють у відповідному напрямку медицини. Серверна частина виконує пошук у таблиці лікарів і повертає лише релевантних спеціалістів. Такий підхід дозволяє уникнути відображення зайвих даних і спрощує взаємодію користувача із системою. Після вибору лікаря пацієнту відображається перелік доступних медичних послуг, які надає обраний спеціаліст. Для цього сервер виконує окремий запит до бази даних і повертає відповідні дані на клієнтську частину. Користувач може обрати одну або кілька послуг залежно від потреб.

Наступним етапом є вибір дати та часу прийому. Клієнтська частина надсилає запит до API для отримання графіка роботи лікаря та доступних часових слотів. Система формує проміжки часу з інтервалом 30 хвилин і повертає список вільних слотів [18].

Фінальним кроком є надсилання POST-запиту для створення нового запису в таблиці appointments. Форма передає ідентифікатор пацієнта, лікаря, обрані послуги, а також дату і час прийому. Перед створенням запису сервер перевіряє доступність вибраного часу, щоб уникнути конфліктів під час одночасного бронювання.

У разі успішної перевірки створюється новий запис у таблиці appointments, а користувач отримує підтвердження успішного запису. У випадку помилки система повертає відповідне повідомлення, яке відображається на клієнтській частині.

Побудована діаграма послідовностей запису на прийом відображає повний цикл взаємодії між користувачем, клієнтським інтерфейсом, серверною логікою та базою даних. Такий підхід забезпечує цілісність даних, запобігає конфліктам і гарантує коректну синхронізацію між компонентами системи.

Після проєктування механізму запису на прийом наступним важливим етапом є безпосереднє проведення медичного прийому лікарем та фіксація результатів консультації в онлайн-режимі. Діаграма послідовностей проведення медичного прийому наведена на рис. 2.4.

У цьому процесі беруть участь лікар, пацієнт, клієнтська та серверна частини, база даних, а також сервіс Firebase Realtime, який використовується для організації онлайн-з’єднання між учасниками прийому.

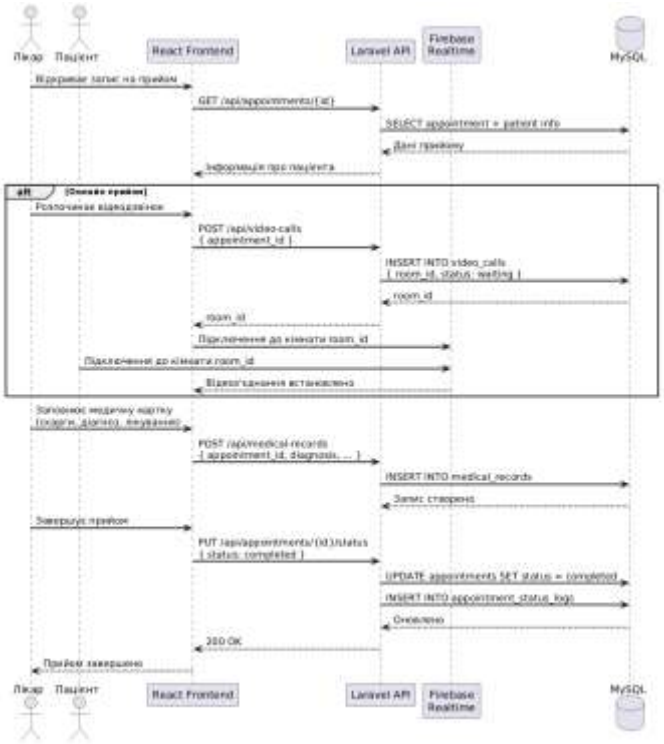


Рисунок 2.4. Діаграма послідовностей створення запису медичної картки

Процес починається з відкриття лікарем детального запису на прийом у системі. Клієнтська частина надсилає GET-запит до серверного API для отримання інформації про конкретний прийом. Серверна частина звертається до бази даних і виконує вибірку даних про запис, пацієнта та пов’язані медичні відомості. Отримана інформація повертається на клієнтську частину у форматі JSON і відображається лікарю для подальшої роботи. Під час проведення прийому в онлайн-форматі система повинна забезпечити механізм створення кімнати для відео-консультації. Лікар ініціює початок консультації, після чого клієнтська частина надсилає POST-запит на створення кімнати до серверної частини. Сервер створює новий запис у таблиці video_calls, де зберігаються унікальний ідентифікатор кімнати та поточний статус з’єднання, і повертає клієнту room_id.

Підключення пацієнта до відповідної кімнати здійснюється через Firebase Realtime. Цей сервіс використовується не лише для автентифікації, а й для синхронізації даних у реальному часі та забезпечення стабільного відеоз'єднання між учасниками прийому без необхідності реалізації власного механізму сигналізації на сервері. Після успішного підключення система підтверджує встановлення з'єднання, і онлайн-консультація може бути розпочата. Під час прийому лікар заповнює електронну медичну картку пацієнта відповідно до полів, визначених на етапі проєктування ER-діаграми (рис. 2.2). Клієнтський застосунок передає ці дані до серверного Laravel API методом POST. Сервер виконує валідацію, зберігає інформацію в таблиці medical_records та після успішного запису повертає підтвердження лікарю.

Наступним етапом є завершення прийому. Лікар змінює його статус з expected на completed. Для цього клієнтська частина надсилає PUT-запит до API. Сервер оновлює статус у таблиці appointments та створює запис у таблиці appointment_status_logs, яка використовується для фіксації змін стану прийомів. Такий механізм дозволяє зберігати історію змін і забезпечує можливість подальшого аудиту дій у системі. Після успішного оновлення статусу сервер повертає відповідь із кодом «200 OK», а клієнтська частина відображає повідомлення про завершення прийому. Таким чином, діаграма демонструє повний цикл проведення медичного прийому — від відкриття його деталей до завершення та збереження результатів в електронній медичній картці пацієнта.

Використання Firebase Realtime для організації з'єднання дозволяє зменшити навантаження на основний сервер Laravel API та забезпечити стабільну роботу функцій реального часу. У свою чергу, централізоване зберігання медичних даних у MySQL гарантує цілісність інформації та можливість подальшого доступу до історії лікування пацієнта.. Додаткові діаграми архітектури системи наведено у додатку Б.

2.4. Проектування інтерфейсу користувача

Інтерфейсу користувача є завершальним етапом в проектуванні системи перед початком її реалізації. Необхідно визначити структуру навігації, склад екранів для кожної ролі користувача та розміщення ключових елементів на сторінках. Прототипи інтерфейсу для медичної приватної клініки MedPlus розроблено в редакторі Figma та відображають фінальний вигляд системи з урахуванням усіх UX-рішень.

Головна сторінка системи є точкою входу для всіх користувачів і виконує інформаційну функцію. У верхній частині розміщено навігаційне меню з розділами «Відділення», «Лікарі», «Послуги», «Контакти», а також кнопка «Увійти в кабінет». Центральна частина сторінки містить hero-секцію з коротким описом та кнопкою заклику до дії «Запис на прийом». Кольорова схема, що є характерним для медичної тематики та асоціюється з довірою та професіоналізмом(рис. 2.5).

Сторінку авторизації спроектовано у мінімалістичному стилі – центрований блок з формою на світло-сірому фоні з двома вкладками: «Авторизація», «Реєстрація». Форма містить лише два поля введення – електронна пошта та пароль і кнопку підтвердження. Такий підхід усуває зайві елементи та дозволяє користувачу зосередитись на єдиній дії. Після успішної авторизації пацієнта буде перенаправляти до свого електронного кабінету, де навігація спроектована у вигляді горизонтальних вкладок «Особистий кабінет», «Медична картка», «Запис на прийом» та «Аналізи».

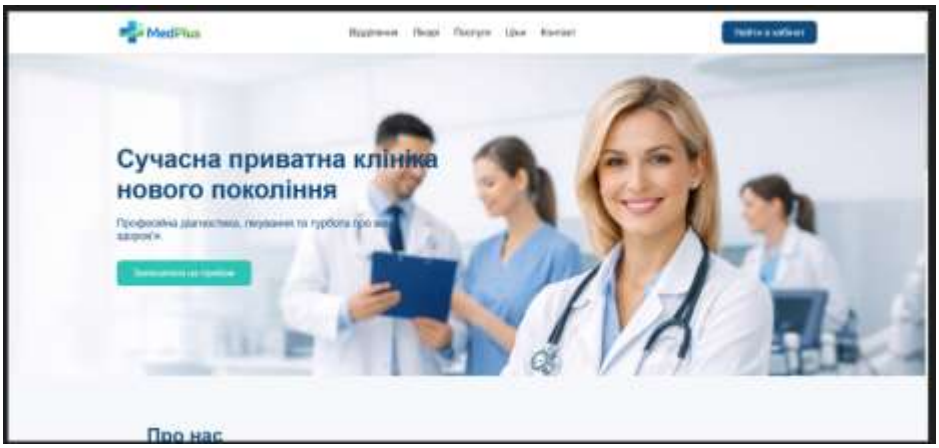


Рисунок 2.5. Макет головної сторінки сайту

Активна вкладка виділяється акцентним кольором. Кран особистої інформації відображає повне ім’я пацієнта, стать, дату народження та вік у вигляді зручної картки з аватаром-ініціалами. Нижче розміщено контактні дані з можливістю зміни електронної пошти та паролю, поле для приміток та блок інформації про прикріпленого сімейного лікаря з його контактними номером. Така структура дозволяє в один клік отримати всі необхідну особисту інформацію, що наведено на рис. 2.6.

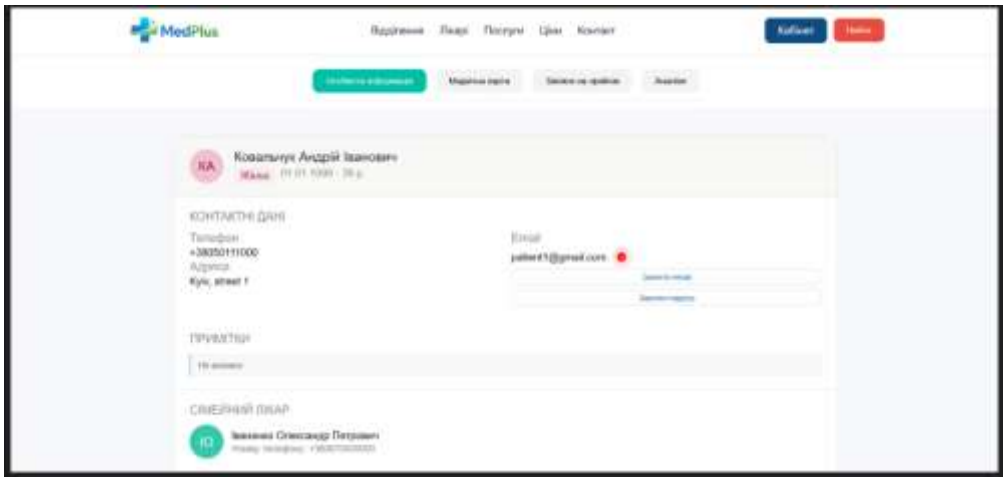


Рисунок 2.6. Макет особистого кабінету

Сторінка запису на прийом спроектована у вигляді двоколонного макету, ліва відображає картку з інформацією пацієнт: ПІБ, телефон та повідомлення про вхід у системі, що дозволяє пацієнт переконатись в актуальності контактних даних. У випадку якщо пацієнт не авторизований у системі, замість відображених даних йому потрібно вручну їх ввести. Права колонка містить покрокову форму вибору параметрів прийому. Поля форми розташовані у логічній послідовності: спочатку обирається тип спеціалізації лікаря, а після чого автоматично буде завантажуватись список спеціалістів відповідно до вибору конкретної області. Далі через випадаючи список додаються необхідні послуги, відображаючись у вигляді кольорових тегів з можністю їх видалення відповідної кнопкою.

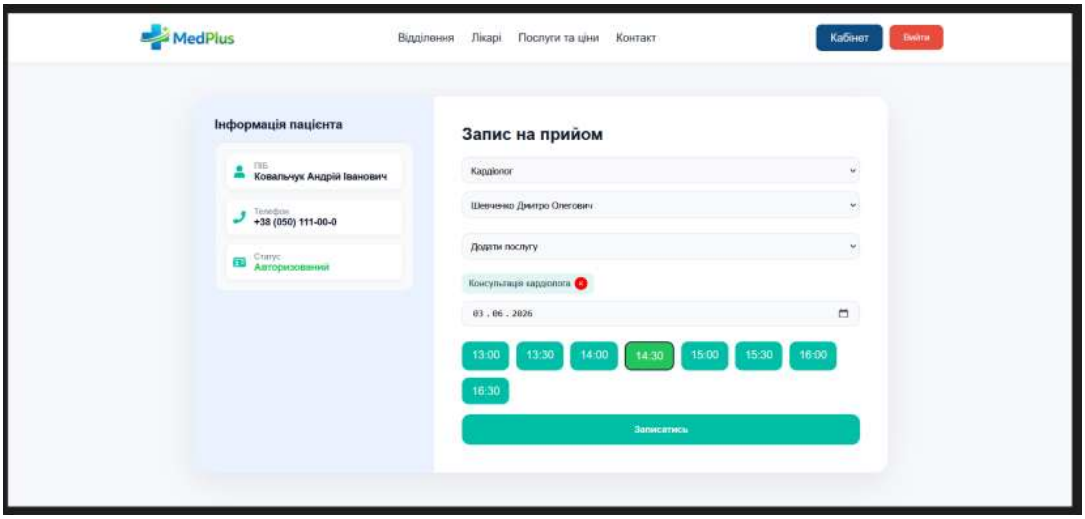


Рисунок 2.7. Макет запису на прийом

Після вибору лікаря та дати система повинна підвантажити автоматично вільні години для прийому обраного спеціаліста з інтервальним часом в 30 хвилин. Обраний слот виділяється темнішим кольором на фоні решти доступних варіантів. Завершує форму кнопка «Записатись», яка стає активною лише після заповнення всіх обов'язкових полів та відображає пацієнту повідомлення про успіх або помилку. Такий підхід проектування принципу покрокового введення даних робить форму для запису на прийом надійною комбінацією та підказує користувачу правильний порядок дії без додаткових інструкцій.

Наступна модель UX-інтерфейсу спроектована для медичного персоналу в медичній панелі системи. Інтерфейс виконаний у темній колірній схемі з темно-сірою бічною навігаційною панеллю, що чітко відрізняє його від кабінету пацієнта. Бічне мені буде мати розділи «Dashboard», «Прийоми», «Пацієнти», «Онлайн консультації», «Аналізи та дослідження», та «Налаштування», воно буде мінятись залежачи від спеціаліста та інших медичних працівників у системі. На Головній сторінці лікар буде бачити свій графік роботи у вигляді календарю, де додатково будуть відображатись зайняти години на прийоми. На календарі він зможе обрати конкретний день або тиждень, що робить зручну можливість планувати робочий час, що наведено на рис. 2.8.

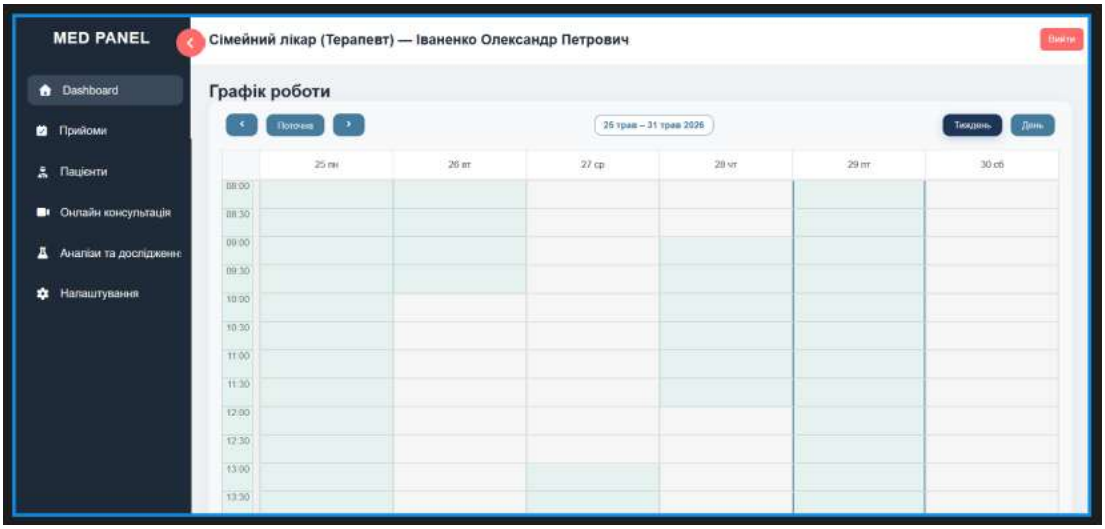


Рисунок 2.8. Макет головної сторінки медичної панелі

Сторінка прийомів відображає повний список записаних пацієнтів у табличному форматі з колонками: пацієнти, дата та час, перелік призначених послуг, статус та кнопка дій. Статус кожного прийому виділяється кольоровим маркером – червоний для «Не з'явився», зелений для «Завершено», сірий для «Скасовано», що дозволяє лікарю миттєво оцінити стан усіх записів без необхідності відкриття кожного окремо. Кнопка «Відкрити» надає доступ до детальної картки прийому, де лікар може заповнити медичну картку або розпочати онлайн-консультації. Більше UI-макетів наведено у додатку А.

2.5 Висновок до Розділу 2

У другому розділі було виконане повне проектування веб-системи прийому пацієнтів медичної установи, яке охопило чотири ключові аспекти: архітектуру системи, структуру бази даних, поведінку та інтерфейс користувача.

Спроектовано загальну архітектуру на основі трирівневої клієнт-серверної моделі з використанням REST API як протоколу взаємодії між компонентами. Визначено розподіл відповідальності між рівнями:

- клієнстький застусунок React відповідає за представлення даних;
- серверна частина Laravel API за обробку бізнес-логіки;
- реляційна база даних MySQL за заберігання інформаці;

Обґрунтовано рішення щодо делегування автентифікації сервісу Firebase Authentication, до дозволяє підвищити безпеку системи без реалізації власного механізму управління обліковими даними. Спроектовані дворівневий захист API через послідовне застосування FirebaseAuth Middleware та RpoleMiddlleware, ща забезпечить перевірку як автентичності токена, так і рольових прав доступу для кожного захищеного запиту. Спроектовано реляційна база даних, що складається з тринадцяти таблиць та охоплює всі ключові сутності предметної області та відповідає усім нормам нормалізації. Змодельована динамічну поведінку системи для трьох ключових сценаріїв за допомогою UML-діаграм послідовностей. Кожна поведінка системи описано з урахуванням всіх важливих аспектів для майбутньої розробки та уникнення можливих конфліктів у системі та дає розуміння як користувач має виконувати ті чи інші дії для досягнення певних задач. Останнім етапом проектування виконувалось прототипи інтерфейсу користувача в редакторі Figma для всіх ролей у системі: публічна частина, електронний кабінет пацієнта з записом на прийом та медичного персоналу.

Результати проектування, отримані в цьому розділі, є вичерпною специфікацією для подальшої розробки системи та охоплюють усі рівні – від архітектури даних до поведінки окремих сценаріїв та зовнішнього вигляду інтерфейсу.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Реалізація бази даних та взаємодія з нею

Одним із ключових етапів розробки інформаційної системи для медичного центру є створення надійної та структурованої бази даних, яка забезпечує зберігання, обробку та швидкий доступ до інформації. База даних є основою для функціонування всієї системи, в якій зберігаються відомості про користувачів, медичні записи, графік роботи лікарів, записи на прийом, результати аналізів та інші дані, необхідні для забезпечення роботи медичного закладу. За допомогою реляційної бази даних MySQL було розроблено з використанням інтегрованої технології Eloquent ORM в фреймворку Laravel. Використання цієї технології дозволяє працювати з таблицями бази даних через об’єктно-орієнтований підхід, що значно спрощує виконання операцій створення, читання, оновлення та видалення записів. Створення бази даних використовувалися механізми міграцій фреймворку, які забезпечують контроль версій структури бази даних та дозволяють автоматизувати процес її розгортання. Будова розробленої моделі бази даних відбувалось згідно спроектованої схему у другому підрозділі з виконанням усіх вимог у вигляді нормалізації для уникнення дублювання інформації, забезпечення цілісності даних та підтримку логічних зав’язків[16; 22].

Дані для автентифікації зберігаються в таблиці users в якій зберігаються uid який зберігається і на сервісі Firebase та роль користувача. В таблиці присутні чотири поля: ідентифікатор запису, firebase_uid та ролі користувача, яка розбивається на чотири відповідні ролі – пацієнт, лікар, працівник реєстратури та адміністратор, що наведено на рис 3.1.

```
Schema::create( table: 'users', function (Blueprint $table) {
    $table->id();
    $table->string( column: 'firebase_uid')->unique();
    $table->enum( column: 'role',['patient', 'doctor', 'receptionist', 'admin']);

});
```

Рисунок 3.1. Міграція таблиці «users»

Зберігання особистих інформацій пацієнт розроблено в таблиці patients та мають зв'язок до таблицею users «один до одного» та з наступними полями:

- user_id – завнішній ключ пов'язаний з таблицею users для привзяання облікового запису до профілю пацієнта, може бути пустим, якщо пацієнт не реєстрував обліковий запис;
- doctor_id – зовнішній ключ, який має зв'язок з таблицею doctors для зберігання тільки смімейного лікаря дял пацієнта, поле може бути пустим у випадку якщо пацієнт не складав відповідну декларацію;
- ПІБ пацієнта(last_name, first_name, middle_name) вказується метод string для створення типу даних VARCHAR, поля мають бути обоввязковими, максимальна довжина символів 50;
- gender – поле яке має тип ENUM і може приймати лише два доступних значень: «male» та «female»;
- поле adress має тип VARCHAR для зберігання адреси проживання пацієнта та має максимальну довжину для типу даних;
- date_of_birth – зберігання дати народження пацієнта, має тип DATE;
- контактний номер(phone) має тип даних VARCAH(12) та має унікальність та обмеження;
- поле notes мають тип поля TEXT та є не обовзяковим для заповнення.

Відповідні таблиці для лікарів та працівників реєстратури було розроблено за аналогічною схемою, як і для пацієнтів(рис. 3.2).

```
Schema::create( table: 'patients', function (Blueprint $table) {
    $table->id();
    $table->foreignId( column: 'user_id')->nullable()->constrained( table: 'users')->nullOnDelete();;
    $table->foreignId( column: 'doctor_id')->nullable()->constrained( table: 'doctors')->onDelete('set null');
    $table->string( column: 'last_name', length: 50);
    $table->string( column: 'first_name' , length: 50);
    $table->string( column: 'middle_name', length: 50);
    $table->enum( column: 'gender', ['male', 'female']);
    $table->string( column: 'address');
    $table->date( column: 'date_of_birth');
    $table->string( column: 'phone', length: 12);
    $table->text( column: 'notes')->nullable();
    $table->timestamps();
});
```

Рисунок 3.2. Міграція таблиці «patients»

Зберігання інформації про запис пацієнтів на прийом до лікаря було розроблено таблицю appointments відповідно до проєктування схеми бази даних(рис. 3.3). Її роль в системі зберігати фіксацію дати та часу на який планується прийом, поточний статус та тип прийому. Первинним ключем таблиці є поле «id», яке створюється методом «$table->id()» і має тип «BIGINT UNSIGNED». Воно є автоінкрементом та забезпечує унікальну ідентифікацію кожного запису про прийом. Поле «patient_id» є зовнішнім ключем, що посилається на таблицю patients, маючи тип «BIGINT UNSIGNED» і створюється методом «$table->foreignId('patient_id')». Зв'язок супроводжується правилом «onDelete('cascade')» для автоматичного видалення всіх записів про прийом пацієнта у разі видалення з таблиці пацієнтів. Аналогічне до цього поля є «doctor_id», яке посилається на таблицю лікарів та вказує до якого лікаря прийом. Поля date та time використовуються для зберігання календарної дати та конкретного часу, коли має початись прийом. Мають тип даних DATE та TIME. Поле «is_online» має тип «boolean» для визначення типу прийому – в онлайн або оффлайн режимі за методом «true/false»[22].

```
Schema::create( table: 'appointments', function (Blueprint $table) {
    $table->id();
    $table->foreignId( column: 'patient_id')->constrained( table: 'patients')->onDelete('cascade');
    $table->foreignId( column: 'doctor_id')->constrained( table: 'doctors')->onDelete('cascade');
    $table->date( column: 'date');
    $table->time( column: 'time');
    $table->boolean( column: 'is_online')->default( value: false);
    $table->enum( column: 'status', ['expected', 'completed', 'cancelled', 'no_show']);
    $table->timestamps();
});
```

Рисунок 3.3. Міграція таблиці прийомів

Поле «status» має тип «ENUM» і визначає поточний стан запису на прийом, воно може набувати одного з доступних значень:

- expected - очікування прийому;
- completed - успішне завершення;
- cancelled - скасування прийому;

- no_show – пацієнт не з’явився на прийом.

Таке обмеження забезпечує контрольовану множину статусів і запобігає некоректним значенням у базі даних.

Таблиця прийомів має зв'язок не лише з таблицями пацієнтів та лікарів, але ще з проміжної таблицею appointments_service(рис. 3.4). В най зберігаються зовнішні ключі «appointment_id» та «service_id» та мають каскадне правило, в якому при видалені запису на прийом або послуг записи в цій таблиці буде видалений. Останнє поле відповідає за збереження цін на послуг на момент створення запису, має тип decimal з обмеженням символів до 10 та кількість цифр після коми значенням 2.

```
Schema::create( table: 'appointment_services', function (Blueprint $table) {
    $table->id();
    $table->foreignId( column: 'appointment_id')->constrained( table: 'appointments')->onDelete('cascade');
    $table->foreignId( column: 'service_id')->constrained( table: 'services')->onDelete('cascade');
    $table->decimal( column: 'price', total: 10, places: 2);

});
```

Рисунок 3.4. Міграції проміжної таблиці «appointments_service»

Наступною не менш найважливішою реалізації є таблиця медичної картки пацієнта(medical_records). Вона є ключовим елементом електронної картки та містить детальну клінічну інформацію, включаючи скарги пацієнта, анамнез, діагноз та призначення лікування(рис. 3.5).

```
Schema::create( table: 'medical_records', function (Blueprint $table) {
    $table->id();
    $table->foreignId( column: 'appointment_id')->constrained( table: 'appointments')->onDelete('cascade');
    $table->string( column: 'chief_complaint');
    $table->string( column: 'anamnesis');
    $table->string( column: 'initial_review')->nullable();
    $table->text( column: 'diagnosis')->nullable();
    $table->text( column: 'treatment')->nullable();
    $table->string( column: 'prescriptions')->nullable();
    $table->text( column: 'notes');
    $table->date( column: 'start_date')->nullable();
    $table->date( column: 'end_date')->nullable();
    $table->timestamps();
});
```

Рисунок 3.5. Міграція таблиці «medical_records»

Первинним ключем в таблиці, як і в усіх інших є поле «id», яке створюється методом «$table->id()» з одним тим самим типом як і в інших таблицях. Зовнішній ключ таблиці є «appointment_id» з типом «BIGINT UNSIGNED». Зв'язок супроводжується з таблицею «appointments» та правилом каскадного видалення, якщо буде видалений запис на прийом. Таке рішення захищає цілісність даних між записами прийому та медичною документацією.

Усі поля для скарги пацієнта, анамнез, діагноз, призначення лікування, рекомендації створюються однаковим методом та мають тип VARCHAR(255) та відповідають за певний крок дії на прийомі. Поле «notes» має тип TEXT для додаткових рекомендацій лікаря і є необов’язковим для заповнення. Для автоматичного обліку часу створення та оновлення записів використовуються службові поля «create_at» та «update_at», які додаються методом «$table->timestamps()» та мають тип «TIMESTAMP» і заповнюються автоматично Laravel.

Організація та управління онлайн-консультацій між пацієнтом та сімейним лікарем у системі буде контролюватись за допомогою таблицею «video_calls»(рис. 3.6). Первинним ключом таблиці є поле «id», яке є автоінкрементним та унікальним. Зовнішній ключ є «appointment_id», що посилається на таблицю «appointments», має автоматичне каскадне видалення запису таблиці у разі видалення прийому. Поле «room_id» має тип «VARCHAR(255)» і використовується для збереження унікального ідентифікатора відео-кімнати. Унікальність гарантує відсутність дублювання кімнат у системі та дозволяє одночасно ідентифікувати кожну відеосесію[30].

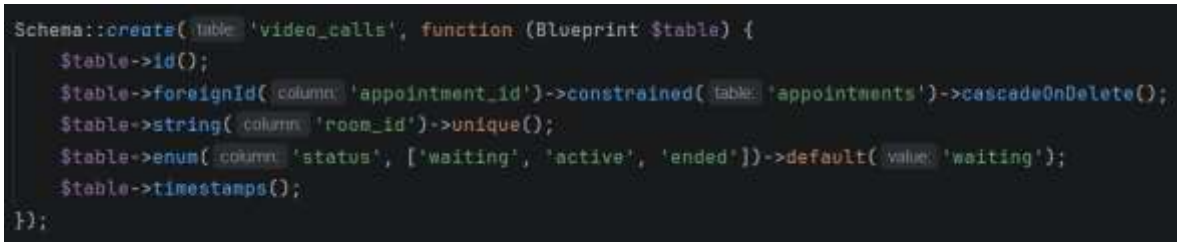


Рисунок 3.6. Міграції таблиці «video_calls»

Поле «status» має тип «ENUM» і визначає поточний стан відео-кімнати та може набувати одного з трьох значень:

- waiting - очікування;
- active - активна відеосесія;
- ended – сессія завершена.

За замовчуванням поле має значення waiting(default('waiting')) при створенні нової кімнати в режимі очікування підключення. Автоматичне збереження часу створення кімнати відбувається аналогічно до таблиці медичної картки пацієнта.

3.2. Розробка моделі даних

Після розробки бази даних на серверній частині необхідно перейти до створення моделей даних з використанням інструменту Eloquent ORM, який входить до складу Laravel. Використання моделей дозволяє інкасулювати бізнес-логіку роботи з даними, спросити виконання запитів до бази даних та забезпечити підтримку складних зав’язків між сутностями предметної області. Центральної сутність системи є модель Appointment, яка представляє запис пацієнта на прийом до лікаря. Модель реалізує зв'язок «багато до одного» з моделями Patient та Doctor, що надає можливість отримати інформацію про учасників прийому без виконання додаткових SQL-запитів. Додатково модель має зв'язок «один до одного» з моделлю MedicalRecord, які містить електрону медичну картку конкретного пацієнта. Можливість реалізувати декількох медичних послуг під час одного візиту використовується проміжна модель AppointmentService(рис. 3.7). Вона реалізує зв'язок «багато до багатьох» мі проймам та послугами. Важливою частиною системи є журнал змін статусів прийомів, реалізований через модель AppointmentStatusLog. Має зв’язок «один до багатьох» з моделлю Appointment, де один прийом, може мати декілька змін у статусі[16; 22].

Представлення спеціалістів у вигляді лікарів використовується модель Doctor та має зв'язок «один до багатьох» з моделлю DoctorSchedules, де один лікар може мати декілька графіків роботи. Додатково модель має зв'язок з таблицею users «Один до одного», де один обліковий запис лікаря може мати один профіль. Має

додаткові зв’язки з таблицями спеціалізацій та прийомами. Модель Patient розроблено аналогічно до моделі лікарів.

```php
public function user()
{
    return $this->belongsTo( related: User::class);
}

studpod
public function specialization()
{
    return $this->belongsTo( related: Specialization::class);
}
studpod
public function schedules()
{
    return $this->hasMany( related: DoctorSchedules::class);
}
studpod
public function appointment()
{
    return $this->hasMany( related: Appointment::class);
}
```

Рисунок 3.7. Частина моделі «Doctor» у Laravel

Модель для зберігання ідентифікатора відеокімнати реалізовано через модель VideoCall, яка має зв'язок з моделлю Appointment «один до одного». Це дозволяє контролювати життєвий цикл відеосесії від моменту створення до завершення прийому. Моделі LabsResult та LabsFile відповідають за збереження результатів аналіз пацієнта та мають між собою зв’язок «один до багатьох», де один аналіз може мати декілька файлів. Через механізм Eloquent пацієнт та лікар можуть отримувати доступ до результатів досліджень безпосередньо з медичної картки.

Таким чином, розроблені моделі даних забезпечує повне відображення предметної області приватної медичної клініки, підтримує складні взаємозв’язки між сутностями та дозволяє ефективно розробляти бізнес-процеси системи за допомогою можливостей ORM Eloquent.

3.3. Розробка серверної логіки на основі контролерів та Middleware

Розробка серверної частини системи виконана на основі архітектури MVC з використанням фреймворку Laravel. Кожен функціональний модуль системи реалізований у вигляді окремого контролера з чітко визначеною зоною відповідальності. Усі контролери організовані в просторах імен відповідно до ролей користувачів:

- App\Http\Controllers\Patient — для пацієнтів;
- App\Http\Controllers\Staff\Doctor — для лікарів;
- App\Http\Controllers\Staff\Receptionist — для працівників реєстратури;
- App\Http\Controllers\Admin — для адміністраторів.

Методи управління профілем пацієнта оброблюються в контролері PersonalOfficeController. Він забезпечує повний цикл операцій CRUD: створення профілю, перегляд і редагування персональної інформації, отримання історії прийомів та доступ до електронної медичної картки. Контролер виступає проміжною ланкою між клієнтською частиною системи та базою даних, виконуючи перевірку прав доступу, валідацію вхідних даних і формування структурованих JSON-відповідей.

Важливим архітектурним рішенням стало розділення логіки методів addProfile() та completeProfile(), що наведено на рис. 3.8. Метод addProfile() використовується для первинного створення профілю пацієнта після авторизації в системі. Перед створенням нового запису виконується перевірка наявності профілю, що запобігає дублюванню даних у базі. Крім того, реалізовано комплексну валідацію введеної інформації, зокрема перевірку обов’язкових полів, коректності формату дати народження та унікальності номера телефону.

Метод completeProfile() використовується під час завершення процесу реєстрації через сервіс Firebase Authentication. Його особливістю є реалізація логіки створення або оновлення профілю залежно від того, чи існує він у системі. Такий підхід коректно оброблює випадки, коли пацієнт раніше записувався на прийом за номером телефону без створення облікового запису або вперше звертається до медичної установи.

Перегляд особистого кабінету виконує метод viewProfile(), який отримує дані поточного авторизованого користувача через механізм Firebase Authentication. У процесі виконання здійснюється перевірка ролі користувача, що обмежує доступ до персональних даних лише власнику профілю. У відповіді також повертається інформація про сімейного лікаря за допомогою механізму зв’язків Eloquent Relationships.

Оновлення персональних даних виконується методом updateProfile(). Він забезпечує цілісність інформації завдяки використанню часткової валідації та дозволяє змінювати лише ті поля, які були передані в запиті. Під час оновлення номера телефону додатково перевіряється його унікальність серед інших пацієнтів, що мінімізує ризик виникнення конфліктів даних [17].

```php
public function addProfile(Request $request){
    $user = Auth::user();
    if ($user->role !== 'patient') {
        return response()->json(['error' => 'Доступ заборонено'], status: 403);
    }
    if (Patient::where('user_id', $user->id)->exists()) {
        return response()->json([
            'error' => 'Профіль вже створено'
        ], status: 400);
    }
    $validated = $request->validate([
        'last_name' => 'required|string|max:255',
        'first_name' => 'required|string|max:255',
        'middle_name' => 'nullable|string|max:255',
        'gender' => 'required|in:male,female',
        'date_of_birth' => 'required|date',
        'address' => 'nullable|string|max:255',
        'phone' => 'required|string|max:13|unique:patients,phone'
    ]);
    $patient = Patient::create([
        'user_id' => $user->id,
        'last_name' => $validated['last_name'],
        'first_name' => $validated['first_name'],
        'middle_name' => $validated['middle_name'] ?? null,
        'gender' => $validated['gender'],
        'date_of_birth' => $validated['date_of_birth'],
        'phone' => $validated['phone'],
        'address' => $request->input( key: 'address')
```

Рисунок 3.8. Частина методу створення запису особистих даних

Однією з найважливіших функцій особистого кабінету є доступ до електронної медичної картки, який розроблений методом «viewMedicalRecords()», який формує зведену інформацію про всі завершені прийоми пацієнта. Кожний

прийом здійснюється завантаження інформації про спеціаліста, який приймав пацієнта, його спеціалізація, первинний огляд пацієнта, встановлений діагноз та призначення лікування. Використовується механізм eager loading, що дозволяє завчасно отримати всі пов’язані сутності одним набором запитів до бази даних, що суттєво підвищує продуктивність системи та зменшує навантаження на сервер. Автоматична фільтрація записів результатів аналіз вдоволяє відображати пацієнту лише змістові записи, що мають практичну цінність для перегляду історії лікування. Прикріпленні файлі результатів аналізів формується публічне посилання на завантаження через файлове сховище Laravel Storage.

Перегляд історій записів на прийом використовується методом viewReception, який повертає список усіх прийомів пацієнті разом з інформацією про лікаря, його спеціалізацію, обрані медичні послуги та дані про онлайн-консультацію. Отримані записи сортируються за датою та часом у зворотному поряду, що робить відображення спочатку актуальних прийомів першими. Контролер PersonalOfficeController реалізує повний набір функцій особистого кабінету пацієнта та забезпечує централізоване управління персональними даними, історією прийомів і медичною документацією.

Наступним ключовим компонентом у розробці інформаційної системи для медичних установ є можливість записатись пацієнту на прийом. Його функціонал реалізований у контролері ReceptionController, який створює записи як для авторизованих користувачі, так і для відвідувачів без облікового запису. Таке рішення спрощує процес взаємодії з медичним закладом та підвищує доступність до електронного сервісу для різних категорій пацієнтів.

Основна бізнес-логіка запису на прийом винесена до приватного методу createAppointments. Він робить дублювання коду не можливим та забезпечує єдиний механізм створення записів незалежно від способу ідентифікації пацієнтів(рис. 3.9). Метод приймає об’єкт HTTP-запиту та модель пацієнт, після виконує комплексну перевірку коректності отриманих даних. На першому етапі здійснюється валідація параметрів запиту, система перевіряє наявність обраного лікаря, списку медичних послуг, дати та часу прийому. Додатково контролюється

існуванням відповідних записів у базі даних та виконується перевірка, щоб дата прийому не є меншою за поточну дату, що виключає можливості створення запису заднім числом. Після виконується перевірка обраного часу пацієнтам на доступність, якщо запис на такий час існує, система блокує створення нового прийому та повертає відповідне повідомлення про зайнятість. Остання перевірка виконується для логіки визначення онлайн-консультацій. Серед обраних послуг присутня послуга яка відповідає за прийом у режимі онлайн і при її виборі система перевіряє на відповідного лікаря при його виборі, яка доступна тільки для сімейних лікарів. Результатом проходження всіх перевірок є створення нового запису у таблиці прийомів, де зберігаються відповідні дані про пацієнта, лікаря, дата і час прийому та обраної послуги. В методі розроблена можливість обирати декілька медичних послуг під час одного візиту до лікаря за допомогою проміжної таблиці зв’язку між прийомами та послугами.

```php
private function createAppointment(Request $request, Patient $patient)
{
    $validated = $request->validate([
        'doctor_id'     => 'required|exists:doctors,id',
        'service_ids'   => 'required|array|min:1',
        'service_ids.*' => 'exists:services,id',
        'date'          => 'required|date|after_or_equal:today',
        'time'          => 'required|date_format:H:i',
    ]);
    $doctor = Doctor::with( relations: 'schedules', 'specialization')
        ->findOrFail($validated['doctor_id']);
    $dayOfWeek = Carbon::parse($validated['date'])->format( format: 'l');
    $scheduleExists = $doctor->schedules()
        ->where( column: 'day_of_week', $dayOfWeek)
        ->where( column: 'start_time', operator: '<=', $validated['time'])
        ->where( column: 'end_time', operator: '>=', $validated['time'])
        ->exists();
    if (!$scheduleExists) {
        return response()->json([
            'error' => 'Лікар не працює в цей час'
        ], status: 422);
    }
    $slotBusy = Appointment::where('doctor_id', $doctor->id)
        ->where('date', $validated['date'])
        ->where('time', $validated['time'])
        ->exists();
    if ($slotBusy) {
        return response()->json([
```

Рисунок 3.9. Загальний метод створення запису на прийом

Після створення запису на прийом система автоматично формує відповідні записи у таблиці appointments_service, де зберігаються ідентифікатори послуг та їх актуальна вартість на момент створення запису. Підтримка неавторизованих користувачів розроблена в методі addReceptionGuest(рис. 3.10), який дозволяє створювати запис на прийом без попередньої реєстрації в системі. Користувачеві необхідно вказувати ПІБ та номер телефону під час створення запису, отримані дані система виконує нормалізацію номеру телефону відповідно до єдиного формату та здійснює пошук існуючого пацієнт. Якщо пацієнт відсутній в базі даних, автоматично створюється новий запис із базовими значенням полів профілю. Надалі ці дані можуть бути доповненні під час повноцінної реєстрації користувача в системі.

```
public function addReceptionGuest(Request $request)
{       $validated = $request->validate([
        'full_name' => 'required|string',
        'phone' => 'required|string',
        'doctor_id' => 'required|exists:doctors,id',
        'service_ids' => 'required|array|min:1',
        'service_ids.*' => 'exists:services,id',
        'date' => 'required|date|after_or_equal:today',
        'time' => 'required|date_format:H:i',        ]);
    $phone = preg_replace( pattern: '/\D+/', replacement: '', $validated['phone']);
    if (!str_starts_with($phone, '38')) {
        $phone = '38' . $phone;        }
    $patient = Patient::firstOrCreate(
        ['phone' => $phone],
        [
            'first_name' => 'не вказано',
            'last_name'  => 'не вказано',
            'middle_name'=> null,
            'gender'     => 'male',
            'address'    => 'не вказано',
            'date_of_birth' => now(),
        ]       );
    if ($patient->last_name === 'не вказано') {
        $parts = explode( separator: ' ', $validated['full_name']);
        $patient->update([
            'last_name'   => $parts[0] ?? 'не вказано',
            'first_name'  => $parts[1] ?? 'не вказано',
            'middle_name' => $parts[2] ?? 'не вказано',
        ]);
```

Рисунок 3.10. Метод створення запису на прийом для відвідувачів

Наступним контролером який буде відповідати за проведення прийому у онлайн режимі, методи які описані в контролері VideoConsultationController, де реалізований функціонал їх проведення та завершення(рис. 3.11). Метод startVideoCall() використовується виключно сімейним лікарем для ініціації онлайн-консультації.

Перед створення відеокімнати система виконує покрокові рівні перевірок безпеки:

- перевіряється авторизацію користувача та його роль;
- здійснюється пошук відповідного запису на прийом;
- перевіряється відповідність лікаря до конкретного прийому, що виключає доступ сторонії користувачів;
- первіряється наявність підтврдження, що онлайн-прийом вказаний як правда в полі «is_online».

```php
public function startVideoCall(Request $request)
{
    $user = auth()->user();
    $doctor = $user->doctor;
    $appointment = Appointment::findOrFail($request->appointment_id);
    if ($appointment->doctor_id !== $doctor->id) {
        return response()->json(['error' => 'Access denied'], status: 403);
    }
    if (!$appointment->is_online) {
        return response()->json(['error' => 'Not online appointment'], status: 400);
    }
    $call = VideoCall::firstOrCreate(
        ['appointment_id' => $appointment->id],
        [
            'room_id' => 'call_' . $appointment->id . '_' . time(),
            'status' => 'waiting'
        ]
    );
    return response()->json([
        'room_id' => $call->room_id
    ]);
}
```

Рисунок 3.11. Метод старту онлайн-консультації в контролері «VideoConsultationController»

Коли усі етапи перевірок пройдено система або отримує вже існуючу відеокімнату(room_id), який формується унікальним ідентифікатором на основі ідентифікатора прийому та поточного часу, за допомогою методу firstOrCreate. За замовчуванням кожен запис має статус очікування(waiting) підключення пацієнта та повертає клієнтській частині «room_id», який використовується для входу у відеосесію. Завершує онлайн-консультацію метод endVideoCall() та оновлює статус у таблицях прийому та відеокімнати на відповідний, перевіривши валідацію вхідних даних. Знаходячи відповідну кімнату та пов'язаний запис на прийом, система змінює її статус на завершено та оновлює статус прийому «completed». Особливість такого підходу централізує керування онлайн-кімнат, контролюючи доступ лише для авторизованих лікарів. Зв'язок відеоконсультації з конкретним медичним прийомом автоматично оновлює статус у системі.

Доступ до електронної картки надається через контролер PatientMedicalController(рис. 3.12), який оброблює запити перегляду, створення та редагування даних. Метод viewMedicalCard() забезпечує перегляд спеціаліста до даних медичної картки пацієнта з урахуванням безпеки та розмежування доступу. Перший етап виконує перевірки ролі користувача, де доступ дозволений лише користувачам з роллю лікаря, далі виконується перевірка профілю лікаря на факт існування в базі даних. Наступним кроком метод виконує завантаження даних пацієнта разом із пов’язанні сутностями, включаючи інформацію про сімейного лікаря та медичні записи. Контроль доступу реалізований наступним чином:

- спеціаліст є сімейним лікарем – дорступ до всіх записів пацієнта;
- спеціаліст не є сімейним лікарем – доступ тільки до власних залишених записів.

Після виконання всіх перевірок, метод формує перелік прийомів пацієнта: дані лікаря, результати медичних оглядів, діагнози та призначення лікування і інформацію про надані медичні послуги. Оптимізація продуктивності виконується механізмом eager loading, який дозволяє завантаження за датою у зворотному порядку, забезпечуючи відображення найновіших записів першими.

Створення нового запису в електронній медичній картці виконується за допомогою методу addMedicalCard(), який перед створенням запису системи виконує перевірку роль користувача та валідацію даних. Додатково метод перевіряє збіги у базі даних, на наявність існуючого запису для прийому, щоб уникнути дублювання даних. Новий запис додається до таблиці після всіх пройдених перевірок та виконує очищення керування для гарантування актуальності даних при наступному запиті до системи. Оновлення вже існуючих записів в медичній картці виконує метод updateMedicalCard, у разі допущених помилкових внесених даних і надає можливість миттєво виправити. Цей метод робить аналогічні кроки, як метод addMedicalCard(), тільки після усіх перевірок цей метод не створює новий запис, а редагує вже існуючі власні записи лікаря.

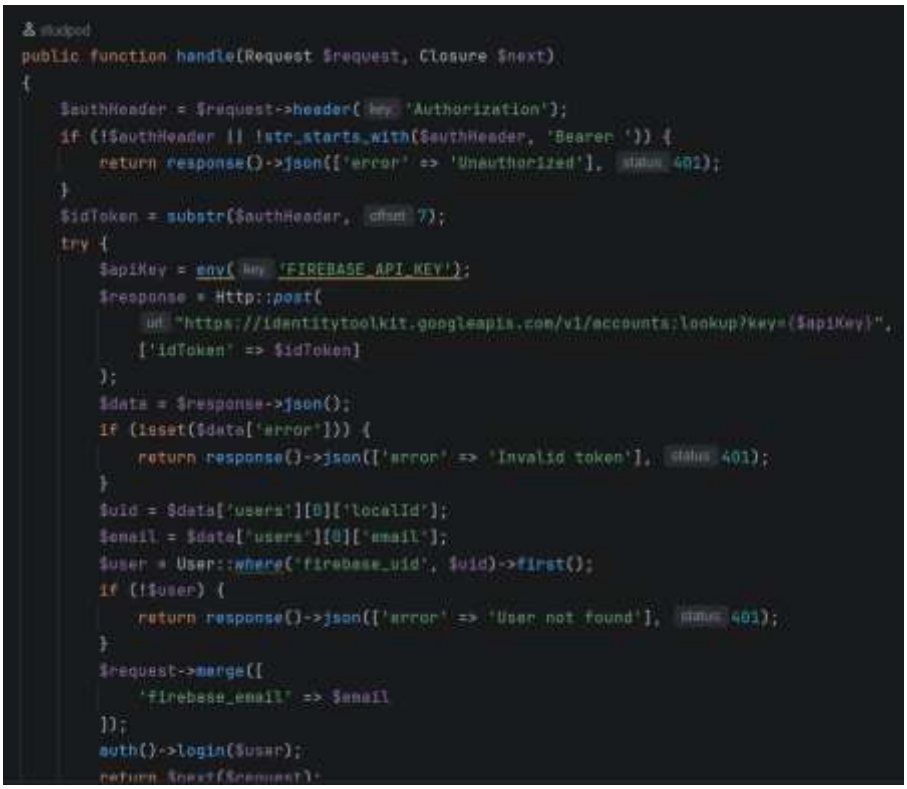
```php
if (!$user || $user->role !== 'doctor') {
    return response()->json(['error' => 'Доступ дозволений лише лікарям'], status: 403);
}

$doctor = Doctor::where('user_id', $user->id)->first();
if (!$doctor) {
    return response()->json(['error' => 'Профіль лікаря не знайдений'], status: 404);
}

$validated = $request->validate([
    'appointment_id'  => 'required|exists:appointments,id',
    'chief_complaint' => 'required|string',
    'initial_review' => 'required|string',
    'anamnesis'       => 'required|string',
    'diagnosis'       => 'required|string',
    'treatment'       => 'required|string',
    'prescriptions'   => 'nullable|string',
    'notes'           => 'nullable|string',
]);
$patient = Patient::find($patientId);
if (!$patient) {
    return response()->json(['error' => 'Пацієнт не знайдений'], status: 404);
}
$appointment = Appointment::where('id', $validated['appointment_id'])
    ->where('patient_id', $patient->id)
    ->first();
```

Рисунок 3.12. Метод створення нового медичного запису

Невід’ємною частиною розробки серверної частини є реалізація middleware для автентифікації користувача через сервіс Firebase та перевірку та контролювання доступу до захищених API-запитів за роллю. Автентифікація виконується з перевірки токену який генерує та передає на клієнтську частину, де потім виконується HTTP-запит до серверної частини для перевірки у відповідному middleware FirebaseAuth(рис 3.13). Спочатку він перевіряє наявність заголовка Authorization у HTTP-запиті очікуючи «Bearer <idToken>». У випадку якщо заголовок відсутній або не відповідає потрібному формату, система одразу повертає відповідь з кодом 401 Unauthorized, щоб запобігти подальшій обробці запиту. Отримавши токена(idToken) виконується його перевірка через REST API Firebase Identity Toolkit за допомогою HTTP-запита «accounts:lookup» з передачою API ключа Firebase, який зберігається у зміна середовища (.env). У відповідь сервіс повертає інформацію про користувача, в яке входить унікальний ідентифікатор, електронна пошта та статус валідності токена. Якщо Firebase повертає помилку або токен не є дійсним, запит завершується з кодом 401 Unauthorized.



```
public function handle(Request $request, Closure $next)
{
    $authHeader = $request->header( key: 'Authorization');
    if (!$authHeader || !str_starts_with($authHeader, 'Bearer ')) {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }
    $idToken = substr($authHeader, offset: 7);
    try {
        $apiKey = env( key: 'FIREBASE_API_KEY');
        $response = Http::post(
            url: "https://identitytoolkit.googleapis.com/v1/accounts:lookup?key={$apiKey}",
            ['idToken' => $idToken]
        );
        $data = $response->json();
        if (isset($data['error'])) {
            return response()->json(['error' => 'Invalid token'], status: 401);
        }
        $uid = $data['users'][0]['localId'];
        $email = $data['users'][0]['email'];
        $user = User::where('firebase_uid', $uid)->first();
        if (!$user) {
            return response()->json(['error' => 'User not found'], status: 401);
        }
        $request->merge([
            'firebase_email' => $email
        ]);
        auth()->login($user);
        return $next($request);
```

Рисунок 3.13. Частина коду Middleware FirebaseAuth

Після успішної валідації токена виконується пошук користувача в локальній базі даних Laravel за полем firebase_uid, який зберігається в таблиці users. Коли користувача не знайдено у системі, доступ до ресурсів теж забороняється та забезпечує синхронізацію Firebase Authentication із внутрішньою системою користувачів. У випадку успішної перевірки middleware додає у відповідь електрону пошту користувача, виконує авторизацію через «auth()->login($user)» і передає на клієнтську частину повідомлення про успішну авторизацію.

Другим по важливості розроблено middleware для перевірки ролей користувача які намагаються виконати запит до захищених API. RoleMiddleware(рис. 3.14) перевіряє користувача на наявність авторизації у системі, де при відсутності повертає null відхиливши його з кодом 401. Перевіривши та впевнившись, що користувач авторизований, виконує перевірку ролі приймаючи перелік дозволений ролей як параметри маршруту, що робить налаштування більш гнучким до доступу до різних частин веб-сайту. Роль користувача порівнюється зі списком дозволених ролей за допомогою функції in_array(). Якщо роль не входить до дозволеного переліку, система повертає відповідь з кодом 403 – доступ заборонено. Особливістю є можливість передачі декілька ролей одночасно, використовувавши тільки один middleware не старуючи для цього їх окремо.

```php
public function handle(Request $request, Closure $next, ...$roles)
{
    $user = auth()->user();

    if (!$user) {
        return response()->json(['error' => 'Неавторизований користувач'], status: 401);
    }

    if (!in_array($user->role, $roles)) {
        return response()->json(['error' => 'Доступ заборонено'], status: 403);
    }

    return $next($request);
}
```

Рисунок 3.14. Частина коду Middleware RoleMiddleware

3.4. Розробка захищених API-запитів на серверній частині

У межах розробки інформаційної системи для приватної клініки на серверній частині було реалізовано REST API з розмежуванням доступу до ресурсів залежно від ролі користувача. Основою архітектури є система маршрутів Laravel, які об'єднані в окремі групи із застосуванням middleware для забезпечення безпеки та контролю доступу.

У системі розроблено чотири основні групи маршрутів для різних категорій користувачів: пацієнтів, лікарів, працівників реєстратури та адміністраторів. Для кожної групи використовується окремий префікс:

- patient;
- doctor;
- receptionist;
- admin.

Кожна група має захист у вигляді middleware для перевірки авторизація та ролі користувача, роль вказується окремо. Приклад захисту middleware приведено на рис. 3.15

```
Route::middleware([FirebaseAuth::class, 'role:patient'])
    ->prefix( prefix 'patient')
    ->group(function () {
        Route::prefix( prefix 'view')->group(function () {
            Route::get( uri '/me', [PersonalOfficeController::class, 'me']);
```

Рисунок 3.15. Захищений маршрут middleware

Група маршрутів для пацієнтів реалізує окремі запити для користувачів із роллю patient. Всередині цієї групи є два основних групи для перегляду(view) та CRUD-запитів(control), кожен з цих запитів відповідає за певний аспект взаємодіє користувача з серверною частиною. Перша група «/patient/view» дає доступу до особистих даних пацієнта, які наведено на рис. 3.14. Вони відповідають за отримання таких даних:

- отримання базової інформації про користувача(/me);

- перегляд особистого профілю(/profile);
- перегляд історій прийомів(/receptions);
- отримання результатів аналізів.

Дані маршрути забезпечують роботу особистого кабінету пацієнта та надають доступ до медичної інформації у зручному структурованому вигляді. Наступною групою маршрутів для пацієнтів є «/patient/control» (рис. 3.15). Вони призначені для створення (POST) та оновлення (PUT) даних у системі.

За допомогою цих маршрутів виконуються функції створення запису на прийом, оновлення персональних даних. Кожен маршрут пов’язаний із відповідним контролером, у якому реалізовано бізнес-логіку та послідовність дій для виконання конкретної операції.

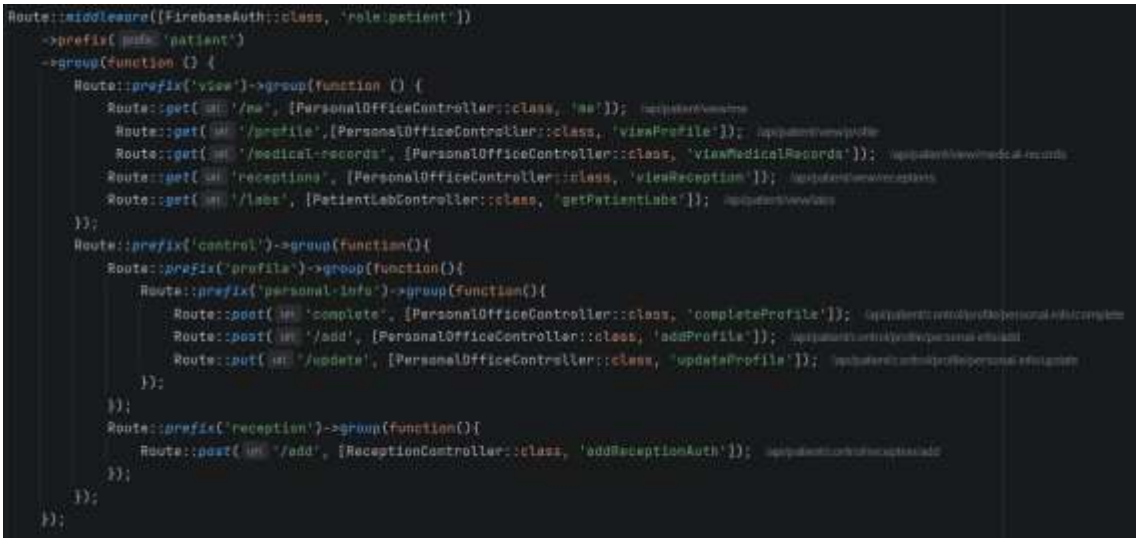


Рисунок 3.16. Захищені маршрути пацієнтів

Після розробки захищених API-запитів для пацієнтів необхідно реалізувати окремий набір захищених маршрутів для користувачів із роллю doctor. Доступ до ресурсів цієї групи надається лише після успішного проходження автентифікації та перевірки ролі користувача. Структура маршрутів лікаря має подібну організацію та також поділяється на групи для перегляду даних і виконання CRUD-операцій. Усередині цих груп використовуються додаткові вкладені маршрути для

запобігання дублюванню однакових префіксів. Приклад API-запитів лікаря наведено в лістингу 3.2.

```
Route::prefix( prefix 'patient')->group(function(){
    Route::get( uri '/search', [PatientController::class, 'searchPatient']);
    Route::get( uri '/all', [PatientController::class, 'viewPatients']);
        Route::prefix( prefix '{patientId}')->group(function(){
            Route::get( uri 'appointments', [AppointmentController::class, 'getPatientAppointments']);
            Route::get( uri '/medical-card', [PatientMedicalController::class, 'viewMedicalCard']);
```

Рисунок 3.17. Групові захищені маршрути лікаря

У наведеному лістингу маршрути об'єднані в спільну групу. Через HTTP-метод GET виконується пошук пацієнтів, перегляд всі пацієнтів, отримання інформації прийомів та доступ до електронної медичної картки. Усі маршрути пов’язані з відповідними контролерами, розташованими в структурі контролерів медичного персоналу (Controllers/Staff/Doctor). Назви маршрутів відповідають назвам методів, реалізованих у цих контролерах.

Основними групами захищених API-маршрутів для лікаря є «/doctor/view» та «/doctor/control». Група «/doctor/view» призначена для отримання інформації з системи. Вона забезпечує доступ до даних про графік роботи лікаря, списки пацієнтів, їхні електронні медичні картки, записи на прийоми, а також інформацію. Для чіткої структури системи, маршрути розподілені між кількома контролерами відповідно до їхнього призначення. Наприклад, усі запити, пов’язані з пацієнтами та їхніми даними, обробляються контролером PatientController.

Група «/doctor/control» використовується для виконання операцій зі створення, оновлення та видалення даних у базі даних. Маршрути цієї групи також розподілені між відповідними контролерами залежно від реалізованого функціоналу, що спрощує підтримку коду та забезпечує чітке розмежування відповідальності між компонентами системи.

3.4. Висновок до Розділу 3

У третьому розділі розглянуто процес розробки серверної частини інформаційної системи для автоматизації роботи приватної клініки. Основну увагу приділено реалізації ключових модулів бізнес-логіки та механізмів взаємодії між підсистемами. Під час розробки використано фреймворк Laravel, який забезпечив необхідні інструменти для побудови структурованої серверної архітектури. Реалізовано набір контролерів для основних категорій користувачів: пацієнтів, лікарів, працівників реєстратури та адміністраторів. Контролери відповідають за обробку запитів клієнтської частини, виконання бізнес-логіки та взаємодію з базою даних через ORM Eloquent.

Особливу увагу приділено реалізованим функціональним модулям системи, зокрема:

- особистий кабінет пацієнта;
- електронна медична картка;
- запис на прийом до лікаря;
- управління медичними послугами;
- робота з результатами аналізів;
- інструменти для лікарів (розклад, перегляд пацієнтів, медична документація);
- модуль онлайн-консультацій.

Захист даних і контроль доступу реалізовано через багаторівневу систему безпеки. Автентифікація користувачів побудована на основі Firebase Authentication із використанням JWT-токенів. Додатково впроваджено middleware для перевірки ролей, що дозволяє розмежувати доступ до функціоналу залежно від типу користувача.

Також розроблено структуровану систему маршрутів, згрупованих за ролями та функціональним призначенням. Це дозволило спростити архітектуру серверної частини, підвищити читабельність коду та забезпечити можливість

подальшого масштабування. Обмін даними між клієнтом і сервером здійснюється у форматі JSON.

Окремим елементом реалізації є модель даних. За допомогою Eloquent ORM створено систему взаємопов’язаних моделей, які відображають основні сутності предметної області: пацієнтів, лікарів, прийоми, медичні картки, послуги, результати лабораторних досліджень та відео-консультації. Використання зв’язків між моделями дозволило ефективно організувати роботу з даними та зменшити складність запитів до бази даних.

У результаті виконання цього етапу реалізовано основні серверні модулі системи, які забезпечують роботу ключових бізнес-процесів медичної клініки. Запропонована архітектура характеризується модульністю, безпекою та можливістю розширення, що відповідає сучасним вимогам до подібних інформаційних систем.

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1. Модульне тестування

Модульне тестування серверної частини медичної інформаційної системи було реалізовано з використанням фреймворку PHPUnit у середовищі Laravel. Основна увага приділялась перевірці окремих модулів API, бізнес-логіки та коректності взаємодії між моделями системи. Для тестування було використано підхід ізольованого виконання кожного сценарію з використанням механізму RefreshDatabase, що забезпечувало автоматичне очищення тестової бази після кожного запуску та виключало вплив попередніх тестів на результати наступних.

На рис. 4.1 наведено частина тестування сценарію створення прийому пацієнтом. У межах цього сценарію перевірялася робота API-методу «/api/patient/control/reception/add», який відповідає за формування нового запису до лікаря. Під час тестування моделювалась ситуація, коли користувач передає необхідні дані для створення прийому:

- передача даних – doctor_id, service_ids, date, time;
- перевірка обов’язковості наявності лікаря та послуг в базі;
- перевірка відповіді API зі статусом 201 Created.

Рисунок 4.1. Тестування моделей системи

У наступному блоці тестування перевірявся механізм автентифікації та контролю доступу до API. Сценарій тестування виконував декілька варіантів взаємодії користувача із системою:

- при зверненні до захищених ендпоінтів без токена доступу система повертала відповідь 401 Unauthorized;
- при наявності коректно автентифікованого користувача запити виконувались успішно зі статусом 200 OK;
- додатково перевірялася робота middleware FirebaseAuth, який відповідає за перевірку токенів користувача;
- окремо тестувалась система ролей, яка обмежує доступ залежно від типу ролі користувача (patient, doctor, admin).

```php
class SyncAuthTest extends TestCase
{
    use RefreshDatabase;

    studpod *
    public function test_user_can_be_created_from_firebase()
    {
        $response = $this->postJson( url: '/api/auth/sync', [
            'uid' => 'firebase_uid_test_123',
            'email' => 'test@example.com',
            'phone' => '380991112233'
        ]);
        $response->assertStatus( status: 200);
        $this->assertDatabaseHas( table: 'users', [
            'firebase_uid' => 'firebase_uid_test_123',
            'role' => 'patient'
        ]);
    }

    studpod
    public function test_sync_fails_without_uid()
    {
        $response = $this->postJson( url: '/api/auth/sync', [
            'email' => 'test@example.com'
        ]);

        $response->assertStatus( status: 400);
    }
}
```

Рисунок 4.2. Модуль тестування SyncAuthTest

Протестований процес синхронізації користувача через Firebase (/api/auth/sync):

- створення нового запису користувача при його відсутності в базі;
- коректну обробку запитів із валідними даними;
- повернення помилки «400 Bad Request» у випадку відсутності обов’язкового параметра UID.

Окремий блок тестування був присвячений перевірці коректності роботи Eloquent-моделей та їх взаємозв’язків. В модулі тестування перевірялось коректність звязків:

- користувачі системи User пов’язані з профілями лікаря та пацієнта;
- лікарі пов’язані зі Specialization;
- прийоми мають зв’язки з пацієнтами та лікарями;
- медичні послуги реалізовані через проміжну таблицю Appointment до Service;
- медичні записи належать конкретному прийому;
- лабораторні результати пов’язані з виконаними послугами та файлами результатів LabsFile;
- відеоконсультації прив’язані до прийомів;
- реєстратори Receptionist пов’язані з користувачами системи.

Рисунок 4.3. Частина модуля тестування ModelsTest

У цьому ж блоці тестувалось коректність створення записів у базі даних, цілісність зовнішніх ключів та правильну роботу каскадних залежностей між таблицями.

У результаті модульного тестування було перевірено ключові компоненти системи: адміністративний, лікарський і пацієнтський модулі, а також механізми автентифікації та роботи базових моделей. На рис. 4.4 приведено результат успішного тестування всіх компонентних модулів.

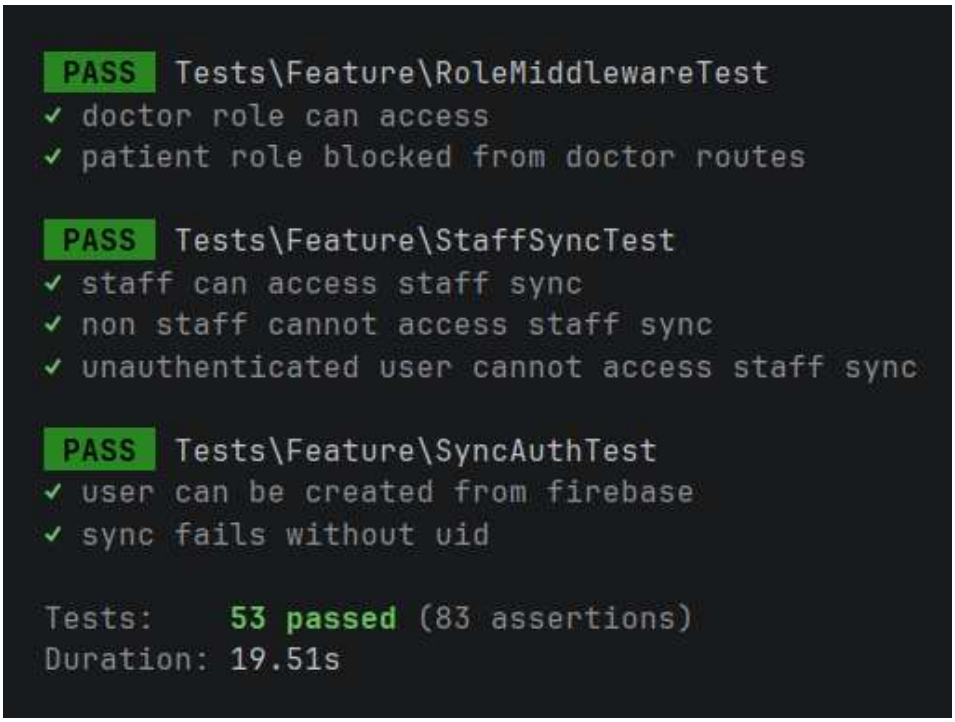


Рисунок 4.4. Результати модульного тестування

4.2. Інтеграційне тестування REST API

Перевірка коректності взаємодії між клієнтською та серверною частинами системи було проведено інтеграційне тестування REST API. Основним інструментом тестування виступив Postman, який дозволяє формувати HTTP-запити, аналізувати відповіді сервера та перевіряти правильність роботи програмного інтерфейсу. Метою цієї перевірки є підтвердження коректності роботи серверних маршрутів, обробки запитів від клієнтської частини та забезпечення правильної взаємодії між контролерами, моделями та базою даних.

Під час тестування приділялась увага основним бізнес-процесів медичної інформаційної системи, зокрема авторизації користувачів, запису на прийом, управління лікарями, роботі реєстратури та перегляд медичної інформації. Для тестування було створено окрему колекцію запитів Postman, що охоплювало всі ключові REST API маршрутів системи. Кожен запит перевірявся як у позитивному сценарію, так і у випадках передачі некоректних або неповних даних[23].

На першому етапі тестування протестована механізм синхронізації користувачів з сервісом Firebase Authentication. Перевірялось реєстрація нового користувача після отримання Firebase UID та правильність зберігання інформації у базі даних(рис. 4.3).

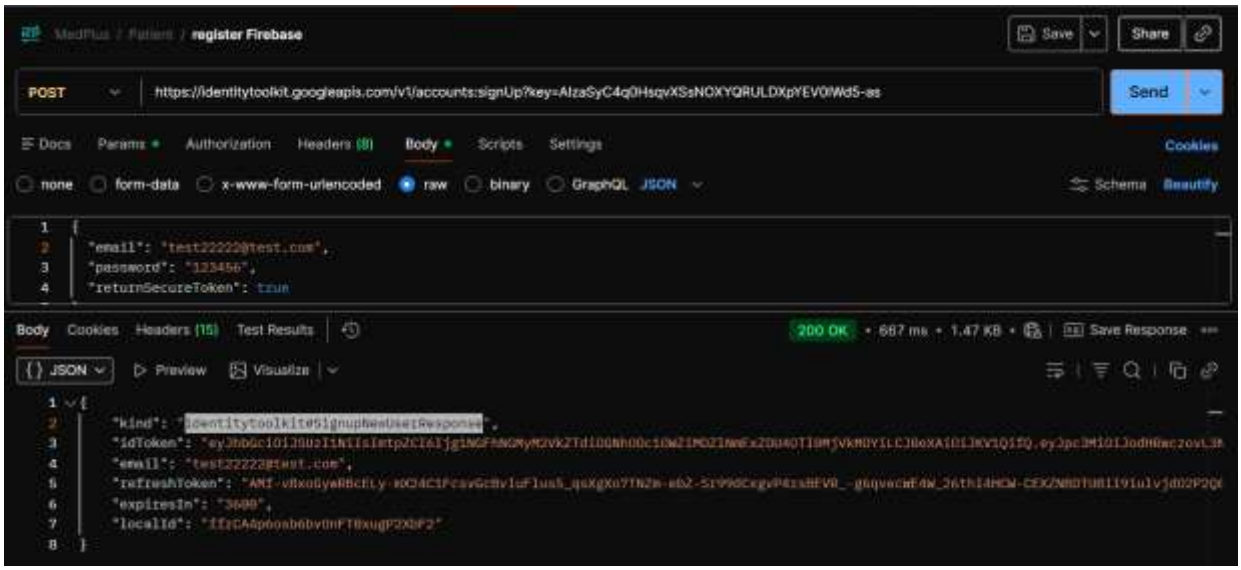


Рисунок 4.5. Тестування отримання токену від Firebase

Функціонал тестування запису пацієнта на прийом до лікаря, надсилаючи POST-запит із зазначенням лікаря, обраних медичних послуг, дати та часу прийому(рис. 4.3). Після успішного виконання запиту сервер надсилав повідомлення про успіх. Вхідні дані запиту формуються у форматі JSON:

- ідентифікатор користувача;
- перелік послуг;
- дата прийому;

- час прийому.

Сервер повернув HTTP-статус кодом 200, що підтверджує створення запису в таблиці прийомів, на виконання якого склав час 1,47 секунди розміром 550 байт. У відповіді міститься повідомлення про успішне створення запису та об’єкт appointment з повними даними створеного запису. Сервер автоматично присвоїв запису унікальний ідентифікатор «64», встановила статус очікування, значення поля is_online встановлено як «false», що вказує на проведення прийому у режимі оффлайн. У висновку сервер успішно оброблює запит створення запису на прийом пацієнтом та повертає коректну відповідь клієнтській частині[23].

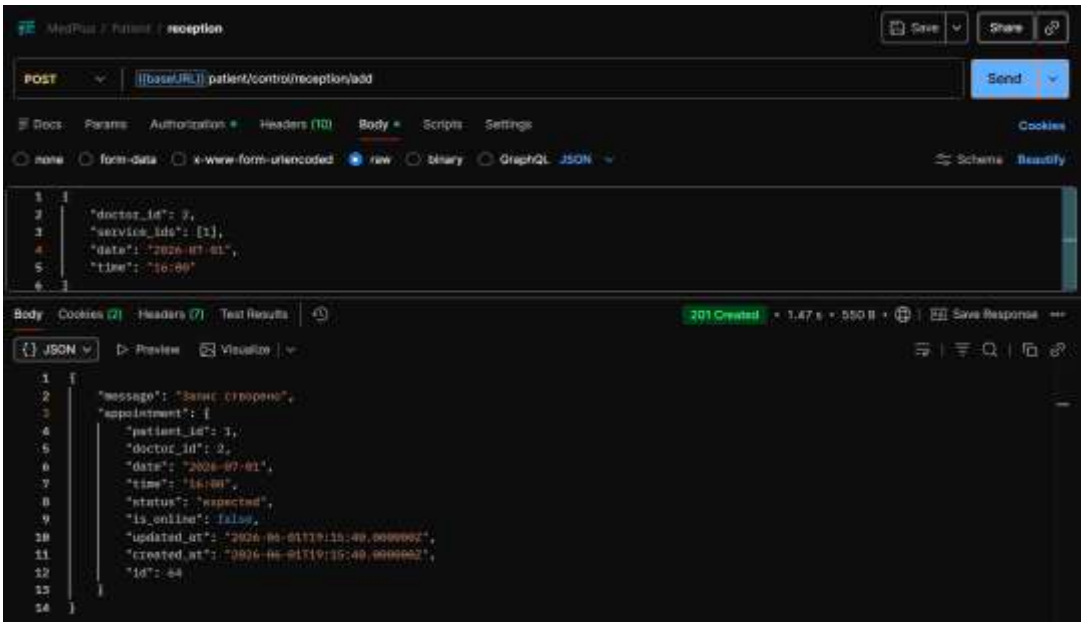


Рисунок 4.6. Тестування створення запису на прийом

Наступним етапом тестування було перевірка створення запису в медичній картці пацієнта за допомогою API-запиту «api/doctor/control/patient/1medical-card/add». У заголовок запиту обов’язково передається токен авторизації у форматі Beer Token, які підтверджує роль користувача та надає доступ до захищеного ендпоінта. Така перевірка забезпечує перевірку того, що операцію може виконувати лише авторизований користувач з роллю лікаря. У тілі запиту(Body) дані передаються у форматі JSON, вони містять основні клінічні параметри прийому: ідентифікатор прийому та його результат проведення. Сервер отримавши

запит виконує перевірка прав доступу користувача, наявність лікаря в системі, відповідність прийому конкретному пацієнту та відсутність дублювання запису в медичній картці для цього прийому, що можна побачити на рис. 4.4. Уразі успішного виконання операції сервер повертає відповідь із HTTP-статусом 201 та повідомлення про успіх створення запису в таблиці. Додатково відповідь повертає створений медичний запис.

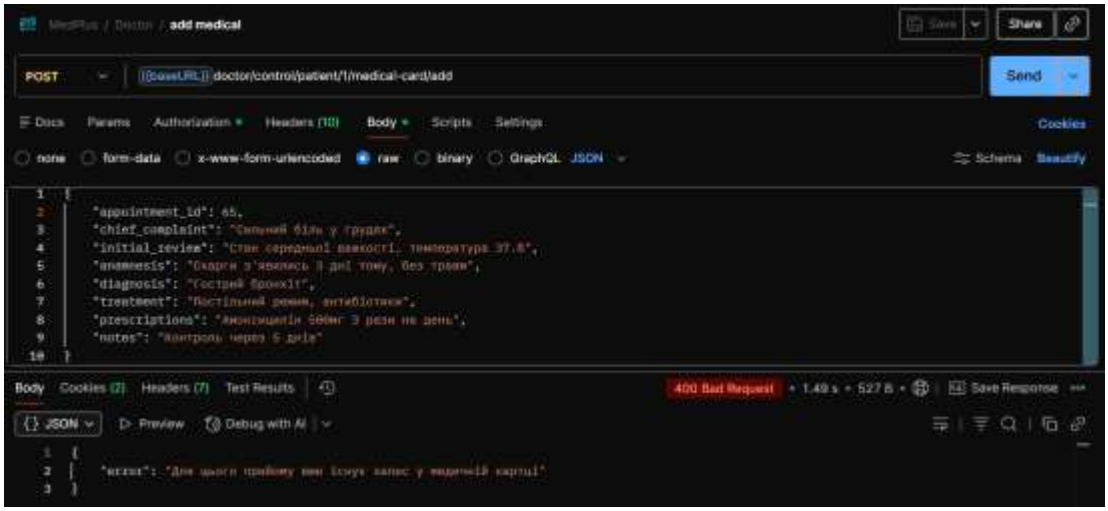


Рисунок 4.7. Тестування запиту створення запису медичної картки

4.3. Встановлення та експериментальне застосування програми

Розроблена веб-система для записів на прийоми в медичних установах є додатком, який не потребує складного встановлення на пристрій користувача. Для роботи з веб-додатком достатньо встановити сучасний браузер(Chrome,Mozilla Firefox та інші.) та стабільне підключення до Інтернету. Система розроблена у форматі клієнт-серверної архітектурної, тому всі обчислення та збереження даних виконуються на сервері. Користувачу не потрібно встановлювати додаткове програмне забезпечення, окрім браузера. Для коректної роботи системи рекомендовано використовувати:

- операційну систему Windows;
- сучасний веб-браузер останніх версій;
- стабільне підключення до мережі інтернет;

- роздільна здатність екрану не меньше 1920x1080.

Початком роботи з веб-додатком користувачеві необхідно перейти за посиланням, де після завантаження сторінки доступності функції авторизації або реєстрації через сервіс автентифікації. Після пацієнт може перейти на сторінку свого електронного кабінету та переглянути або редагувати особисті дані, або по бажанню змінити електронну пошту та пароль. На цій ж сторінці переключитись на медичну картку та переглянути свою медичну картку. Додатково пацієнт може передивитись історію прийомів та результати свої аналізів. Для запису на прийом, пацієнту необхідно перейти на головну сторінку, де на початку буде кнопка з переправленням або спочатку перейти на сторінку лікарів та ознайомитись з лікарями перед тим як записатись до когось на прийом.

Медичний персонал переходить за посиланням до окремої медичної панелі, де авторизувавшись вони залежно від своєї ролі отримують доступ до певних можливостей в панелі. Лікаря на головні сторінці панелі можуть ознайомитись з календарем, де розписаний тиждень та можуть заздалегідь готуватись до прийомів. Можуть перейти на сторінку всіх прийомів до лікаря у відповідному мені зліва, та передивившись усі список, які відображаються спочатку нові прийому які мають статус очікування, а після вже закриті прийоми. Медичний персонал може оновити свої дані в медичній панелі, змінити пароль або електронну пошту у разі необхідності.

4.4. Висновки до розділу 4

У даному розділі проведено комплексне тестування реалізованих модулів медичної інформаційної системи. Воно включало модульне, інтеграційне та функціональне тестування, перевірку REST API за допомогою інструменту Postman і аналіз роботи окремих сценаріїв використання.

Під час модульного тестування перевірено коректність роботи основних компонентів системи, бізнес-логіки та зв’язків між сутностями. Використання фреймворку PHPUnit і механізму RefreshDatabase забезпечило ізольоване

виконання тестів і стабільність результатів. Усі реалізовані модульні тести виконано успішно, що підтверджує коректність роботи базових компонентів.

Інтеграційне тестування REST API, виконане за допомогою Postman, дозволило перевірити взаємодію між клієнтською та серверною частинами. Основну увагу було приділено ключовим реалізованим процесам, зокрема:

- автентифікації користувачів через Firebase Authentication;
- запису пацієнтів на прийом;
- роботі реєстратури;
- управлінню медичними даними.

Результати тестування показали, що всі основні API-ендпоінти коректно обробляють запити, повертають відповідні HTTP-статуси та забезпечують правильну структуру відповіді у форматі JSON. Також підтверджено коректність роботи механізмів автентифікації та розмежування доступу за ролями користувачів.

Окремо проаналізовано поведінку реалізованих модулів у межах основних сценаріїв використання, зокрема створення та оновлення медичних записів, формування прийомів, робота з результатами аналізів та відеоконсультаціями. Отримані результати свідчать про стабільну та передбачувану роботу реалізованого функціоналу.

Підсумовуючи, можна зазначити, що проведене тестування підтвердило відповідність реалізованих модулів функціональним вимогам. Виявлені під час розробки незначні помилки були усунені, що дозволило підвищити якість та надійність програмної реалізації. Реалізований функціонал готовий до використання та подальшого розширення

ВИСНОВКИ

Результатом виконання дипломної роботи стала розробка та практична реалізація веб-системи для автоматизації процесу прийому пацієнтів у медичних установах. У ході роботи було досягнуто поставленої мети та виконано основні завдання, що дозволило створити програмне рішення для автоматизації ключових процесів взаємодії між пацієнтами та медичним персоналом. Розроблені модулі спрямовані на підвищення ефективності роботи медичних закладів, спрощення ведення медичної документації та покращення якості надання медичних послуг.

На етапі проєктування було проведено аналіз предметної області, досліджено особливості функціонування сучасних медичних інформаційних систем та визначено основні вимоги до програмного продукту. На основі цього обрано технологічний стек: Laravel для серверної частини, React для клієнтського інтерфейсу та MySQL для зберігання даних. Використання клієнт-серверної архітектури забезпечило чіткий розподіл відповідальності між компонентами та підвищило масштабованість системи.

У межах практичної реалізації було розроблено основні функціональні модулі системи. Для пацієнтів реалізовано:

- реєстрацію та автентифікацію користувачів;
- перегляд інформації про медичний заклад;
- запис на прийом до лікаря;
- доступ до електронної медичної картки;
- перегляд результатів лабораторних досліджень.

Для лікарів реалізовано окремий функціональний модуль, який включає:

- перегляд розкладу та записів на прийом;
- ведення електронної медичної документації;
- внесення результатів оглядів;
- формування медичних висновків.

Окрему увагу приділено модулю онлайн-консультацій між пацієнтом і лікарем. Реалізований механізм відеозв’язку дозволяє проводити дистанційні консультації в реальному часі, що підвищує доступність медичних послуг і зменшує потребу в особистому відвідуванні клініки.

Під час тестування перевірено роботу основних реалізованих модулів, зокрема автентифікації та авторизації, запису на прийом, ведення медичних карток, перегляду результатів аналізів та проведення відеоконсультацій. Результати тестування підтвердили коректність роботи функціоналу, відповідність вимогам та стабільність реалізованих рішень.

Підсумовуючи, можна зробити висновок, що реалізовані модулі забезпечують автоматизацію ключових процесів роботи медичної установи. Розроблене програмне рішення має модульну структуру, можливість масштабування та може бути розширене у майбутньому шляхом інтеграції з державними медичними сервісами, впровадження електронних рецептів, мобільного застосунку та додаткових аналітичних інструментів.

Посилання на репозиторій з кодом програми URL: https://github.com/studpod/medplus

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Законодавство України про захист персональних даних. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 12.04.2026).

2. Міністерство охорони здоров'я України — офіційний портал. URL: https://moz.gov.ua (дата звернення: 10.04.2026).

3. Офіційний сайт медичного центру «Асклепій». URL: https://asklepiy.com (дата звернення: 14.04.2026).

4. Офіційний сайт медичної мережі «Добробут». URL: https://dobrobut.com/ua (дата звернення: 14.04.2026).

5. Офіційний сайт медичної мережі «Medibor». URL: https://medibor.ua (дата звернення: 25.04.2026).

6. Офіційний сайт медичного центру «Нікамед». URL: https://nikamed.com.ua/ua (дата звернення: 25.04.2026).

7. Офіційний сайт медичної мережі «Oxford Medical» Житомир. URL: https://zhitomir.oxford-med.com.ua/ua (дата звернення: 25.04.2026).

8. Acceldata. What Is a Relational Database? URL: https://www.acceldata.io/blog/what-is-a-relational-database-architecture-features-and-real-world-applications (дата звернення: 20.04.2026).

9. Agora — Real-Time Engagement Platform Documentation. URL: https://docs.agora.io/en (дата звернення: 08.05.2026).

10. Axios — Promise Based HTTP Client Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 07.05.2026).

11. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley, 2021. 480 p.

12. Cloudexis Technolabs. Creating RESTful APIs in Laravel 12 for React Applications. URL: https://cloudexistechnolabs.com/creating-restful-apis-in-laravel-12-for-react-applications/ (дата звернення: 15.04.2026).

13. Firebase Authentication Documentation. URL: https://firebase.google.com/docs/auth (дата звернення: 09.05.2026).

14. Firebase Realtime Database Documentation. URL: https://firebase.google.com/docs/database (дата звернення: 09.05.2026).

15. Fowler M. Refactoring: Improving the Design of Existing Code. 3rd ed. Boston: Addison-Wesley, 2022. 448 p.

16. Laravel — Eloquent ORM Documentation. URL: https://laravel.com/docs/eloquent (дата звернення: 06.05.2026).

17. Laravel — Official Documentation. URL: https://laravel.com/docs (дата звернення: 05.05.2026).

18. Laravel — RESTful API Resources Documentation. URL: https://laravel.com/docs/eloquent-resources (дата звернення: 06.05.2026).

19. Laravel — Sanctum Authentication Documentation. URL: https://laravel.com/docs/sanctum (дата звернення: 07.05.2026).

20. Laravel with Firebase Authentication Guide. URL: https://ilaydadastan.com/laravel-with-firebase-authentication-c48e67e24837 (дата звернення: 12.05.2026).

21. MDN Web Docs — WebRTC API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (дата звернення: 08.05.2026).

22. MySQL — Official Documentation. URL: https://dev.mysql.com/doc (дата звернення: 07.05.2026).

23. Pressman R., Maxim B. Software Engineering: A Practitioner’s Approach. 9th ed. New York: McGraw-Hill, 2021. 976 p.

24. OneUptime Blog. How to Use MySQL with Laravel. URL: https://oneuptime.com/blog/post/2026-03-31-mysql-laravel-php/view (дата звернення: 10.04.2026).

25. React Documentation. URL: https://react.dev (дата звернення: 10.05.2026).

26. React Router — Official Documentation. URL: https://reactrouter.com/en/main (дата звернення: 10.05.2026).

27. Sass (SCSS) — Official Documentation. URL: https://sass-lang.com/documentation/ (дата звернення: 09.05.2026).

28. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2023. 810 p.

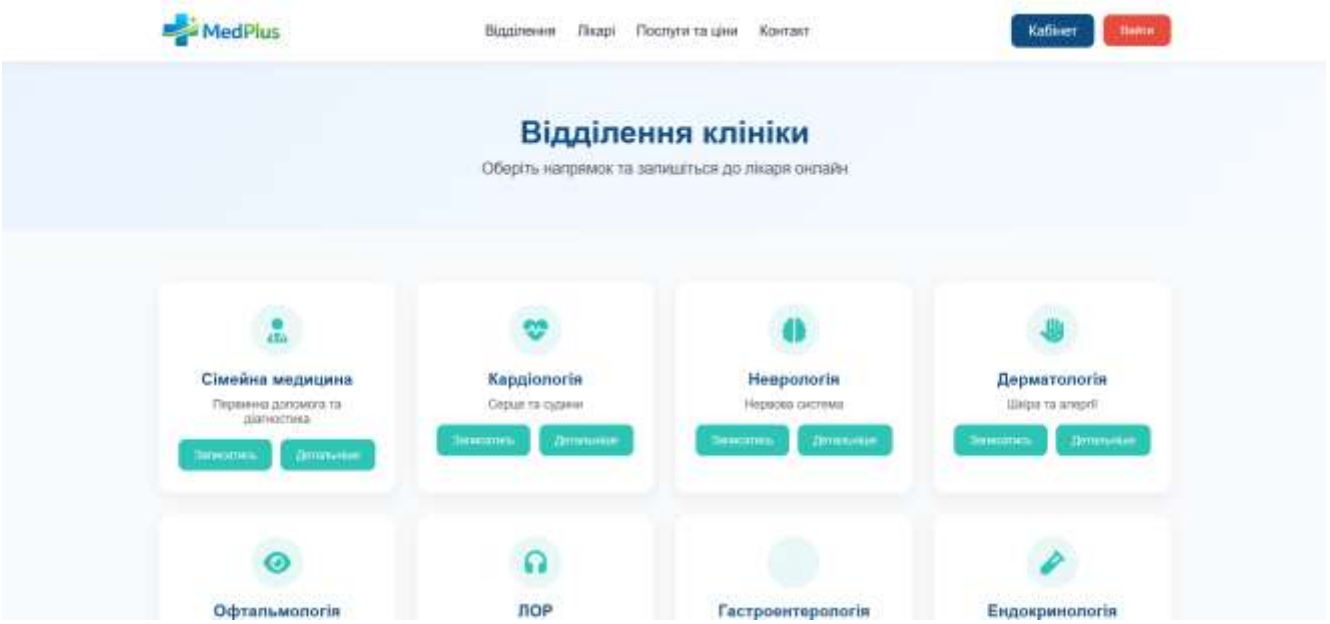
29. Testing: Getting Started. URL: https://laravel.com/docs/13.x/testing (дата звертання: 11.02.2026)

30. WebRTC.org - Official WebRTC Project Site. URL: https://webrtc.org (дата звернення: 08.05.2026).

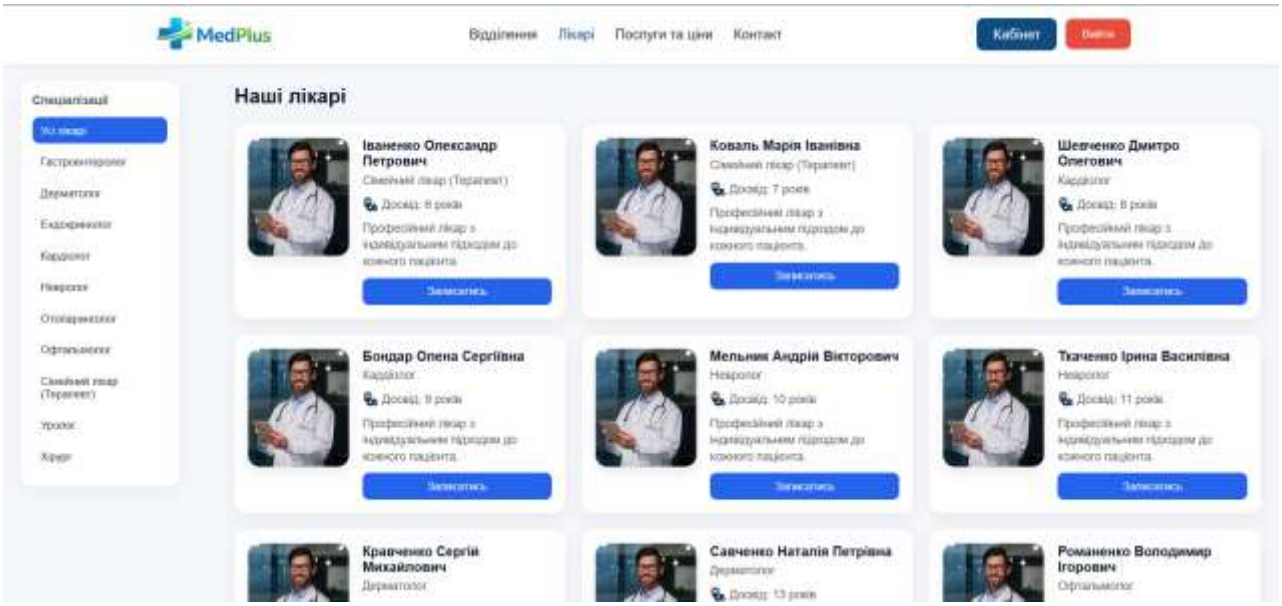
ДОДАТОК А

Інтерфейс користувача

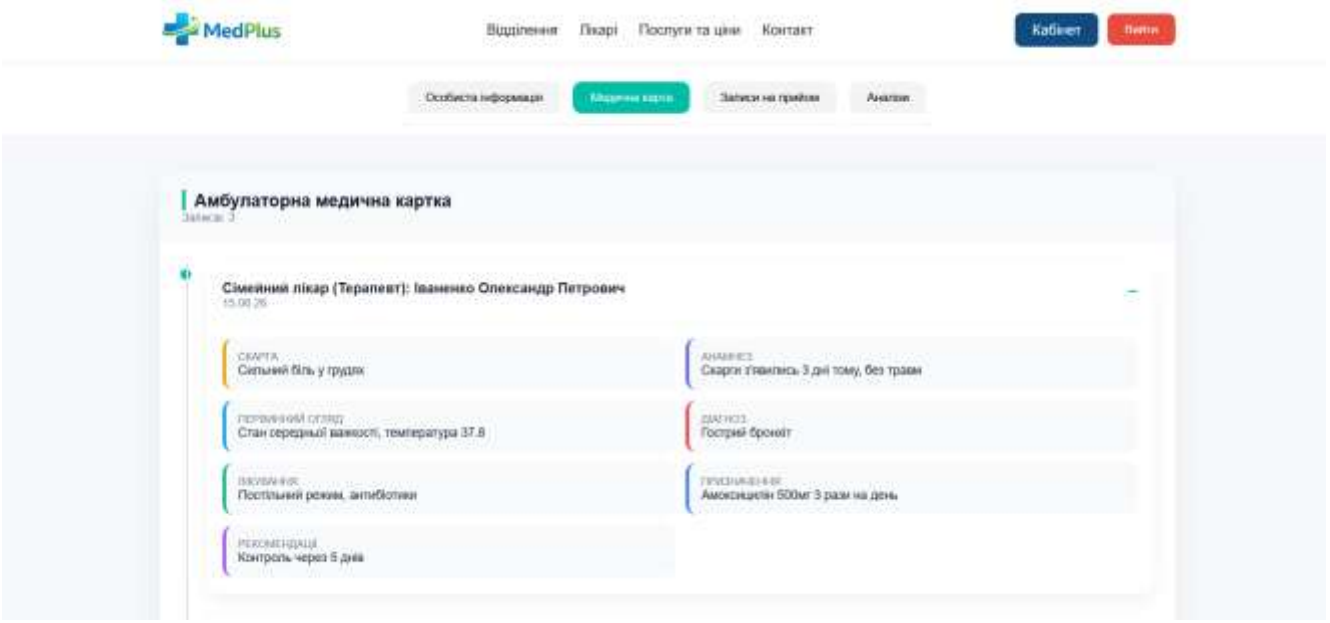
А.1 Сторінка відділення



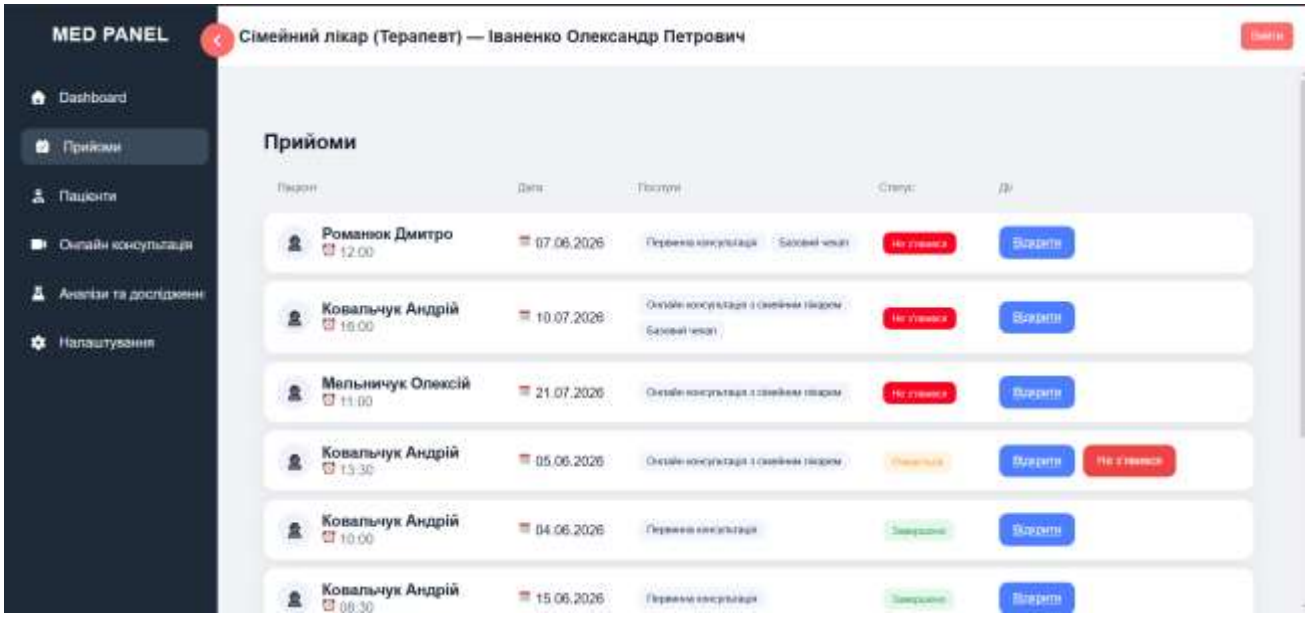
А.2 Сторінка лікарів



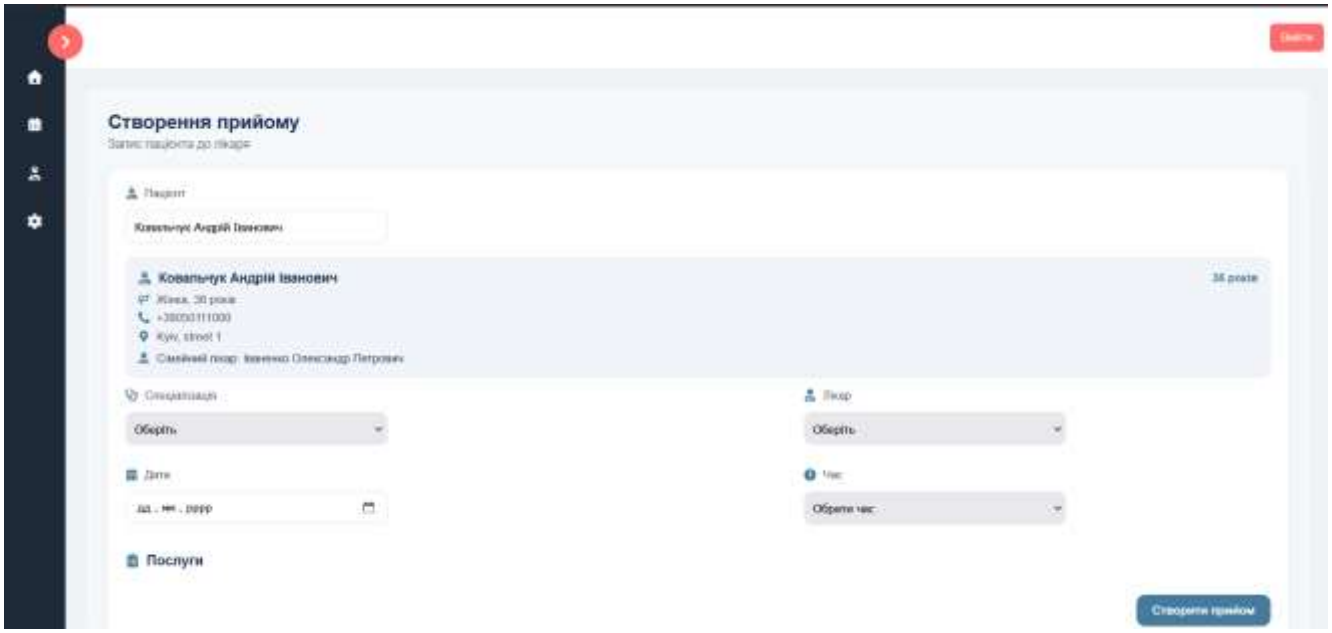
А.3 Сторінка Електронна медична картка пацієнта в електронному кабінеті



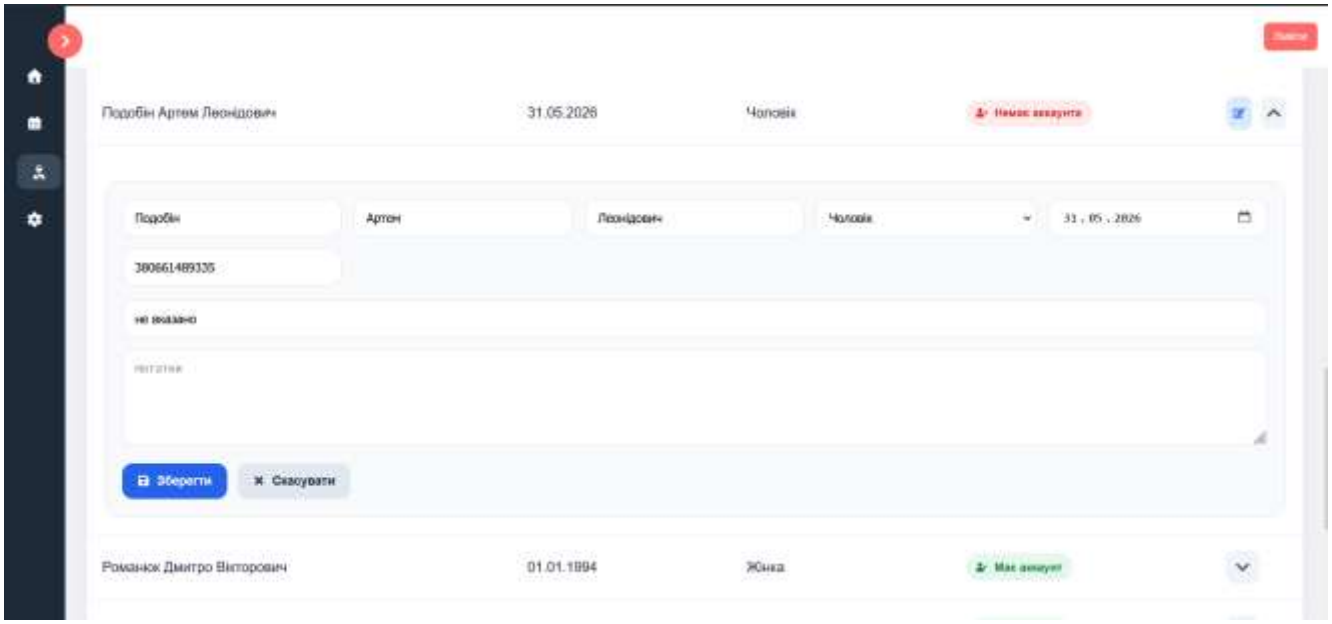
А.4 Сторінка списку прийомів до лікаря у медичній панелі



А.5 Сторінка запису пацієнта на прийом працівником реєстратури



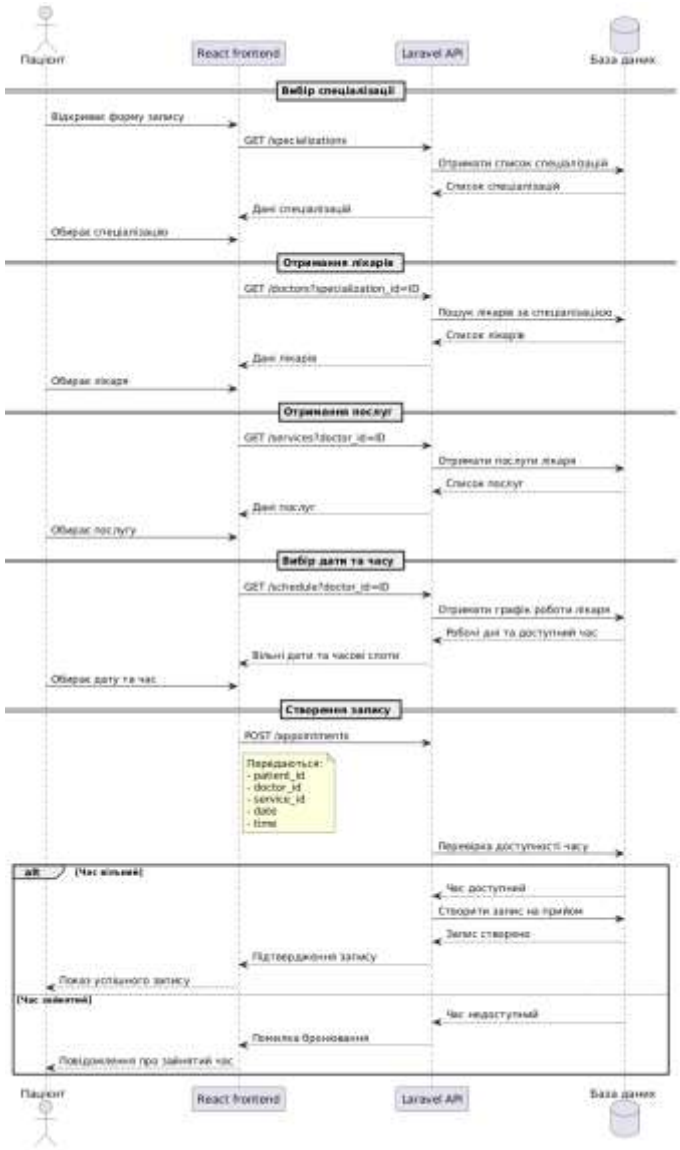
А.6 Сторінка створювання профілю пацієнтів



ДОДАТОК Б

Схеми архітектури системи

Б.1 Діаграма послідовностей створення запису на прийом до лікаря



Б.2 UML-діаграма діяльності процесу онлайн-запису на прийом